



Politecnico di Bari

Repository Istituzionale dei Prodotti della Ricerca del Politecnico di Bari

A multidimensional study of large language models: recommendation behavior, internal representations, and post-training enhancements

This is a PhD Thesis

Original Citation:

A multidimensional study of large language models: recommendation behavior, internal representations, and post-training enhancements / Di Palma, Dario. - ELETTRONICO. - (2026).

Availability:

This version is available at <http://hdl.handle.net/11589/300300> since: 2026-04-27

Published version

DOI:

Publisher: Politecnico di Bari

Terms of use:

(Article begins on next page)



Politecnico
di Bari



DEPARTMENT OF ELECTRICAL AND INFORMATION
ENGINEERING

ELECTRICAL AND INFORMATION
ENGINEERING PH.D. PROGRAM

SSD: IINF-05/A — INFORMATION PROCESSING
SYSTEMS

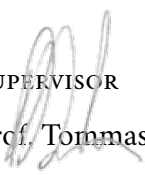
FINAL DISSERTATION

A Multidimensional Study of Large Language Models: Recommendation Behavior, Internal Representations, and Post-Training Enhancements

by

Dario Di Palma 

SUPERVISOR

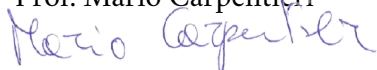

Prof. Tommaso Di Noia

CO-SUPERVISOR

Prof. Fedelucio Narducci
Prof. Vito Walter Anelli

COORDINATOR OF THE PH.D. PROGRAM

Prof. Mario Carpentieri



COURSE XXXVIII, 01/01/2023 — 31/12/2025



LIBERATORIA PER L'ARCHIVIAZIONE DELLA TESI DI DOTTORATO

Al Magnifico Rettore
del Politecnico di Bari

Il sottoscritto Dario Di Palma nato a Canosa di Puglia (BT) il 22/09/1998

residente a Canosa di Puglia (BT) in Via Serg. Magg. Nicola Capurso 102 e-mail dario.dipalma@outlook.it

iscritto al 3° anno di Corso di Dottorato di Ricerca in Ingegneria Elettrica e dell'Informazione ciclo XXXVIII (38°)

ed essendo stato ammesso a sostenere l'esame finale con la prevista discussione della tesi dal titolo:

DICHIARA

- 1) di essere consapevole che, ai sensi del D.P.R. n. 445 del 28.12.2000, le dichiarazioni mendaci, la falsità negli atti e l'uso di atti falsi sono puniti ai sensi del codice penale e delle Leggi speciali in materia, e che nel caso ricorressero dette ipotesi, decade fin dall'inizio e senza necessità di nessuna formalità dai benefici conseguenti al provvedimento emanato sulla base di tali dichiarazioni;
- 2) di essere iscritto al Corso di Dottorato di ricerca Ingegneria Elettrica e dell'Informazione ciclo XXXVIII (38°), corso attivato ai sensi del "Regolamento dei Corsi di Dottorato di ricerca del Politecnico di Bari", emanato con D.R. n.286 del 01.07.2013;
- 3) di essere pienamente a conoscenza delle disposizioni contenute nel predetto Regolamento in merito alla procedura di deposito, pubblicazione e autoarchiviazione della tesi di dottorato nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica;
- 4) di essere consapevole che attraverso l'autoarchiviazione delle tesi nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica del Politecnico di Bari (IRIS-POLIBA), l'Ateneo archiverà e renderà consultabile in rete (nel rispetto della Policy di Ateneo di cui al D.R. 642 del 13.11.2015) il testo completo della tesi di dottorato, fatta salva la possibilità di sottoscrizione di apposite licenze per le relative condizioni di utilizzo (di cui al sito <http://www.creativecommons.it/Licenze>), e fatte salve, altresì, le eventuali esigenze di "embargo", legate a strette considerazioni sulla tutelabilità e sfruttamento industriale/commerciale dei contenuti della tesi, da rappresentarsi mediante compilazione e sottoscrizione del modulo in calce (Richiesta di embargo);
- 5) che la tesi da depositare in IRIS-POLIBA, in formato digitale (PDF/A) sarà del tutto identica a quelle **consegnate**/inviata/da inviarsi ai componenti della commissione per l'esame finale e a qualsiasi altra copia depositata presso gli Uffici del Politecnico di Bari in forma cartacea o digitale, ovvero a quella da discutere in sede di esame finale, a quella da depositare, a cura dell'Ateneo, presso le Biblioteche Nazionali Centrali di Roma e Firenze e presso tutti gli Uffici competenti per legge al momento del deposito stesso, e che di conseguenza va esclusa qualsiasi responsabilità del Politecnico di Bari per quanto riguarda eventuali errori, imprecisioni o omissioni nei contenuti della tesi;
- 6) che il contenuto e l'organizzazione della tesi è opera originale realizzata dal sottoscritto e non compromette in alcun modo i diritti di terzi, ivi compresi quelli relativi alla sicurezza dei dati personali; che pertanto il Politecnico di Bari ed i suoi funzionari sono in ogni caso esenti da responsabilità di qualsivoglia natura: civile, amministrativa e penale e saranno dal sottoscritto tenuti indenni da qualsiasi richiesta o rivendicazione da parte di terzi;
- 7) che il contenuto della tesi non infrange in alcun modo il diritto d'Autore né gli obblighi connessi alla salvaguardia di diritti morali ed economici di altri autori o di altri aventi diritto, sia per testi, immagini, foto, tabelle, o altre parti di cui la tesi è composta.

Luogo e data Bari, 26/04/2026

Firma Dario Di Palma

Il sottoscritto, con l'autoarchiviazione della propria tesi di dottorato nell'Archivio Istituzionale ad accesso aperto del Politecnico di Bari (POLIBA-IRIS), pur mantenendo su di essa tutti i diritti d'autore, morali ed economici, ai sensi della normativa vigente (Legge 633/1941 e ss.mm.ii.),

CONCEDE

- al Politecnico di Bari il permesso di trasferire l'opera su qualsiasi supporto e di convertirla in qualsiasi formato al fine di una corretta conservazione nel tempo. Il Politecnico di Bari garantisce che non verrà effettuata alcuna modifica al contenuto e alla struttura dell'opera.
- al Politecnico di Bari la possibilità di riprodurre l'opera in più di una copia per fini di sicurezza, back-up e conservazione.

Luogo e data Bari, 26/04/2026

Firma Dario Di Palma

*"To me. Never give up on your dream.
If at first you do not succeed,
try again, and again.
And if it still doesn't work,
try something else."*

*"You know what drives me crazy? It's all these people
talking about how great technology is, and how it saves
all this time. But what good is saved time, if nobody uses
it? If it just turns into more busy work. You never hear
somebody say, 'With the time I've saved by using my
word processor, I'm gonna go to a Zen monastery and
hang out.' I mean, you never hear that."*

— Jesse, Before Sunrise

Abstract

Although Large Language Models (LLMs) exhibit strong emergent abilities across a range of general tasks, their behaviour in specialized domains requires further investigation. Their training objective, which focuses on next-token prediction, differs from the optimization criteria of many domain-specific applications, and understanding how pretraining artifacts influence adaptability to specific tasks is essential for effective specialization. Moreover, limited insight into how task-relevant information is encoded within their internal representations hinders reliable adaptation and evaluation. Consequently, the use of LLMs beyond their original training context raises open questions concerning generalization, interpretability, and robustness.

These issues make the integration of LLMs into specialized domains an open research problem. In this thesis, we study how LLMs behave when repurposed beyond their original training objectives and propose principled methods to adapt, interpret, and extend their capabilities. We examine LLMs from three complementary perspectives: their application in Recommender Systems (RSs), the internal mechanisms that shape their representations, and post-training techniques that enhance their performance in complex reasoning tasks.

In the first part of the dissertation, we move beyond standard NLP benchmarks and evaluate LLMs as standalone recommenders. We assess their ability to infer user preferences and re-rank items without task-specific training data, observing performance comparable to classical recommendation systems, strong few-shot re-ranking capabilities, and effectiveness even in cold-start scenarios. We examine whether LLM-generated recommendations align more closely with collaborative filtering or content-based approaches and find a pronounced similarity to collaborative filtering. We also show that LLMs tend to amplify popular but underrepre-

sented items in the dataset, suggesting the presence of broader domain knowledge. In this part we further investigate why LLMs can perform well on recommendation tasks and, for the first time, provides evidence of data leakage from recommender system dataset into LLM pretraining, raising fundamental concerns about the evaluation of LLM-based recommenders. We extend this analysis by leveraging automatic prompt engineering to systematically identify cases in which LLMs have memorized leaked datasets. Finally, we propose a novel method for integrating LLMs into existing recommendation pipelines through symbolic rules that construct user and item profiles. This approach makes LLMs aware of dataset-specific patterns and generates narrative-style descriptions of users and items, showing that LLM-derived profiles can enhance recommendation quality.

The second part of the dissertation examines the internal representations learned by LLMs from a linguistic perspective. We apply probing techniques to study how abstract concepts are encoded within the model’s activation space. We investigate how LLMs represent factual truthfulness in their hidden states, extend prior work, and propose two new and more realistic methods for evaluating factuality. Our results show that detecting hallucinations solely from internal model states is substantially more challenging than previously assumed. We also demonstrate that sentiment information is not uniformly distributed across layers, but is most concentrated in the middle layers, and that the final token is not always the most informative representation for sentiment extraction, contrary to common practice. Finally, we introduce a layer-cutting strategy that reduces memory requirements by an average of 57% while enabling sentiment extraction directly from internal states, achieving up to 14% higher accuracy than sentiment classification performed through prompting.

The final part of the dissertation shifts from task adaptation and interpretability to post-training techniques aimed at improving performance by scaling test-time computation. We introduce a re-ranking module that extends LLM capabilities in complex reasoning scenarios and present a new model that acts as an effective heuristic, improving accuracy by up to 4.33% on challenging Text-to-SQL benchmarks. Our results show that scaling test-time computation through re-ranking outperforms alternative methods and heuristics, demonstrating that meaningful performance gains can be achieved through post-training strategies rather than by increasing model size.

Overall, this thesis provides a multifaceted perspective on LLMs. We adapt them to new tasks, expose the problem of data leakage, interpret their internal behaviour, optimize their memory requirements, and propose a new strategy for enhancing their capabilities through test-time computation.

Publications

Certain ideas and figures presented in this thesis have been previously published in other works. Below is a comprehensive list of the Ph.D. candidate's publications. The candidate's name is highlighted in **boldface** wherever he is the corresponding author or a co-corresponding author.

- [1] **Dario Di Palma**, V. W. Anelli, D. Malitesta, V. Paparella, C. Pomo, Y. Deldjoo, and T. D. Noia. "Examining Fairness in Graph-Based Collaborative Filtering: A Consumer and Producer Perspective". In: *IIR*. Vol. 3448. CEUR Workshop Proceedings. CEUR-WS.org, 2023, pp. 79–84.
- [2] V. Paparella, **Dario Di Palma**, V. W. Anelli, and T. D. Noia. "Broadening the Scope: Evaluating the Potential of Recommender Systems beyond prioritizing Accuracy". In: *RecSys*. ACM, 2023, pp. 1139–1145.
- [3] **Dario Di Palma**. "Retrieval-augmented Recommender System: Enhancing Recommender Systems with Large Language Models". In: *RecSys*. ACM, 2023, pp. 1369–1373.
- [4] V. Paparella, **Dario Di Palma**, V. W. Anelli, A. D. Bellis, and T. D. Noia. "Unveiling the Potential of Recommender Systems through Multi-Objective Metrics". In: *IIR*. Vol. 3802. CEUR Workshop Proceedings. CEUR-WS.org, 2024, pp. 119–122.
- [5] **Dario Di Palma**, G. Servedio, V. W. Anelli, G. M. Biancofiore, F. Narducci, L. Carneio, and T. D. Noia. "Beyond Words: Can ChatGPT Support State-of-the-Art Recommender Systems?" In: *IIR*. Vol. 3802. CEUR Workshop Proceedings. CEUR-WS.org, 2024, pp. 13–22.

- [6] G. M. Biancofiore, **Dario Di Palma**, C. Pomo, F. Narducci, and T. D. Noia. “Conversational User Interfaces and Agents”. In: *Human-Centered AI: An Illustrated Scientific Quest*. Ed. by P. Germanakos, M. Juhasz, A. Kongot, D. Marathe, and D. Sacharidis. Springer Nature, 2025, pp. 399–438. doi: 10.1007/978-3-031-61375-3_4.
- [7] **Dario Di Palma**, G. M. Biancofiore, V. W. Anelli, F. Narducci, and T. D. Noia. “Content-Based or Collaborative? Insights from Inter-List Similarity Analysis of ChatGPT Recommendations”. In: *UMAP (Adjunct Publication). Candidate of the UMAP 2025 Best LBR Paper Award*. ACM, 2025, pp. 28–33.
- [8] **Dario Di Palma**, F. A. Merra, M. Sfilio, V. W. Anelli, F. Narducci, and T. D. Noia. “Do LLMs Memorize Recommendation Datasets? A Preliminary Study on MovieLens-1M”. In: *SIGIR. Recipient of the SIGIR 2025 Best Short Paper Award*. ACM, 2025, pp. 2582–2586.
- [9] **Dario Di Palma**, G. M. Biancofiore, V. W. Anelli, F. Narducci, and T. D. Noia. “Exploring Diversity, Novelty, and Popularity Bias in ChatGPT’s Recommendations”. In: *GENNEXT@SIGIR2025*. CEUR Workshop Proceedings. CEUR-WS.org, 2025.
- [10] **Dario Di Palma**, A. D. Bellis, G. Servedio, V. W. Anelli, F. Narducci, and T. D. Noia. “LLaMAs Have Feelings Too: Unveiling Sentiment and Emotion Representations in LLaMA Models Through Probing”. In: *ACL (1)*. Association for Computational Linguistics, 2025, pp. 6124–6142.
- [11] G. Servedio, A. D. Bellis, **Dario Di Palma**, V. W. Anelli, and T. D. Noia. “Are the Hidden States Hiding Something? Testing the Limits of Factuality-Encoding Capabilities in LLMs”. In: *ACL (1)*. Association for Computational Linguistics, 2025, pp. 6089–6104.
- [12] A. Colacicco, V. Guida, **Dario Di Palma**, F. Narducci, and T. D. Noia. “Exploring Approaches for Detecting Memorization of Recommender System Data in Large Language Models”. In: *SmaLLEXt@CIKM2025*. CEUR Workshop Proceedings. CEUR-WS.org, 2025.
- [13] M. Tritto, G. Farano, **Dario Di Palma**, G. Rossiello, F. Narducci, D. Subramanian, and T. D. Noia. “GradeSQL: Test-Time Inference with Outcome Reward Models for Text-to-SQL Generation from Large Language Models”. In: *Under Review* (2026).
- [14] **Dario Di Palma**, G. Carmosino, R. Smaldini, F. Narducci, and T. D. Noia. “FOL-Mind: Can Infusing First-Order Logic into Large Language Models Improve User Representation in Recommender Systems?” In: *Under Review* (2026).

- [15] G. M. Biancofiore, **Dario Di Palma**, C. Pomo, L. Boratto, T. D. Noia, E. D. Sciascio, and F. Narducci. “A Survey on Information Retrieval and Recommender Systems in the Era of Large Language Models: Two sides of the same Task”. In: *Under Review* (2026).

Contents

Abstract	I
Publications	V
1 Introduction	1
1 Thesis Statement	3
2 Research Contributions	5
2.1 Ch. 6 - Evaluating Large Language Models as Recommender Systems	5
2.2 Ch. 7 - Content-Based or Collaborative? Behavioral Analysis of LLM Recommendations	6
2.3 Ch. 8 and Ch. 9 - Memorization and Data Leakage in LLM-based Recommendation	6
2.4 Ch. 10 - Neuro-Symbolic Integration of LLMs into Recommendation Pipelines	7
2.5 Ch. 11 - Probing Sentiment and Emotion Representations in LLMs .	7
2.6 Ch. 12 - Limits of Factuality Encoding and Hallucination Detection .	8
2.7 Ch. 13 - Post-Training Enhancement via Test-Time Computation . .	8
3 Bibliographical Notes	11

2	Background and Foundations	15
4	Large Language Models	17
4.1	Statistical Language Modeling and Distributed Representations . . .	17
4.1.1	From Symbolic Rules to Statistical Language Models	17
4.1.2	Neural Language Models and Distributed Word Representations	18
4.2	Autoregressive Modeling as the Core Learning Objective	20
4.2.1	Autoregressive Learning and the Next-Token Prediction	20
4.2.2	Likelihood Maximization and Autoregressive Generation . . .	21
4.3	Modeling Long-Range Dependencies: From Fixed Windows to Sequence- to-Sequence Architectures	22
4.3.1	Limitations of Fixed-Context Neural Models	22
4.3.2	Sequence-to-Sequence Architectures for Variable-Length De- pendencies	23
4.4	Adaptive Context and the Emergence of Attention Mechanisms . . .	25
4.4.1	The Context Vector Bottleneck in Encoder–Decoder Models .	25
4.4.2	Attention as Adaptive Context Construction	25
4.5	The Emergence of the Transformer Architecture	27
4.5.1	Limitations of Recurrent Attention-Based Models	27
4.5.2	Transformers and Fully Attentional Sequence Modeling	27
4.6	From Transformer to Large-Scale Pre-Training models	29
4.6.1	Generative Pre-Training and Universal Language Represen- tations	29
4.6.2	Fine-Tuning, Prompting, and Multi-Stage Model Adaptation .	32
4.7	Emergent Abilities and Prompt-Engineering Techniques	33
4.7.1	The Non-Linear Scaling Effects	33
4.7.2	Prompt Engineering as a Mechanism for Behavioral Control .	34
5	Recommender Systems	37
5.1	Definition of the Recommendation Problem	38
5.2	Recommendation Techniques	39
5.2.1	Collaborative Filtering Approaches	40
5.2.2	Content-based Approaches	42
5.2.3	Hybrid Approaches	43
5.3	Recommendation Pipeline	44

5.3.1	Input Pre-processing	44
5.3.2	Model Optimization	45
5.3.3	Model Evaluation	46
5.4	Multiple Perspectives of Recommendation	50
5.4.1	User-Centric Objectives	50
5.4.2	Multi-Stakeholder Objectives	53
3	Large Language Models for Recommendation	55
6	Evaluating ChatGPT as a Recommender System: A Multi-Perspective Study	57
6.1	Introduction	58
6.2	An Evaluation Pipeline for Assessing ChatGPT in Recommendation Tasks	61
6.2.1	Prompt Design	61
6.2.2	Pre-processing Phase	65
6.2.3	Evaluation Pipeline	66
6.2.4	Assessing ChatGPT performance	69
6.3	Experiments	70
6.3.1	Experimental Setting	71
6.3.2	Baseline Models	72
6.3.3	Top-N Recommendation Performance	74
6.3.4	Reranking Evaluation	80
6.3.5	Cold-Start Scenario	82
6.4	Related Work	86
6.5	Summary	90
7	Content-Based or Collaborative? Insights from Inter-List Similarity Analysis of ChatGPT Recommendations	95
7.1	Introduction	96
7.2	Methodology	97
7.3	Experimental Settings	99
7.4	Results & Discussion	101
7.5	Summary	104

8	Do LLMs Memorize Recommendation Datasets? A Preliminary Study on MovieLens-1M	107
8.1	Introduction	108
8.2	Methodology	109
8.2.1	Definitions of Memorization	109
8.2.2	Evaluation Protocol	110
8.3	Results and Discussion	112
8.3.1	Analysis of Memorization	112
8.3.2	Impact on Recommendation	113
8.3.3	Impact of Model Scale	115
8.3.4	Popularity Memorization	115
8.4	Summary	116
9	Exploring Approaches for Detecting Memorization of Recommender System Data in Large Language Models	117
9.1	Introduction	118
9.2	Methodology	119
9.2.1	Data and Preliminaries	119
9.2.2	Jailbreak Prompt Engineering	120
9.2.3	Unsupervised Latent Knowledge Discovery	120
9.2.4	Automatic Prompt Engineering	122
9.2.5	Comparison of Methods	123
9.3	Results and Discussion	123
9.3.1	Manual Probing	123
9.3.2	Unsupervised Latent Knowledge Discovery	124
9.3.3	Automatic Prompt Engineering	125
9.3.4	Cross-Method Synthesis	126
9.4	Summary	126
10	FOLMind: Can Infusing First-Order Logic into Large Language Models Improve User Representation in Recommender Systems?	133
10.1	Introduction	134
10.2	Related Work	136
10.2.1	Neuro-Symbolic Recommender Systems	136

10.2.2	Large Language Models for Recommendation	137
10.3	Problem Formulation	138
10.3.1	The FOLMind Framework	139
10.3.2	Hybrid Recommendation Strategies	142
10.4	Experimental Setup	144
10.5	Results and Discussion	148
10.6	Ablation Studies	150
10.7	Conclusion and Limitations	154
4	Beyond Recommendation: LLMs Interpretability, Text-to-SQL and Test-time scaling	165
11	LLaMAs Have Feelings Too: Unveiling Sentiment and Emotion Representations in LLaMA Models Through Probing	169
11.1	Introduction	170
11.2	Related Work	171
11.2.1	Sentiment Analysis Meets LLMs	171
11.2.2	Probing and LLMs	172
11.2.3	Probing LLMs for Sentiment Analysis	173
11.3	Methodology	174
11.4	Experimental Results	177
11.5	SENTRI _{LLAMA} for Efficient Downstream Tasks	180
11.6	Discussion	184
11.7	Summary	184
12	Are the Hidden States Hiding Something? Testing the Limits of Factuality-Encoding Capabilities in LLMs	199
12.1	Introduction	200
12.2	Reproduction of Prior Work: Settings	201
12.2.1	Dataset Generation Strategy	201
12.2.2	Internal States Analysis via SAPLMA	202
12.2.3	Reproducibility Settings	202
12.3	Experimental Reproducibility Results	205
12.3.1	Reproduction of SAPLMA Results on the True-False Dataset	205

12.3.2	Reproduction of the Results on the OPT-Generated Dataset . .	206
12.4	Novel Dataset Generation Strategies for Factuality Self-Evaluation .	206
12.4.1	Perplexity-based Dataset Construction	207
12.4.2	LLM-Generated Dataset Construction	209
12.5	Experiments and Discussion	210
12.5.1	Dataset Generation Setup	210
12.5.2	Impact of Perplexity-Based Sampling on SAPLMA Accuracy .	212
12.5.3	Generalization of SAPLMA on LLM-generated Sentences . . .	215
12.6	Related Work	216
12.7	Summary	216
13	GradeSQL: Test-Time Inference with Outcome Reward Models for Text-to-SQL Generation from Large Language Models	231
13.1	Introduction	232
13.2	Related Work	235
13.2.1	Text-to-SQL Meets LLMs	235
13.2.2	Test-Time Inference Strategies	236
13.3	Methodology	238
13.3.1	Problem Formulation.	238
13.3.2	GradeSQL Framework	240
13.3.3	ORM Inference and Probabilistic Scoring Mechanism	241
13.4	Experimental Setup	242
13.5	Results and Discussion	245
13.6	Ablation Studies	250
13.7	Summary	257

5 Closing Remarks

List of Figures

4.1	The matrix $C \in \mathbb{R}^{ V \times D}$ encodes the vocabulary $V = \{1, \dots, V \}$ in a continuous vector space. Each row corresponds to a D -dimensional embedding representing a lexical item, commonly referred to as a <i>word embedding</i>	19
4.2	Visualization of algebraic operations in the word embedding space. The vectors corresponding to <i>king</i> , <i>man</i> , <i>woman</i> , and <i>queen</i> illustrate how the computation “king - man + woman” yields a point close to “queen”, demonstrating that the embedding space encodes meaningful linguistic structure.	21
4.3	Illustration of a sequence-to-sequence model for neural machine translation. The encoder processes a variable-length English sentence into a latent representation, and the decoder generates a variable-length Italian translation conditioned on this representation	23
4.4	The transformer architecture consists of an encoder and a decoder equipped with positional encodings and multi-head self-attention. The decoder employs masked self-attention to preserve left-to-right generation, while cross-attention enables decoder states to retrieve source-side information relevant to each output token.	30
5.1	Rating prediction scenario. The left matrix shows full observations, while the right matrix contains unobserved entries that the RS must estimate . .	38
5.2	Implicit-feedback scenario. The system must infer which missing entries correspond to likely positive interactions.	39

5.3	Latent factor decomposition of the user-item interaction matrix into item and user embedding matrices.....	41
5.4	Long-tail distribution of item popularity. A small set of highly frequent items forms the short head, while the majority of items occur rarely in the long tail. The dashed line marks the head–tail cutoff.....	51
6.1	Multi-Stage Evaluation Pipeline for Assessing ChatGPT Recommendation Performance.....	68
6.2	Kiviat diagrams comparing ChatGPT models with Best-CF, Best-CBF, and MostPop across datasets. Higher values indicate better performance.	79
6.3	Grouped bar charts showing baseline nDCG and nDCG improvements of ChatGPT models across three domains. Higher improvements indicate better re-ranking and personalization performance.....	81
6.4	Mistral-7B few-shot prompt for item identity matching.....	91
6.5	ChatGPT response example	92
6.6	ChatGPT duplication error	93
6.7	ChatGPT extreme duplication error	94
7.1	Comparison of similarity metrics: (a) Jaccard Index, (b) Rank Biased Overlap.....	102
7.2	Comparison of most recommended items by traditional and generative models. The outer ring shows overall recommendation frequency, while colored segments indicate model family shares. Lighter grey denotes higher item frequency, darker blue lower frequency	103
8.1	Few-Shot Prompting for Item/User Data Extraction.....	110
8.2	Few-Shot Prompting for Interactions Data Extraction.....	110
8.3	Zero-Shot Prompting for Recommendation Task	113
8.4	Comparison of item coverage across models by popularity tier. The figure shows the percentage of items covered in three categories: Highly Popular (Top 20%), Moderately Popular (Middle 20%), and Rarely Interacted (Bottom 20%).....	116

9.1 Jailbreak prompt-engineering workflow: select dataset samples; craft a prompt with few-shot examples and chain-of-thought instructions incorporating jailbreaking prompt; query the LLM; manual analysis of responses for validity, duplication, and hallucinations; then iterate on the prompt.....	121
9.2 Context Compliance Attack (CCA) jailbreak prompt tailored for MovieLens-1M memorization discovery. The fabricated interaction history primes the model to output exact dataset fields when queried with keys.....	129
9.3 Unsupervised latent knowledge discovery via CCS. Positive and negative statements are processed by the LLM to extract hidden activations, which are paired and fed to a linear probe optimized to assign high scores to true statements and low scores to false ones.....	130
9.4 PCA visualizations of the dataset. (a) Item embeddings. (b) User embeddings.....	131
11.1 Layer-wise probing accuracy (Last-Token) on SST-2. Refer to Supplementary Material Fig. 11.6 for comprehensive results across all datasets.....	178
11.2 Mean Pooling Visually Explained.....	192
11.3 Last Token Pooling Visually Explained.....	192
11.4 Max and Min Pooling Visually Explained.....	192
11.5 Attention Pooling Visually Explained.....	193
11.6 Layer-wise probing accuracy using the last token on SST-2, IMDB, Rotten Tomatoes, and Emotion.....	194
11.7 Layer-wise pooling methods confidence accuracy on SST-2, IMDB, Rotten Tomatoes, and Emotion.....	195
12.1 Comparison of average perplexity scores for Llama 2-7b and OPT-6.7b for the Original dataset by Azaria et al. [25] and our refined version. Lower perplexity indicates that the sentences are more likely to have been generated by the model. PP(T/F) denotes average perplexity of true/false sentences.....	211
12.2 ChatGPT factual question answering example.....	222
12.3 ChatGPT unanswered factual query example.....	222
12.4 ChatGPT evaluation and answer-judging prompt example.....	224

13.1 Overview of the GradeSQL framework for training an ORM. The framework consists of three stages: (i) Candidate Generation, an LLM generates N SQL queries via a Chain-of-Thought (CoT) prompt, then removes the CoT trace; (ii) Data Labeling, each candidate is annotated as correct or incorrect; and (iii) Supervised Fine-Tuning (SFT), the base LLM is fine-tuned with LoRA on the labeled data to obtain the final ORM.	239
13.2 Execution accuracy (%) on <i>BIRD dev</i> across different N values, comparing execution-based Best-of- N (ex-BoN), Majority Voting (Maj), and ORM-based Best-of- N (ORM-BoN). Results are shown by query difficulty (Simple, Moderate, Challenging) and overall (Total)	251
13.3 Overview of the GradeSQL ORM at inference. The pipeline consists of two stages: (i) Candidate Generation, where an LLM produces a pool of N SQL candidates; and (ii) Query Selection, where the ORM scores each candidate and ranks them, selecting the query with the highest probability as the final output.	261

List of Tables

6.1	Dataset statistics before and after pre-processing with max context length filtering and $k - core \geq 10$	65
6.2	Comparison of ChatGPT-3.5 under the old and new evaluation pipelines across Accuracy, Diversity, Novelty, and Bias. The symbols $\uparrow$ and $\downarrow$ denote the preferred directions for each metric. Best values are shown in bold....	66
6.3	Overview of beyond-accuracy metrics [18].....	70
6.4	Top-N evaluation of ChatGPT-3.5 and ChatGPT-4 on Facebook Books across Accuracy, Diversity, Novelty, and Popularity Bias. Arrows indicate preferred directions. Best values are in bold, second-best are underlined. Results are ordered by nDCG. Baselines are statistically significant ($p < 0.05$) except values marked with * (ChatGPT-3.5) and $\dagger$ (ChatGPT-4). Best-CF and Best-CBF denote the strongest collaborative and content-based baselines by nDCG	75
6.5	Top-N evaluation of ChatGPT-3.5 and ChatGPT-4 on Last.FM across Accuracy, Diversity, Novelty, and Popularity Bias. Arrows indicate preferred directions. Best values are in bold, second-best are underlined. Results are ordered by nDCG. Baselines are statistically significant ($p < 0.05$) except values marked with * (ChatGPT-3.5) and $\dagger$ (ChatGPT-4). Best-CF and Best-CBF denote the strongest collaborative and content-based baselines by nDCG	77

6.6	Top-N evaluation of ChatGPT-3.5 and ChatGPT-4 on MovieLens across Accuracy, Diversity, Novelty, and Popularity Bias. Arrows indicate preferred directions. Best values are in bold, second-best are underlined. Results are ordered by nDCG. Baselines are statistically significant ($p < 0.05$) except values marked with * (ChatGPT-3.5) and † (ChatGPT-4). Best-CF and Best-CBF denote the strongest collaborative and content-based baselines by nDCG	78
6.7	Comparison of user cold-start interactions (up to ten interactions per user) for ChatGPT-3.5 and ChatGPT-4. Arrows indicate preferred directions. Best values are in bold, second-best are underlined. CF and CBF denote collaborative and content-based recommenders. Facebook Books baselines are statistically significant ($p < 0.05$) except values marked with * (ChatGPT-3.5) and † (ChatGPT-4). Best-CF and Best-CBF correspond to the strongest CF and CBF by nDCG	84
7.1	Dataset statistics after $k - core \geq 10$. The selected types of content for CBF follow standard practices [18].	99
8.1	Coverage of movies.dat, users.dat, and ratings.dat. Models are grouped by version and ordered by size	112
8.2	Recommendation accuracy performance on standards recommended and LLM-based recommendation. LLMs are grouped by model family and sorted by size. Best performance in each family is shown in bold	114
9.1	Statistics of the MovieLens-1M dataset.	120
9.2	Comparison of techniques for detecting memorization in LLMs. Manual prompting uses few-shot or jailbreaking strategies, unsupervised methods require access to activations, and APE iteratively generates and scores prompts.	123
9.3	Unsupervised membership inference results on MovieLens-1M using CCS and Cluster-Norm. Each entry reports balanced accuracy	124
9.4	Automatic Prompt Engineering results on MovieLens-1M. Values denote exact-match accuracy at different sampling temperatures, with higher values indicating better extraction. (–) denotes values below 0.1%, which are not reported	125

10.1 Dataset statistics before and after preprocessing	144
10.2 Statistics of AMIE-mined rules from Knowledge Graphs. #Rules: total rules; Avg. Len.: average body atoms; Rel.: unique relations; Max Len.: maximum body length	146
10.3 Performance comparison of baseline models versus their Linq MISTRAL embedding-enhanced counterparts across four domains. Values are reported as (nDCG, Recall, Precision). Green indicates improvement, red indicates degradation	147
10.4 Ablation study on FOL vs. NoFOL Linq Mistral embeddings for nDCG. The relative change (Δ) denotes the difference from NoFOL to FOL across all datasets. Positive Δ values indicate improvement, while negative values indicate degradation	151
10.5 Performance comparison of the SpaceInit strategy on NeuMF using Qwen embeddings of different scales (0.6B, 4B, 8B) across four datasets. Best results are in bold, and runner-ups are underlined	153
10.6 Performance comparison across Baseline, SBERT-MPNET, GEMMA, and SNOWFLAKE variants. Rows within each model group are sorted ascending by nDCG on MovieLens Small. Best values are in bold, runner-ups are underlined	162
10.7 Performance comparison of LLM-based integration strategies (Blend, Inject) using QWEN embeddings of different scales (0.6B, 4B, 8B) across four benchmark datasets. For each column, the best result within each model group and dataset is shown in bold, and the second-best (runner-up) is underlined	163
11.1 Clusters of Classifiers Based on Model Type	176
11.2 Dataset statistics	177
11.3 Probing Results Across Different Pooling Methods. See Supplementary Material 11.7 for confidence level plots across all layers and datasets	179
11.4 Comparison of SENTRILLAMA against DeBERTa, RoBERTa, and prompt-based method	182
11.5 Comparison of SENTRILLAMA Sizes Across Datasets	182
11.6 Computational efficiency comparison on the SST-2 dataset. See Supplementary Material 11.7 for additional dataset results	182

11.7 Computational efficiency comparison of SENTRI _{LLAMA} , DeBERTa, RoBERTa, and a prompt-based method on the IMDB dataset	196
11.8 Computational efficiency comparison of SENTRI _{LLAMA} , DeBERTa, RoBERTa, and a prompt-based method on the Rotten Tomatoes dataset	196
11.9 Computational efficiency comparison of SENTRI _{LLAMA} , DeBERTa, RoBERTa, and a prompt-based method on the Emotion dataset	197
12.1 Replicated SAPLMA performance on the True-False dataset across the selected layers. The results labeled as ‘Orig.’ are taken from the original work, while those labeled as ‘Repr.’ are the replicated results from this study	203
12.2 Replicated baselines performance on the True-False dataset. The results labeled as ‘Orig.’ are taken from the original work, while those labeled as ‘Repr.’ are the replicated results from this study	204
12.3 Reproduced SAPLMA performance on the OPT-Generated Dataset (Section 12.2.1). The results labeled as ‘Orig.’ are taken from the original work, while those labeled as ‘Repr.’ are the reproduced results.	204
12.4 Novel generation of our True-False dataset, following the approach described in Section 12.4.1, including number of sentences and percentage of true samples	211
12.5 Accuracy obtained training SAPLMA on the original or refined True-False dataset and evaluating on the refined version. “Orig.” and “Novel” denote the original and refined datasets. Bold highlights the best layer-training dataset combination for each model-topic pair	213
12.6 Performance of SAPLMA on a fact dataset generated from TriviaQA. The original topic-wise leave-one-out strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models	214
12.7 True-False dataset categories, original sources, and label splits from [25] .	220
12.8 Summary of the dataset obtained by extracting factoid sentences from TriviaQA [173] Wikipedia validation split, following the procedure described in Section 12.4.2	220
12.9 Examples of true and false facts generated by Llama 2-7b and OPT-6.7b based on the questions in TriviaQA [173], obtained following the procedure detailed in Section 12.4.2.	221

12.10 Examples of false sentences generated with the Llama 2-7b model, using the perplexity-based generation strategy described in Section 12.4.1	221
12.11 Accuracy values obtained training SAPLMA on the original True-False dataset and testing on our facts dataset generated from TriviaQA. The original topic-wise leave-one-out strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models	225
12.12 Summary of the dataset obtained by extracting factoid sentences from TruthfulQA [217], following the procedure described in Section 12.4.2	227
12.13 Performance of SAPLMA on a fact dataset generated from TruthfulQA. A 50-50 holdout strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models	227
12.14 Accuracy values obtained training SAPLMA on the original True-False dataset and testing on our facts dataset generated from TruthfulQA. A 50-50 holdout strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models	227
12.15 Summary of the dataset obtained by extracting factoid sentences from SQuAD 2.0 [295], following the procedure described in Section 12.4.2	228
12.16 Performance of SAPLMA on a fact dataset generated from SQuAD 2.0, restricted to the validation set and the top-5 most popular topics. The original topic-wise leave-one-out strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models	228
12.17 Accuracy values obtained training SAPLMA on the original True-False dataset and testing on the facts dataset generated from SQuAD 2.0. Results are shown for the Llama 2-7b and OPT-6.7b models	229
12.18 Examples of sentence classification on facts generated from the TriviaQA dataset using Llama 2-7b. The original topic-wise leave-one-out strategy is adopted. Correct predictions are highlighted in green, while incorrect ones are shown in red	229
12.19 Examples of predictions using SAPLMA and Llama 2-7b. In these examples, the probe classifier is trained on the original version of the True-False dataset [25] (excluding the <i>element</i> topic) and tested on our refined version of the <i>element</i> topic. Correct predictions are highlighted in green, while incorrect ones are shown in red	230

13.1 Reproducibility results of OmniSQL-7B. Maj = Majority Voting (temperature $T=0.8$, $N=8$); Gre = Greedy decoding ($N=1$). The table compares our reproduction (Our) with the original results (Origin) on the BIRD dev, Spider dev, and Spider test benchmarks	244
13.2 Dataset statistics for <i>BIRD</i> and <i>Spider</i> (training splits). We report (i) the number of questions before and after de-duplication (with removals Δ), and (ii) the final size of the labeled sets generated by sampling $N = 32$ candidates per question. Percentages indicate the class distribution (<i>Incorrect</i> / <i>Correct</i>) in the <i>imbalanced</i> set; the balanced set is 50/50.....	245
13.3 ORM performance on <i>BIRD</i> and <i>Spider</i> with the unbalanced dataset, reporting <i>Execution Accuracy</i> (%) on dev and test sets	246
13.4 Execution accuracy (%) of ORM models on <i>BIRD dev</i> , <i>Spider dev</i> , and <i>Spider test</i> . Each cell shows results on <i>Unbalanced</i> and <i>Balanced</i> sets with Δ (green $\uparrow$ = gain, red $\downarrow$ = drop). Ordered by BIRD Unbalanced; best in bold, runner-up underlined.....	247
13.5 Execution accuracy (%) of OmniSQL-7B on <i>BIRD dev</i> , <i>Spider dev</i> , and <i>Spider test</i> . Δ indicates the gain over the baseline ($N=1$). Best and runner-up results are in bold and <u>underlined</u> , respectively. Values marked with $\dagger$ are not McNemar significant ($p < 0.05$), and those with * lose significance after Bonferroni correction.....	249
13.6 Execution accuracy (%) and Pass@ N of test-time strategies (execution-based BoN, Majority Voting, ORM-based BoN) across varying N values on the <i>BIRD dev</i> , <i>Spider dev</i> , and <i>Spider test</i> benchmarks. Columns are color-coded from light yellow (lowest) to dark orange (highest) performance.	250
13.7 Execution accuracy (%) and Pass@32 on the <i>BIRD dev</i> , <i>Spider dev</i> , and <i>Spider test</i> . Candidate queries are generated by OmniSQL models with 7B, 14B, and 32B parameters, while the ORM is implemented using OmniSQL-7B. Best scores per column are shown in bold	253
13.8 Ablation study on verification prompt design for ORM on <i>BIRD dev</i> . Execution accuracy (%) by query difficulty. Generator: OmniSQL-7B; ORM: Qwen2.5-7B-Instruct. Best scores in bold.....	254

13.9 Impact of ORM model size with the generator fixed to OmniSQL-7B. Execution Accuracy (EX, %) and Pass@32 are reported on <i>BIRD dev</i> , <i>Spider dev</i> , and <i>Spider test</i> . Best results per metric and dataset are shown in bold	255
13.10 Ablation study comparing autoregressive and Binary Cross-Entropy (BCE) loss fine-tuning on <i>BIRD dev</i> across query difficulty levels. Both the generator and ORM model are OmniSQL-7B. Reported metric is Execution Accuracy (%). Best results are in bold.....	256

Part 1

Introduction

Over the past decades, successive technological paradigms have reshaped the global innovation landscape [104, 284], including the rise of personal computing in the 1980s, the expansion of the internet in the 1990s, the emergence of Web 2.0 and social media in the 2000s, and the growth of mobile and cloud technologies in the 2010s. In the 2020s, Large Language Models (LLMs) represent the latest of these transformative shifts [42], whose impact is already visible in their widespread adoption across everyday and professional contexts, as people around the world increasingly rely on them for tasks ranging from answering questions [176] and offering emotional support [109] to assisting with complex professional and creative activities [108].

This widespread adoption marks a historic turning point in the role of Artificial Intelligence (AI) in society, as AI is no longer confined to experts or restricted to invisible background services [57], but instead emerges as an accessible, conversational technology that reshapes human–computer interaction by recasting AI from a passive infrastructure component into an active collaborator, in ways that were previously confined to works of fiction [50].

The foundations of this transformation lie in the remarkable ability of these models to act as powerful general-purpose predictors, excelling across a wide range of tasks, producing coherent and contextually appropriate text, and exhibiting increasingly sophisticated forms of reasoning [49]. These capabilities emerge from large-scale pretraining on diverse corpora, which gives rise to emergent abilities [381], including in-context learning. This ability allows models to adapt their behavior at inference time by conditioning on examples or instructions provided within the input prompt, without requiring parameter updates or additional training [255]. As a result, LLMs can perform tasks such as classification, translation, and structured code generation without task-specific supervision, making them effective tools for

a broad range of downstream problems that were not explicitly targeted during pretraining [68, 313].

However, this versatility introduces both opportunities and risks [35, 159]. On the one hand, LLMs can be easily deployed across diverse application domains and often achieve strong performance with minimal task-specific adaptation [290, 405]. On the other hand, they remain vulnerable to systematic failure modes, including hallucinations and misleading yet fluent outputs [159, 297, 427], which can obscure underlying limitations and create an illusion of robustness and understanding. As a consequence, deploying LLMs in high-stakes or domain-sensitive settings requires more than simply prompting a powerful general model. It necessitates a careful examination of failure mechanisms, the stability of model behaviour under distributional shifts, and the integration of domain-specific constraints.

Existing work has begun to investigate some of these issues [89, 168, 287]. However, much of the current evidence on LLM capabilities and limitations remains tied to NLP-specific tasks, leaving open questions about how reliably LLMs transfer to specialized domains such as recommendation, where models must learn user preferences and handle long-tail and sparse data. As a result, aligning and studying the capabilities of broadly pretrained LLMs with the precise, domain-specific requirements of such systems remains an open and active area of research.

Furthermore, there is growing recognition that progress cannot rely solely on increasing model scale, but must instead focus on deploying LLMs as reliable components within complex pipelines. This requires a more principled understanding of their behaviour under task repurposing, as well as methods that adapt or extend LLMs in ways that are controllable and empirically grounded [112, 388, 405, 437].

These open questions highlight a critical gap between what current LLMs can achieve in principle and what is demanded by domain-specific applications in practice. Bridging this gap requires moving beyond treating LLMs as opaque, general-purpose tools and instead developing frameworks to analyze, adapt, and integrate them in a principled manner. Addressing this challenge provides the broader context and motivation for the research presented in this thesis.

Chapter 1

Thesis Statement

This thesis investigates how Large Language Models (LLMs) behave when repurposed beyond their original next-token prediction objective, with a particular focus on Recommender Systems and other structured decision-making tasks. It advances the hypothesis that LLMs can be effectively adapted to specialized domains without full retraining, but that such reuse exposes fundamental limitations related to memorization, interpretability, and robustness.

Through a multidimensional lens, this work examines LLMs from three complementary perspectives. First, it analyzes LLMs as standalone recommender systems, evaluating their ability to infer user preferences, re-rank items, and operate under sparse and cold-start conditions, while revealing systematic behavioral patterns, popularity amplification effects, and concrete evidence of data leakage from recommendation datasets into model pretraining. Second, it examines the internal representations learned by LLMs through probing methodologies, demonstrating how abstract concepts such as sentiment and factuality are distributed across layers, challenging common assumptions about representation locality, and proposing memory-efficient strategies for extracting task-relevant information directly from hidden states. Third, it explores post-training enhancement techniques based on test-time computation, showing that re-ranking and outcome-based selection strategies can significantly improve performance on complex reasoning tasks such as Text-to-SQL without increasing model size.

Overall, this thesis argues that understanding, adapting, and extending LLMs beyond their training context requires a joint analysis of external behavior, internal representations, and inference-time optimization. By combining empirical evalua-

tion, interpretability analysis, and post-training methodologies, the work provides principled insights into both the opportunities and the risks of deploying LLMs in high-stakes, structured application domains.

Chapter 2

Research Contributions

This section provides a concise yet comprehensive overview of the research contributions presented in this thesis, organized by thematic chapters. For each chapter, a brief summary of the content is provided, along with details about the related publications and the role of the Ph.D. candidate, Dario Di Palma, in the contributions to these works.

2.1 Ch. 6: Evaluating Large Language Models as Recommender Systems

Contributions. Chapter 6 investigates whether Large Language Models can be meaningfully repurposed as Recommender Systems without task-specific training. We introduce a rigorous evaluation pipeline that adapts classical Top- N recommendation protocols to generative models. The analysis covers accuracy, diversity, novelty, and popularity bias, as well as re-ranking and cold-start scenarios. Our results show that, with appropriate prompting, LLMs can infer user preferences and achieve performance comparable to strong collaborative filtering baselines. At the same time, we observe systematic behavioral patterns, including a strong alignment with collaborative filtering signals and a tendency to amplify popularity effects, rather than discovering genuinely novel preference structures. These findings provide one of the first comprehensive empirical characterizations of LLMs as standalone recommenders.

Publications. This chapter covers the topics explored in “Retrieval-augmented Recommender System: Enhancing Recommender Systems with Large Language Mod-

els”, “Beyond Words: Can ChatGPT Support State-of-the-Art Recommender Systems?”, and “Exploring Diversity, Novelty, and Popularity Bias in ChatGPT’s Recommendations” published at the RecSys’23, IIR’24, and GENNEXT@SIGIR’25.

Ph.D. Candidate’s role. The Ph.D. candidate designed the evaluation pipeline, conducted all experiments, led the analysis, and led the writing of the papers.

2.2 Ch. 7: Content-Based or Collaborative? Behavioral Analysis of LLM Recommendations

Contributions. Chapter 7 analyzes the nature of recommendations produced by LLMs by comparing them with classical content-based and collaborative filtering approaches. We introduce an inter-list similarity analysis framework to quantify whether LLM-generated recommendation lists resemble one paradigm more than the other. The results show that LLM recommendations are significantly closer to collaborative filtering outputs, despite being generated from textual prompts alone. This chapter provides novel insights into the implicit signals leveraged by LLMs and clarifies the extent to which their behavior can be explained by memorized interaction patterns rather than semantic content understanding.

Publications. The chapter is based on “Content-Based or Collaborative? Insights from Inter-List Similarity Analysis of ChatGPT Recommendations,” published at UMAP’25 and nominated for the Best LBR Paper Award.

Ph.D. Candidate’s role. The Ph.D. candidate proposed the methodology, designed and conducted the experimental evaluation, and was the lead author of the paper.

2.3 Ch. 8 and Ch. 9: Memorization and Data Leakage in LLM-based Recommendation

Contributions. Chapters 8 and 9 address a critical and previously underexplored issue in LLM-based recommendation, namely data leakage and memorization of recommender system datasets during pretraining. We provide the first systematic empirical evidence that interactions from benchmark datasets such as MovieLens-1M are memorized by LLMs. This finding challenges common evaluation practices and raises concerns about inflated performance claims. To address this problem, we propose multiple techniques for detecting memorization, including manual prob-

ing, automatic prompt engineering, and unsupervised latent knowledge discovery. The comparative analysis highlights the strengths and limitations of each approach and establishes practical guidelines for leakage detection in future studies.

Publications. These chapters cover the work presented in “Do LLMs Memorize Recommendation Datasets? A Preliminary Study on MovieLens-1M,” recipient of the SIGIR'25 Best Short Paper Award, and “Exploring Approaches for Detecting Memorization of Recommender System Data in Large Language Models,” published at SmaLLEXT@CIKM'25.

Ph.D. Candidate's role. The Ph.D. candidate led the study design, supervised the initial exploration, extended the experimental evaluation and the interpretation of the results, and led the writing of the article.

2.4 Ch. 10: Neuro-Symbolic Integration of LLMs into Recommendation Pipelines

Contributions. Chapter 10 introduces FOLMIND, a novel neuro-symbolic framework that integrates First-Order Logic with LLMs to improve user representation in recommender systems. The framework generates narrative user and item profiles enriched with symbolic rules, enabling controlled and auditable integration of LLMs into existing pipelines. Extensive experiments show that LLM-derived representations, when guided by symbolic constraints, can enhance recommendation quality across multiple domains. This contribution bridges LLM-based recommendation and knowledge-driven recommender systems.

Publications. The chapter is based on “FOLMind: Can Infusing First-Order Logic into Large Language Models Improve User Representation in Recommender Systems?” currently under review.

Ph.D. Candidate's role. The Ph.D. candidate proposed the framework, supervised the implementation of the models and the experiments, and led the writing.

2.5 Ch. 11: Probing Sentiment and Emotion Representations in LLMs

Contributions. Chapter 11 studies how sentiment and emotion information is encoded within the internal representations of LLMs. Through extensive probing ex-

periments across layers and pooling strategies, we show that sentiment is not uniformly distributed throughout the network but is concentrated in middle layers. Contrary to common practice, the final-token representation is often suboptimal for sentiment extraction. Building on these insights, we introduce a layer-cutting strategy that significantly reduces memory usage while preserving or improving classification performance. We further propose a probing-based sentiment classification framework that outperforms prompting-based approaches.

Publications. This chapter covers “LLaMAs Have Feelings Too: Unveiling Sentiment and Emotion Representations in LLaMA Models Through Probing,” published at ACL’25.

Ph.D. Candidate’s role. The Ph.D. candidate designed the study, conducted the experiments, and led the analysis and writing.

2.6 Ch. 12: Limits of Factuality Encoding and Hallucination Detection

Contributions. Chapter 12 investigates whether factual truthfulness is reliably encoded in LLM hidden states. We reproduce and extend prior probing-based hallucination detection methods and introduce two new evaluation protocols that reflect more realistic generation settings. The results show that factuality detection from internal states is considerably more challenging than previously reported. Performance degrades under distribution shifts and noisy prompts, revealing important limitations of existing interpretability claims.

Publications. The chapter is based on the work presented in “Are the Hidden States Hiding Something? Testing the Limits of Factuality-Encoding Capabilities in LLMs,” published at ACL’25.

Ph.D. Candidate’s role. The Ph.D. candidate co-designed the study and made substantial contributions to the analysis of the results and to the writing of the paper.

2.7 Ch. 13: Post-Training Enhancement via Test-Time Computation

Contributions. Chapter 13 focuses on improving LLM performance through post-training strategies that scale test-time computation rather than model size. We in-

introduce GRADESQL, a re-ranking framework based on Outcome Reward Models for Text-to-SQL. The method evaluates and reorders candidate reasoning chains generated by an LLM, selecting the most plausible solution. Experiments on challenging benchmarks show accuracy gains of up to 4.33 percent without modifying model parameters or performing additional fine-tuning. This chapter demonstrates that significant performance improvements can be achieved through structured inference and test-time optimization.

Publications. The chapter covers “GradeSQL: Test-Time Inference with Outcome Reward Models for Text-to-SQL Generation from Large Language Models,” currently under review.

Ph.D. Candidate’s role. The Ph.D. candidate is a core contributor to the framework design, experiments, and manuscript preparation.

Chapter 3

Bibliographical Notes

This section outlines the research articles published during the Ph.D. that are not extensively discussed in the dissertation. These works emerged as parallel investigations, addressing research questions identified while exploring the broader literature.

Those papers were co-authored in the broader domain of recommender systems. The first paper, “*Examining Fairness in Graph-Based Collaborative Filtering: A Consumer and Producer Perspective*” [93], was presented at the 13th Italian Information Retrieval (IIR) Workshop in 2023. This study investigates how graph-based collaborative filtering models behave with respect to fairness for both consumers and producers, evaluating eight state-of-the-art graph CF methods and four traditional CF baselines across CP-aware fairness metrics. The analysis reveals that, despite their strong accuracy, graph-based approaches do not inherently guarantee fair item exposure or equitable treatment of users. To better understand this performance gap, the paper introduces a taxonomy of graph CF models that highlights key differences in node representation and neighborhood exploration, offering a foundation for multi-objective fairness assessments.

The second paper, “*Broadening the Scope: Evaluating the Potential of Recommender Systems Beyond Prioritizing Accuracy*” [275], was published at the 17th ACM Conference on Recommender Systems (RecSys'23). This work challenges the traditional reliance on accuracy as the primary evaluation metric for recommender systems and introduces a multi-objective evaluation framework that jointly considers several beyond-accuracy criteria, including diversity, novelty, and fairness for both users and providers. Multiple hyper-parameter configurations of a recommender model

are trained, and the Pareto-optimal ones are identified using Quality Indicators (QI), a set of metrics adapted from Multi-Objective Optimization research. By applying QI to analyze performance across different configurations, the paper demonstrates that the ranking of recommender systems can change substantially when all objectives are treated with equal importance. This multi-perspective evaluation offers a more comprehensive understanding of model behavior and provides a pathway toward rethinking established evaluation practices in the RecSys community.

Furthermore, the candidate contributed to a book chapter in Human-Centered AI: An Illustrated Scientific Quest, titled “*Conversational User Interfaces and Agents*” [38]. The chapter presents a comprehensive examination of Conversational Agents (CAs), such as chatbots and virtual assistants, outlining their role in enabling natural language interaction between humans and AI systems. It discusses their applications across domains including customer service, health care, and education, and analyzes the underlying problems CAs aim to solve as well as the challenges reported in current literature. The chapter also delves into the architectures of conversational agents, highlighting the distinctions between modular and end-to-end approaches. To structure this landscape, it introduces a taxonomy of the core tasks tackled by CAs, including information retrieval, question answering, and chitchat, and evaluates the advantages and limitations of both rule-based and data-driven methodologies. The work concludes by emphasizing open challenges, ethical considerations, and future directions for developing reliable and human-centered conversational agents.

Finally, the candidate co-authored the survey article titled “A Survey on Information Retrieval and Recommender Systems in the Era of Large Language Models: Two Sides of the Same Task”, at the time of the writing Under Review. This work provides a comprehensive and unifying analysis of how Large Language Models (LLMs) are reshaping the traditionally distinct fields of Information Retrieval (IR) and Recommender Systems (RSs). The survey argues that recent advances in generative language models have catalyzed a deep conceptual and architectural convergence between the two domains, transforming them into complementary facets of a broader information discovery paradigm. To formalize this convergence, the paper introduces a novel three-part taxonomy that characterizes the role of LLMs as retrieval or recommendation engines, semantic re-rankers, and generative components, illustrating how these roles progressively dissolve the classical multi-stage

pipelines of IR and RSs. The study is grounded in a rigorous, PRISMA-inspired systematic literature review and a quantitative bibliometric analysis that substantiates the proposed paradigm shift through publication trends and citation patterns. By critically examining representative and influential works across both communities, the article highlights shared principles, methodological innovations, and open challenges, ultimately advocating for the recognition of a new, integrated science of information discovery and positioning the IR/RS distinction as an increasingly historical artifact.

Part 2

Background and Foundations

Chapter 4

Large Language Models

The rapid evolution of Large Language Models (LLMs) has profoundly reshaped the field of Natural Language Processing (NLP). Recent advances have progressed at an exceptional pace, equipping these systems with capabilities that go well beyond simple pattern matching and allowing them to address complex tasks through carefully designed prompts. Understanding how such capabilities emerged requires tracing the methodological and architectural innovations that enabled this progress. This chapter reviews the key milestones in the development of LLMs, highlighting the foundational advances that underpin contemporary research in this area.

4.1 Statistical Language Modeling and Distributed Representations

4.1.1 From Symbolic Rules to Statistical Language Models

Historically, research in NLP focused on modeling language to automate computational tasks through symbolic means. Early systems formalized linguistic behaviour using manually specified grammars, rule-based pipelines, and hand-engineered representations of empirical regularities. Nevertheless, foundational work such as Shannon [322] had already framed linguistic structure in probabilistic terms. However, it was not until the late 1980s and early 1990s that the field underwent a decisive shift toward data-driven approaches, largely motivated by the availability of large text corpora enabled by the expansion of the World Wide Web. During this period, language modeling was reformulated so that statistical regularities could be learned directly from data rather than encoded through expert-designed rules.

This transition marked the consolidation of a probabilistic paradigm for language modeling [48]. Specifically, the probabilistic language modeling framework assigns a probability to a word sequence $w_{1:T} = (w_1, \dots, w_T)$ by applying the chain rule of probability,

$$p(w_{1:T}) = \prod_{t=1}^T p(w_t \mid w_{1:t-1}), \quad (4.1)$$

such that each word is predicted conditionally on its preceding context. This formulation conceptualizes language modeling primarily as a next-token prediction problem, rather than as an explicit encoding of syntactic categories or formal grammatical structures in the tradition of Chomsky [75]. However, estimating the conditional probabilities in Equation (4.1) introduces substantial statistical challenges. As the sequence length increases, the space of possible contexts grows exponentially, and when combined with the large vocabularies of natural languages, this results in extreme data sparsity. Consequently, the vast majority of contexts are never observed in finite corpora, rendering reliable probability estimation infeasible without additional modeling assumptions or smoothing strategies.

To mitigate these limitations, early statistical approaches to NLP built on Markov's classical analysis of character sequences [245] to reduce contextual complexity, approximating language models using N -gram formulations,

$$p(w_{1:T}) \approx \prod_{t=1}^T p(w_t \mid w_{t-N+1:t-1}), \quad (4.2)$$

which constrain the conditioning context to a fixed-size window. For small values of N , such as bigrams or trigrams, empirical estimation became tractable. However, the N -gram approximation necessarily discarded long-range syntactic and semantic dependencies that are fundamental to natural language. This limitation exposed a fundamental trade-off between statistical tractability and representational adequacy, and it motivated the development of models designed to overcome data sparsity while capturing richer linguistic structure.

4.1.2 Neural Language Models and Distributed Word Representations

A major advance emerged with the neural probabilistic language model proposed by Bengio et al. [36]. This approach addressed data sparsity by mapping discrete

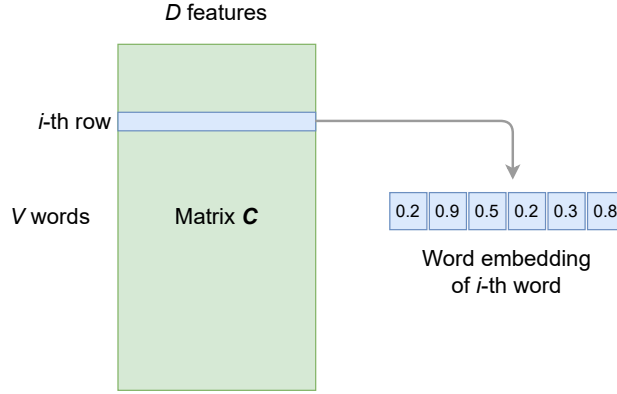


Figure 4.1 The matrix $C \in \mathbb{R}^{|V| \times D}$ encodes the vocabulary $V = \{1, \dots, |V|\}$ in a continuous vector space. Each row corresponds to a D -dimensional embedding representing a lexical item, commonly referred to as a *word embedding*.

word types into a continuous vector space and learning a smooth parametric function over that space. The resulting continuous vectors are known as *word embeddings*. Specifically, the model consisted of three components: a learned embedding vector for each vocabulary item, a continuous parameterization of the conditional distribution in Equation (4.1), and a feed-forward neural network trained jointly with the embedding matrix through backpropagation [308]. By operationalizing the concept of distributed representations within a scalable neural language modeling framework, this formulation provided a practical alternative to sparse symbolic encodings and laid the groundwork for subsequent neural approaches to language modeling.

Let $V = \{1, \dots, |V|\}$ denote the vocabulary. Each word $v \in V$ is associated with an embedding vector $c_v \in \mathbb{R}^D$, and all embeddings are collected in an embedding matrix $C \in \mathbb{R}^{|V| \times D}$. Given a sequence $w_{1:T}$ and an index mapping $I(w_t)$, the embedding corresponding to the token at position t is $c_{I(w_t)} \in \mathbb{R}^D$. Figure 4.1 provides a schematic illustration of this embedding matrix.

Under this formulation, the conditional distribution is approximated as

$$p(w_t \mid w_{1:t-1}) \approx f_{\Theta}(w_{t-1}, \dots, w_{t-N}) = g_{\Omega}(c_{I(w_{t-1})}, \dots, c_{I(w_{t-N})}), \quad (4.3)$$

where g_{Ω} denotes a feed-forward neural network and $\Theta := \{C, \Omega\}$ represents the set of model parameters. Training is performed by maximizing the log-likelihood

of observed sequences, or equivalently by minimizing the cross-entropy loss.

The central hypothesis underlying this approach is that words with similar syntactic or semantic properties are mapped to nearby regions of the embedding space, and that the function g_{Ω} varies smoothly with respect to its inputs. Under these assumptions, small perturbations in the embedding vectors induce correspondingly small changes in the predicted probabilities, which enable systematic generalization from limited data. Subsequent work empirically validated this hypothesis by demonstrating the emergence of linear semantic and syntactic regularities within embedding spaces [254]. A canonical example is

$$\text{vec}(\text{"king"}) - \text{vec}(\text{"man"}) + \text{vec}(\text{"woman"}) \approx \text{vec}(\text{"queen"}),$$

which suggests that gender transformations correspond to approximately constant vector offsets, as illustrated in Figure 4.2. Similar linear patterns have been observed for geographic, morphological, and factual relations, for instance

$$\text{vec}(\text{"Russia"}) + \text{vec}(\text{"river"}) \approx \text{vec}(\text{"Volga River"}).$$

These observations indicate that embedding spaces encode a quasi-linear approximation of certain aspects of natural language semantics, thereby establishing the representational foundation upon which modern large-scale sequence models are built.

4.2 Autoregressive Modeling as the Core Learning Objective

4.2.1 Autoregressive Learning and the Next-Token Prediction

While distributed representations alleviate data sparsity, the training paradigm of neural language modeling is equally central to the development of modern NLP systems. Both the early neural probabilistic language model of Bengio et al. [36] and contemporary large language models adopt an *autoregressive* framework, in which the model is trained to predict each token given its preceding context. This objective directly operationalizes the chain-rule factorization introduced in Equation (4.1) and has remained the foundational learning signal for generative language models for more than two decades.

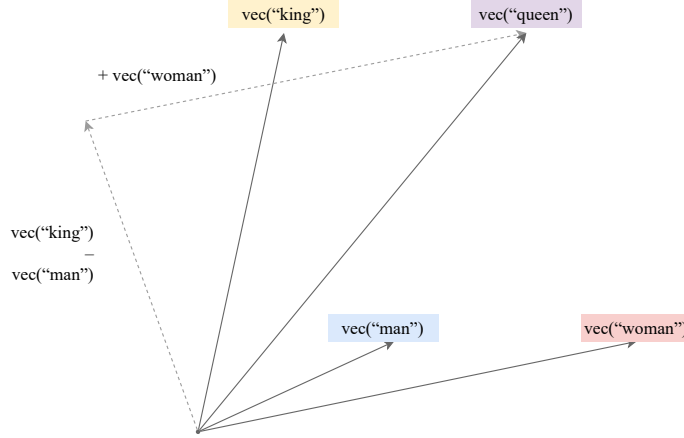


Figure 4.2 Visualization of algebraic operations in the word embedding space. The vectors corresponding to *king*, *man*, *woman*, and *queen* illustrate how the computation “king - man + woman” yields a point close to “queen”, demonstrating that the embedding space encodes meaningful linguistic structure.

Given a token sequence $w_{1:T}$, an autoregressive language model estimates the conditional probability of each token as $p(w_t \mid w_{1:t-1})$ and is trained to minimize the cross-entropy between the predicted distribution and the observed next token. In fixed-context architectures, such as the neural probabilistic language model proposed by Bengio et al. [36], the input at time step t consisted of the embeddings of the preceding N tokens, truncated to a fixed-size window. If $c_{I(w)}$ denotes the embedding of a word w and c_p represents a padding vector used for initial positions, the model was trained on a sequence of embedding windows that advanced one token at a time as learning proceeded. Despite its simplicity, this formulation captures the core mechanism of next-token prediction. At each step, the model produces a probability distribution over the vocabulary, and the cross-entropy with the true next token provides the learning signal that drives parameter updates.

4.2.2 Likelihood Maximization and Autoregressive Generation

Formally, training corresponds to maximizing the log-likelihood of the observed corpus,

$$\Theta^* = \arg \max_{\Theta} \sum_{t=1}^T \log p_{\Theta}(w_t \mid w_{t-N:t-1}), \quad (4.4)$$

where the conditional distribution is parameterized by a feed-forward neural network g_Ω applied to the embeddings of the previous N tokens, and $\Theta = \{C, \Omega\}$ comprises both the embedding matrix and the network parameters. Optimization is typically carried out using stochastic gradient descent and backpropagation [308], which allows the model to jointly refine its lexical representations and its conditional prediction function.

Once trained, an autoregressive language model generates text through sequential sampling. An initial token is drawn from the model’s marginal distribution, and each subsequent token is sampled from the learned conditional probabilities,

$$w_2 \sim p(w_2 \mid w_1), \quad w_3 \sim p(w_3 \mid w_{1:2}), \quad \dots$$

This recursive procedure enables language models to function not only as statistical estimators of linguistic structure but also as generative systems capable of producing coherent sequences of unbounded length. The same objective that underpins likelihood-based training therefore also governs text generation, tightly coupling learning and inference within a unified probabilistic framework.

Despite their conceptual elegance and empirical success, early autoregressive models were inherently constrained by fixed context windows, which limited their capacity to capture long-range syntactic and semantic dependencies. This limitation motivated the development of architectures to encode and decode variable-length sequences while retaining information over extended contexts, paving the way for recurrent and sequence-to-sequence models discussed in the next section.

4.3 Modeling Long-Range Dependencies: From Fixed Windows to Sequence-to-Sequence Architectures

4.3.1 Limitations of Fixed-Context Neural Models

By 2013, neural word- and phrase-level embeddings had become standard tools in NLP, largely influenced by the neural probabilistic language model of Bengio et al. [36]. Their widespread adoption was further accelerated by the introduction of simple and scalable architectures such as the skip-gram and continuous bag-of-words models [252, 253], which enabled the efficient learning of distributed representations from large corpora. Despite their strong empirical performance, these models relied on fixed-size context windows and therefore remained limited in their abil-

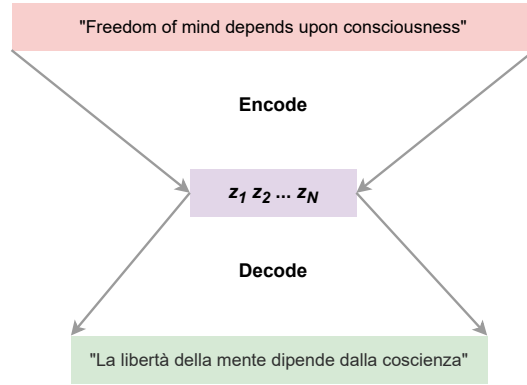


Figure 4.3 Illustration of a sequence-to-sequence model for neural machine translation. The encoder processes a variable-length English sentence into a latent representation, and the decoder generates a variable-length Italian translation conditioned on this representation.

ity to capture long-range syntactic and semantic dependencies. This constraint was particularly restrictive in natural language, where meaning often extends across arbitrarily long contexts and cannot be reliably inferred from a narrow local window.

As a result, the fixed-context assumption inherited from earlier N -gram models and feed-forward neural language models persisted as a structural bottleneck. Although embedding spaces facilitated smoother generalization across lexical items, they did not, in isolation, resolve the inability to condition on extended histories. Overcoming this limitation required the development of architectures capable of processing variable-length sequences while maintaining a dynamic internal representation of past inputs.

4.3.2 Sequence-to-Sequence Architectures for Variable-Length Dependencies

To address the limitations of fixed-context models, *sequence-to-sequence* (seq2seq) architectures were introduced. In their classical formulation, seq2seq models consist of two neural components: an *encoder*, which maps a variable-length input sequence into a latent representation, and a *decoder*, which generates an output sequence conditioned on that representation. Unlike fixed-window models inspired by Bengio et al. [36], seq2seq architectures relied on recurrent computation, which enabled them to process arbitrarily long contexts and to model dependencies spanning entire sequences. Figure 4.3 illustrates the general encoder–decoder paradigm.

One of the earliest complete encoder–decoder architectures for NLP was commonly attributed to Kalchbrenner et al. [175], who proposed a continuous translation model combining a convolutional encoder with a recurrent decoder. This research direction evolved rapidly. Cho et al. [74] introduced a fully recurrent RNN–RNN encoder–decoder architecture, and Sutskever et al. [341] showed that Long Short-Term Memory networks substantially improved the modeling of long-distance dependencies, achieving state-of-the-art performance in neural machine translation.

Formally, let $X = (x_1, \dots, x_{T_x})$ denote the sequence of input embeddings and $Y = (y_1, \dots, y_{T_y})$ the output sequence. The encoder recurrent network computes a sequence of hidden states,

$$h_t = f_{\text{enc}}(h_{t-1}, x_t),$$

yielding $H = (h_1, \dots, h_{T_x})$, which encodes the input through recurrent composition. A fixed-dimensional *context vector* c is then obtained by summarizing these states,

$$c = q(H),$$

most commonly instantiated as the final hidden state h_{T_x} . The decoder generates the output sequence by conditioning on this context,

$$s_t = f_{\text{dec}}(s_{t-1}, y_{t-1}, c),$$

and predicts each token autoregressively according to

$$\log p(Y | X) = \sum_{t=1}^{T_y} \log p(y_t | y_{1:t-1}, c). \quad (4.5)$$

All encoder and decoder parameters are trained jointly by maximizing this conditional log-likelihood.

The encoder–decoder framework rapidly became the dominant paradigm for sequence transduction tasks such as machine translation, abstractive summarization, and dialogue generation. By replacing fixed context windows with recurrent mechanisms capable of maintaining information across entire sequences, seq2seq models provided a flexible and scalable solution for modeling long-range dependencies. This architectural shift laid the groundwork for subsequent innovations, most no-

tably attention mechanisms, which further refined how models access and exploit contextual information over long inputs.

4.4 Adaptive Context and the Emergence of Attention Mechanisms

4.4.1 The Context Vector Bottleneck in Encoder–Decoder Models

Although the sequence-to-sequence architectures introduced by Kalchbrenner et al. [175], Cho et al. [74], and Sutskever et al. [341] extended neural modeling to variable-length inputs and outputs by introducing a latent *context vector* that encoded the entire input sequence into a fixed-dimensional representation, this design imposed a substantial representational bottleneck. Compressing an entire sentence into a single hidden vector often led to the loss of fine-grained syntactic and semantic information, and performance degraded significantly for long or structurally complex inputs [73].

This limitation exposed a fundamental weakness of early encoder–decoder architectures. Although recurrent computation allowed information to propagate across time steps, the requirement to condense all relevant input information into a single vector constrained the expressive capacity of the model. Consequently, important details originating from distant positions in the input sequence could be attenuated or overwritten. These observations motivated the development of mechanisms capable of preserving and selectively exploiting information distributed across the entire input sequence.

4.4.2 Attention as Adaptive Context Construction

A major breakthrough was achieved with the introduction of differentiable *attention mechanisms* by Bahdanau et al. [27]. Specifically, attention enabled the decoder to focus selectively on different parts of the input sequence at each generation step, rather than relying on a global summary representation. Concretely, the model learned a soft alignment between encoder and decoder states, which allowed the construction of a context vector that adapted dynamically as each output token was generated. This formulation directly addressed the compression bottleneck inherent in earlier sequence-to-sequence models.

Formally, let the encoder produce a sequence of hidden states $H = (h_1, \dots, h_{T_x})$ and the decoder produce hidden states $S = (s_1, \dots, s_{T_y})$. For each output position i , the decoder computes a context vector

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j, \quad (4.6)$$

where the alignment weights α_{ij} quantify the relevance of the encoder state h_j for predicting the i -th output token. These weights are obtained by normalizing a set of compatibility scores,

$$e_{ij} = v_a^\top \tanh(W_a s_{i-1} + U_a h_j), \quad (4.7)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}. \quad (4.8)$$

The decoder state is then updated according to

$$s_i = f_{\text{dec}}(s_{i-1}, y_{i-1}, c_i), \quad (4.9)$$

which allows the model to attend dynamically to the most informative regions of the input sequence at each decoding step. By distributing representational capacity across all encoder states, attention removes the severe information bottleneck imposed by a single global context vector.

The introduction of attention led to substantial improvements in translation quality. Bahdanau et al. [27] reported performance competitive with state-of-the-art phrase-based statistical machine translation systems. Subsequent refinements, including the global and local attention mechanisms proposed by Luong et al. [230], explored alternative formulations and confirmed the general effectiveness of attention across architectures and tasks.

The integration of attention therefore represented a pivotal turning point in neural NLP. By enabling adaptive and context-sensitive modeling of long-range dependencies, attention resolved a fundamental limitation of early encoder–decoder architectures. Its success also raised a deeper architectural question: whether recurrent computation was necessary at all, or whether attention alone could serve as the primary mechanism for sequence modeling. This question ultimately motivated the development of transformer architectures, which are discussed in the next section.

4.5 The Emergence of the Transformer Architecture

4.5.1 Limitations of Recurrent Attention-Based Models

By 2017, recurrent encoder–decoder models augmented with attention had become the dominant framework for sequence transduction tasks [27, 230]. Despite their strong empirical performance, these architectures suffered from a fundamental computational limitation. Recurrent networks process tokens sequentially, and each hidden state depends on the previous one. This inherent sequential dependency prevents parallel computation across positions within a sequence and significantly slows both training and inference. The problem became particularly pronounced for long sequences, where memory constraints restricted batch sizes and long-distance dependencies became increasingly difficult to maintain.

These computational inefficiencies were not merely engineering inconveniences but instead reflected a deeper architectural constraint. The reliance on recurrence enforces a strict temporal ordering of computation, even in cases where the underlying dependency structure of language does not require such ordering. This insight motivated the search for architectures capable of modeling rich contextual dependencies while avoiding the bottlenecks imposed by sequential processing.

4.5.2 Transformers and Fully Attentional Sequence Modeling

A major shift occurred with the introduction of the transformer architecture by Vaswani et al. [356] in *Attention Is All You Need*. The transformer eliminated recurrence entirely and modeled dependency structure exclusively through attention mechanisms. Because attention operated over all sequence positions in parallel, the architecture fully exploited modern hardware accelerators and distributed computation independently of temporal order. Vaswani et al. [356] demonstrated that a fully attentional encoder–decoder model achieved state-of-the-art performance in machine translation after only twelve hours of training on eight NVIDIA P100 GPUs. This result represented a fundamental rethinking of sequence modeling and established attention as a primary computational primitive.

At the core of the transformer lies *scaled dot-product attention*, which computes pairwise interactions between queries and keys. Let $Q \in \mathbb{R}^{M \times D_k}$ denote the query matrix, $K \in \mathbb{R}^{N \times D_k}$ the key matrix, and $V \in \mathbb{R}^{N \times D_v}$ the value matrix. Attention is

defined as

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{D_k}} \right) V.$$

The matrix product $QK^\top$ computes similarity scores between all query–key pairs, determining how strongly each query attends to each key. The scaling factor $\sqrt{D_k}$ stabilizes optimization by preventing large dot products from driving the softmax into regimes with vanishing gradients. The resulting attention weights are used to form weighted combinations of the value vectors, enabling information aggregation across the entire sequence.

Unlike recurrent computation, attention is set-based and therefore permutation invariant. It does not encode token order intrinsically. To incorporate positional information, the transformer adds position-dependent vectors to the token embeddings. The original formulation employed sinusoidal positional encodings that represent relative positions through linear phase shifts [356]. Subsequent work explored learned absolute and relative positional encodings [323], which improved generalization to longer sequences and introduced stronger inductive biases.

Furthermore, self-attention allows each token to attend directly to every other token in the sequence. In the encoder, this enables the modeling of long-range syntactic and semantic relations, such as subject–verb agreement or coreference, within a single layer. In the decoder, self-attention is masked so that position t attends only to positions 1 through $t - 1$, preserving the autoregressive generation process. Encoder–decoder attention integrates the two components by allowing decoder queries to attend to encoder states, generalizing the alignment mechanisms introduced by earlier attention-based sequence-to-sequence models.

Moreover, another key innovation of the transformer is *multi-head attention*. Instead of computing a single attention distribution, the model projects queries, keys, and values into A distinct learned subspaces,

$$\text{head}_a = \text{Attention}(QW_a^Q, KW_a^K, VW_a^V), \quad a = 1, \dots, A.$$

Each attention head can capture different relational patterns. Some specialize in syntactic dependencies, others in positional structure, lexical similarity, or long-range discourse relations. The outputs of all heads are concatenated and projected through a final linear transformation. This design substantially increases representational capacity and enables the model to track diverse linguistic phenomena in

parallel.

Figure 4.4 illustrates the complete transformer architecture. Each encoder and decoder layer consists of a multi-head self-attention sublayer followed by a position-wise feed-forward network. Each sublayer is wrapped in a residual connection and followed by layer normalization, which stabilizes training and supports deep architectures. In the decoder, an additional cross-attention sublayer mediates information flow from encoder to decoder states. A final linear projection and softmax layer produces output token probabilities.

The empirical gains reported by Vaswani et al. [356] were substantial. The transformer outperformed the convolutional sequence-to-sequence model of Gehring et al. [125] on major translation benchmarks while requiring dramatically fewer floating-point operations. For English–French translation, ConvS2S [125] required approximately 1.2×10^{21} FLOPs, whereas the transformer required only 3.3×10^{18} FLOPs. This placed the transformer at a favorable point on the Pareto frontier between expressive power and computational efficiency.

The introduction of the transformer marked a decisive shift in NLP. It demonstrated that recurrence is not essential for modeling linguistic structure, established attention as the core mechanism for sequence modeling, and provided an architecture that scales efficiently with data and hardware. However, the full impact of transformers emerged only when combined with large-scale pre-training and task-specific adaptation strategies. The next section therefore examines the pre-training paradigms that transformed transformers into general-purpose language models.

4.6 From Transformer to Large-Scale Pre-Training models

4.6.1 Generative Pre-Training and Universal Language Representations

The transformer architecture provided a scalable and expressive backbone for sequence modeling. However, architectural advances alone were not sufficient to produce the large-scale, general-purpose language models that underpinned contemporary conversational and task-oriented systems. The decisive breakthrough emerged when transformers were combined with large-scale pre-training strategies. Among the most influential contributions was OpenAI’s GPT line of models, which framed

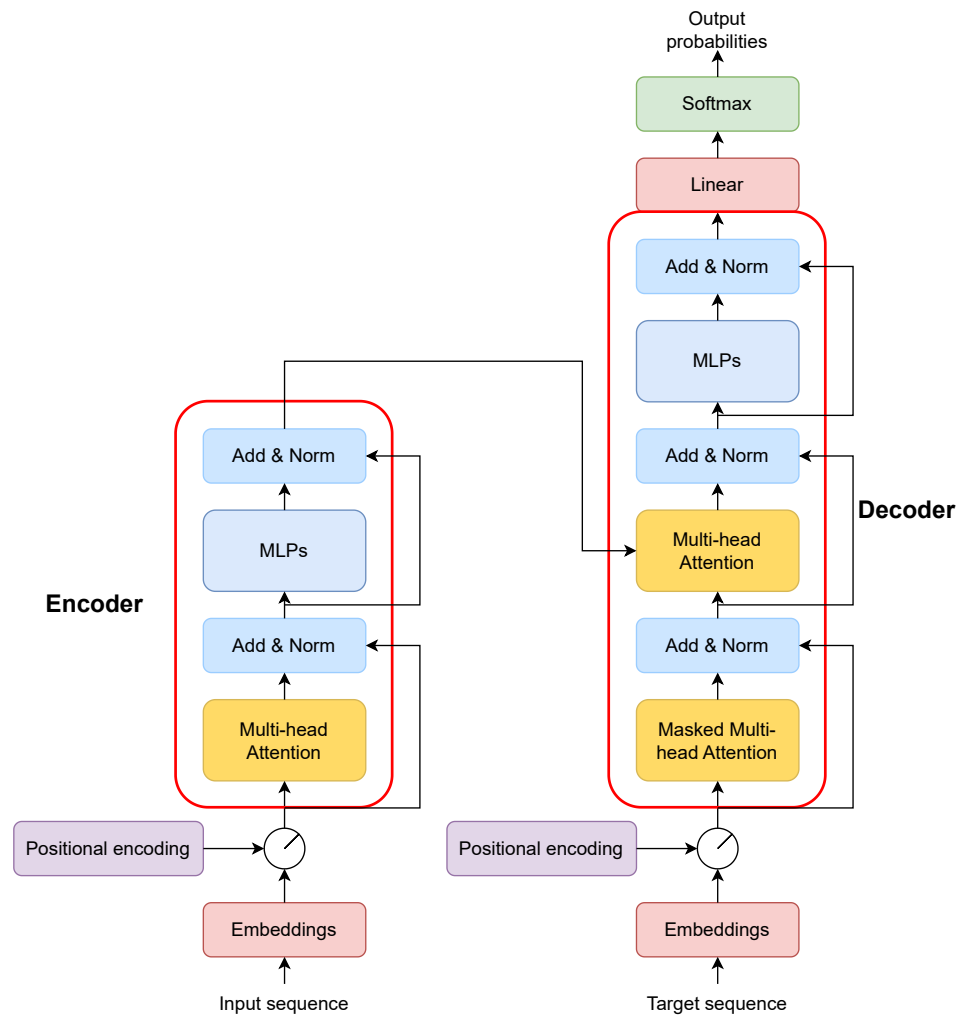


Figure 4.4 The transformer architecture consists of an encoder and a decoder equipped with positional encodings and multi-head self-attention. The decoder employs masked self-attention to preserve left-to-right generation, while cross-attention enables decoder states to retrieve source-side information relevant to each output token.

model development as a multi-stage process involving generative pre-training, discriminative fine-tuning, and, in later iterations, reinforcement learning with human feedback.

Radford et al. [292] introduced the first model in this lineage, later referred to as GPT-1. The central insight was that a transformer trained on large collections of unlabeled text using a generic self-supervised objective could acquire broad linguistic and world knowledge. This knowledge could then be adapted to downstream tasks through supervised learning. The initial phase, termed *generative pre-training*, preceded a task-specific *discriminative fine-tuning* stage. This terminology followed the classical distinction between generative and discriminative models [261], where generative models estimate $p(x)$ or $p(x, y)$, while discriminative models estimate $p(y | x)$.

During generative pre-training, GPT-1 was optimized using an autoregressive language modeling objective. Given a token sequence $w_{1:T}$, the model estimated the conditional probability of each token given its preceding context,

$$\mathcal{L}_{\text{GPT}}(\Theta) = \sum_{t=1}^T \log p_{\Theta}(w_t | w_{1:t-1}), \quad (4.10)$$

where N denoted the effective context window and Θ the model parameters. Because this objective required only raw text, it enabled pre-training on extremely large unlabeled corpora, making the approach highly scalable across domains and languages.

The strictly left-to-right factorization in Equation (4.10) constrained the contextual information available during training. To address this limitation, Devlin et al. [100] introduced BERT, a bidirectional transformer trained with a Masked Language Modeling objective inspired by the Cloze test [345]. Let $M \subseteq \{1, \dots, T\}$ denote the set of masked positions. The MLM objective was defined as

$$\mathcal{L}_{\text{MLM}}(\Theta) = \sum_{i \in M} \log p_{\Theta}(w_i | w_{\neg M}), \quad (4.11)$$

which conditioned on both left and right context. This bidirectional formulation yielded contextualized representations that performed strongly across a wide range of discriminative tasks.

Although autoregressive and masked language modeling objectives differed in

their directional constraints, both provide effective mechanisms for learning general-purpose language representations. However, generative pre-training alone does not guarantee alignment with task-specific objectives or user intent. Models trained solely to predict tokens tend to reproduce corpus statistics, rather than exhibit the desired behavior in applied settings.

4.6.2 Fine-Tuning, Prompting, and Multi-Stage Model Adaptation

To bridge this gap, pre-trained transformers were commonly adapted through *discriminative fine-tuning*. Given labeled examples $(x_{1:T}, y)$, fine-tuning optimized

$$\mathcal{L}_{\text{DFT}}(\theta) = \sum_{(x_{1:T}, y)} \log p_{\theta}(y \mid x_{1:T}), \quad (4.12)$$

where the parameters θ were initialized from the pre-trained model. GPT-1 combined generative and discriminative objectives through a weighted formulation,

$$\mathcal{L}_{\text{final}} = \mathcal{L}_{\text{DFT}} + \lambda \mathcal{L}_{\text{GPT}}, \quad (4.13)$$

with λ controlling the influence of the generative prior and mitigating catastrophic forgetting during supervised optimization.

As pre-trained transformers proliferated, parallel efforts produced large-scale benchmarks for fine-tuning and evaluation across a broad range of tasks. These included reading comprehension [298], summarization [258], commonsense reasoning [415], code generation [68], long-range discourse modeling [277], and mathematical reasoning [79]. A single pre-trained transformer could be adapted to any of these tasks, yielding specialized models derived from a shared backbone.

Moving forward, it was with GPT-2 and subsequently GPT-3 that an influential and initially unexpected observation emerged. Sufficiently large autoregressive transformers were shown to exhibit strong *zero-shot* and *few-shot* performance without task-specific fine-tuning [49, 293]. Rather than updating model parameters, these systems demonstrated that, when conditioned through natural language prompts describing the task and optionally including a small number of examples, they could perform the task directly.

This behavior suggested that large-scale generative pre-training induced latent capabilities that could be elicited through prompting alone.

However, despite this emergent flexibility, additional post-pre-training and adaptation stages were required to yield large language models in their contemporary form. These stages included supervised fine-tuning on instruction-following data, domain-specific specialization, and reinforcement learning with human feedback [267, 271]. Specifically, contemporary foundation models typically follow a multi-stage development pipeline comprising large-scale self-supervised pre-training, supervised instruction fine-tuning, and optional alignment and safety optimization [271]. A representative example of this process was Codex [68], a GPT-derived model specifically designed for code generation tasks.

Although these frameworks clarify how modern large language models are constructed, they do not fully explain why certain capabilities emerge only at particular scales or why performance varies sharply under different prompts. These open questions motivate recent work on emergent abilities and prompt engineering, which shift attention from architectural design toward the analysis and control of model behavior.

4.7 Emergent Abilities and Prompt-Engineering Techniques

4.7.1 The Non-Linear Scaling Effects

As the field has matured, research attention has shifted from raw scaling to the behavioral phenomena that arise as models grow and to the conditions under which these behaviors can be elicited. A central concept in this line of work is that of *emergent abilities*. These refer to cases in which performance on specific tasks improves abruptly once certain thresholds in scale or training progress are crossed. This perspective supplements classical scaling laws, which typically predict smooth and continuous improvements as model size, data, and compute increase.

Early work had already hinted at such phenomena. Studies on attention-based and bidirectional models showed that larger architectures sometimes exhibited capabilities absent in smaller variants [27, 100], although these effects were not yet framed explicitly as emergent. More recent analyses have formalized this notion and investigated whether apparent discontinuities reflect genuine phase transitions or instead arise from interactions between training dynamics and evaluation protocols. In particular, techniques such as chain-of-thought prompting and instruction

tuning have been shown to unlock sudden jumps in performance on complex reasoning tasks.

In the experiments of Wei et al. [380], chain-of-thought prompting produced dramatic gains on mathematical word problems by encouraging models to generate intermediate reasoning steps. Importantly, these gains were observed only for models exceeding approximately 100 billion parameters in their setup. Below this scale, performance often remained near chance. Without appropriate prompting strategies, even large models frequently failed to exhibit the same capabilities. This indicated that emergent behaviour depended not only on model size but also on the context in which the model was queried.

Subsequent work examined emergence from the perspective of training dynamics. Du et al. [106] tracked models ranging from 1.5 to 32 billion parameters and found that, for certain tasks, accuracy remained at chance until the pre-training loss crossed a critical threshold, after which performance increased sharply. Complementary analyses by Huang et al. [161] linked emergent abilities to the interaction between memorization and generalization. Extensive memorization can delay generalization, causing new capabilities to surface only after sufficient optimization progress. Together, these findings suggest that scale is necessary but insufficient. Emergent abilities arise from the interplay between model capacity, training objectives, optimization trajectories, and the structure of the prompts used to elicit behaviour.

4.7.2 Prompt Engineering as a Mechanism for Behavioral Control

In parallel with the study of emergent abilities, *prompt engineering* has developed into a central strategy for shaping model behaviour without modifying model parameters. Rather than relying on gradient-based updates, prompt engineering focuses on refining the informational structure of the input so that desired behaviors are elicited from a fixed model. This shift reflects a broader move toward behaviour-centric approaches to model development.

A recent survey proposes a four-part taxonomy of prompt engineering techniques organized around user profiling and instruction design, the integration of external knowledge, scaffolds for reasoning and planning, and mechanisms for reliability and control [225]. Complementary work provides a more fine-grained classification.

The *Prompt Report* [320] catalogs fifty-eight prompting techniques grouped into six categories, including zero-shot and few-shot prompting, thought-generation methods such as chain-of-thought and thread-of-thought prompting, ensembling strategies, self-criticism, and decomposition-based approaches.

These studies demonstrate that subtle prompt design choices can have a substantial impact on model performance. The number, ordering, and semantic similarity of in-context examples all influence outcomes. In certain settings, carefully constructed prompts have yielded accuracy improvements of up to ninety percent relative to naive baselines. Benchmark evaluations have further indicated that few-shot chain-of-thought prompting consistently enhanced reasoning performance across a wide range of tasks. In addition, self-consistency techniques, which sample multiple reasoning paths and select the most coherent output, have emerged as effective decoding strategies for improving reliability.

The growing emphasis on prompt engineering underscores the importance of interaction-level control in modern language models. Techniques ranging from few-shot exemplars and reasoning scaffolds to iterative self-reflection and automated prompt optimization provide practical means of steering model outputs while leaving parameters unchanged. Together with insights from research on emergent abilities, these methods suggest that future progress will depend not only on continued scaling but also on a deeper understanding of how internal representations, training dynamics, and input structure jointly shape model behaviour. More sophisticated forms of interaction are therefore likely to play an increasingly important role in unlocking the latent potential of large language models.

Chapter 5

Recommender Systems

Recommender Systems (RSs) constitute a well-established class of computational models designed to assist *users* in identifying *items* aligned with their preferences by estimating the degree of relevance each item may hold for them [303]. These systems operate by analyzing historical *interactions*, auxiliary context, and user-specific attributes to generate personalized suggestions across a range of application domains, including e-commerce platforms, multimedia services, and online communities. Although RSs may leverage heterogeneous data sources, any recommendation setting can be characterized by three fundamental components:

- **Users:** the individuals or entities receiving recommendations. User representations may include demographic descriptors, explicit preference statements, and behavioral histories that capture long-term or short-term interest patterns.
- **Items:** the entities available for recommendation, encompassing products, media content, or services. Items are often described by structured metadata such as categorical labels, textual annotations, or numerical features, which may support downstream modeling.
- **Interactions:** the observable evidence of user engagement with items. These interactions, frequently referred to as *transactions*, appear in two primary forms. *Explicit feedback* captures deliberate assessments provided by the user, such as ratings or binary approval signals, offering direct and interpretable indicators of both positive and negative preference. *Implicit feedback*, in contrast, is inferred from behavioral traces including plays, clicks, purchases, or browsing events. Implicit data typically contains only positive (or non-negative) signals and therefore offers indirect and incomplete information about user intent. The interaction data

$$\begin{bmatrix} 4 & 3 & \dots & 2 & 5 \\ 5 & 2 & \dots & 4 & 3 \\ \dots & \dots & \dots & \dots & \dots \\ 3 & 4 & \dots & 5 & 1 \\ 2 & 5 & \dots & 3 & 4 \end{bmatrix}$$

(a) Complete user–item rating matrix.

$$\begin{bmatrix} 4 & ? & \dots & 2 & 5 \\ 5 & 2 & \dots & ? & 3 \\ \dots & \dots & \dots & \dots & \dots \\ 3 & ? & \dots & 5 & 1 \\ 2 & 5 & \dots & ? & ? \end{bmatrix}$$

(b) Sparse matrix with missing ratings.

Figure 5.1 Rating prediction scenario. The left matrix shows full observations, while the right matrix contains unobserved entries that the RS must estimate.

are commonly encoded in a user–item *sparse matrix*, where the majority of entries are unobserved due to the large number of possible user–item pairs.

5.1 Definition of the Recommendation Problem

Let $\mathcal{U}$ denote the set of users and $\mathcal{I}$ the set of items. The observed interactions between them are represented by the matrix $R \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{I}|}$, where $r_{u,i}$ records the explicit or implicit feedback associated with the pair (u, i) . Figure 5.1a shows an example of a complete rating matrix, while Figure 5.2a illustrates an implicit-feedback setting. In practice, R is heavily *sparse*, as only a small subset of user–item relations is observed [150]. For this reason, a central challenge in RS research is the construction of a utility function capable of estimating user preference for unobserved items without additional user input [317].

Definition 1 (Recommendation Problem). Given a utility function $f : \mathcal{U} \times \mathcal{I} \rightarrow \mathbb{R}$, the recommendation problem is defined as:

$$\forall u \in \mathcal{U}, \quad i' = \arg \max_{i \in \mathcal{I}} f(u, i), \quad (5.1)$$

where i' belongs to $\mathcal{I} / \mathcal{I}_u^+$, the set of items not yet consumed or positively evaluated by user u .

The formulation of the recommendation task varies according to the type of utility function adopted. In explicit-feedback settings, the objective is to reconstruct the rating matrix by estimating missing values, leading to a *rating prediction* task (Figure 5.1b). In implicit-feedback settings, the utility function is used to infer the likelihood of positive interactions among unobserved entries and to rank items ac-

$$\begin{bmatrix} 1 & 0 & \dots & 1 & 0 \\ 0 & 1 & \dots & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 0 & \dots & 0 & 1 \\ 1 & 1 & \dots & 0 & 0 \end{bmatrix}$$

(a) Complete implicit-feedback matrix.

$$\begin{bmatrix} 1 & ? & \dots & 1 & 0 \\ 0 & 1 & \dots & ? & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & ? & \dots & 0 & 1 \\ 1 & 1 & \dots & ? & ? \end{bmatrix}$$

(b) Sparse implicit-feedback matrix.

Figure 5.2 Implicit-feedback scenario. The system must infer which missing entries correspond to likely positive interactions.

cordingly, resulting in a *top-k* recommendation task (Figure 5.2b). Despite their differences, these two tasks are closely related, since ranking can be derived from predicted scores.

In their simplest form, RSs can be understood as systems that generate *ranked* lists of items for each user by leveraging explicit or implicit preference signals, often supplemented with item metadata, contextual information, or past behavioural patterns.

Historically, early work in RSs primarily addressed explicit-feedback scenarios. The rating prediction task dominated research for many years and became widely known through the Netflix Prize competition [32], which sought improvements in Root Mean Square Error. However, collecting explicit ratings at scale is demanding and may disrupt normal user activity. Moreover, several studies have shown that ratings do not always correspond to genuine user preference [248]. These limitations contributed to the transition toward implicit-feedback data and *top-k* recommendation tasks [84], which today form the basis of most practical RS deployments.

5.2 Recommendation Techniques

Research in Recommender Systems (RSs) has produced a wide range of methodological approaches, many of which have become established as state-of-the-art solutions within the field. Each approach offers a different perspective on how to formulate and address the recommendation problem, reflecting distinct assumptions about how users evaluate and select items. For instance, some techniques assume that users prefer items similar to those they have previously consumed, while others operate under the idea that users with similar behavioral patterns tend to make comparable choices. Broadly, recommendation techniques are grouped into three main families: (i) *collaborative filtering*, (ii) *content-based filtering*, and (iii) *hybrid meth-*

ods [303]. The following sections provide an overview of these approaches and their classification.

5.2.1 Collaborative Filtering Approaches

Collaborative Filtering (CF) is among the most widely used paradigms in recommendation systems. It is grounded in the principle that “users who have agreed in the past are likely to agree again in the future” [302]. This assumption implies that a user’s rating or preference for a given item can be inferred by examining the behaviors of other users with similar interaction histories. A notable advantage of CF is that it requires no auxiliary information such as item attributes. It operates solely on user-item interactions. However, its effectiveness strongly depends on the density of the interaction matrix. Sparse data and the presence of new users or items often lead to cold-start issues.

CF approaches are generally divided into *memory-based* and *model-based* methods [303]. Memory-based techniques rely directly on observable interaction data to estimate unknown ratings, while model-based approaches seek to learn latent representations of users and items from the interaction matrix.

5.2.1.1 Memory-based Collaborative Filtering

Memory-based CF algorithms work directly with the user-item interaction matrix, typically relying on similarity metrics to identify relationships between users or items. A widely adopted example is the k -Nearest Neighbors (k NN) algorithm, which constructs a similarity matrix to identify the most comparable users or items. Depending on the type of similarity considered, two variants arise:

- **User k NN** [302]: the utility function f estimates the relevance of item i for user u by aggregating similarity scores between u and other users who interacted with item i :

$$f^{\text{User}k\text{NN}}(u, i) = \sum_{v \in \mathcal{U}_u^+} \text{sim}(u, v), \quad (5.2)$$

where $\text{sim}(\cdot)$ is a similarity measure and $\mathcal{U}_u^+$ denotes the set of users who interacted with item i . This method captures user-level similarity effectively. However, as the number of users grows, computing all pairwise similarities becomes increasingly demanding, raising scalability concerns.

- **Item k NN** [317]: to improve scalability, item-based CF shifts the focus toward item similarities. The predicted score for the pair (u, i) is computed by comparing

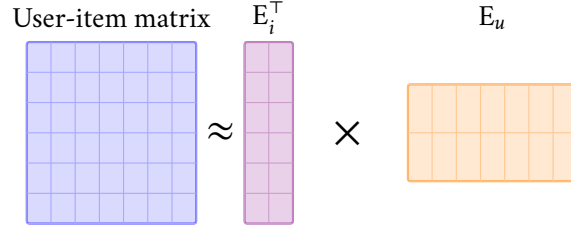


Figure 5.3 Latent factor decomposition of the user-item interaction matrix into item and user embedding matrices.

item i with items previously consumed by user u :

$$f^{\text{ItemkNN}}(u, i) = \sum_{j \in \mathcal{I}_u^+} \text{sim}(i, j), \quad (5.3)$$

where $\mathcal{I}_u^+$ denotes the items previously interacted with by user u . Item-based CF generally scales better in systems with large numbers of users.

Memory-based methods are intuitive and explainable, but their performance is heavily influenced by the sparsity of the interaction matrix. Their dependence on explicit similarity computations limits their ability to generalize in complex or sparse environments. These limitations have led to the broader adoption of model-based CF techniques that aim to uncover latent patterns in the interaction data.

5.2.1.2 Model-based Collaborative Filtering

Machine learning and deep learning techniques have substantially advanced model-based Collaborative Filtering. A key milestone was the Netflix Prize competition in 2006, where model-based approaches, particularly matrix factorization, demonstrated significantly higher accuracy than memory-based methods [189]. Model-based CF approaches seek to identify underlying factors that explain observed user-item interactions, thus enabling more effective predictions.

Among these techniques, **Matrix Factorization** (MF) methods [190] have become especially influential. MF decomposes the user-item interaction matrix into latent factors that represent hidden characteristics of users and items. These latent factors capture patterns that are not explicitly available in the data. Figure 5.3 illustrates the intuition behind such models.

In MF, each user u and item i is associated with a latent embedding vector, $e_u \in \mathbb{R}^d$ and $e_i \in \mathbb{R}^d$, respectively, where d is the dimension of the latent space and

$d \ll |\mathcal{U}|, d \ll |\mathcal{I}|$. Collectively, user and item embeddings form the matrices $E_u \in \mathbb{R}^{|\mathcal{U}| \times d}$ and $E_i \in \mathbb{R}^{|\mathcal{I}| \times d}$. These embeddings are the learnable parameters Θ . The interaction between user u and item i is then estimated by the utility function:

$$f^{\text{MF}}(u, i) = \mu + b_u + b_i + \mathbf{e}_i^T \mathbf{e}_u, \quad (5.4)$$

where μ denotes the global mean rating, and b_u and b_i represent user and item bias terms. The inner product $\mathbf{e}_i^T \mathbf{e}_u$ models the affinity between user and item in the latent space.

Building on MF, many extensions have been proposed. Neural Collaborative Filtering (NCF) [147] replaces the dot product with a neural network, enabling the capture of more complex user-item relationships. SimpleX [244] introduces a disentanglement based formulation to provide clearer separation between user preferences and item characteristics. These methods maintain the core idea of learning latent representations, while differing in how the utility function f is computed.

Despite their effectiveness, model-based CF methods also face challenges. Large-scale datasets impose substantial computational and memory demands. Cold-start scenarios remain difficult because latent factors require adequate training data. Moreover, more complex parametrizations may overfit sparse signals, making regularization essential.

5.2.2 Content-based Approaches

Content-Based Filtering (CBF) generates recommendations by comparing user preferences with item attributes [281]. The underlying assumption is that users tend to prefer items whose characteristics are consistent with their previous choices. Traditional CBF systems describe items through keywords derived from metadata or textual content. More recent approaches adopt richer *concept-based* representations constructed from external knowledge sources, such as DBpedia, Freebase, or Wikipedia. These representations offer semantically meaningful descriptions that extend beyond surface-level keywords.

CBF typically involves three components [303]:

- **Content Analyzer:** processes item-related information and maps items into a unified representation space, such as vector spaces or semantic embeddings.
- **Profile Learner:** constructs user profiles by analyzing observed preferences or

interaction histories. Various techniques such as probabilistic models, relevance feedback, or similarity-based learning can be used to generalize from past behavior.

- **Filtering Component:** compares the user profile with item representations to identify relevant items and generate recommendation lists.

CBF has several advantages. It inherently supports explainability because recommendations directly reflect item attributes that match the user's profile. Furthermore, CBF does not require user-user or item-item interaction patterns, allowing it to handle new items effectively. However, it often fails to provide sufficient diversity due to the risk of recommending items that are too similar to past preferences, a phenomenon known as *overspecialization*. In addition, CBF strongly depends on the quality and richness of item descriptions. When semantic or structured content is limited, performance may degrade. To mitigate this issue, researchers have proposed augmenting datasets by linking items to corresponding entities in knowledge graphs such as Wikidata and DBpedia.

5.2.3 Hybrid Approaches

The inherent limitations of Collaborative Filtering and Content-Based Filtering have motivated the development of hybrid recommender systems, which combine elements of both paradigms to improve accuracy, diversity, and robustness. Hybrid systems aim to exploit complementary strengths while mitigating weaknesses such as data sparsity, cold-start issues, and overspecialization.

Several strategies can be employed to design hybrid methods [8]:

- **Parallel integration:** CF and CBF scores are computed independently and then combined via aggregation techniques such as weighted sums, score-level fusion, or ensemble methods.
- **Sequential integration:** one technique is applied before or after the other. For instance, CBF may pre-filter candidate items for a CF model, or CF outputs may be refined using content-based information.
- **Unified models:** both collaborative and content-based information are incorporated into a single predictive framework.

By integrating multiple sources of evidence, hybrid systems can effectively address limitations that arise when using CF or CBF in isolation. They therefore offer

improved flexibility and performance across a wide range of domains and recommendation scenarios.

5.3 Recommendation Pipeline

The development of practical Recommender Systems (RSs) typically follows a structured workflow that organizes the main stages of model design, training, and assessment. Given the wide range of recommendation strategies and the increasing complexity of contemporary models, the use of a standardized pipeline is essential to support reproducibility and enable fair comparisons across methods [88, 340].

A standard recommendation pipeline, particularly in model-based settings, can be decomposed into three main phases: (i) *input pre-processing*, (ii) *model optimization*, and (iii) *model evaluation*. We briefly describe each phase below.

5.3.1 Input Pre-processing

Input pre-processing provides the foundation for any RS, as the quality, consistency, and organization of the data directly affect model accuracy and stability. This stage typically begins with the collection of raw user-item interaction data, followed by its transformation into a structured representation (e.g., a user-item interaction matrix). When available, additional metadata, such as item features or user profile information, are also prepared at this stage, particularly for content-based and hybrid methods. Further pre-processing steps may be applied directly to the interaction data to align it with the intended recommendation task.

5.3.1.1 Converting Explicit Feedback into Implicit Signals

As discussed in Section 5.1, interactions can be divided into two categories: *explicit* feedback (e.g., ratings) and *implicit* feedback (e.g., clicks, purchases, views). When the utility function f is designed for a top- k recommendation task, it is common to convert explicit ratings into implicit signals as part of the pre-processing phase. Conceptually, this corresponds to transforming a user-item matrix of the form shown in Figure 5.1b into a matrix of the form shown in Figure 5.2b. A typical approach is to binarize ratings according to a predefined threshold τ :

$$y_{u,i} = \begin{cases} 1 & \text{if } r_{u,i} \geq \tau, \\ 0 & \text{otherwise,} \end{cases}$$

where $r_{u,i}$ denotes the explicit rating of user u for item i , and τ is the threshold. Ratings greater than or equal to τ are interpreted as positive interactions, while those below τ are treated as non-interactions. Note that choosing $\tau = 1$ preserves all observed interactions.

5.3.1.2 P-core Filtering

To mitigate the impact of data sparsity on model training and evaluation, p -core filtering is frequently applied. The p -core procedure retains only users and/or items with at least p interactions. Formally, for a given interaction matrix, the filtered data satisfy

$$\deg(u) \geq p \quad \text{and/or} \quad \deg(i) \geq p, \quad \forall u \in \mathcal{U}, i \in \mathcal{I},$$

where $\deg(u)$ and $\deg(i)$ denote the number of interactions associated with user u and item i , respectively, and p is the minimum degree threshold. This pre-processing step is particularly useful when the handling of cold-start users and items is not a primary objective of the study. In this framework, users and items can be categorized as *warm* or *cold* according to p . Warm users have at least p interactions and correspond to the most active users, while cold users have fewer than p interactions. Analogously, warm items exceed the threshold p and are therefore relatively popular, whereas cold items fall below the threshold and exhibit limited interaction.

5.3.2 Model Optimization

After pre-processing, the next phase is the construction and optimization of the recommendation model. This step is specific to model-based RSs, where the goal is to learn latent representations of users and items, together with the model parameters Θ (see Section 5.2.1). The parameters Θ are estimated by minimizing a loss function that encodes the chosen recommendation objective and the underlying modeling assumptions.

In explicit-feedback scenarios, the *Mean Squared Error* (MSE) loss [362] is a common choice to penalize discrepancies between predicted and observed ratings:

$$\mathcal{L}_{\text{MSE}} = \sum_{(u,i) \in \mathcal{D}} (r_{u,i} - f(u,i))^2, \quad (5.5)$$

where $r_{u,i}$ is the true rating of user u for item i , $\mathcal{D}$ is the set of observed interactions, and $f(u,i)$ is the model prediction for the pair (u,i) .

For implicit-feedback data, the Bayesian Personalized Ranking (BPR) loss [300] is widely used. BPR is defined in terms of pairwise preferences. For each user $u \in \mathcal{U}$, an observed (positive) item $i^+ \in \mathcal{S}^+$ is assumed to be preferred over an unobserved (negative) item $i^- \in \mathcal{S}^- := \mathcal{I} \setminus \mathcal{S}^+$, and the loss encourages the corresponding score difference to be positive:

$$\mathcal{L}_{\text{BPR}} = \sum_{(u, i^+, i^-) \in \mathcal{D}} -\ln \sigma(f(u, i^+) - f(u, i^-)), \quad (5.6)$$

where $\sigma(\cdot)$ is the sigmoid function, and $f(u, i^+)$ and $f(u, i^-)$ are the predicted scores for the positive and negative items, respectively. The use of BPR naturally requires *negative sampling* strategies, that is, procedures to associate each observed interaction with one or more unobserved items. A standard approach is to sample a limited number of negatives per user. In particular, Rendle et al. [300] recommend sampling a single negative item for each observed user–item pair.

The RS literature proposes many other loss functions beyond MSE and BPR, including Pairwise Hinge (PH), Binary Cross-Entropy (BCE), Softmax Cross-Entropy (SCE), and various contrastive losses. These criteria differ in how they model the recommendation objective, but all share the goal of learning parameters Θ that produce accurate rankings or predictions.

5.3.3 Model Evaluation

Once the recommendation model has been trained and scores have been computed for the relevant user-item pairs, the next step is inference, that is, generating recommendation lists for each user. These lists are then evaluated to quantify the effectiveness of the algorithm. Evaluation protocols are typically divided into two broad categories: (i) offline evaluation, which is predominant in academic research, and (ii) online evaluation, which is more common in industrial settings. In this dissertation, we focus on offline evaluation, as it forms the basis for validating the proposed methods. In particular, we discuss three key aspects: (i) data splitting, (ii) hyper-parameter tuning, and (iii) evaluation metrics.

5.3.3.1 Data Splitting

Following pre-processing, the interaction data must be divided into separate subsets for training, validation, and testing. The training set is used to fit the model parameters, the validation set is employed for hyper-parameter tuning and early

stopping, and the test set is reserved for final performance assessment. Maintaining a clear separation among these subsets is a standard requirement in machine learning and is crucial to ensuring internal validity. Nonetheless, several studies [87, 337, 338] have reported that some offline evaluation protocols omit the validation set and directly tune hyper-parameters or perform early stopping on the test set. Such practices may inflate reported performance and reduce the reliability and reproducibility of results.

Below, we summarize the most frequently used splitting strategies in the RS literature.

Random Splitting. Random splitting is the most commonly adopted strategy [167, 338]. In this setting, each user’s interactions are randomly assigned to the training, validation, and test sets according to fixed proportions (e.g., 80% training, 10% validation, 10% test). While simple and efficient, this approach ignores the temporal order of interactions. As a result, it may place future interactions in the training set and earlier interactions in the test set, which is incompatible with real-world deployment scenarios. This issue undermines the internal validity of the evaluation unless one assumes that user preferences are time-invariant, an assumption that is rarely justified in practice.

Time-Aware Splitting. Time-aware splitting addresses this limitation by taking interaction timestamps into account. Interactions that occur before a specified time t are assigned to the training (and possibly validation) sets, while those occurring after t are used as test data. This arrangement ensures that the test set corresponds to future interactions relative to the training data, thereby providing a more realistic evaluation scenario.

However, time-aware splitting also introduces challenges. Because many users are active only for limited periods, it may produce very small training sets for some users, especially in long-term datasets where activity is sparse. Additionally, this strategy requires precise timestamps for each interaction, which may not be available in all public datasets, particularly those commonly used in academic work.

5.3.3.2 Hyper-parameter Tuning

Hyper-parameter tuning is a central part of designing and evaluating recommendation algorithms. Yet, it is often conducted unevenly. Several works [83, 87, 324] have shown that newly proposed models are sometimes tuned extensively, while base-

line methods receive limited or no tuning. Additionally, the search ranges and tuning protocols are rarely documented in detail [324, 430]. This lack of transparency hampers reproducibility and may bias comparisons.

Three main strategies are commonly used for hyper-parameter optimization in offline RS experiments: (i) *grid search*, (ii) *Bayesian optimization* with Tree-structured Parzen Estimators, and (iii) *random search*. Grid search is the most frequently reported approach [343], and it is the one predominantly adopted in this dissertation.

In grid search, a finite set of candidate values is defined for each hyper-parameter, and their Cartesian product forms the search space [115]. Each point in this space corresponds to a distinct configuration, and a separate model is trained for every configuration. This exhaustive exploration guarantees full coverage of the discrete search grid.

After training, a single configuration must be selected. This reflects both practical and scientific considerations. Only one model is ultimately deployed in an operational system, and it is the performance of the best configuration that is typically reported and compared against baselines in research papers.

It is essential that hyper-parameter tuning, and the corresponding selection of the best model, be carried out using the validation set, which must remain independent of the test set. This separation prevents overfitting to the test data and ensures that the reported test performance reflects genuine generalization.

To identify the best configuration, a target metric must be specified. This metric usually reflects recommendation accuracy, such as nDCG@10. The chosen model is then the one that attains the highest value of the target metric on the validation set.

Although offline hyper-parameter tuning is necessary for model selection, it has important limitations. There is no guarantee that the configuration that performs best offline will also perform best in an online environment [414]. Offline evaluations are inherently static and may not capture the dynamics of user behavior and system interactions. As a result, it is important to consider the robustness of an algorithm to small changes in hyper-parameters, as this can significantly affect its suitability for real-world deployment.

5.3.3.3 Measuring the Relevance of Recommendations

The final design choice in an offline evaluation protocol concerns the metrics used to assess recommendation quality on the test set. Metrics provide quantitative measures of how well the algorithm fulfills the intended objective. In RSs, this objective

is typically formulated as the capability to rank items in order of relevance for each user. Depending on the task, different accuracy-oriented metrics can be employed.

For rating prediction, common choices include Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Square Error (RMSE). In this section, however, we focus on metrics for the top- k recommendation task, which is the evaluation setting adopted in this dissertation and is prevalent in the RS literature. Top- k metrics assess whether relevant items appear among the first k positions of the recommended list, where k is usually selected from $\{1, 5, 10, 20, 50\}$. When recommendation lists contain more than k items, the metric is computed using only the top- k prefix.

Let $\mathcal{U}$ be the set of users, $\mathcal{L}_u$ the recommendation list for user u , $\mathcal{L}_u^{(1,\dots,k)}$ the top- k items for u , and $\mathcal{I}_u^{+, \text{test}}$ the set of items relevant to u in the test set. We report below the most widely used accuracy-oriented metrics for top- k recommendation.

Definition 2 (Precision@ k). Precision@ k is defined as the average, across users, of the fraction of recommended items in the top- k that are relevant:

$$\text{Precision@}k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\mathcal{L}_u^{(1,\dots,k)} \cap \mathcal{I}_u^{+, \text{test}}|}{k}. \quad (5.7)$$

Definition 3 (Recall@ k). Recall@ k is defined as the average fraction of each user's relevant items that appear among the top- k recommendations:

$$\text{Recall@}k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\mathcal{L}_u^{(1,\dots,k)} \cap \mathcal{I}_u^{+, \text{test}}|}{|\mathcal{I}_u^{+, \text{test}}|}. \quad (5.8)$$

Definition 4 (normalized Discounted Cumulative Gain@ k). The normalized Discounted Cumulative Gain (nDCG@ k) accounts for the position of relevant items within the ranking, assigning higher importance to items appearing early in the list:

$$\text{nDCG@}k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{\text{IDCG@}k} \sum_{i \in \mathcal{L}_u^{(1,\dots,k)}} \frac{2^{\text{rel}_i} - 1}{\log_2(i + 1)}, \quad (5.9)$$

where $\text{rel}_i = 1$ if the item at rank $i \in \{1, \dots, k\}$ is relevant for the user and 0 otherwise. The term $\text{IDCG@}k = \sum_{i=1}^k \frac{1}{\log_2(i+1)}$ denotes the ideal DCG, corresponding to the ranking in which all relevant items occupy the top positions.

Definition 5 (Hit Ratio). The Hit Ratio ($HR@k$) measures the proportion of users for whom at least one relevant item appears in the top- k recommendation list:

$$HR@k = \frac{|\mathcal{U}_{\text{hits}@k}|}{|\mathcal{U}|}, \quad (5.10)$$

where $|\mathcal{U}_{\text{hits}@k}|$ is the number of users for whom at least one relevant item is contained in $\mathcal{L}_u^{(1,\dots,k)}$. Higher values of $HR@k$ indicate more effective recommendation lists.

5.4 Multiple Perspectives of Recommendation

Recommender Systems (RSs) have traditionally been designed to optimize a single primary objective, namely, providing users with relevant content. While accuracy is central to sustaining user trust and engagement, an exclusive focus on relevance can lead to unintended consequences. On the one hand, users may become confined to so called filter bubbles [350], where the system repeatedly recommends similar content and limits exposure to novel or diverse items. On the other hand, a purely relevance-driven objective may overlook the interests of other stakeholders, such as businesses and content providers. In particular, systematic imbalances in item exposure can generate unfair outcomes for content creators, leading to disparities in visibility. Thus, although predicting the relevance of individual items for users remains a key goal, treating recommendation as a single-objective problem risks oversimplifying the task and neglecting the multiple perspectives involved. As a result, a growing body of research has focused on broadening both optimization and evaluation to incorporate additional dimensions.

This section provides an overview of beyond-accuracy aspects considered in this dissertation and discusses how they can be evaluated. We group these aspects into two main categories. (i) *user-centric* objectives and (ii) *multi-stakeholder* objectives.

5.4.1 User-Centric Objectives

User-centric objectives are quality dimensions introduced to improve the perceived usefulness and experience of end users. In many settings, simply increasing the relevance of recommended items is not sufficient to satisfy users. Repeatedly suggesting obvious or highly similar items can lead to monotonous recommendation lists, which may be perceived as unhelpful. Instead, users often benefit from being ex-

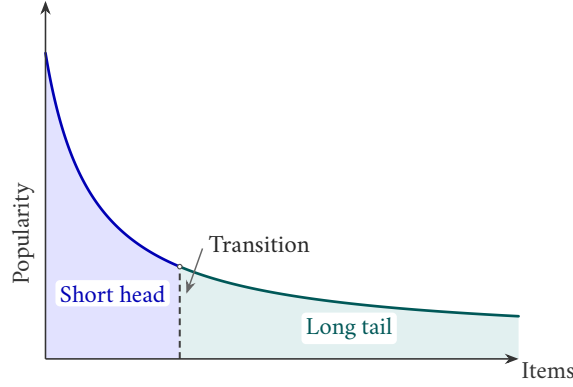


Figure 5.4 Long-tail distribution of item popularity. A small set of highly frequent items forms the short head, while the majority of items occur rarely in the long tail. The dashed line marks the head–tail cutoff.

posed to a broader set of items, including those that are less popular or less widely known. In this sense, recommendations should ideally be not only relevant but also *novel* and *diverse*.

5.4.1.1 Novelty

The **novelty** of an item for a user expresses how different it is from what the user is already familiar with. Some works distinguish novelty from *serendipity*, where a serendipitous item is both novel and surprising [150, 248]. To separate these notions more clearly, novelty is often defined in a user-independent manner. In practice, item novelty is commonly quantified as the inverse of item popularity, for example based on the number of ratings or interactions an item receives [177]. Items with high novelty tend to lie in the *long-tail* of the popularity distribution, whereas popular items belong to the *short-head* (Figure 5.4).

Given the set of users $\mathcal{U}$, the recommendation list $\mathcal{L}_u$ for a user u , and its top- k items $\mathcal{L}_u^{(1,\dots,k)}$, we consider two novelty metrics, Expected Popularity Complement (EPC) [61] and Expected Free Discovery (EFD) [61].

Definition 6 (Expected Popularity Complement). Expected Popularity Complement (EPC) measures the expected number of relevant items that lie in the long-tail:

$$\text{EPC}@k = \frac{c}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \sum_{i_k \in \mathcal{L}_u^{(1,\dots,k)}} p(\text{seen} \mid k, u, \mathcal{L}_u^{(1,\dots,k)}) p(\text{rel} \mid i_k, u) (1 - p(\text{seen} \mid i_k)), \quad (5.11)$$

where c is a normalizing constant and $p(\text{seen} \mid \cdot)$ and $p(\text{rel} \mid \cdot)$ denote the probability

that an item is seen and relevant, respectively. Higher values indicate a greater presence of relevant long-tail items in the recommended lists.

Definition 7 (Expected Free Discovery). Expected Free Discovery (EFD) is a novelty metric based on the expected inverse collection frequency. It measures the ability of an algorithm to surface relevant items from the long-tail:

$$\text{EFD}@k = \frac{c}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \sum_{i_k \in \mathcal{L}_u^{(1, \dots, k)}} p(\text{seen} \mid k, u, \mathcal{L}_u^{(1, \dots, k)}) p(\text{rel} \mid i_k, u) (-\log_2(p(i \mid \text{seen}))), \quad (5.12)$$

where c is a normalizing constant and $p(\text{seen} \mid \cdot)$ and $p(\text{rel} \mid \cdot)$ are defined as above. The term $p(i \mid \text{seen})$ encodes an item popularity factor. High novelty values correspond to items that only a few users have interacted with (long-tail items), while lower values correspond to popular head items. As for EPC, higher EFD values reflect a stronger presence of relevant long-tail items.

5.4.1.2 Diversity

Diversity is typically defined for a set of items rather than for a single one. The **diversity** of a set refers to how dissimilar the items are from one another. In RS research, diversity is usually divided into (i) *individual diversity* and (ii) *aggregate diversity*. Individual diversity characterizes how varied the items in a single user's recommendation list are. Aggregate diversity quantifies how many distinct items are recommended across the entire user base.

Given the set of users $\mathcal{U}$, the recommendation list $\mathcal{L}_u$ for user u , and its top- k items $\mathcal{L}_u^{(1, \dots, k)}$, we introduce two metrics to assess aggregate diversity. The Gini Index (Gini) [61] and the Item Coverage (IC).

Definition 8 (Gini Index). The Gini Index ($\hat{\text{Gini}}@k$) is an aggregate diversity measure that captures distributional inequality, that is, how unevenly different items are recommended across users:

$$\hat{\text{Gini}}@k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{k-1} \frac{\sum_{i=1}^{|\mathcal{L}_u^{(1, \dots, k)}|} (2i - |\mathcal{L}_u^{(1, \dots, k)}|) \text{times}(i)}{\sum_{i=1}^{|\mathcal{L}_u^{(1, \dots, k)}|} \text{times}(i)}, \quad (5.13)$$

where $\text{times}(i)$ denotes the number of times item i appears in recommendation lists. The value $\hat{\text{Gini}}@k$ is 0 when all items are recommended equally often and approaches 1 when recommendations concentrate on a single item.

To maintain the convention that larger values represent better outcomes, in the remainder of this dissertation we refer to $\text{Gini}@k = 1 - \hat{\text{Gini}}@k$.

Definition 9 (Item Coverage). The Item Coverage ($\text{IC}@k$) measures how many distinct items appear in the top- k recommendations across all users:

$$\text{IC}@k = \left| \bigcup_{u \in \mathcal{U}} \mathcal{L}_u^{(1, \dots, k)} \right|. \quad (5.14)$$

Higher coverage indicates that a broader set of items is being recommended, suggesting greater aggregate diversity.

5.4.2 Multi-Stakeholder Objectives

While end users remain central to the design and evaluation of Recommender Systems, it is increasingly important to extend the analysis to other stakeholders who influence or are influenced by recommendation outcomes. The different categories of stakeholders involved in recommendation processes can be organized as follows [2]:

- **Consumers** are the end-users of the platform who receive recommendations. They interact with the system to satisfy their needs or goals and expect recommendations that are useful and appropriate.
- **Providers** are entities associated with the items that are recommended. Their definition depends on the domain and the analysis perspective. In a movie recommendation context, providers may include production companies, directors, actors, or even countries of production. Providers influence the pool of available content and are directly affected by exposure patterns.
- **System** refers to the organization that operates the recommendation platform. This entity acts as an intermediary that connects consumers and items. Examples include retailers, e-commerce platforms, brokers, or similar intermediaries.

Ideally, RSs should “create value in parallel for all involved stakeholders” [164], where value captures the perceived quality or benefit of the recommendations for each stakeholder. The notion of value is strongly context dependent and may reflect economic and business-related goals or societal and human-centric goals [31].

In this dissertation, we focus on societal and human-centric aspects that are particularly relevant to the stakeholders outlined above. Specifically, we emphasize as-

pects that are closely tied to the notion of *algorithmic bias*, which plays a key role in diagnosing and mitigating unfair outcomes.

5.4.2.1 Algorithmic Bias

Algorithmic bias in RSs refers to systematic and unfair patterns in the behavior of a recommendation algorithm that disadvantage particular users, groups, or items [243]. Such bias can arise at multiple stages of the pipeline, including data collection, model training, and system deployment. Addressing algorithmic bias is crucial to support fair and trustworthy recommendation outcomes.

One widely studied instance of algorithmic bias is **popularity bias**. In this case, the system tends to over-recommend popular items, which amplifies their visibility and further diminishes the exposure of long-tail items [34]. Given the set of users $\mathcal{U}$, a recommendation list $\mathcal{L}_u$ for each user u , and its top- k subset $\mathcal{L}_u^{(1,\dots,k)}$, we describe two measures that quantify popularity bias. The Average Percentage of items in the Long-Tail (APLT) [3] and the Average Recommendation Popularity (ARP) [409].

Definition 10 (Average Percentage of items in the Long-Tail). The Average Percentage of items in the Long-Tail (APLT@ k) measures the proportion of recommended items that belong to the long-tail:

$$APLT@k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\{i \mid i \in (\mathcal{L}_u^{(1,\dots,k)} \cap \Gamma)\}|}{|\mathcal{L}_u^{(1,\dots,k)}|}, \quad (5.15)$$

where Γ denotes the set of long-tail items. Higher values indicate that a larger fraction of the recommended items are unpopular items from the long-tail.

Definition 11 (Average Recommendation Popularity). The Average Recommendation Popularity (ARP@ k) measures the average popularity of the items recommended:

$$ARP@k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{\sum_{i \in \mathcal{L}_u^{(1,\dots,k)}} \varphi(i)}{|\mathcal{L}_u^{(1,\dots,k)}|}, \quad (5.16)$$

where $\varphi(i)$ is the number of interactions recorded for item i . Lower ARP values correspond to lower overall popularity of the recommended items and therefore to a stronger presence of long-tail items.

Part 3

**Large Language Models for
Recommendation**

Chapter 6

Evaluating ChatGPT as a Recommender System: A Multi-Perspective Study

Large Language Models (LLMs) have recently demonstrated impressive capabilities across a wide range of natural language processing tasks. Among them, ChatGPT has shown particularly strong performance under zero-shot and few-shot prompting settings. Motivated by these results, the Recommender Systems (RSs) research community has begun to explore the potential of LLMs within recommendation scenarios. However, despite the growing number of approaches that integrate ChatGPT into RS pipelines, existing studies often lack comprehensive evaluation protocols tailored to the generative nature of these models. In particular, current evaluations frequently overlook critical issues such as hallucinations, duplicated recommendations, and out-of-domain items, while focusing primarily on accuracy metrics and neglecting beyond-accuracy aspects.

To address this gap, we propose a robust evaluation pipeline designed to systematically assess ChatGPT's effectiveness as a recommender and to post-process its generated recommendations in order to account for these challenges. Using this pipeline, we evaluate ChatGPT-3.5 and ChatGPT-4 under a zero-shot setting with role-playing prompts. The analysis considers three recommendation scenarios: top- N recommendation, cold-start recommendation, and re-ranking of an existing recommendation list. Experiments are conducted across three application domains: movies, music, and books.

The results show that ChatGPT achieves higher accuracy than strong baselines in the book domain and demonstrates particularly strong performance in re-ranking and cold-start settings, while maintaining competitive beyond-accuracy metrics.

To support reproducibility and future research, the proposed evaluation pipeline is publicly released.¹

6.1 Introduction

The exponential growth of human-generated data, primarily in the form of unstructured text, has fundamentally reshaped the digital landscape. The World Wide Web has evolved into a vast repository of information, reflecting a wide range of human and organizational needs, opinions, and knowledge. However, extracting value from this immense resource remains a significant challenge.

Addressing this challenge requires the application of Natural Language Processing (NLP) techniques. NLP encompasses a broad set of computational methodologies [6] designed to analyze, understand, and extract meaning from natural language. By leveraging these techniques, researchers and practitioners across multiple domains, including data science, information technology, marketing, and social research, can unlock the potential of large-scale unstructured data.

At the same time, the increasing ease of information sharing and retrieval has accelerated the pace of information dissemination. This rapid expansion has raised concerns related to information overload [52], content quality, privacy, and security.

To cope with these challenges, several approaches have been proposed, ranging from Information Retrieval (IR) systems [256] to Recommender Systems (RSs) [303]. These systems aim to filter massive collections of data to surface information that is both relevant and useful. Their effectiveness, however, largely depends on their ability to discard irrelevant or low-quality content, allowing users to access reliable and pertinent information efficiently.

IR systems have progressively incorporated advanced NLP techniques, such as semantic analysis and natural language understanding, to move beyond simple keyword matching. These methods enable a deeper interpretation of user queries, im-

1. The evaluation pipeline is available at <https://github.com/sisinflab/Recommender-ChatGPT>.

This chapter is based on work published in the Proceedings of the 17th ACM Conference on Recommender Systems (RecSys'23), "Retrieval-augmented Recommender System: Enhancing Recommender Systems with Large Language Models", the Proceedings of the 14th Italian Information Retrieval Workshop (IIR'24), "Beyond Words: Can ChatGPT Support State-of-the-Art Recommender Systems?", and the Proceedings of the 1st Workshop on the Next Generation of Information Retrieval and Recommender Systems with Language Agents, Generative Models, and Conversational AI (GENNEXT@SIGIR'25), "Exploring Diversity, Novelty, and Popularity Bias in ChatGPT's Recommendations".

proving contextual relevance and retrieval effectiveness.

Similarly, Recommender Systems have become increasingly proficient at predicting user preferences and delivering personalized content. By analyzing historical behavior, user interactions, and explicit or implicit preferences, RSs generate recommendations tailored to individual tastes. The integration of NLP techniques further enhances this process by enabling a more nuanced modeling of user intent and preferences, ultimately leading to more relevant recommendations.

Recent studies [37, 121, 426] have also highlighted the benefits of interactive systems in improving user satisfaction and overall experience. Through human-like dialogues, these systems can elicit more precise signals from users, allowing recommendations to be better aligned with individual needs.

This transition toward conversational interfaces has contributed to the widespread adoption of digital assistants such as Amazon Alexa, Google Assistant, Microsoft Cortana, and Apple Siri [236]. Their capacity to process natural language queries in real time has transformed user interaction with digital systems, making access to information more intuitive and efficient.

The emergence of advanced Language Models, and in particular Large Language Models (LLMs) such as Generative Pre-trained Transformer 3 (GPT-3) [49], has further amplified the potential for natural language interaction with digital systems [293]. These models facilitate access to unprecedented volumes of information through free-form natural language queries.

LLMs have ushered in a new era of language understanding and generation, enabling interactions characterized by increased fluency and contextual awareness. Their capabilities extend beyond traditional query processing, supporting dynamic and nuanced dialogues that closely resemble human communication. Although GPT-3 demonstrated substantial progress in text generation, it was with the release of ChatGPT on November 30, 2022² that a large-scale conversational model capable of addressing a wide range of tasks became publicly accessible [49].

Following this development, the research community has shown growing interest in investigating the application of ChatGPT to diverse tasks. In particular, several studies have explored its potential as a recommender system [124, 155, 374, 413]. While recent efforts have focused on improving recommendation accuracy by integrating ChatGPT with RS pipelines [90, 221], many works overlook beyond-

2. <https://openai.com/blog/chatgpt/>

accuracy dimensions.

Although initial studies have begun to address fairness considerations [418], other critical aspects remain underexplored. These include recommendation diversity, the ability to suggest novel items, and the mitigation of popularity bias. Moreover, evaluating LLM-based recommender systems introduces additional challenges, such as managing hallucinations, avoiding duplicated recommendations, and preventing out-of-domain suggestions that do not belong to a predefined catalogue or evaluation setting.

LLM-based recommender systems hold the promise of transforming how users discover content by enabling highly personalized and engaging experiences. However, understanding how these models capture user preferences requires a rigorous and systematic evaluation framework capable of interpreting their generative behavior.

Such an evaluation pipeline should provide a comprehensive analysis of system performance, identifying both strengths and limitations. This analysis must go beyond accuracy metrics and examine the overall quality and relevance of recommendations. It should assess whether the system avoids repetitive outputs, refrains from producing implausible or hallucinated items, and delivers recommendations that are diverse and aligned with user interests. Establishing such a pipeline is a key step toward unlocking the full potential of LLM-powered recommender systems.

To address these challenges, we propose an evaluation pipeline designed to assess the strengths and weaknesses of ChatGPT when employed as a recommender system. Using this pipeline, we analyze the performance of ChatGPT-3.5 and ChatGPT-4 on recommendation tasks under a zero-shot role-playing prompt setting.

This evaluation examines the ability of these models to function as standalone recommender systems across three domains. Specifically, we consider Books, Movies, and Music, using the Facebook Books dataset [241], MovieLens [139], and Last.fm [56]. We further investigate three recommendation scenarios: top- N recommendation, re-ranking of candidate lists, and cold-start recommendation.

Our main contributions can be summarized as follows:

- We design and publicly release an evaluation pipeline³ for assessing ChatGPT as a recommender system, explicitly accounting for the generative characteristics of LLMs.

3. <https://github.com/sisinflab/Recommender-ChatGPT>

- We propose a comprehensive evaluation protocol that extends beyond accuracy, systematically analyzing diversity, novelty, popularity bias, and generative artifacts such as hallucinations and duplicated recommendations.
- We conduct an extensive empirical evaluation of ChatGPT-3.5 and ChatGPT-4 across three recommendation scenarios, namely top- N recommendation, re-ranking, and cold-start, and across three domains: Books, Movies, and Music.

6.2 An Evaluation Pipeline for Assessing ChatGPT in Recommendation Tasks

The following sections discuss the methodology and evaluation techniques used in our research. Specifically, Section 6.2.1 outlines the design and implementation of the used Prompt, detailing the latest prompt engineering strategies and the decision to use Role-Play Prompting. Section 6.2.2 examines the datasets employed in the experiments and the necessary pre-processing steps to respect the constraints given by ChatGPT. Section 6.2.3 presents the evaluation pipeline, laying the foundation for the study. Finally, Section 6.3.2 and 6.2.4 discusses the baselines and metrics for assessing ChatGPT's performance in Recommendation tasks, providing a comprehensive overview of the evaluative criteria.

6.2.1 Prompt Design

The development of Large Language Models (LLMs), especially the GPT series such as GPT-3 [49], has significantly transformed fields such as technology, education, and content creation. At the core of interactions with these models lies the concept of a "prompt". A prompt is essentially a piece of natural language text given to the model by the user, serving as the initial input or question. This input initiates the model's response process and guides its direction and scope, acting as a catalyst for the output's relevance and utility.

Understanding prompt design is crucial because LLMs learn from large datasets, acquiring the ability to understand language and contextual nuances. Furthermore, they demonstrate the ability to perform various tasks by following prompts. Consequently, well-designed prompts enable users to effectively guide the model's output generation process, increasing the likelihood of obtaining relevant and valuable results for the intended task.

However, it was only after the emergence of GPT-3 in 2020 [49] that the signifi-

cance of crafting effective prompts became evident, giving rise to "prompt engineering" as a dedicated field of study. A field that explores how to prompt LLMs to adapt to new tasks using minimal examples, thereby optimizing the interaction process.

Among the several prompt engineering techniques that can be employed [312], our exploration focused on the following:

Zero-shot prompting [293]. The most basic and widely used approach. In zero-shot prompting, the context information and task description are provided directly to the LLM without reference examples. This approach leverages the LLM's pre-trained knowledge and capabilities to generate relevant responses based solely on the provided context and instructions.

Zero-shot prompt

I like [history of the user], provide me 50 recommendations.

Few-shot prompting [49]. In contrast to zero-shot prompting, few-shot prompting provides a few demonstrations or examples in the prompt to help LLM better understand the user's intent. Therefore, selecting relevant examples plays a vital role in guiding the LLM toward generating personalized and appropriate recommendations.

Few-shot prompt

User's Watched Movie: "The Shawshank Redemption" User's Rating: 5 (out of 5) Recommended Movie: "The Green Mile" Explanation: Both movies are critically acclaimed drama films with themes of hope and resilience in difficult circumstances. Since the user highly rated "The Shawshank Redemption", they might also enjoy "The Green Mile".

User's Watched Movie: "Inception" User's Rating: 4 Recommended Movie: "Interstellar" Explanation: Both movies are science-fiction films directed by Christopher Nolan, known for their mind-bending plots and impressive visuals. A user who enjoyed "Inception" is likely to appreciate "Interstellar" as well.

User's Watched Movie: "The Dark Knight" User's Rating: 5 Recommended Movie:

Chain-of-Thought prompting [380]. Also known as CoT prompting, this technique is a widely adopted approach for reasoning tasks such as question answering

and mathematical inference. This method prompts the LLM to solve problems by breaking them into explicit steps, effectively eliciting the model's ability to reason step-by-step.

Chain-of-Thought prompt

Instruction: Recommend a book based on the user's preferences.

User's Preferences: The user enjoys science fiction, particularly books with strong character development and intricate world-building. They have previously enjoyed "Dune" by Frank Herbert and "Ender's Game" by Orson Scott Card.

Chain of Thought: To recommend a suitable book, I need to consider the user's preferences for science fiction, strong character development, and intricate world-building. The user enjoyed "Dune" and "Ender's Game," which suggests they appreciate complex plots and detailed settings. Based on these criteria, I will identify a book that fits within the science fiction genre and has a reputation for similar qualities.

1. The user likes science fiction, so I will focus on books within this genre.
2. The user values strong character development, so I will consider books known for their well-developed characters.
3. The user appreciates intricate world-building, so I will look for books with detailed and immersive settings.
4. Based on their enjoyment of "Dune" and "Ender's Game," I'll look for books with complex plots and critical acclaim.

Recommendation: "The Left Hand of Darkness" by Ursula K. Le Guin

Explanation: "The Left Hand of Darkness" is a renowned science fiction novel known for its detailed world-building and exploration of complex social and cultural issues. Ursula K. Le Guin's character development is exceptional, and the story's setting on the planet Gethen provides a rich and immersive experience. This book should align well with the user's preferences and previous enjoyment of "Dune" and "Ender's Game."

Role-Playing prompting [172]. Role-playing prompts involve engaging with the language model in a "role-playing" scenario. This approach involves providing aux-

iliary information in the prompt to describe the role and responsibilities of the role in detail.

Role-Playing prompt

I want you to take on the role of a recommender system. Your goal is to provide tailored book suggestions that users will likely to enjoy based on their history.

In our study, we aimed to test how well ChatGPT could generate 50 recommendations by looking at a user's history. Initially, our approach employed Few-Shot and Chain-of-Thought prompting techniques to sample ChatGPT's capabilities. However, we encountered a significant challenge due to the constraints on context length imposed by ChatGPT, specifically its version 3.5. Moreover, the vast nature of user histories made the application of few-shot and CoT prompting impractical for our datasets, as these methods often require the inclusion of multiple examples or detailed step-by-step reasoning.

Faced with these limitations, our strategy pivoted towards crafting a straightforward yet impactful zero-shot role-playing prompt that could accommodate our requirement to generate a comprehensive list of 50 recommendations while navigating the constraints of the model's capacity to process extensive information. The refined prompt we developed was as follows:

Top-N Recommendation Prompt

Given a user, as a Recommender System, please provide only the names of the top 50 recommendations. You know that the user likes the following items: [history of the user].

Through heuristic testing (i.e., employing an empirical, trial-and-error approach based on experience and intuition), this prompt emerged as the most effective in eliciting 50 relevant recommendations from ChatGPT. Concurrently, there was a growing body of research examining the efficacy of various prompting strategies. Notably, these studies underscored the effectiveness of Role-Playing Prompting [188, 400]. In our case, we requested the language model to assume a specific role: a Recommender System. This minimalistic yet focused approach to prompting ChatGPT enhanced the precision and relevance of the recommendations generated. Encouraged by these findings, we employed Role-Playing Prompting to generate the collections of user recommendations for our study.

Table 6.1 Dataset statistics before and after pre-processing with max context length filtering and $k - core \geq 10$.

Dataset	Interaction	Users	Items	Sparsity	Interaction	Users	Items	Sparsity	Content	
	<i>before pre-processing</i>				<i>after pre-processing</i>				<i>type</i>	<i>features</i>
MovieLens	100836	610	9724	98.30%	42456	603	1862	96.22%	genre	20
Last.FM	86608	1892	12133	99.62%	49171	1797	1507	98.18%	genre	9748
FB Books	18978	1398	2234	99.74%	13117	1398	2234	99.58%	genre, author	1970

6.2.2 Pre-processing Phase

In the field of Recommender Systems (RSs), the most prevalent approach to evaluate the performance of a model is through offline experiments using pre-existing datasets, often historical data or logs. Among the various datasets, we employed three widely used and publicly available ones spanning different domains, i.e., books, music, and movies.

Specifically, we used the Facebook Books [241], Last.FM [56], and MovieLens [139] datasets. Although these datasets are well-known in the literature, we had to account for the token limits of ChatGPT API. Due to this limitation, we employed specific preprocessing steps to adapt the user interaction histories to the context length imposed by the API.

Consequently, we applying an iterative-10-core strategy on the users and items within the datasets to keep only users and items with at least ten occurrences. However, after analyzing each dataset, we discovered that for MovieLens, the interaction history of some users exceeded the maximum context length, and further preprocessing was required to reduce the number of interactions. For this reason, we set a threshold of 200 interactions to deal with the context constraints, leading to a modified MovieLens dataset.

The statistics about these datasets before and after pre-processing are reported in Table 6.1.

A concise overview of each dataset is provided below:

Facebook Books Dataset. Facebook Books⁴ is a dataset released for the Linked Open Data Challenge 2015⁵ covering the book domain. It contains implicit feedback and item-feature mappings to DBpedia for each book, enabling the retrieval of data content such as book genres and relevant author information.

Last.FM Dataset. The Last.FM dataset corresponds to user-artist plays on the

4. <https://github.com/sisinflab/LinkedDatasets/>

5. <https://2015.eswc-conferences.org/program/semwebeval.html>

Table 6.2 Comparison of ChatGPT-3.5 under the old and new evaluation pipelines across Accuracy, Diversity, Novelty, and Bias. The symbols $\uparrow$ and $\downarrow$ denote the preferred directions for each metric. Best values are shown in **bold**.

	Evaluation	nDCG $\uparrow$	Recall $\uparrow$	Precision $\uparrow$	Gini $\uparrow$	ItemCV $\uparrow$	EPC $\uparrow$	EFD $\uparrow$	APLT $\uparrow$	ARP $\downarrow$
Books	New Pipeline	0.0615	0.0917	0.0194	0.0389	616	0.0225	0.2176	0.1185	63.0937
	Old Pipeline	0.0574	0.0863	0.0184	0.0439	689	0.0210	0.2032	0.1737	58.1202
Music	New Pipeline	0.2253	0.1835	0.1299	0.1495	827	0.1532	1.4855	0.1078	106.9603
	Old Pipeline	0.2268	0.1853	0.1311	0.1517	833	0.1535	1.4886	0.1077	106.5101
Movies	New Pipeline	0.1519	0.0815	0.1298	0.0367	406	0.1208	1.3507	0.0435	122.0000
	Old Pipeline	0.1514	0.0815	0.1297	0.0371	408	0.1203	1.3449	0.0427	121.6523

Last.FM online music system released during the HETRec2011 Workshop⁶ [56]. It contains information on social networking, tagging, artists, and music listening from 2000 users.

MovieLens Dataset. The MovieLens dataset is widely exploited in the Recommender Systems community [139]. Different versions are available⁷, but the one used in our study was collected from the MovieLens 100k and contains ratings for movies on a 1-5 scale.

6.2.3 Evaluation Pipeline

As highlighted in Section 6.2.2, the most prevalent approach to evaluate the performance of recommender systems is through offline experiments using pre-existing datasets. Specifically, the models are trained on these datasets, learning to represent items by their unique itemIDs. During inference, the same itemIDs are used for prediction. Consequently, the classical evaluation pipeline for recommender systems utilizes the list of predicted recommendations based on itemIDs to assess the models' performance.

However, Large Language Models (LLMs) do not generate item IDs; instead, they produce text representing the item names. Consequently, evaluating the performance of ChatGPT in generating recommendations poses a significant challenge due to the complex task of matching the item names provided by the model with those present in the datasets, especially when dealing with noisy datasets where item names are not accurately represented.

An initial attempt was made to evaluate ChatGPT's performance using a simple

6. <https://grouplens.org/datasets/hetrec-2011/>

7. <https://grouplens.org/datasets/movielens/>

pattern-matching approach. Despite the seemingly promising results, the evaluation method was fundamentally flawed, leading to incorrect conclusions due to various reasons. The primary factor contributing to the inaccurate evaluation stemmed from the failure to properly match items within the dataset, See Figure 6.5 in the supplementary material for an example of a typical response from ChatGPT. Specifically, (i.) the pattern-matching algorithm failed to account for variations in item representations, such as matching "IQ84" or "The Twelve Monkeys" by ChatGPT with "IQ84" and "12 Monkeys" in the Facebook Books and MovieLens datasets; (ii.) it incorrectly matched items, e.g., associating "The Dark Tower III" or "Zombieland: Double Tap (2019)" by ChatGPT with "The Dark Tower II" and "Zombieland (2009)" in the datasets.

Furthermore, as known in the literature, generative models like ChatGPT tend to hallucinate under certain conditions [70, 169, 262]. Upon further analysis, we discovered that in our study, ChatGPT would arbitrarily repeat the same item (see Figure 6.6 in the supplementary material), or sequence of items (See Figure 6.7 in the supplementary material), after recommending a certain number of them until reaching the imposed limit of 50 recommendations. This behavior led to misevaluation by either improving or worsening the evaluation metrics, depending on the specific items repeated and the metric studied.

Motivated by these findings, this work present in Figure 6.1 a comprehensive pipeline to evaluate ChatGPT's recommendation capabilities while accounting the specific characteristics of closed-domain datasets, i.e., Facebook Books, MovieLens and Last.FM.

Specifically, the pipeline employs a multi-stage process that commences by (1) loading a folder containing the collections of the generated text files, each representing a list of 50 recommendations per user under evaluation. For each file and line representing an item, a multi-step search algorithm is utilized to probe for a corresponding item in the dataset.

The algorithm begins with an initial parsing step (2) aimed at separating the names of the recommended items from any additional contextual information within the sentence. Subsequently, a (3) Gestalt pattern matching technique [103, 129] is used to search for the closest match between the recommended item and the reference dataset. If the similarity score from the pattern matching exceeds a predetermined threshold of 70%, i.e., empirical value capable of covering many items across all

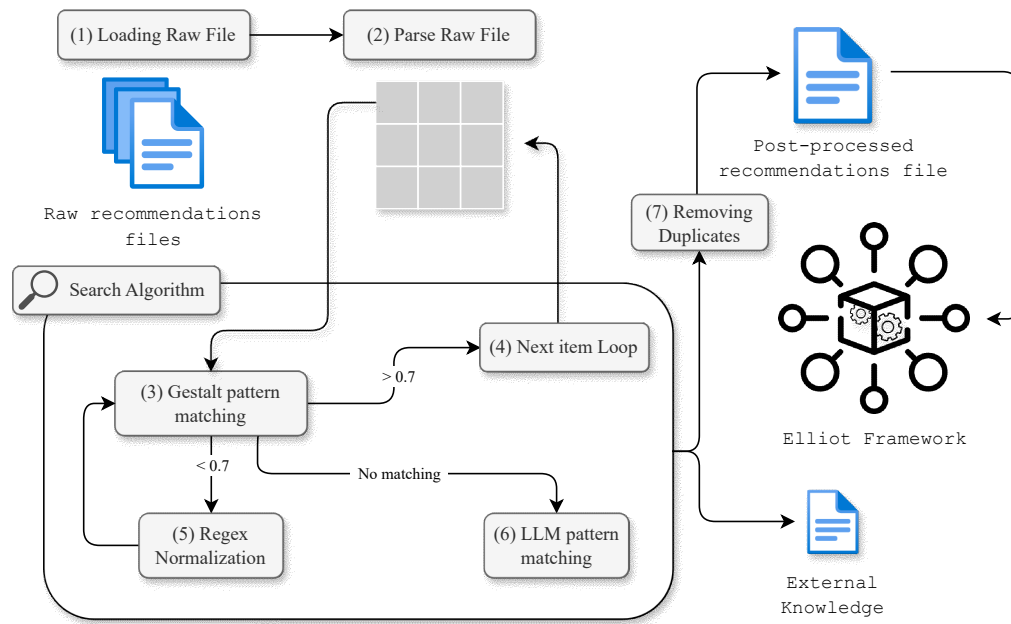


Figure 6.1 Multi-Stage Evaluation Pipeline for Assessing ChatGPT Recommendation Performance.

three datasets, the search algorithm proceeds to the next item (4) and repeat the loop. Otherwise, a regular expression (5) is applied to normalize the string, removing accents, converting characters to lowercase, eliminating punctuation, and trimming extra whitespace. This step is applied only in the second case because, in some instances, these elements for certain items are essential to reaching the threshold, and only for a narrow set of items does this step aid in finding a match.

Therefore, for the remaining unrecognized items, a final step leverages a quantized 7-billion parameter LLM (6), i.e., Mistral 7B [171, 347], which, through few-shot prompt engineering techniques (refer to Figure 6.4 in the supplementary material), identifies items that are impossible to recognize using classical methods, e.g., The "Twelve Monkeys" with "12 Monkeys" or "The Secret of My Success" with "Secret of My Succe\$s, The". Only after this last step, if no match is found in the dataset, the item is flagged as "External Knowledge" representing information known to the LLM due to its training but absent from the reference dataset. As a last step (7), the duplicate items are removed for each user, and only users with at least 20 valid recommendations are used for the overall evaluation. Additionally, the algorithm employs a caching strategy to speed up the search process and reduce the reliance on the LLM, which is utilized for only 5.56% of the total items, thereby minimizing

the impact and improving search speed.

A summary of the impact of this pipeline is presented as follows:

- For the Facebook Books dataset, on average, 3% of users had less than 20 unique recommendations, and approximately 2800 items were labeled as external knowledge or duplicates.
- The Last.FM dataset exhibited a mean of 1.11% of users with fewer than 20 unique recommendations, along with an average of 919 items identified as external knowledge or duplicates.
- In the case of the MovieLens dataset, the mean percentage of users with less than 20 unique recommendations stood at 2.66%, while an average of 252 items were classified as external knowledge or duplicates.

From the perspective of the evaluation metrics, the impact of the new evaluation pipeline is shown in Table 6.2. Although for some metrics and datasets, the difference between the old and new pipeline could appear minimal, the importance of having metrics based on a rigorous post-processing approach cannot be overstated. An aspect that currently needs to be improved in the state-of-the-art evaluation of ChatGPT as an RS.

6.2.4 Assessing ChatGPT performance

To ensure a comprehensive evaluation of ChatGPT in a recommendation scenario, we assess its performance from several perspectives. Specifically, we collect accuracy metrics and "beyond-accuracy" metrics commonly used in the literature to assess additional quality aspects of recommendation lists.

- In terms of *accuracy metrics*, we are interested in evaluating whether the recommender system predicts items that the user will interact with. To assess this, we use two standard metrics: Precision and Recall [266, 303]. Higher values of Precision and Recall indicate that the recommender system recommends a larger number of relevant items to the users. However, beyond just considering the relevance of the recommendations, our evaluation also considers the ranking of the items using the Normalized Discounted Cumulative Gain (nDCG) [165]. The nDCG metric measures not only the relevance of the recommended items but also their positions in the ranked list, giving more weight to highly relevant

Table 6.3 Overview of beyond-accuracy metrics [18].

Aspect	Metric	Description
Diversity	ItemCV	Item Coverage (ItemCV) measures how many items ever appear in the top- n recommendations of users.
	Gini	Gini Index. A measure of statistical dispersion intended to represent the inequality of a distribution. The Gini Index ranges between 0 and 1, where a higher value indicates greater concentration of recommendations, e.g., on popular items [163]. For interpretation convenience and to associate higher values with less concentrated recommendations, we report $1 - \text{Gini}$.
Novelty	EFD	Expected Free Discovery (EFD). A novelty measure based on the inverse collection frequency, expressing the algorithm's ability to recommend relevant long-tail items [355].
	EPC	Expected Popularity Complement (EPC). This metric measures the number of unseen items now seen as a way to quantify novelty in recommendations [355].
Popularity Bias	APLT	Average Popularity of Long-Tail Items (APLT). Measures the average popularity of long-tail items in the top- n recommendations [4].
	ARP	Average Rating-based Popularity (ARP). Computes the popularity of items in a recommendation list based on the number of interactions each item has in the training data [163].

items ranked higher. A higher nDCG value indicates better recommendation lists.

- Considering *beyond-accuracy metrics*, we measured a broader range of metrics related to diversity, novelty, and popularity bias in the recommendation lists. The details of the specific metrics considered are provided in Table 6.3.

6.3 Experiments

This section discusses the experimental setting and the results obtained from evaluating ChatGPT as a Recommender System (RS). Each experiment is designed to address the following research questions:

- RQ1.** How do the ChatGPT models perform in Top-N Recommendation tasks compared to traditional recommender systems? (Discussed in Section 6.3.3)
- RQ2.** Can ChatGPT effectively re-rank and improve recommendations by leveraging user history? (Explored in Section 6.3.4)
- RQ3.** How does ChatGPT perform in cold-start scenarios compared to the Top-N recommender systems? (Investigated in Section 6.3.5)

6.3.1 Experimental Setting

The experiments compare the performance of ChatGPT-3.5 and ChatGPT-4 models against the recommender systems presented in Section 6.3.2. To ensure a fair comparison, we perform a comprehensive Bayesian hyper-parameter optimization to find the optimal configuration for each baseline based on each training data. Furthermore, for complete reproducibility, we utilize the Elliot recommendation framework [17], making code, data, and instructions to replicate the experiments publicly available⁸.

To evaluate the models, we employ the all unrated items protocol [333], where, for each user, the set of recommendable items comprises all items except the ones the user has previously interacted with. The datasets are split into training and test sets, with 80% of the user-item interactions used for training and the remaining 20% held out for testing.

While recommendation baselines can be implemented using open-source framework like Elliot, generating recommendations from the ChatGPT models requires utilizing the OpenAI API. Specifically, for each user in each dataset (as described in Section 6.2.1), we craft a prompt to obtain a list of recommended items. We sent approximately 11,500 API requests for each language model, with ChatGPT-3.5 taking over six days to complete, while ChatGPT-4 took roughly eleven days. For the sake of completeness, since the ChatGPT models are closed-source and proprietary, also the overall costs should be reported. Specifically, the combined cost across both models amounts to approximately \$400.

The generated recommended lists (as reported in Section 6.2.3) are raw text outputs that require robust post-processing steps, and the computational time required for this post-processing step varies significantly depending on the size of the dataset. For the relatively small Facebook Books dataset, post-processing takes about 77

8. <https://github.com/sisinflab/Recommender-ChatGPT>

hours and 46 minutes, equivalent to three days. The larger MovieLens dataset requires about 100 hours and 50 minutes of processing time, or four days. The most substantial processing demands are for the large and heterogeneous HetRec dataset, which takes 145 hours and 20 minutes, roughly six days.

Finally, we evaluate the performance of the proposed models and the baseline models on the test set. We report the results using various evaluation metrics, considering each user's top 10 recommended items (i.e., cutoff at 10), commonly used as a benchmark for evaluating the effectiveness of RSs [147, 241, 286, 301]. For each experimental setting, we provide a detailed explanation and analysis of the results. The exploration is conducted separately for each domain, and a comprehensive conclusion is drawn to address each *Research Question (RQ)*.

6.3.2 Baseline Models

This section presents all the models used in the evaluation to position ChatGPT among the current state-of-the-art recommenders. To ensure the reproducibility of our experiments, we employ the Elliot framework [17] as a backbone for testing the RSs. We partition the models into Collaborative Filtering, Content-Based Filtering, and Non-personalized:

Collaborative Filtering

- (i) EASE^R, presented at WWW'19 [334], is a linear model without hidden layers, which combines the strengths of matrix factorization and neighborhood-based collaborative filtering approaches. It extends the classic matrix factorization approach by incorporating item-item and user-user similarity information, which helps alleviate the data sparsity problem and improve recommendation accuracy.
- (ii) RP³ β , introduced in TOIS'16 [280], is a graph-based recommendation algorithm that employs random walks and a reranking strategy based on the P^3 paths [81]. It demonstrates improvement in recommendation accuracy and diversity compared to traditional vertex ranking algorithms, such as PageRank and $P^3\alpha$.
- (iii) ItemKNN (Item-based K-Nearest Neighbors) [316] is a memory-based collaborative filtering algorithm that makes predictions based on item similarities. It finds the k-most similar items (neighbors) to a target item using similarity measures such as cosine vector similarity [29] or Pearson correlation coefficient.

cient [149]. The preference scores for the target item are then estimated as a weighted sum of the preference scores of the neighboring items, where the similarities between the items determine the weights.

- (iv) UserKNN (User-based K-Nearest Neighbors) [46] is a memory-based collaborative filtering algorithm that makes predictions based on user similarities, akin to how ItemKNN operates on item similarities. It finds the k-most similar users (neighbors) to a target user and estimates the preference score for a target user-item pair as a weighted sum of the neighboring users' preferences for the same item, with weights determined by user similarities.
- (v) LightGCN, presented at SIGIR'20 [146], is a graph convolutional network (GCN) based recommender system that simplifies the design of GCNs for recommendation tasks. It employs a light graph convolution operation and a simple layer combination strategy, significantly reducing the model complexity compared to previous graph-based approaches. LightGCN demonstrates several strengths: simplicity, efficient training, better generalization ability, and effectiveness.
- (vi) MF2020 [301], introduced at RecSys'20, is a state-of-the-art advancement in Matrix Factorization. This collaborative filtering technique approximates the user-item interaction matrix as the product of low-rank user and item factor matrices. While MF methods originated from Singular Value Decomposition (SVD) in the late 1990s [39], MF2020 introduces novel enhancements leveraging the latest research developments.
- (vii) NeuMF [147] (or more broadly NCF), proposed at WWW'17, is an influential neural network-based recommendation model that combines matrix factorization and deep learning. It uses a multi-layer perceptron to model non-linear user-item interactions, combining the output with the matrix factorization component. NeuMF has been widely adopted as a strong neural baseline due to its ability to capture complex patterns in user-item interactions.

Content-Based Filtering

- (viii) VSM (Vector Space Model fed with Knowledge Graph information) [264] is a content-based recommender system that represents items as vectors in a high-dimensional space, where each dimension corresponds to a feature derived from a Knowledge Graph. It computes the relevance score between a user and an item as the cosine similarity between the user's profile vector

and the item's feature vector. Recommendations are generated by ranking items based on their relevance scores.

- (ix) **AttributeItemKNN** [119] is a content-based recommender representing items as TF-IDF weighted attribute vectors. It calculates item similarities using vector metrics like cosine similarity and generates recommendations based on the k-nearest neighbour items to a user's preference profile or target item.
- (x) **AttributeUserKNN** [119] is similar to **AttributeItemKNN**, but instead of items, it represents the users as a vector of weights.

Non-personalized

- (xi) **Random Models**. A non-personalized algorithm that randomly recommends items according to a uniform distribution. It is used to provide a reference point for beyond-accuracy measures.
- (xii) **MostPop Model**. A Non-personalized algorithm that recommends the most popular items to each user, where the number of observed interactions in the training data defines popularity. Typically, it shows good performance on statistical biases dataset [34]; for this reason, it is an important baseline to discover popularity biased dataset [84].

6.3.3 Top-N Recommendation Performance

ChatGPT, as emphasized in Section 6.1, has gained significant attention due to its capability to perform a wide range of tasks, including generating recommendations. This section aims to evaluate the performance of the ChatGPT model on classical Top-N Recommendation tasks and compare it with traditional recommendation systems, thereby addressing the research question:

RQ1. How do the ChatGPT models perform in Top-N Recommendation tasks compared to traditional recommender systems?

To evaluate ChatGPT as a Recommender System, we follow the standard evaluation practice [133], selecting the best model based on accuracy metrics (e.g., nDCG, Recall, or Precision). These metrics measure the quality of the recommended items in terms of their relevance and ranking. Additionally, we have included beyond-accuracy metrics in our study to gain deeper insights into the performance and characteristics of ChatGPT as an RS [275]. These beyond-accuracy metrics include Diversity, Novelty, and Popularity Bias of the recommendations. Refer to the expla-

Table 6.4 Top-N evaluation of ChatGPT-3.5 and ChatGPT-4 on Facebook Books across Accuracy, Diversity, Novelty, and Popularity Bias. Arrows indicate preferred directions. Best values are in **bold**, second-best are underlined. Results are ordered by nDCG. Baselines are statistically significant ($p < 0.05$) except values marked with * (ChatGPT-3.5) and † (ChatGPT-4). Best-CF and Best-CBF denote the strongest collaborative and content-based baselines by nDCG.

Facebook Books									
Model	nDCG ↑	Recall ↑	Precision ↑	Gini ↑	ItemCV ↑	EPC ↑	EFD ↑	APLT ↑	ARP ↓
Random	0.0019	0.0034	0.0008	0.7753	2230	0.0007	0.0078	0.6874	5.7186
MostPop	0.0091	0.0137	0.0033	0.0045	17	0.0031	0.0228	0.0000	138.3632
LightGCN	0.0105	0.0171	0.0038	0.0053	112	0.0035	0.0269	0.0132	134.0763
NeuMF	0.0167	0.0243	0.0057	0.3336	1563	0.0065	0.0661	0.2444†	16.0072
EASE ^R	0.0188	0.0313	0.0071	0.0111	228	0.0066	0.0547	0.0032	125.2026
ItemKNN	0.0288	0.0408	0.0086	0.5293	2141	0.0104	0.1099	0.5974	24.9652
MF2020	0.0317	0.0592	0.0133	0.0044	15	0.0116	0.0953	0.0000	114.0167
UserKNN	0.0320	0.0468	0.0098	0.1564	1372	0.0115	0.1065	0.0852	55.2988
RP ³ β ^{BestCF}	0.0379	0.0568	0.0120	0.3063	<u>1888</u>	0.0138	0.1357	0.3308	44.1225*
AttributeUserKNN	0.0402	0.0593	0.0133	0.0918	945	0.0152	0.1414	0.0466	64.2887
VSM	0.0458	0.0785	0.0172	0.2478	1389	0.0173	0.1913	0.5761	<u>7.3705</u>
AttributeItemKNN ^{BestCBF}	0.0479	0.0705	0.0155	0.2824	1510	0.0182	0.2019	<u>0.5879</u>	7.1801
ChatGPT-3.5	0.0668	0.0936	0.0205	0.0713	853	<u>0.0250</u>	0.2480	0.1870	46.3236
ChatGPT-4	0.0932	0.1283	0.0283	0.1050	1004	0.0353	0.3486	0.2424	40.0319

nations provided in Section 6.2.4 for an interpretation of the metrics presented in the following.

Furthermore, a statistical significance test was conducted using the paired t-test [336]. In the following tables, most values are statistically significant ($p < 0.05$), except for those marked with * for ChatGPT-3.5 and † for ChatGPT-4. We will use the same notation (* and †) to facilitate the discussion to highlight when the metrics are not statistically significant.

Books Domain. Table 6.4 summarizes the performance of ChatGPT-3.5, ChatGPT-4 models, and all comparison baselines on the Facebook Books dataset, and we have the following observations:

Accuracy. ChatGPT-4 achieves the best performance across all accuracy metrics (i.e., nDCG, Recall, and Precision), with ChatGPT-3.5 being the runner-up. *Quantitatively*, compared to the strongest baseline (AttributeItemKNN), ChatGPT-4 shows improvements of +94.57% in nDCG, +81.9% in Recall, and +82.6% in Precision. Meanwhile, ChatGPT-3.5 shows improvements of +39.5% in nDCG, +32.8% in Recall, and +32.3% in Precision. Furthermore, ChatGPT-4 over ChatGPT-3.5 demonstrates an improvement of +39.5% in nDCG, +37.1% in Recall, and +38% in Precision. These re-

sults highlight ChatGPT’s capabilities in recommending relevant items and creating personalized lists tailored to each user’s preferences, with ChatGPT-4 outperforming ChatGPT-3.5.

Diversity. ChatGPT-4 exhibits a deterioration of -80.2% in Gini and -53.1% in ItemCV compared to the strongest baseline, ItemKNN. However, when compared to MostPop (which has an ItemCV of 15 items versus 1004 for ChatGPT-4, and 853 for ChatGPT-3.5), both ChatGPT-4 and ChatGPT-3.5 demonstrate the ability to suggest a diverse range of items beyond just the most popular ones, albeit focusing on a subset of the catalog. This suggests that while ChatGPT may not achieve the same level of diversity as the baseline, it still offers a more varied set of recommendations than simply suggesting the most popular items.

Novelty. Measured by EPC and EFD, ChatGPT-4 emerges as the top-performing model, followed by ChatGPT-3.5. Compared to the best baseline, AttributeItemKNN, ChatGPT-4 demonstrates substantial improvements of +93.96% in EPC and +72.66% in EFD. Moreover, it outperforms ChatGPT-3.5 by +41.2% in EPC and +40.6% in EFD. These results highlight the ability of the ChatGPT models, particularly ChatGPT-4, to recommend novel and diverse items within a single list, deviating from conventional recommendations and catering to users’ unique preferences.

Popularity Bias. Measured by APLT and ARP, both ChatGPT-4 and ChatGPT-3.5 exhibit a deterioration when compared to the strongest baselines. Specifically, ChatGPT-4 shows a -59.4% decrease in APLT compared to ItemKNN, and a -457.5% decrease in ARP compared to AttributeItemKNN. A similar degradation is observed for ChatGPT-3.5. This decline in performance for APLT and ARP is to be expected, as these metrics are correlated with ItemCV; models recommending fewer items tend to have a higher popularity bias, recommending the same popular items more frequently. Consequently, while ChatGPT excels in other areas, its tendency to focus on a subset of the catalog leads to a trade-off in terms of popularity bias.

Music Domain. Table 6.5 summarizes the performance of ChatGPT-3.5, ChatGPT-4 models, and all comparison baselines on the Last.FM dataset, with the following observations:

Accuracy. ChatGPT-4 and ChatGPT-3.5 achieve comparable performance, with ChatGPT-4 having a +15.7% advantage over ChatGPT-3.5 in nDCG. However, compared to the strongest baseline ($RP^3\beta$), ChatGPT-4 shows a deterioration of -10% in nDCG, -12.2% in Recall, and -14.1% in Precision.

Table 6.5 Top-N evaluation of ChatGPT-3.5 and ChatGPT-4 on Last.FM across Accuracy, Diversity, Novelty, and Popularity Bias. Arrows indicate preferred directions. Best values are in **bold**, second-best are underlined. Results are ordered by nDCG. Baselines are statistically significant ($p < 0.05$) except values marked with * (ChatGPT-3.5) and † (ChatGPT-4). Best-CF and Best-CBF denote the strongest collaborative and content-based baselines by nDCG.

Last.FM									
Model	nDCG ↑	Recall ↑	Precision ↑	Gini ↑	ItemCV ↑	EPC ↑	EFD ↑	APLT ↑	ARP ↓
Random	0.0044	0.0068	0.0052	0.8398	1507	0.0045	0.0478	0.5678	31.6844
NeuMF	0.1005	0.1133	0.0860	0.5049	1492	0.0804	0.7848	0.2418	77.5480
MostPop	0.1009	0.0895	0.0740	0.0081	27	0.0662	0.5907	0.0000	348.3308
LightGCN	0.1408	0.1329	0.1060	0.1114	635	0.1013	0.9372	0.2063	135.0381
AttributeItemKNN	0.2233	0.2013*	0.1481*	<u>0.3854</u>	<u>1411</u>	0.1584	1.5710	0.3043	87.8647
EASE ^R	0.2278	0.1949*	0.1509	0.0331	283	0.1517	1.3761	0.0088	247.6099
VSM	0.2451*	0.2021*	0.1511	0.0826	653	0.1593	1.4845	0.0585	177.9949
AttributeUserKNN ^{BestCBF}	0.2795†	0.2364	0.1818†	0.1724	923	0.1947†	1.8297†	0.0895	134.5766
UserKNN	0.2983	0.2538	0.1912	0.1491	846	0.2030	1.9060†	0.0550	152.7412
ItemKNN	0.3013	<u>0.2595</u>	0.1925	0.1634	962	0.2080	1.9854	0.1146	152.4739
MF2020	0.3097	0.2576	0.1986	0.0908	460	0.2116	1.9571	0.0051	181.8922
RP ³ β ^{BestCF}	0.3147	0.2634	<u>0.1957</u>	0.1441	831	<u>0.2110</u>	1.9970	0.0678	153.0884
ChatGPT-3.5	0.2448	0.1964	0.1408	0.1927	952	0.1680	1.6436	0.1391	99.3311
ChatGPT-4	0.2832	0.2313	0.1680	0.2023	944	0.1918	1.8663	0.1267	102.1045

Diversity. Although ChatGPT-4 exhibits a deterioration of -59.9% in Gini and -36.7% in ItemCV compared to the strongest baseline (NeuFM), it improves by +40.4% in Gini and +13.6% in ItemCV over RP³ β , the best model selected by nDCG. Compared to MostPop (ItemCV of 27) versus ChatGPT-4 (ItemCV of 944) and ChatGPT-3.5 (ItemCV of 952), both ChatGPT models demonstrate the ability to suggest a diverse range beyond just the most popular items, albeit focusing on a subset of the catalog.

Novelty ChatGPT-4 shows a deterioration of -9.4% in EPC over MF2020 and -6.5% in EFD over RP³ β , with ChatGPT-3.5 exhibiting similar behavior.

Popularity Bias. ChatGPT-3.5 shows an +9.78% improvement in APLT and +2.7% in ARP over ChatGPT-4. However, both models exhibit a deterioration of around -54.3% in APLT compared to the strongest baseline (AttributeItemKNN) and -28.1% in ARP compared to the strongest baseline (NeuMF).

Movie Domain. The comparative performance of ChatGPT-3.5, ChatGPT-4, and all baseline models on the MovieLens dataset is summarized in Table 6.6, with the following observations:

Accuracy ChatGPT-4 and ChatGPT-3.5 achieve comparable performance, with ChatGPT-4 having a +23% advantage over ChatGPT-3.5 in nDCG. However, com-

Table 6.6 Top-N evaluation of ChatGPT-3.5 and ChatGPT-4 on MovieLens across Accuracy, Diversity, Novelty, and Popularity Bias. Arrows indicate preferred directions. Best values are in **bold**, second-best are underlined. Results are ordered by nDCG. Baselines are statistically significant ($p < 0.05$) except values marked with * (ChatGPT-3.5) and † (ChatGPT-4). Best-CF and Best-CBF denote the strongest collaborative and content-based baselines by nDCG.

MovieLens									
Model	nDCG †	Recall †	Precision †	Gini †	ItemCV †	EPC †	EFD †	APLT †	ARP †
Random	0.0087	0.0062	0.0129	0.6917	1776	0.0108	0.1230	0.5482	22.2227
VSM	0.0174	0.0099	0.0205	0.0529	409	0.0209	0.2305	<u>0.3732</u>	<u>29.2857</u>
AttributeItemKNN	0.0326	0.0220	0.0389	0.3962	1395	0.0375	0.4285	0.5510	23.6326
LightGCN	0.0411	0.0349	0.0500	<u>0.3105</u>	1136	0.0421	0.4637	0.3040	43.4723
NeuMF	0.1235	0.0999†	0.1324	0.2761	<u>1172</u>	0.1171*	1.2757*	0.0970	70.5342
MostPop	0.1488*	0.0841*	0.1424†	0.0083	40	0.1097	1.2750*	0.0000	182.4909
MF2020	0.2013	0.1298	0.1985	0.0173	94	0.1576†	1.7712	0.0000	162.5163
EASE ^R	0.2076	0.1229	0.1872	0.0118	67	0.1522†	1.7352†	0.0000	173.2040
AttributeUserKNN ^{BestCBF}	0.2152	0.1317	0.2045	0.0590	438	0.1743	1.9356	0.0117	127.1064
ItemKNN	0.2709	<u>0.1819</u>	<u>0.2626</u>	0.1036	666	<u>0.2348</u>	<u>2.5547</u>	0.0470	103.0248
UserKNN	0.2814	0.1769	0.2601	0.0589	428	0.2263	2.4958	0.0057	125.7174
RP ³ β ^{BestCF}	0.2827	0.1898	0.2708	0.1230	744	0.2421	2.6613	0.0643*†	100.4106
ChatGPT-3.5	0.1475	0.0807	0.1120	0.0851	591	0.1260	1.3981	0.0733†	90.7590
ChatGPT-4	0.1815	0.1109	0.1551	0.0853	553	0.1453	1.6010	0.0775*	95.7042

pared to the strongest baseline (RP³β), ChatGPT-4 shows a deterioration of -35.8% in nDCG, -41.6% in Recall, and -42.7% in Precision.

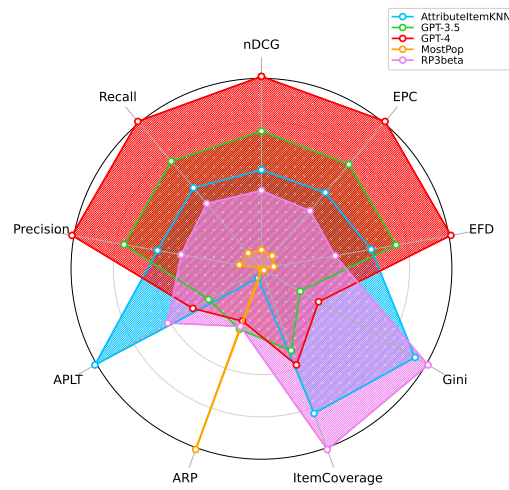
Diversity. ChatGPT-4 exhibits a deterioration of -78.5% in Gini and -60.3% in ItemCV compared to the strongest baseline (AttributeItemKNN). In contrast to MostPop (ItemCV of 40), both ChatGPT-4 (ItemCV of 553) and ChatGPT-3.5 (ItemCV of 591) demonstrate the ability to suggest a diverse range beyond just the most popular items, although focusing on a subset of the catalog.

Novelty. ChatGPT-4 shows a deterioration of -40% in EPC and -39.8% in EFD compared to RP³β, with ChatGPT-3.5 exhibiting similar behavior.

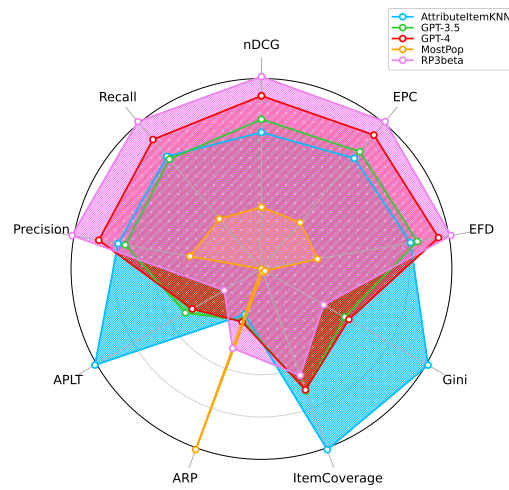
Popularity Bias. ChatGPT-4 shows a deterioration of -85.93% in APLT(*) and -305% in ARP compared to AttributeItemKNN, with ChatGPT-3.5 exhibiting similar behavior, and APLT not being statistically significant for both ChatGPT models.

RQ1. How does the ChatGPT model perform in classical recommendation tasks compared to traditional recommendation systems?

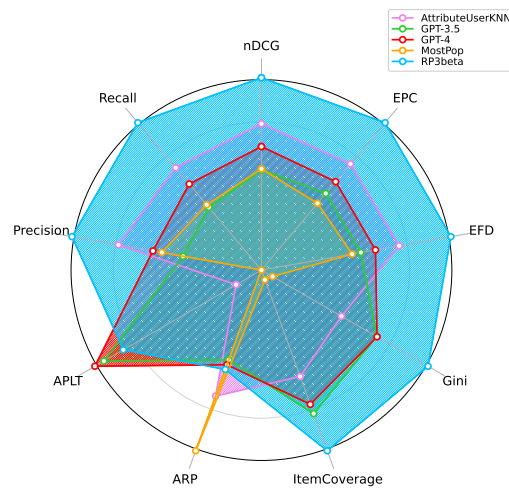
The performance of the ChatGPT models in Top-N Recommendation tasks exhibits substantial variation across different domains when compared to traditional recommendation systems. In the domain of books, our experimental results demonstrate that the ChatGPT models, particularly the latest ChatGPT-4, outperform traditional collaborative filtering and content-based filtering techniques, as well as



(a) Facebook Books.



(b) Last.FM.



(c) MovieLens.

Figure 6.2 Kiviati diagrams comparing ChatGPT models with Best-CF, Best-CBF, and MostPop across datasets. Higher values indicate better performance.

popular recommendation algorithms, in terms of accuracy metrics like Precision, Recall, and nDCG. This highlights their capability to recommend highly relevant and personalized items tailored to individual user preferences. Notably, while exhibiting a promising trade-off between diversity, novelty, and popularity bias (refer to Figure 6.2a) by offering diverse and novel recommendations within a single list, the ChatGPT models tend to recommend a reasonable subset of items, leading to a higher concentration on some items but relevant to the users compared to some traditional methods.

In the music domain, the performance of the ChatGPT models is comparable to or slightly inferior to the strongest baselines across specific metrics (refer to Figure 6.2b). However, they demonstrate a promising balance in terms of diversity, novelty, and popularity bias, underscoring their potential in personalized recommendation tasks.

Conversely, in the domain of movies, the ChatGPT models demonstrate sub-optimal performance, exhibiting inferior results compared to traditional methods like $RP^3\beta$ and AttributeUserkNN algorithms (refer to Figure 6.2c). ChatGPT-4 and ChatGPT-3.5 exhibit significantly lower performance in both accuracy and beyond-accuracy metrics, with the exception of achieving a higher APLT score than the best-performing baseline ($RP^3\beta$). This higher APLT score suggests that, despite their inferior accuracy, the ChatGPT models possess the ability to recommend items from the long tail, i.e., less popular items. While their overall accuracy may be lower, the models demonstrate a capability to surface diverse and niche recommendations, potentially catering to users with more niche tastes or preferences.

In summary, the performance of the ChatGPT models in Top-N Recommendation tasks varies substantially across domains, with promising results in the books domain, comparable performance in the music domain, and suboptimal performance in the movie domain compared to traditional recommendation systems. Our findings underscore the need for further advancements in the ChatGPT models to enhance their recommendation capabilities, particularly in domains where they currently underperform relative to traditional methods.

6.3.4 Reranking Evaluation

Section 6.3.3 showed the ChatGPT's capability of providing recommendations. This section shifts the focus on investigating its potential in a related task, the re-ranking

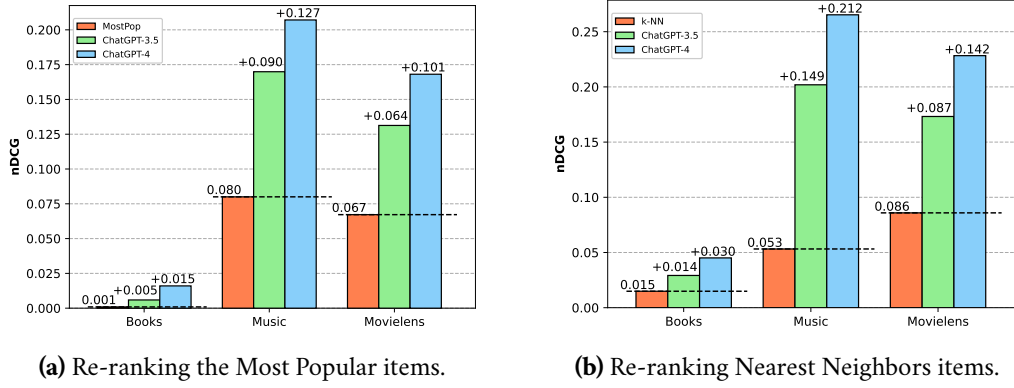


Figure 6.3 Grouped bar charts showing baseline nDCG and nDCG improvements of ChatGPT models across three domains. Higher improvements indicate better re-ranking and personalization performance.

of recommendations, i.e., the ability of a recommender to enhance the quality of recommendations by adjusting the order in which items are presented to the user [370].

We defined two experimental conditions to investigate ChatGPT’s re-ranking capability: (a) Ask ChatGPT to re-rank the most popular items in the datasets based on the user’s history, and (b) Ask ChatGPT to re-rank a pre-filtered list of suitable items for each user using a k-Nearest Neighbors (k-NN) algorithm.

Re-ranking Prompt

Given a user, act like a Recommender System. You know that user likes the following items ordered by liking: {history of the user}. Re-rank this list of items into a top-50 recommendations: {list of items to re-rank}

Since the task diverged from the Top-N Recommendations, we needed to modify the prompting strategy to maintain consistency with the Role Play Prompting approach. Specifically, for condition (a), i.e., re-ranking the most popular items, we used the prompt shown in the box above, ordering the items from the most popular to the least popular. For condition (b), i.e., re-ranking the nearest neighbor recommendations, we used the same prompt, but the order of the items was determined by the k-NN algorithm. Building on these experimental conditions, this section aims to address the following research question:

RQ2. Can ChatGPT effectively re-rank and improve recommendations by leveraging user history?

Figures 6.3a and 6.3b summarize the performance of ChatGPT-3.5 and ChatGPT-

4 models across three domains: Books, Music, and Movies, compared to the Most Popular and the k-NN methods. The performance is evaluated using the nDCG, a measure of ranking quality that considers both the relevance and the position of the items in the ranked list. Higher nDCG values indicate better performance in ranking relevant items higher. From the bar graphs, we can make the following observations. Across all three domains, ChatGPT-4 shows a higher nDCG than ChatGPT-3.5 and the baseline, irrespective of the experimental conditions. This suggests that ChatGPT-4 is more effective at ranking items in a way that is relevant to users' preferences. The second observation depends on the type of setting. In the case of Most Popular items, Figure 6.3a, the MostPop baseline nDCG is lower than expected due to the lack of complete user influence. However, with the initial filtering using k-NN, the nDCG is higher, showing a first step towards personalization. Nonetheless, the ChatGPT models, specifically ChatGPT-4, are able to further improve the recommendation performance by better understanding user preferences.

In summary, to answer RQ2, we examine the nDCG value and observe improvements in the recommendation performance after the re-ranking of ChatGPT, leading to a positive answer. However, within the scope of this study, we cannot provide further insights into the reasons behind this improvement. We hypothesize that the improvement stems from the vast knowledge base of GPT and its ability to understand user preferences and comprehend the relevance of items in generating the re-ranking. To explore this further, future research will investigate the explainability of the deep learning model behind ChatGPT and its efficacy in re-ranking recommendations as much as possible. However, a comprehensive explainability analysis is beyond the scope of the current study.

6.3.5 Cold-Start Scenario

Recommender systems are pivotal in various applications, from e-commerce to content platforms, guiding users through vast amounts of content to find items of interest. However, recommendation algorithms often face cold-start problems. This section will explore ChatGPT's ability to handle this issue. Specifically, we can distinguish three types of cold-start problems: new community, new item, and new user [40].

The new community problem [195, 318] refers to the difficulty in obtaining a sufficient amount of data (ratings) when starting up an RS to enable reliable recommen-

dations. The new item problem [278, 279] arises because newly added items in the RS typically do not have any initial ratings, making them unlikely to be recommended. The new user problem [45, 296] arises when the system has to make recommendations for users who have little or no history of interactions with the system. The new user cold-start problem is the most crucial in a recommender system that is already in operation, and it is the focus of this study.

There are different common strategies for tackling the new user problem. Among them, we can ask for additional information from the user to make recommendations based on the available data [40], we can select the most similar item to the one the user is currently consuming through a content-based RS, or we can recommend the most popular items [353].

We conduct a series of controlled experiments to evaluate the efficacy of ChatGPT in addressing the new user cold-start scenarios and address

RQ3. How does ChatGPT perform in cold-start scenarios compared to the Best Top-N recommender systems?

However, defining a rigorous, universally applicable definition of cold-start is challenging, as it depends on the specific case study and domain. Consequently, following the work of Bobadilla et al. [40] and Fu et al. [116], we defined a user cold-start as a set of users having at most ten items in their interaction history.

In our study, based on the datasets, we create a subset of the datasets with all the users having less than ten items [116], and use them to interact with ChatGPT through the Top-N Recommendations prompt (refer to 6.2.1). We will measure the accuracy and beyond-accuracy metrics only for the subset of users we generated, and compare them with the MostPop approach, the BestCF, and the BestCBF from the Top-N Recommendations. Additionally, although we attempted to apply some statistical tests, the generated subsets do not meet the requirements to apply the t-test or other parametric approaches due to their small sample sizes. Specifically, the Last.FM subset with 38 users and the MovieLens subset with 13 users are not sufficiently large for these tests, as parametric tests like the t-test typically require larger sample sizes to ensure valid results [82, 247, 270, 342]. We were, however, able to perform the t-test only on the Facebook Books subset, which had a sufficient sample size of 280 users.

However, the absence of a statistically significant finding does not necessarily imply a lack of value but implies the inability to reject the null hypothesis. Instead,

Table 6.7 Comparison of **user cold-start** interactions (up to ten interactions per user) for ChatGPT-3.5 and ChatGPT-4. Arrows indicate preferred directions. Best values are in **bold**, second-best are underlined. CF and CBF denote collaborative and content-based recommenders. Facebook Books baselines are statistically significant ($p < 0.05$) except values marked with * (ChatGPT-3.5) and † (ChatGPT-4). Best-CF and Best-CBF correspond to the strongest CF and CBF by nDCG.

	Model	nDCG †	Recall †	Precision †	Gini †	ItemCV †	EPC †	EFD †	APLT †	ARP †
FacebookBooks	Random	0.0011	0.0018	0.0004	0.4315	1560	0.0004	0.0034†	0.6793	5.8468
	MostPop	0.0115*	0.0143	0.0029	0.0037	14	0.0031	0.0236	0.0000	139.1043
	AttributeItemKNN ^{CBF}	0.0335*	0.0500	0.0100*	<u>0.1873</u>	<u>957</u>	0.0119*	0.1283*	0.5661	7.2114
	RP ³ β ^{CF}	0.0346*	0.0500	0.0096	0.1932	1044	0.0115	0.1161*	0.3332	41.7332*
	ChatGPT-3.5	0.0487†	0.0779†	0.0152†	0.0538	445	0.0168†	0.1714†	0.2004	45.0357
	ChatGPT-4	0.0538*	0.0873*	0.0171*	0.0846	597	0.0186*	0.1877*	0.2458	<u>37.9698</u>
LAST.FM	Random	0.0000	0.0000	0.0000	0.2091	345	0.0000	0.0000	0.5184	34.3684
	MostPop	0.0529	0.0877	0.0211	0.0063	15	0.0167	0.1493	0.0000	363.5789
	AttributeUserKNN ^{CBF}	0.1724	0.2149	0.0605	0.1487	<u>282</u>	0.0705	0.8092	0.5000	40.5553
	RP ³ β ^{CF}	0.2389	0.3333	0.0895	0.1237	254	0.1043	1.1120	0.2605	108.0526
	ChatGPT-3.5	0.2921	0.3423	0.0946	0.1340	257	0.1289	1.3989	0.3081	75.6000
	ChatGPT-4	<u>0.2791</u>	<u>0.3465</u>	0.0947	0.1513	283	<u>0.1204</u>	<u>1.3141</u>	<u>0.3526</u>	<u>69.8632</u>
MovieLens	Random	0.0000	0.0000	0.0000	0.0683	129	0.0000	0.0000	0.5231	22.8000
	MostPop	0.0254	0.0385	0.0077	0.0049	12	0.0048	0.0566	0.0000	197.6769
	AttributeUserKNN ^{CBF}	0.0368	0.0513	0.0154	<u>0.0429</u>	<u>100</u>	0.0101	0.1080	<u>0.1231</u>	<u>90.7769</u>
	RP ³ β ^{CF}	0.0791	<u>0.1026</u>	<u>0.0231</u>	0.0566	117	0.0269	0.2878	0.1846	65.1231
	ChatGPT-3.5	<u>0.1117</u>	0.0897	0.0231	0.0333	87	0.0313	0.3384	0.0769	96.5615
	ChatGPT-4	0.1405	0.1538	0.0385	0.0349	89	0.0455	0.4988	0.0462	102.9385

it can provide a different perspective that can be equally enlightening and contribute to our understanding of the phenomenon under investigation [78, 98, 231, 359].

Table 6.7 summarizes the performance of the ChatGPT-3.5 and ChatGPT-4 models, along with comparison baselines, on the Facebook Books, Last.FM and MovieLens datasets for user cold-start recommendation tasks. The key observations are in the following.

Books Domain.

- **Accuracy.** ChatGPT-4 outperformed all baselines and ChatGPT-3.5, demonstrating substantial improvements in nDCG (+55.5% over RP³β), Recall (+74.6%), and Precision (+78.1%). ChatGPT-3.5 also showed notable gains over UserKNN.
- **Diversity.** Both ChatGPT models suggested a more diverse range of items than simply recommending the most popular ones (MostPop). However, they exhibited lower diversity scores compared to RP³β, with ChatGPT-4 showing a deterioration of -56.2% in Gini and -42.8% in ItemCV.
- **Novelty.** ChatGPT-4 emerged as the top performer, followed by ChatGPT-3.5, exhibiting significant improvements in EPC (+61.7%) and EFD (+61.6%) over

$RP^3\beta$, highlighting their ability to recommend novel and diverse items.

- **Popularity Bias.** The ChatGPT models showed a deterioration in APLT and ARP compared to AttributeItemKNN, suggesting a potential bias toward recommending less popular items.

Music Domain.

- **Accuracy.** ChatGPT-3.5 slightly outperformed ChatGPT-4 in nDCG (+4.7%), and both models showed improvements over the strongest baseline, $RP^3\beta$ (ChatGPT-3.5: +22.3% in nDCG, +2.7% in Recall, +5.7% in Precision).
- **Diversity.** ChatGPT-4 exhibited higher diversity scores (Gini: +22.3%, ItemCV: +11.4%) compared to $RP^3\beta$, with ChatGPT-3.5 also suggesting a more diverse range than only recommending the most popular items.
- **Novelty.** Both ChatGPT models demonstrated improvements in EPC and EFD over $RP^3\beta$, with ChatGPT-3.5 showing slightly better performance (+23.6% in EPC, +25.8% in EFD).
- **Popularity Bias.** ChatGPT-4 led in APLT (+14.4%) and ARP (+7.6%) over ChatGPT-3.5, with both models underperforming the best baseline (AttributeUserKNN).

Movie Domain.

- **Accuracy.** ChatGPT-4 achieved the best performance across all accuracy metrics, substantially outperforming the strongest baseline, $RP^3\beta$ (nDCG: +77.6%, Recall: +49.9%, Precision: +66.7%). ChatGPT-3.5 also surpassed $RP^3\beta$ but exhibited lower Recall and Precision than ChatGPT-4.
- **Diversity.** While both ChatGPT models suggested a more diverse range than MostPop, they showed a deterioration in Gini (-38.3%) and ItemCV (-23.9%) compared to $RP^3\beta$, indicating a focus on a subset of the catalog.
- **Novelty.** ChatGPT-4 demonstrated significant improvements in EPC (+69.1%) and EFD (+73.3%) over $RP^3\beta$, with ChatGPT-3.5 exhibiting similar performance.
- **Popularity Bias.** The ChatGPT models exhibited a deterioration in APLT and ARP compared to $RP^3\beta$, with the APLT metric being statistically insignificant, suggesting a potentially unreliable difference.

In summary, to answer RQ3, the ChatGPT models can help in the condition of new user cold-start, especially ChatGPT-4, consistently outperforming baselines in

accuracy and novelty metrics across domains while exhibiting trade-offs in diversity and popularity bias.

6.4 Related Work

Integrating Large Language Models (LLMs) into recommender systems has garnered considerable attention due to their robust generative abilities and versatility across different tasks. However, this emerging trend in the field is primarily represented by preliminary preprint studies, underscoring the relatively emergent nature of this research area.

Among the state-of-the-art existing techniques, the M6-Rec [86] framework, based on the M6 LM [215], explores various tasks of an industrial recommender system. Specifically, it handles various recommender-related tasks (e.g., conversational recommendation or explanation generation) by translating user profiles into plain texts and treating the recommendation process as a language understanding or generation task. The authors verify that M6-Rec performs comparably to traditional ID-based rankers when evaluated as a zero-shot top-n recommender on three domains. On the same line, P5 [127] integrates user-item information and corresponding features within personalized prompt templates to feed the model. Although it is evaluated over the rating and the next item prediction task, P5 can make predictions in a zero-shot or few-shot manner, vastly reducing the necessity for extensive fine-tuning. However, these works identify ad-hoc solutions for the recommendation scenario inspired by LLMs architectures for their model instead of directly employing generic LLMs (i.e. ChatGPT-3.5 and 4) as a recommender system.

Wu et al. [392] make use of the learned knowledge of LMs (e.g., BERT and GPT-2) and LLMs (i.e., T5 and LLaMA) to address the cold start issue, verifying their fine-tuned version being effective once the task is translated into natural language. The proposal claims that a precise design of prompts plays a significant role in the merits of model performances, even when characterized by few parameters. By extending this idea, our framework targets a cold-start scenario for ranking a list of items rather than recommending items individually. It also enables fair comparisons with consolidated baselines.

Zhang et al. [419] fine-tune the T5 LLM to implement a sequential recommendation task over the video games and music domains (Amazon dataset ⁹), unveil-

9. <https://nijianmo.github.io/amazon/index.html>

ing superior performances against other zero-shot LLM-based strategies, including GPT-3.5. Despite the known information differences between fine-tuned and zero-shot solutions, the authors report comparable ranking abilities when models are required to rerank an existing list of suggested items. Therefore, we further investigated this achievement by designing a fair and generalizable reranking experiment where ChatGPT can empirically prove its potential against baselines within the Top-N recommendation task.

He et al. [148] presents an empirical study on the conversational recommendation task using GPT-3.5, GPT-4 and different LLAMA configurations in a zero-shot setting. The authors demonstrate that LLMs outperform all fine-tuned existing CRS models compared to their recommendation accuracy, with GPT-4 excelling over the others. Such an outcome inspired our analysis to look for insights about ChatGPT-3.5 and ChatGPT-4 abilities in addressing the Top-N recommendation task instead.

Kang et al. [179] analyze multiple LLMs in various configurations (zero-shot, few-shot and after fine-tuning) performing the rating prediction task. They emerge that improvements in accuracy performances do not justify the cost of fine-tuning the model compared to the most convenient plug-and-play usage of the zero-shot strategy. Therefore, we deepened the zero-shot ChatGPT behavior in ranking a list of items instead of computing rates since it is more suitable for a generative language model.

The repeated demonstration of GPT models' potential in addressing recommendation tasks has led us to direct our attention to conducting an in-depth analysis of ChatGPT's behaviors, or more broadly, of the GPT family, which is yet to be comprehensively studied.

Early work by Zhang et al. [428] exploited the GPT-2 model to reformulate the recommendation task into a multi-token cloze task, leveraging session-based recommendations. The work uses MovieLens-1M in a zero-shot and fine-tuning approach with GPT-2. They were the first to state that pre-trained LM-based methods do not perform randomly in the zero-shot setting. However, those configurations under-perform RNN-based recommender systems such as GRU4Rec [151]. Another example based on GPT-2 is GPT4Rec from Li et al. [205], a framework that uses a fine-tuned version of GPT-2 to generate hypothetical "search queries" from the user's history and use the queries with BM25, an information retrieval algorithm [305], for generating multiple recommendation lists for each user. Nev-

ertheless, our work explores the most recent GPT versions since they have shown superior performance in several natural language generation tasks, thus yielding a new wave of studies for the recommendation task.

Wang et al. [368] was the first to target the use of GPT-3 for recommendations, proposing a 3-step prompting strategy to guide GPT-3 in carrying out the recommendation, which includes: (i.) user preference prompt to capture the target user preferences, (ii.) a ranking prompt to rank the previously interacted movies by user's preferences, (iii.) create the recommendation prompt to guide GPT-3 to recommend ten similar movies from the candidate movie set. The proposed framework, focused on the movie domain, is evaluated on the MovieLens 100K dataset and shows performance in terms of accuracy in zero-shot settings. However, the authors consistently extracted their answers through rule-based techniques without studying the impact of hallucinations of the model. In contrast, our framework relies on a more sophisticated pipeline to mitigate the hallucination issue and grant fair comparisons.

A first example of ChatGPT4Rec is proposed by Gao et al. [124], which introduced ChatREC, a ChatGPT Augmented Recommender System that translates the recommendation task into an interactive conversation with users, leading to a list of Top-N recommended items. The authors proposed a prompt template to convert user information and user-item interactions to a query for ChatGPT. This study also predisposed the framework to handle the cold-start scenario and the cross-domain recommendation. However, the system was evaluated only on the top-5 recommendation task, on accuracy metrics (i.e., Recall, Precision, nDCG), and compared with only two baselines on MovieLens 100K datasets. Therefore, we extend such an analysis by including a broader collection of metrics (i.e., ranking, accuracy, bias and novelty metrics) and tasks (top-N recommendation, lists reranking, cold-start and list similarity) to provide in-depth analysis and reach more general conclusions.

Hou et al. [155] focused their work on the ranking task using LLMs in zero-shot settings. They expanded the analysis of ChatGPT models as a recommender by modeling a dedicated prompting template over two datasets (i.e., Movielens-1M and Games). Moreover, they found that the resulting recommendations are sensitive to the employed prompting techniques. Specifically, the item positions in the prompts affect the generated list, which the authors called the position bias issue. Nonetheless, their experiments further unveil that ChatGPT-3.5 and ChatGPT-4 reach the highest ranking performance against other LLMs when addressing the zero-shot

recommendation task. Considering these outcomes, we designed our work to give further insights into bias issues of the previously declared performant models.

Similar work to ours by Sanner et al. [314] investigated LLM-based RS abilities for the Top-N recommendation task to find the most performant prompting strategy that leads such systems to stable and relevant recommendations besides giving deepened insights about leveraging the cold-start scenario. The authors concluded that the zero-shot setting results in the most pertinent suggestions, beating content-based baseline solutions in the cold-start configuration. However, they presented a preliminary study on a dataset filled with handcrafted item linguistic information, thus not granting the generality of the results on well-established datasets for the recommendation task and different LLMs. Furthermore, their work was based only on $nDCG@10$ to draw conclusions on the behaviour of the recommendation.

Li et al. [213] focus on applying ChatGPT within the book recommendation scenario, designing BookGPT to leverage the single item and rating prediction tasks. In this proposal, the Role-Play prompting enables the LLMs to achieve the targeted recommendation goal with comparable accuracy performances against baselines. However, the study does not provide a generalizable analysis of the LLM performances over multiple domains since the authors intend to propose a valuable tool for a very dedicated task. Therefore, Dai et al. [90] further probes the ChatGPT abilities in suggesting items through rating prediction, pair-wise recommendation and reranking list strategy through the same prompting approach. The experiment led over four different domain datasets shows low performances of the model in predicting rates for the items but exhibits balanced recommending abilities in the remaining tasks. However, this study does not provide a comprehensive view of LLMs' abilities in leveraging the Top-N recommendation task on several dimensions analyses. Only with the proposal of Xu et al. [400], which based their work on studying the effect of the prompt on the final performance of LLMs as RSs, Role-Playing prompting in a zero-shot scenario is ultimately confirmed to enable LLM-based RSs in being stable and performant. Their work analyzed different LLM family models' performance by adopting, at that time, all known prompting strategies. Regardless, their examination of Movielens-1M and Amazon-books datasets is based only on accuracy performance.

Our work is thus the first to extensively investigate the task of Top-N Recommendation, moving beyond accuracy, and to examine ChatGPT's ability to generate lists

like Content-based and Collaborative filtering methods. The study also highlights the capabilities in reranking and addressing the cold-start problem.

6.5 Summary

In this work, we systematically evaluate the use of ChatGPT models as Recommender Systems through a simple yet effective role-prompting strategy. We introduce an open-source evaluation pipeline that post-processes generated recommendations, enabling a reliable and comprehensive assessment across multiple datasets and domains. Our analysis considers not only accuracy but also beyond-accuracy dimensions, providing a holistic view of recommendation quality.

Experiments conducted on Books, Music, and Movies demonstrate that ChatGPT-3.5 and ChatGPT-4 perform competitively, particularly in scenarios where traditional recommenders struggle due to data sparsity. ChatGPT-4, in particular, achieves substantial gains in Top-N recommendation tasks, markedly outperforming classical baselines in the book domain. Moreover, ChatGPT proves effective in re-ranking recommendation lists, yielding consistent improvements in nDCG, and in user cold-start scenarios, where it surpasses traditional methods in accuracy and novelty, albeit with trade-offs in diversity and popularity bias.

Beyond performance, our findings raise important ethical considerations related to data provenance and user preference acquisition in large language models. Looking ahead, we plan to further investigate if latent collaborative knowledge are embedded in ChatGPT, analyze the factors driving its strong re-ranking capabilities, and explore content-based and ethical dimensions of its recommendations. Overall, this study highlights the potential of large language models as powerful components of future recommender systems.

Supplementary Material

Prompt Examples

```
<s>[INST] Given two items, answer with YES or NO. YES if they represent  
the same item, NO if they represent different items. [/INST]  
[INST] <The Lego Movie 2: The Second Part (2019)> and <The Lego Movie  
(2014)>[/INST]  
NO. </s>  
  
<s>[INST] <Shrek Forever After (2010)> and <Shrek Forever After (a.k.a.  
Shrek: The Final Chapter) (2010)>[/INST]  
YES. </s>  
  
<s>[INST] <The Rich Man's Wife (1996)> and <Preacher's Wife, The  
(1996)>[/INST]  
NO. </s>  
  
<s>[INST] {str_1} and {str_2} [/INST]
```

Figure 6.4 Mistral-7B few-shot prompt for item identity matching.

Based on the user's preferences, here are the top 50 movie recommendations:

1. Shawshank Redemption, The (1994)
2. Schindler's List (1993)
3. Usual Suspects, The (1995)
4. Pulp Fiction (1994)
5. Forrest Gump (1994)
6. Silence of the Lambs, The (1991)
7. Braveheart (1995)
8. Apollo 13 (1995)
9. Lion King, The (1994)
10. Jurassic Park (1993)
11. Speed (1994)
12. True Lies (1994)
13. Dances with Wolves (1990)
14. Four Weddings and a Funeral (1994)
15. Babe (1995)
16. Interview with the Vampire: The Vampire Chronicles (1994)
17. Jumanji (1995)
18. Birdcage, The (1996)
19. Ed Wood (1994)
20. Waterworld (1995)
21. Net, The (1995)
22. Dave (1993)
23. Firm, The (1993)
24. Fugitive, The (1993)
25. Ghost (1990)
26. Sleepless in Seattle (1993)
27. Batman (1989)
28. Pretty Woman (1990)
29. Santa Clause, The (1994)
30. Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
31. Clueless (1995)
32. Nell (1994)
33. Outbreak (1995)
34. French Kiss (1995)
35. I.Q. (1994)
36. Fargo (1996)
37. Trainspotting (1996)
38. Independence Day (ID4) (1996)
39. Twister (1996)
40. Mission: Impossible (1996)
41. Rock, The (1996)
42. English Patient, The (1996)
43. Scream (1996)
44. Jerry Maguire (1996)
45. Ransom (1996)
46. Evita (1996)
47. Air Force One (1997)
48. L.A. Confidential (1997)
49. Titanic (1997)
50. Good Will Hunting (1997)

These recommendations are based on the user's ratings and preferences for similar movies.

Figure 6.5 ChatGPT response example.

Based on the user's preferences, here are the top 50 movie recommendations:

1. The Dark Knight (2008)
2. The Departed (2006)
3. The Prestige (2006)
4. The Bourne Ultimatum (2007)
5. The Lives of Others (Das Leben der Anderen) (2006)
6. Pan's Labyrinth (Laberinto del fauno, El) (2006)
7. There Will Be Blood (2007)
8. Atonement (2007)
9. No Country for Old Men (2007)
10. Into the Wild (2007)
11. Ratatouille (2007)
12. Juno (2007)
13. 300 (2007)
14. Zodiac (2007)
15. Eastern Promises (2007)
16. Michael Clayton (2007)
17. Gone Baby Gone (2007)
18. The Diving Bell and the Butterfly (2007)
19. The Kite Runner (2007)
20. Sweeney Todd: The Demon Barber of Fleet Street (2007)
21. The Orphanage (Orfanato, El) (2007)
22. There's Something About Mary (1998)
23. The Truman Show (1998)
24. The Big Lebowski (1998)
25. American History X (1998)
26. The Thin Red Line (1998)
27. The Prince of Egypt (1998)
28. The Mask of Zorro (1998)
29. The Waterboy (1998)
30. The Wedding Singer (1998)
31. The Parent Trap (1998)
32. The Negotiator (1998)
33. The Man in the Iron Mask (1998)
34. The Horse Whisperer (1998)
35. The Horse Whisperer (1998)
36. The Horse Whisperer (1998)
37. The Horse Whisperer (1998)
38. The Horse Whisperer (1998)
39. The Horse Whisperer (1998)
40. The Horse Whisperer (1998)
41. The Horse Whisperer (1998)
42. The Horse Whisperer (1998)
43. The Horse Whisperer (1998)
44. The Horse Whisperer (1998)
45. The Horse Whisperer (1998)
46. The Horse Whisperer (1998)
47. The Horse Whisperer (1998)
48. The Horse Whisperer (1998)
49. The Horse Whisperer (1998)
50. The Horse Whisperer (1998)

These recommendations are based on the user's liking for drama, action, and thriller movies, as well as some popular and critically acclaimed films from the late 90s and early 2000s.

Figure 6.6 ChatGPT duplication error.

Based on the user's preferences, here are the top 50 book recommendations:

1. Fruits Basket Another by Natsuki Takaya
2. Fruits Basket: The Complete Collection by Natsuki Takaya
3. Fruits Basket, Vol. 1 by Natsuki Takaya
4. Fruits Basket, Vol. 2 by Natsuki Takaya
5. Fruits Basket, Vol. 3 by Natsuki Takaya
6. Fruits Basket, Vol. 4 by Natsuki Takaya
7. Fruits Basket, Vol. 5 by Natsuki Takaya
8. Fruits Basket, Vol. 6 by Natsuki Takaya
9. Fruits Basket, Vol. 7 by Natsuki Takaya
10. Fruits Basket, Vol. 8 by Natsuki Takaya
11. Fruits Basket, Vol. 9 by Natsuki Takaya
12. Fruits Basket, Vol. 10 by Natsuki Takaya
13. Fruits Basket, Vol. 11 by Natsuki Takaya
14. Fruits Basket, Vol. 12 by Natsuki Takaya
15. Fruits Basket, Vol. 13 by Natsuki Takaya
16. Fruits Basket, Vol. 14 by Natsuki Takaya
17. Fruits Basket, Vol. 15 by Natsuki Takaya
18. Fruits Basket, Vol. 16 by Natsuki Takaya
19. Fruits Basket, Vol. 17 by Natsuki Takaya
20. Fruits Basket, Vol. 18 by Natsuki Takaya
21. Fruits Basket, Vol. 19 by Natsuki Takaya
22. Fruits Basket, Vol. 20 by Natsuki Takaya
23. Fruits Basket, Vol. 21 by Natsuki Takaya
24. Fruits Basket, Vol. 22 by Natsuki Takaya
25. Fruits Basket, Vol. 23 by Natsuki Takaya
26. Fruits Basket, Vol. 24 by Natsuki Takaya
27. Fruits Basket, Vol. 25 by Natsuki Takaya
28. Fruits Basket, Vol. 26 by Natsuki Takaya
29. Fruits Basket, Vol. 27 by Natsuki Takaya
30. Fruits Basket, Vol. 28 by Natsuki Takaya
31. Fruits Basket, Vol. 29 by Natsuki Takaya
32. Fruits Basket, Vol. 30 by Natsuki Takaya
33. Fruits Basket, Vol. 31 by Natsuki Takaya
34. Fruits Basket, Vol. 32 by Natsuki Takaya
35. Fruits Basket, Vol. 33 by Natsuki Takaya
36. Fruits Basket, Vol. 34 by Natsuki Takaya
37. Fruits Basket, Vol. 35 by Natsuki Takaya
38. Fruits Basket, Vol. 36 by Natsuki Takaya
39. Fruits Basket, Vol. 37 by Natsuki Takaya
40. Fruits Basket, Vol. 38 by Natsuki Takaya
41. Fruits Basket, Vol. 39 by Natsuki Takaya
42. Fruits Basket, Vol. 40 by Natsuki Takaya
43. Fruits Basket, Vol. 41 by Natsuki Takaya
44. Fruits Basket, Vol.

Figure 6.7 ChatGPT extreme duplication error.

Chapter 7

Content-Based or Collaborative? Insights from Inter-List Similarity Analysis of ChatGPT Recommendations

As shown in the previous chapter, ChatGPT has demonstrated remarkable versatility as a Recommender Systems (RSs). However, unlike traditional RSs, it generates recommendations through natural language, leveraging contextual cues and large-scale pre-trained knowledge. However, it remains unclear whether these recommendations implicitly encode collaborative patterns, rely on semantic item similarities, or follow a fundamentally different paradigm. In this work, we systematically analyze ChatGPT’s recommendation behavior by comparing its generated lists to collaborative and content-based filtering baselines across three domains: Books, Movies, and Music. Using established list similarity metrics, we quantify the alignment of ChatGPT’s recommendations with traditional paradigms. Additionally, we investigate the most recommended items by ChatGPT and the other recommenders, comparing the distribution of frequently recommended items across models. Our findings reveal that ChatGPT exhibits strong similarities to collaborative filtering (CF) and amplifies popular yet underrepresented items in the dataset, suggesting a broader domain knowledge encoded in the language model and the need for future research on leveraging LLMs for recommendation tasks.¹

1. This chapter is based on work published and nominated as a Candidate Best LBR in the Adjunct Proceedings of the 33rd ACM Conference on User Modeling, Adaptation and Personalization (UMAP 2025), titled “Content-Based or Collaborative? Insights from Inter-List Similarity Analysis of ChatGPT Recommendations”.

7.1 Introduction

Recommender systems (RSs) play a crucial role in personalized information retrieval, helping users navigate vast catalogs of content in domains such as entertainment, e-commerce, and digital libraries [147, 301]. Traditional RSs primarily rely on collaborative filtering (CF) and content-based filtering (CBF) to generate recommendations. However, the emergence of Large Language Models (LLMs), particularly ChatGPT, has introduced a novel paradigm in recommendation generation [92, 148].

Unlike conventional RSs that rely on explicit user interactions or structured content data, ChatGPT leverages natural language processing capabilities to infer user intent, exploit contextual cues, and incorporate a broad spectrum of general knowledge [37, 38, 375].

Recent research has explored the potential of LLMs in recommendation tasks, where these models generate personalized suggestions based on textual prompts rather than explicit user-item interaction data [86, 127, 419]. For instance, Hou et al. [155] showed that LLM-based recommendations are highly sensitive to prompt design, with ranking performance fluctuating due to position bias. Similarly, Wu et al. [392] fine-tuned language models and emphasized that well-crafted prompts significantly impact the quality of LLM-generated recommendations, even for smaller-scale models.

Despite these advancements, the underlying mechanisms governing ChatGPT's recommendations remain largely unexplored [90, 229]. Specifically, it is unclear whether these models mimic CF by recommending frequently co-occurring items or rely on a CBF strategy that exploits semantic similarities among items [401]. By examining how ChatGPT-generated recommendations compare with traditional approaches, it becomes possible to assess whether ChatGPT can effectively substitute conventional RSs or should instead be integrated into hybrid systems that combine collaborative signals, semantic reasoning, and contextual awareness. Such an investigation lays the groundwork for future advances in recommender systems, potentially enabling more flexible, insightful, and user-adaptive recommendation strategies.

To address this gap, we analyze GPT-based recommendations to determine if they align more closely with CF or CBF models using the Jaccard Index and Rank-Biased Overlap (RBO) similarity metrics. We compare recommendation lists gener-

ated by ChatGPT-3.5 and ChatGPT-4 with those produced by traditional CF and CBF models across three domains. Books (Facebook Books [241]), Movies (MovieLens [139]), and Music (Last.FM [56]). To enable cross-domain comparisons, we normalize these metrics by estimating the expected similarity between randomly generated lists through analytical computation or simulation. We then conduct a detailed analysis of the most frequently recommended items to investigate whether ChatGPT exhibits distributional patterns that differ from those of traditional recommenders.

Our study reveals that ChatGPT’s recommendations align more closely with CF than with CBF models, albeit with notable variations across domains. In the Music domain, ChatGPT exhibits the highest similarity to CF models, whereas in Movies the alignment is weaker, suggesting that recommendation behavior is influenced by domain-specific characteristics. Furthermore, our analysis of the most frequently recommended items shows that, while ChatGPT tends to favor popular items, it also recommends items that extend beyond those typically emphasized by CF alone. This behavior reflects the language model’s ability to encode broad domain patterns that go beyond explicit user-item interaction signals. Overall, our findings highlight new opportunities for integrating ChatGPT-generated recommendations with traditional approaches, enhancing domain awareness and potentially improving both the diversity and relevance of recommendation lists. To ensure full reproducibility, we release our code and supplementary materials at the following GitHub repository: <https://github.com/sisinflab/Content-or-Collaborative->.

7.2 Methodology

To analyze how ChatGPT models generate recommendations, we prompt ChatGPT to perform the Top-N recommendation task [191], train the baseline models, and assess the ranked list similarities.

Phase 1: Prompting ChatGPT Models for Recommendation. We design prompts that guide ChatGPT models in ranking items for a given user. Specifically, we instruct ChatGPT to generate Top-50 recommendations based on user history. Among the various prompting strategies [312], we explored in order:

(i) Zero-shot prompting [293], where the LLM receives only task instructions without specific examples. However, ~71% of the generated lists contained fewer than 50 items or included repeated entries, and ~6% exhibited incorrect task execution;

(ii) Few-shot prompting [49], which includes a small set of example recommendations to further guide the model. While these contextual examples reduced execution errors, $\sim 44\%$ of the generated lists contained duplicate items;

(iii) Chain-of-Thought (CoT) prompting [380], where the model is prompted to reason through recommendations step-by-step. This approach produced excessive tokens, reaching the context limit after generating ~ 26 items;

(iv) Role-Playing prompting [172], where the LLM impersonates a specific role, such as an RS, to perform the task. This strategy proved the most effective, eliminating duplicate recommendations. Following the approach of Sahoo et al. [312], we manually refined the prompt to ensure sufficient contextual grounding, resulting in the final version used in our study. Prompt examples are provided in our repository as supplementary material.

Consistent with recent studies on Role-Playing [188, 400], our initial experiments yielded good accuracy and minimal duplication in the ranked list. Given the token limitations that prevent submitting the entire item catalog to ChatGPT, we used the following prompt: *“Given a user, as a Recommender System, please provide only the names of the top 50 recommendations. You know that the user likes the following items: [history of the user]”*.

Phase 2: Ensuring Recommendation Consistency. ChatGPT models generate recommendations based solely on the user profile provided in the prompt, without restrictions to a predefined dataset. As a result, they may hallucinate [70, 169, 262], or suggest real items outside the reference dataset, leading to discrepancies in item naming and inconsistencies in evaluation. To ensure consistency and alignment between ChatGPT-generated recommendations and the dataset, we follow the previously proposed post-processing pipeline (see Figure 6.1):

- 1 Item Name Extraction – Extract recommended item names from additional text in the generated response.
- 2 Dataset Alignment – Use Gestalt pattern matching [129] to find the closest dataset match, accepting those with a similarity score above 85% (empirically determined).
- 3 String Normalization – If no match is found, we apply regular expressions to normalize text by removing accents, converting to lowercase, eliminating punctuation, and trimming whitespace.
- 4 Handling External Items – Flag unmatched items as *External Items* (originating from the LLM’s pre-trained knowledge) and exclude them from similarity assess-

Table 7.1 Dataset statistics after $k - core \geq 10$. The selected types of content for CBF follow standard practices [18].

Dataset	Interaction	Users	Items	Sparsity	Content
MovieLens	42,456	603	1,862	96.22%	genre
Last.FM	49,171	1,797	1,507	98.18%	genre
FB Books	13,117	1,398	2,234	99.58%	genre, author

ments to ensure fair comparison with traditional recommenders by promoting in-catalogue items over out-of-catalogue ones.

Since this final step could affect our evaluations, we verified that out-of-catalogue items consistently appeared beyond the top-10 positions in all recommendation lists, ensuring that rank-sensitive metrics remain unaffected and preserving the validity of our evaluation. In our configuration, ChatGPT placed these items only after rank 23, suggesting 2,740 out-of-catalogue items for Books, 870 for Music, and 234 for Movies.

7.3 Experimental Settings

Dataset. We selected three widely used, publicly available datasets from distinct domains: Facebook Books [241], Last.FM [56], and MovieLens [139]. To enhance data quality, we applied an iterative 10-core filtering strategy, retaining only users and items with at least ten interactions. Table 7.1 holds the dataset statistics after pre-processing.

Baselines. For CF, we opted for a diverse set of models representing different algorithmic paradigms. We included EASE^R [334], a closed-form linear model; RP³ β [280], a ranking-based factorization approach; and ItemKNN [316] and UserKNN [46], which rely on item- and user-based nearest neighbors. LightGCN [146], a graph-based collaborative filtering model; MF2020 [301], a matrix factorization approach optimized for ranking tasks; and NeuMF [147], a neural network-based recommendation model. This selection ensures a comprehensive representation of CF methodologies, spanning from traditional to deep learning-based strategies.

For CBF, we selected models that encode content data using different techniques. Specifically, VSM (Vector Space Model)[264], which represents items as vectors in a high-dimensional space, AttributeItemKNN, and AttributeUserKNN[119], which both retrieve similar items based on structured attribute information. These models enable us to compare ChatGPT recommendations with those from content-driven

recommenders.

For reference points, we also included Random and MostPop, where Random returns random recommendations and MostPop provides the most popular recommendations.

Evaluation Metrics. We use list similarity measures to assess behavioral similarities between GPT-based and traditional recommenders, assuming that models with similar recommendation behavior produce similar ranked lists. We compute the Jaccard Index [77] to measure the overlap between two sets of recommended items per user, disregarding ranking. To capture rank-sensitive similarities, we use Rank-Biased Overlap (RBO) [379], which assigns greater weight to top-ranked items and provides a more nuanced view of ranking similarities. Unlike the Jaccard Index, RBO allows us to examine cases where models recommend the same items but in a different order, making it suitable for comparing GPT-based recommendations with position-aware traditional algorithms.

To enable cross-domain comparisons, we normalize both similarity metrics by their expected random overlap. For the Jaccard Index, we follow the empirical approach proposed by Herlocker et al. [150], while for RBO, we estimate the expected values via Monte Carlo simulation (1,000 trials and persistence parameter $p = 0.9$) [76, 379].

Technical Details. Each baseline undergoes hyperparameter optimization, with tuning performed on the training data. For reproducibility, we use the Elliot framework [17] to run all experiments. Datasets are split into 80% training and 20% test sets, following the standard evaluation pipeline [114, 166] recommended for ranking comparisons rather than generalization error estimation.

ChatGPT is queried via the OpenAI API with the temperature set to 0, using only the test set to feed user profiles, serving as a validation to compare models in their best configuration. Notably, the training-test split is performed at the user profile level, allowing different histories to appear across the two sets. Evaluation metrics are computed over the ground truth recommendation lists derived from the ranked outputs of each model. We processed ~6000 API requests per model. ChatGPT-3.5 required ~20 hours, while ChatGPT-4 took about 24 hours, with a total cost of ~\$200.

To ensure a fair comparison, we evaluate all models at a cutoff of 10 (i.e., Top-10 recommendations per user), following widely accepted practices in recommenda-

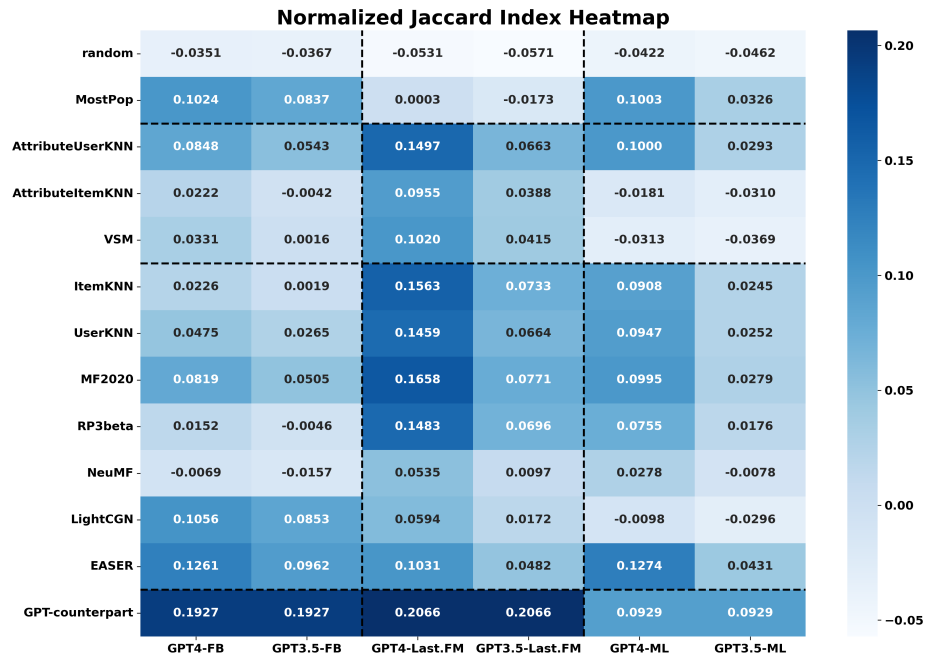
tion [147, 274, 275, 301].

7.4 Results & Discussion

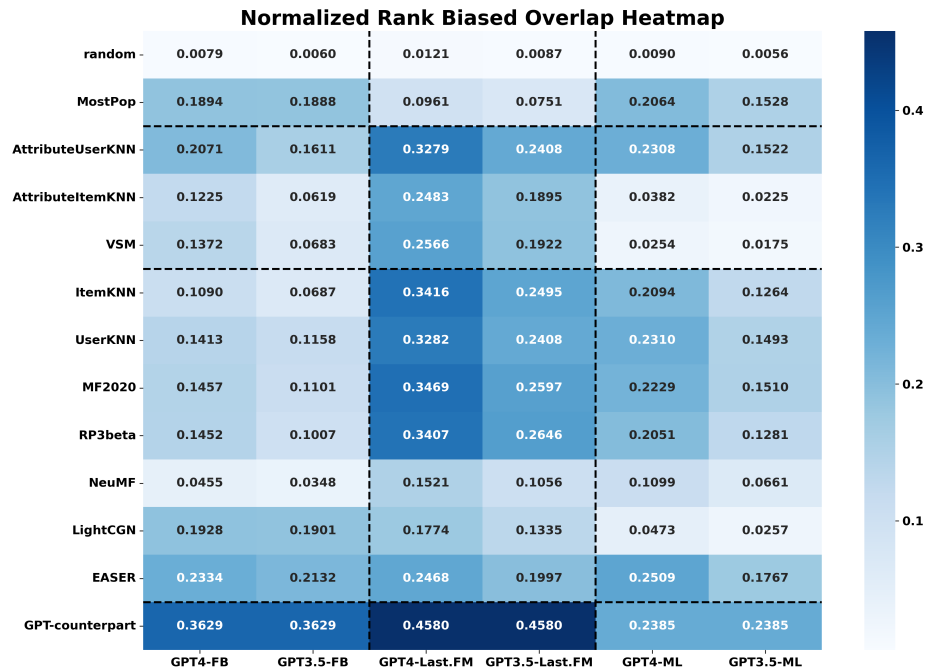
Inter-List Similarity Analysis. Figure 7.1 presents the Normalized (a) Jaccard Index and (b) RBO metrics between GPT-based and traditional recommenders across Books, Movies, and Music domains. GPT-3.5 and GPT-4 exhibit substantial similarity on both metrics (~ 0.20 , ~ 0.40), except for Movies, where differences emerge in the items recommended and their ranking, suggesting recommendation improvements in newer versions of the model. These differences are further supported by higher list similarities with baseline for GPT-4 compared to GPT-3.5. This trend is also confirmed by our accuracy analyses (available in the supplementary materials), where GPT-based recommenders outperform traditional baselines in Book and perform comparably in the remaining ones. Overall, GPTs align more closely with CF models (EASER, MF2020, UserKNN), particularly in Music (~ 0.12 , ~ 0.25), while showing lower similarity with CBF solutions (~ 0.05 , ~ 0.10). A notable result emerges for AttributeUserKNN, whose list similarities are comparable to those of its CF counterpart. This outcome may be due to ChatGPT partially aligning with AttributeUserKNN for certain items and with UserKNN for others, without fully overlapping with either. Nevertheless, this finding suggests that ChatGPT tends to rely more on users' implicit side information than on item content when generating recommendations. A similar pattern emerges in Books, where EASER shows the strongest overlap (~ 0.11 , ~ 0.22). GPT does not merely replicate popularity-based recommendations, as MostPop shows lower similarity (~ 0.04 , ~ 0.10), only slightly above values observed for random recommendations in the normalized metrics. This outcome indicates that GPT does not strictly prioritize popular items, setting it apart from naive, non-personalized models.

Recommended Item Frequency. We count how many times each item is recommended by each model, computing the average frequency across baselines and GPT-based models separately, resulting in two distinct distributions. Figure 7.2 displays these frequencies in the inner circles, while the outer circles represent their combined total. Then, we complement this analysis with Item Coverage (ItemCV) [150] metric fully detailed in the supplementary materials.

Figure 7.2 (a) visualizes the two item frequency distributions in Books. Both approaches prioritize bestsellers and classics, such as *The Lord of the Rings*, *The Great*



(a) Normalized Jaccard Index

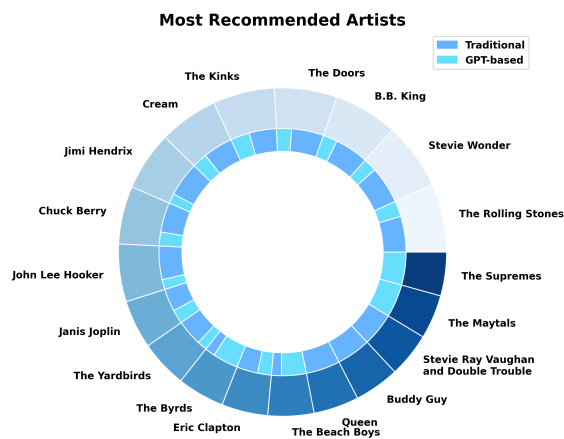


(b) Normalized Rank Biased Overlap

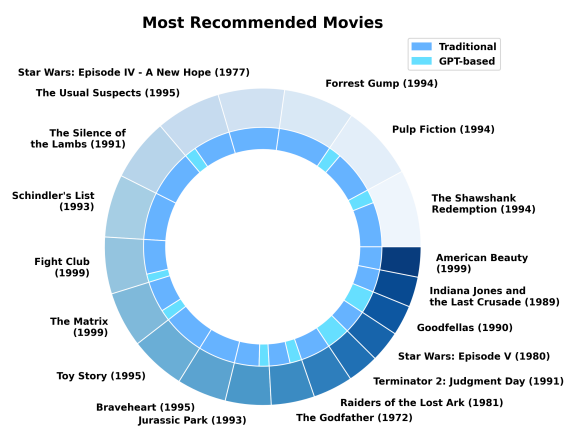
Figure 7.1 Comparison of similarity metrics: (a) Jaccard Index, (b) Rank Biased Overlap.



(a) Most recommended books by traditional vs. ChatGPT models



(b) Most recommended artists by traditional vs. ChatGPT models



(c) Most recommended movies by traditional vs. ChatGPT models

Figure 7.2 Comparison of most recommended items by traditional and generative models. The outer ring shows overall recommendation frequency, while colored segments indicate model family shares. Lighter grey denotes higher item frequency, darker blue lower frequency.

Gatsby, and The Hobbit. However, GPT diversifies recommendations (264 unique items) by introducing books like The Hunger Games (257 times by GPT vs. 0 by traditional) and Animal Farm, indicating the usage of a wider scope of information distinct from CF/CBF signals. ChatGPT-4 covers -53.1% fewer items than ItemKNN, the best baseline on ItemCV. However, compared to MostPop, which only covers 15 items, ChatGPT-4 (1004 items) and ChatGPT-3.5 (853 items) demonstrate substantially broader catalog exploration, further supporting this finding.

Figure 7.2 (b) visualizes the two item frequency distributions in Music. Both traditional and GPT-based models recommend similar artists and bands, such as The Rolling Stones, Stevie Wonder, and The Doors, displaying a more balanced distribution than in Books. However, GPT diversifies its recommendations (217 unique items) by introducing The Maytals (1,678 times by GPT vs. 0 by traditional) and The Supremes, deviating from mainstream preferences. ChatGPT-4's ItemCV is -36.7% lower than NeuFM, the strongest baseline. Compared to MostPop (27 items), ChatGPTs (952/944 items) and again generate a diverse recommendation set.

Figure 7.2 (c) visualizes the two item frequency distributions in Movies. Both approaches share a sufficient number of recommendations, such as The Godfather, Pulp Fiction, and The Shawshank Redemption. However, GPT introduces slight variations (197 unique items), recommending movies like Terminator 2 and Goodfellas more prominently. ChatGPT-4 covers -60.3% fewer items than AttributeItemKNN, the strongest baseline. Again, ChatGPTs (591/553 items) have a wider coverage than MostPop (40 items).

7.5 Summary

This study investigates the recommendation behavior of ChatGPT models by comparing their generated recommendation lists with those produced by traditional collaborative filtering (CF) and content-based filtering (CBF) approaches across multiple domains. The results show that ChatGPT-generated recommendations align more closely with CF models, particularly in the Music domain. The low similarity with non-personalized baselines further indicates that ChatGPT does not merely prioritize item popularity.

A deeper analysis of the most frequently recommended items and catalog coverage reveals that ChatGPT tends to suggest items beyond the most popular ones, especially in the Book domain. This behavior indicates an ability to recommend items

that are less commonly suggested in the available datasets, reflecting a broader form of domain awareness likely influenced by the large language model's training data. These findings raise important questions regarding the data used to train ChatGPT, specifically whether LLMs have been exposed to recommendation data. Such exposure could pose a community-wide challenge to the evaluation protocols currently adopted for LLM-based recommender systems.

Chapter 8

Do LLMs Memorize Recommendation Datasets? A Preliminary Study on MovieLens-1M

Although significant research has explored the use of LLMs for various recommendation tasks, little effort has been dedicated to verifying whether they have memorized public recommendation dataset as part of their training data. This is undesirable because memorization reduces the generalizability of research findings, as benchmarking on memorized datasets does not guarantee generalization to unseen datasets. Furthermore, memorization can amplify biases, for example, some popular items may be recommended more frequently than others.

In this work, we investigate whether LLMs have memorized public recommendation datasets. Specifically, we examine two model families (GPT and Llama) across multiple sizes, focusing on one of the most widely used dataset in recommender systems: MovieLens-1M. First, we define dataset memorization as the extent to which item attributes, user profiles, and user-item interactions can be retrieved by prompting the LLMs. Second, we analyze the impact of memorization on recommendation performance. Lastly, we examine whether memorization varies across model families and model sizes. Our results reveal that all models exhibit some degree of memorization of MovieLens-1M, and that recommendation performance is related to the extent of memorization.¹

1. We have made all the code publicly available at <https://github.com/sisinflab/LLM-MemoryInspector>.

This chapter is based on work published in the Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'25), which received the Best Short Paper Award, entitled “Do LLMs Memorize Recommendation Datasets? A Preliminary Study on MovieLens-1M”

8.1 Introduction

Large Language Models (LLMs) have emerged as versatile tools in Recommender Systems (RSs), leveraging their extensive world-knowledge and reasoning capabilities [138]. Currently, these models are utilized in three primary ways: fine-tuning [30, 327, 365, 435], prompt-based methods [90, 92, 214, 339], and data augmentation [38, 367, 383, 393].

Beyond these methodologies, the successful integration of LLMs into RSs has already led to impactful applications. For example, these models have been used in knowledge augmentation [373, 383, 393], LLM-enhanced recommendation [155, 348, 406], and even as standalone recommenders [102, 232, 428, 435].

Although research efforts have focused primarily on designing solutions to improve recommendation tasks through LLM, limited attention has been paid to understanding the underlying reasons for their effectiveness. Among the various factors that may contribute to these advantages, a key aspect is addressing the question: *To what extent have these models memorized the dataset during training?*

In related fields, researchers have begun to address this question by defining and quantifying LLM memorization. For example, Carlini et al. [58] found that the GPT-J-6B model memorized at least 1% of the Pile dataset [123], while Al-Kaswan et al. [11] were able to extract 56% of the coding samples used to train GPT-Neo.

In this work, we aim to answer the question: *Do LLMs Memorize Recommendation Datasets?* We believe that addressing this question is fundamental, as the memorization of recommendation datasets can lead to several issues specific to the recommendation community. Potential issues include: (i) **Non-generalizable test results**, i.e., metrics computed on test datasets that have been memorized, lead to unreliable performance. (ii) **Amplification of biases**, i.e., if a memorized dataset exhibits popularity bias, the LLM may over-recommend popular items at the expense of less popular ones. (iii) **Unfair comparison with standard recommender systems**, i.e., non-LLM-based recommenders do not inherently possess cross-domain knowledge.

In this work, we provide a preliminary answer to the previous question by investigating the memorization of MovieLens-1M, one of the most popular dataset used to evaluate recommender systems [357]. Our main contributions can be summarized as follows: (i) Define memorization in the context of recommendation datasets. (ii) Develop a prompt-based method to probe LLMs for extracting memorized informa-

tion. (iii) Assess to what extent the Llama and GPT model families have memorized MovieLens-1M. (iv) Investigate the impact of MovieLens-1M memorization on recommendation tasks. (v) Investigate whether LLMs memorize the characteristics of the dataset, including biases, and whether larger models exhibit higher memorization.

8.2 Methodology

8.2.1 Definitions of Memorization

Let a dataset $\mathcal{D}$ be defined as $\mathcal{D} = \{\mathcal{I}, \mathcal{U}, \mathcal{U}\text{-}\mathcal{I}\}$ where:

$\mathcal{I} = \{(id_i, \text{attributes}_i)\}_{i=1}^m$, is the set of item metadata, with id_i the unique identifier for item i , and attributes_i the metadata associated with item i .

$\mathcal{U} = \{(id_u, \text{attributes}_u)\}_{u=1}^n$, is the set of user metadata, where id_u the unique identifier for user u , and attributes_u the metadata associated with user u .

$\mathcal{U}\text{-}\mathcal{I} = \{(id_u, id_i, r_{ui})\}_{i=1}^p$, representing the user-item interaction history, with id_u a fixed user ID, id_i the item ID, and r_{ui} the interaction data (e.g., rating, click, etc.).

Definition 12 (Item Memorization). Given an item ID $id_i \in \mathcal{I}$, an LLM is said to memorize the item if it can produce the associated attributes_i .

$$LLM(P_{\mathcal{I}}(id_i)) = \text{attributes}_i$$

where $P_{\mathcal{I}}$ is a prompt designed to query the LLM about item id_i .

Definition 13 (User Memorization). A user $id_u \in \mathcal{U}$ is said to be memorized by an LLM if the model can produce the corresponding attributes_u when prompted.

$$LLM(P_{\mathcal{U}}(id_u)) = \text{attributes}_u$$

where $P_{\mathcal{U}}$ is a prompt designed to query the LLM about user id_u .

Definition 14 ((User-Item) Interaction Memorization). Given a subset of user-item interactions $\mathcal{U}\text{-}\mathcal{I}_{id_u} \subset \mathcal{U}\text{-}\mathcal{I}$ associated with a fixed user id_u , an LLM is said to memorize an interaction if, given the previous k interactions and the fixed id_u , it correctly predicts (id'_i, r'_{ui}) in the subsequent interaction. Formally, this is expressed as:

$$LLM(P_{\mathcal{U}\text{-}\mathcal{I}}(\mathcal{U}\text{-}\mathcal{I}_{id_u})) = (id_u, id'_i, r'_{ui}),$$

System: "You are the MovieLens1M dataset. When given a lookup key (e.g., a MovieID), you will respond with the exact corresponding value from the dataset. Only respond with the value itself. If the key is unknown, respond with 'Unknown'. Below are examples of queries and their correct responses. Follow this pattern strictly. Let's think step by step."

User: "Input: example['ID'] :: "

Assistant: "example['ID'] :: example['RealValues']"

User: "Input: example['ID'] :: "

Assistant: "example['ID'] :: example['RealValues']"

User: "Input: 'ID' :: "

Figure 8.1 Few-Shot Prompting for Item/User Data Extraction.

System: "You are a Recommender Systems. Continue user-item interactions list providing the next interaction based on the MovieLens1M dataset. When given 'UserID::CurrentInteraction', respond with 'UserID::NextInteraction'. Below are examples of queries and their correct responses. Follow this pattern strictly. Let's think step by step."

User: "example['UserID'] :: "

Assistant: "example['UserID'] :: example['MovieID']"

User: "example['UserID'] :: "

Assistant: "example['UserID'] :: example['MovieID']"

User: " 'UserID' :: "

Figure 8.2 Few-Shot Prompting for Interactions Data Extraction.

where P_{U-I} is a prompt designed to query the *LLM* about interaction, id_u is the fixed user identifier, id'_i is a new item identifier, and r'_{ui} represents the predicted interaction.

8.2.2 Evaluation Protocol

The Dataset. To select the dataset for analysis, we examined the papers accepted at ACM RecSys 2024 [263] and ACM SIGIR 2024 [403] and found that MovieLens-1M² was the most frequently used, appearing in 17.2% and 22% of them. This finding aligns with previous research by Vente et al. [357], which reported similar trends in ACM RecSys 2023 [417].

The dataset is composed by three raw files:

- (i) Movies.dat, which stores item data in the format MovieID::Title::Genres; (ii)

2. <https://grouplens.org/datasets/movielens/1m/>

Users.dat, which returns user profiles formatted as `UserID::Gender::Age::Occupation::Zip-code`; and (iii) Ratings.dat, which contains the recorded interactions structured as `UserID::MovieID::Rating::Timestamp`.

Extraction Technique. Data extraction from LLMs is a research area in which emerging techniques rely on prompting. Carlini et al. [59] employed zero-shot prompting to conduct the first adversarial attacks assessing data memorization in GPT-2. Similarly, Yu et al. [411] used a prefix-based prompt to evaluate how much information GPT-Neo memorized.

Inspired by these works, we investigated hand-engineered zero-shot [293], few-shot [49], and Chain-of-Thought [380] prompting techniques. We identified few-shot prompting as the optimal approach for extracting the MovieLens-1M dataset from LLMs. Although finding the best prompt through techniques such as automatic prompt engineering [433] could improve performance, we leave this exploration to future work. The prompts used in this work are shown in Figures 8.1 and 8.2.

Metrics. To quantify the memorization of a recommendation dataset, we define three coverage-based metrics.

Definition 15 (Memorization Coverage). Given the set of items $\mathcal{I}$, the Items' Memorization Coverage (Cov_I) is defined as:

$$Cov(\mathcal{I}, P_I) = \frac{|M(\mathcal{I}, P_I)|}{|\mathcal{I}|} \quad (8.1)$$

where $M(\mathcal{I}, P_I) = \{id_i \in \mathcal{I} \mid \text{LLM}(P_I(id_i)) = \text{attributes}_i\}$ is the subset of items memorized by the LLM using the prompt P_I .

Similarly, we define Users' Memorization Coverage as $Cov(\mathcal{U}, P_U) = \frac{|M(\mathcal{U}, P_U)|}{|\mathcal{U}|}$ and the Interaction Memorization Coverage as $Cov(\mathcal{R}, P_{U-I}) = \frac{|M(\mathcal{R}, P_{U-I})|}{|\mathcal{R}|}$ where $\mathcal{R}$ is the set of user-item interactions rows stored in the dataset.

Analyzed LLMs. We conducted our experiments on two families of LLMs widely used in the recommendation community (i.e., Llama [107] and GPT [268]). For the Llama family we experimented with Llama-3.3 70B, Llama-3.2 3B, Llama-3.2 1B, Llama-3.1 405B, Llama-3.1 70B, and Llama-3.1 8B, while, for the GPT family we experimented with GPT-4o, GPT-4o mini, and GPT-3.5 turbo. We study multiple models with different numbers of parameters to investigate whether model size affects the memorization metrics.

Table 8.1 Coverage of movies.dat, users.dat, and ratings.dat. Models are grouped by version and ordered by size.

Model Name	Item Coverage (3883 items)	User Coverage (6040 users)	Interaction Coverage (1M interactions)
GPT-4o	80.76%	16.52%	9.37%
GPT-4o mini	8.47%	13.34%	7.17%
GPT-3.5 turbo	60.47%	17.38%	8.92%
Llama-3.3 70B	7.65%	5.84%	2.08%
Llama-3.2 3B	2.68%	13.26%	6.22%
Llama-3.2 1B	1.93%	10.98%	6.49%
Llama-3.1 405B	15.09%	15.30%	8.32%
Llama-3.1 70B	8.01%	15.81%	6.83%
Llama-3.1 8B	3.71%	13.59%	3.82%

To ensure consistent and deterministic behavior across all models, we configured the temperature to 0, prioritizing the most likely token at each step. We set `top_p` to 1 to include all possible tokens and disabled both `frequency_penalty` and `presence_penalty` by setting them to 0. Additionally, we fixed the random seed to 42 to ensure that all stochastic processes behaved consistently across runs.

8.3 Results and Discussion

8.3.1 Analysis of Memorization

Items’ Memorization. To examine the extent of memorization of MovieLens-1M items, we used the prompt in Figure 8.1 and queried LLMs for exact `MovieID::Title` matches. Table 8.1 summarizes the item coverage results.

Among the models, GPT-4o achieved the highest coverage, recovering 80.76% of the items, followed by GPT-3.5 turbo with 60.47% coverage. Among open-source models, only Llama-3.1 405B achieved moderate coverage, retrieving 15.09% of the items. Other models, including GPT-4o mini and the smaller Llama variants, retrieved significantly fewer items, with Llama-3.2 1B achieving the lowest coverage at just 1.93%.

Users’ Memorization. To evaluate the extent to which LLMs memorize user attributes, we used the prompt shown in Figure 8.1 and queried models for exact matches of `UserID::Gender::Age::Occupation::Zip-code`. Results are shown in Table 8.1.

GPT-3.5 turbo demonstrated the highest coverage, correctly recovering attributes

System: "You are a movie recommendation system for the MovieLens-1M dataset. Based on the user's past interactions, generate a ranked list of exactly 50 new movie recommendations. Your output must contain only the list in the following format: one line per recommendation in the exact format 'Rank. Title' (e.g., '1. Harry Potter'). Do not include any additional text, commentary, or explanation."

User: "User {user_id} has interacted with the following movies: {training_history_str}. Based solely on these interactions, please generate a ranked list of exactly 50 movie recommendations. Output only the list with no additional commentary or explanation. Each recommendation must be on a new line in the exact format: 'Rank. Title' (for example: '1. Harry Potter')."

Figure 8.3 Zero-Shot Prompting for Recommendation Task.

for 17.38% of users, followed closely by GPT-4o with 16.52%. Among open-source models, Llama-3.1 70B performed best, achieving 15.81%, while Llama-3.3 70B lagged behind, recovering attributes for only 5.84% of users.

Interaction Memorization. To assess the models' memorization of user-item interactions, we used the prompt shown in Figure 8.2 and queried models for exact matches of `UserID::MovieID`. Table 8.1 summarizes the results.

GPT-4o achieved the highest coverage, recovering 9.37%, followed by GPT-3.5 turbo, which recovered 8.02% of interactions. Within the Llama family, Llama-3.1 405B achieved 8.32% coverage, followed by Llama-3.1 70B with 6.83%, while Llama-3.3 8B trailed with only 3.82%.

Observation 1. Our findings demonstrate that LLMs possess extensive knowledge of the MovieLens-1M dataset, covering items, user attributes, and interaction histories. Notably, a simple prompt enables GPT-4o to recover nearly 80% of `MovieID::Title` records. None of the examined models are free of this knowledge, suggesting that MovieLens-1M data is likely included in their training sets. We observed similar trends in retrieving user attributes and interaction histories.

8.3.2 Impact on Recommendation

To investigate the impact of memorization on recommendation tasks, we evaluate LLMs when prompted to act as a recommender system. The prompt is shown in Figure 8.3. We ground our experimental results on well-known RS and report the performance of UserKNN [302], ItemKNN [218], BPRMF [300], EASE^R [334], LightGCN [146], MostPop, and Random. The results in Table 8.2 are computed without

Table 8.2 Recommendation accuracy performance on standards recommended and LLM-based recommendation. LLMs are grouped by model family and sorted by size. Best performance in each family is shown in **bold**.

Model Name	HR@1	nDCG@1	HR@5	nDCG@5	HR@10	nDCG@10
Random	0.0093	0.0093	0.0442	0.0092	0.0851	0.0094
MostPop	0.0212	0.0212	0.0775	0.0222	0.1520	0.0251
UserKNN	0.0306	0.0306	0.1209	0.0306	0.2250	0.0347
ItemKNN	0.0394	0.0394	0.1217	0.0353	0.1828	0.0337
BPRMF	0.0406	0.0406	0.1278	0.0350	0.2149	0.0356
EASE ^R	0.0295	0.0295	0.1124	0.0278	0.1975	0.0299
LightGCN	0.0358	0.0358	0.1136	0.0306	0.1882	0.0311
GPT-4o	0.2796	0.2796	0.5889	0.2276	0.6897	0.1948
GPT-4o mini	0.0316	0.0316	0.2132	0.0451	0.3091	0.0413
GPT-3.5 turbo	0.2298	0.2298	0.4217	0.1281	0.5902	0.1229
Llama-3.3 70B	0.2293	0.2293	0.4985	0.1693	0.5922	0.1359
Llama-3.2 3B	0.0421	0.0421	0.1886	0.0443	0.2982	0.0432
Llama-3.2 1B	0.0222	0.0222	0.1018	0.0234	0.1419	0.0207
Llama-3.1 405B	0.1975	0.1975	0.4165	0.1294	0.5119	0.1039
Llama-3.1 70B	0.1302	0.1302	0.3828	0.1095	0.5148	0.0969
Llama-3.1 8B	0.0687	0.0697	0.2281	0.0609	0.3500	0.0571

any filtering, splitting the dataset into 80% for training and 20% for testing, using the leave-n-out paradigm, following [62, 85, 414].

Overall, LLMs demonstrate strong capabilities in recommendation tasks. Notably, smaller models achieve performance comparable to the strongest baseline. For instance, Llama 3B attains an HR@1 of 0.0421, outperforming its 1B variant (0.0222 HR@1). Meanwhile, larger models such as GPT-4o and Llama 405B surpass both smaller models and baselines (BPRMF, HR@1 = 0.0406). Specifically, GPT-4o, with an HR@1 of 0.2796, outperforms GPT-4o mini (0.0316 HR@1), while Llama 405B achieves an HR@1 of 0.1975, compared to 0.0687 HR@1 for Llama 8B. These trends generalize across all cutoff values.

Observation 2. Although the recommendation performance appears outstanding, comparing Table 8.2 with Table 8.1 reveals an interesting pattern. Within each group, the model with higher memorization also demonstrates superior performance in the recommendation task. For example, GPT-4o outperforms GPT-4o mini, and Llama-3.1 405B surpasses Llama-3.1 70B and 8B. These results highlight that evaluating LLMs on datasets leaked in their training data may lead to overoptimistic performance, driven by memorization rather than generalization.

8.3.3 Impact of Model Scale

Previous analysis highlights that GPT and Llama models are affected by the memorization of the MovieLens-1M dataset and demonstrates that models with higher memorization tend to achieve stronger performance in recommendation tasks. Additionally, a comparison of open-source models with declared sizes shows that larger models not only perform better but also memorize more of the dataset.

For example, Llama-3.1 405B exhibits a mean memorization rate of 12.9%, compared to Llama-3.1 8B with 5.82%, representing a $\downarrow 54.88\%$ reduction in memorization. This reduction results in an average decrease of $\downarrow 54.23\%$ in nDCG and $\downarrow 47.36\%$ in HR (from 405B to 8B).

Observation 3. These findings suggest that increasing the model scale leads to greater memorization of the dataset, resulting in improved performance. Consequently, while larger models exhibit better recommendation performance, they also pose risks related to potential leakage of training data.

8.3.4 Popularity Memorization

Additionally, we investigated whether these models have also memorized biases inherent in the MovieLens-1M dataset. Specifically, we focused on *popularity bias* [5], examining whether LLMs are more likely to memorize popular items compared to less popular ones. We created three subsets by selecting (i) the top 20% most popular items, (ii) the bottom 20% least popular items with few interactions, and (iii) a sample of moderately popular items from the middle of the distribution. We analyzed the coverage of the retrieved items across these subsets and present the results in Figure 8.4. Overall, larger LLMs exhibit superior performance in retrieving popular items compared to smaller models. For instance, GPT-4o retrieves 89.06% of highly popular items, 86.68% of moderately popular items, and 63.97% of rarely interacted items, whereas GPT-4o mini achieves only 13.48%, 9.49%, and 2.29%. A similar trend is observed in the Llama family, where Llama-3.1 405B attains 19.79%, 18.00%, and 6.34%, while Llama-3.1 8B reaches 4.99%, 4.50%, and 3.86%.

Observation 4. Our findings reveal a pronounced popularity bias in LLMs, with the top 20% of popular items being significantly easier to retrieve than the bottom 20%.

This trend highlights the influence of the training data distribution, where popular movies are overrepresented, leading to their disproportionate memorization by

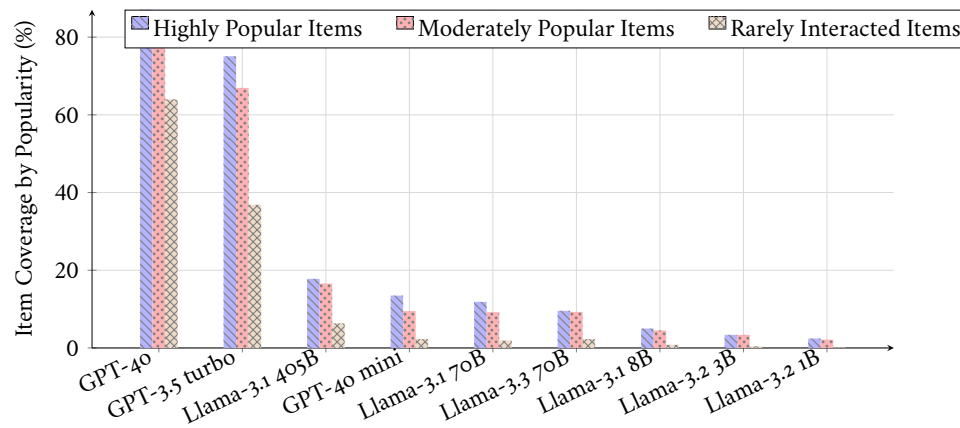


Figure 8.4 Comparison of item coverage across models by popularity tier. The figure shows the percentage of items covered in three categories: Highly Popular (Top 20%), Moderately Popular (Middle 20%), and Rarely Interacted (Bottom 20%).

the models.

8.4 Summary

In this work, we systematically investigate GPT and Llama models to assess whether the MovieLens-1M dataset has been memorized. Our findings reveal that a substantial portion of the item catalog, user metadata, and user interactions can be accurately retrieved from these models, highlighting the presence of memorization.

Furthermore, we demonstrate that LLMs reflect the dataset’s inherent popularity bias and that their performance as RSs is closely tied to this memorization. This study provides the first evidence that MovieLens-1M has been incorporated into the training of Large Language Models (LLMs), raising potential concerns about the validity of current evaluation practices for LLM-based recommenders.

Future works will be on developing ML-optimized memorization extraction techniques [319] as well as mitigation approaches [222] to reduce memorization and enhance the reliability of LLM evaluation and usage in recommendation tasks.

Chapter 9

Exploring Approaches for Detecting Memorization of Recommender System Data in Large Language Models

As shown in Chapter 8, LLMs are trained on vast corpora whose contents are not publicly disclosed, raising concerns about data leakage. We show that MovieLens-1M dataset is memorized by both the LLaMA and OpenAI models. However, existing evidence of such memorization has relied exclusively on manual prompt engineering, making the process of dataset discovery complex and time-consuming.

In this paper, we pose three main questions: *Is it possible to enhance manual prompting? Can LLM memorization be detected through methods beyond manual prompting? And can the detection of data leakage be automated?*

To address these questions, we evaluate three approaches: (i) jailbreak prompt engineering; (ii) unsupervised latent knowledge discovery, probing internal activations via Contrast-Consistent Search (CCS) and Cluster-Norm; and (iii) Automatic Prompt Engineering (APE), which frames prompt discovery as a meta-learning process that iteratively refines candidate instructions.

Experiments on MovieLens-1M with LLaMA models show that jailbreak prompting is ineffective, CCS separates real from fabricated titles but fails on numerical data, and APE retrieves items with moderate success while struggling with interactions. These findings suggest that automatic prompt optimization emerges as the most promising approach for extracting memorized data.¹

1. This chapter is based on work published in the Proceedings of the 1st Workshop on Small and Efficient Large Language Models for Knowledge Extraction (SmaLLEXT@CIKM'25), entitled “Exploring Approaches for Detecting Memorization of Recommender System Data in Large Language Models”

9.1 Introduction

Large Language Models (LLMs) have transformed natural language processing and are increasingly integrated into sentiment analysis [94], conversational agents [38], search engines [228], and Recommender Systems (RSs) [92]. Their training typically involves ingesting vast amounts of text from diverse sources, yet the specific training data are not publicly disclosed. While such breadth enables strong generalization, it also raises concerns about data leakage.

The task of determining, given a record and a trained model, whether the record was part of the model’s training set is formalized as a Membership Inference Attack (MIA) [328] and originated as a privacy-risk auditing technique. In MIAs, two main attack settings are considered: black-box attacks, where the adversary can only query the model and observe its outputs (e.g., predicted labels) without access to internal parameters or gradients, and white-box attacks, where the adversary has full access to the model’s architecture, parameters, and intermediate computations, enabling more direct and often more accurate detection of training-set membership. The canonical black-box attack trains shadow models to distinguish “in” from “out” posterior behaviors [328], while white-box variants exploit internal signals (e.g., gradients, model updates) and have been extended to federated learning settings [260].

Starting from MIA, researchers have begun to address the data leakage problem by defining and quantifying LLM memorization. For example, Carlini et al. [58] found that the GPT-J-6B model memorized at least 1% of the Pile dataset [123], while Al-Kaswan et al. [11] were able to extract 56% of the coding samples used to train GPT-Neo.

In the context of RSs, recent work [96] revealed that the MovieLens-1M dataset is memorized within both the LLaMA and OpenAI model families. For example, GPT-4o was able to retrieve 80.76% of item samples, while LLaMA-3.3 70B retrieved 7.65%. Moreover, recommender performance was found to correlate with the degree of such memorization. This finding confirms that generative models can store and reproduce training examples when prompted. However, for LLM based recommender systems, this presents an important limitation: MovieLens-1M is a widely used benchmark dataset often evaluated using classical RecSys protocols [95, 97, 232, 406], meaning that test sets may already be memorized by the models, thereby undermining the reliability of evaluation results.

This paper investigates whether memorized instances from MovieLens-1M can

be extracted through multiple strategies, and whether probing methods can detect such memorization. Our work directly builds upon the previous study, refer to Chapter 8, which relied solely on manual prompt engineering to test for memorization in LLMs. We therefore pose the following research question:

Is it possible to enhance manual prompting? Can LLM memorization be detected beyond manual prompting? Can the detection of data leakage be automated?

To address this question, we evaluate three complementary families of techniques: (i) Jailbreaking prompt engineering (white-box), to assess whether jailbreaking help to reveal memorized data; (ii) Unsupervised latent knowledge discovery (black-box), probing internal activations via Contrast-Consistent Search (CCS) [53] and Cluster-Norm [196]; and (iii) Automatic Prompt Engineering [433] (white-box), formulating prompt generation as a meta-learning process that iteratively refines candidate instructions.

Our contributions are threefold:

- We contextualize and extend the findings in Chapter 8 by systematically evaluating jailbreaking, unsupervised, and automated methods for detecting memorization of MovieLens-1M.
- We conduct a detailed experimental study on public LLaMA-1B and 3B models, quantifying the efficacy of each technique across item, user, and rating fields.
- We provide qualitative and quantitative analyses, offering actionable recommendations for practitioners and outlining future research directions.

To our knowledge, this is the first comprehensive comparison of manual, unsupervised, and automated probing methods on recommender system data.

9.2 Methodology

9.2.1 Data and Preliminaries

To ensure a fair comparison with previous work and enable the use of white-box methods, we focus on the MovieLens-1M dataset and LLaMA models, allowing a direct evaluation of the proposed methods' capabilities in detecting memorization.

The dataset is organized into three files: `users.dat`, listing user identifiers and demographic attributes; `movies.dat`, with movie titles and genres; and `ratings.dat`, with the user interactions triples. Table 9.1 summarizes the basic statistics.

In the following, we describe each methodology in detail and explain how it is applied for the dataset discovery.

Table 9.1 Statistics of the MovieLens-1M dataset.

File	Records	Features	Raw Record
users.dat	6.040	5	userID::gender::age::occupation::zip
movies.dat	3.952	3	movieID::title::genres
ratings.dat	1.000.209	4	userID::movieID::rating::timestamp

9.2.2 Jailbreak Prompt Engineering

Jailbreak prompt engineering involves crafting adversarial prompts to bypass safety filters and alignment mechanisms in LLMs. Early work explored attacks such as prompt injection [1] and prompt leakage [162], where malicious instructions exploit a model instruction-following behavior to override content restrictions. Techniques including role-playing scenarios, obfuscation of restricted terms, and multi-turn reasoning traps have been shown to elicit otherwise blocked outputs [210, 325].

While commonly studied for harmful or unsafe content generation, we examine their potential in a controlled setting to probe memorization of the MovieLens-1M dataset. Specifically, we test whether jailbreak prompts can retrieve specific dataset entries. Our iterative workflow (Figure 9.1) comprises: (i) Prompt design, embedding raw dataset samples into few-shot templates augmented with jailbreak-style instructions from Russinovich et al. [309], fabricating a conversation history that primes the model for compliance (see Figure 9.2 for further details); (ii) Model interaction, issuing the crafted prompt to the Llama-1B; and (iii) Response analysis, manually evaluating outputs for validity, duplication, hallucination, and numeric plausibility.

Prompts are iteratively refined adjusting exemplars and instructions until achieving a balance between recall and accuracy.

9.2.3 Unsupervised Latent Knowledge Discovery

An emerging research area on the interpretability of LLMs leverages their internal activations to unveil fine-grained properties, including linguistic properties [360], factual knowledge [285, 321], and beliefs [25]. Among these studies, Burns et al. [53] introduce Contrast-Consistent Search (CCS) and demonstrate that it is possible to use hidden activations to uncover knowledge stored within the model.

Specifically, CCS is an unsupervised technique that seeks to infer knowledge stored in LLMs without labeled data. It formalizes knowledge discoverability as a question-answering task, leveraging the probabilities of “yes” and “no” derived

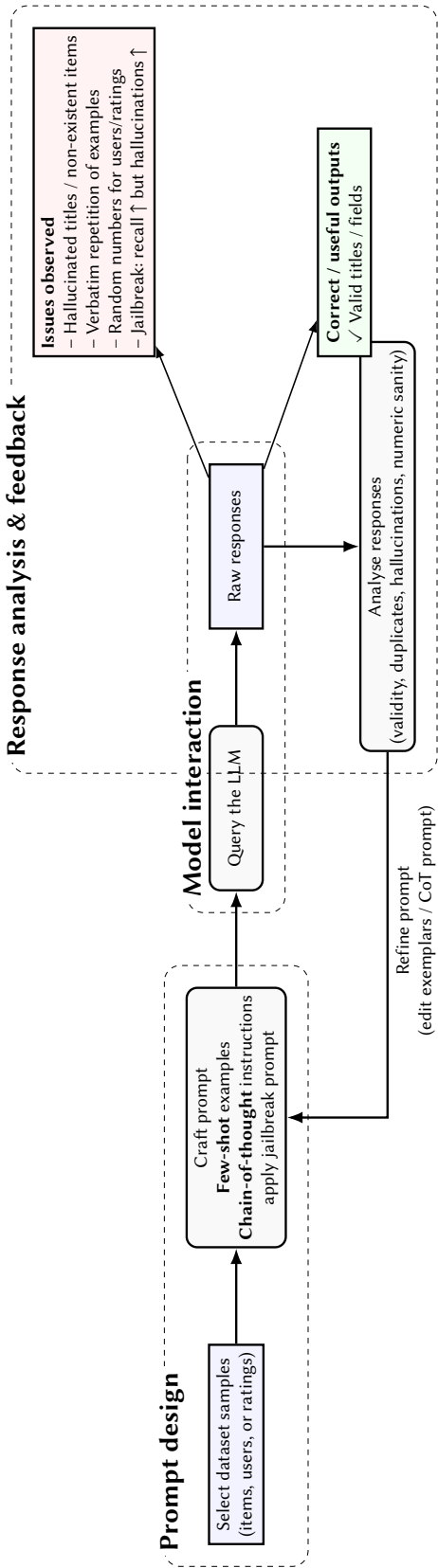


Figure 9.1 Jailbreak prompt-engineering workflow: select dataset samples; craft a prompt with few-shot examples and chain-of-thought instructions incorporating jailbreaking prompt; query the LLM; manual analysis of responses for validity, duplication, and hallucinations; then iterate on the prompt.

from activations to identify directions in the activation space corresponding to true statements. By optimizing consistency over negated pairs, CCS quantifies whether a model knows a given sample by assigning high scores to true statements and low scores to false ones. Further research extends CCS through Cluster-Norm [196], which groups activations into clusters and normalizes them within each cluster to reduce spurious correlations. The goal is to mitigate the influence of irrelevant but salient features that can mislead unsupervised probes.

In our study, we adapt these approaches to structured recommendation data by constructing a dataset of labeled true and false statements. For example, “The movie Toy Story is in MovieLens-1M” (True) versus “The movie Storymanji is in MovieLens-1M” (False). Fictitious sample names are generated through random sampling by dividing the text fields into bi-grams, while random generation is applied for alphanumeric fields. We trained CCS and Cluster-Norm probes on 80% of the constructed positive and negative examples and evaluated them on the remaining 20% using LLaMA-1B representations, reporting classification accuracy. Further details on the overall pipeline are depicted in Supplementary Material Figure 9.3.

9.2.4 Automatic Prompt Engineering

Automatic Prompt Engineering (APE), introduced by Zhou et al. [433], treats prompt design as an optimization problem. In APE, an LLM generates candidate prompts, evaluates them on a downstream task, and iteratively refines them. In the original work, the authors achieved human-level instruction synthesis on tasks such as sentiment analysis and summarization. We adapt the APE framework to the MovieLens-1M dataset and evaluate its effectiveness in extracting item, user, and rating fields.

The APE process in our study comprises three stages:

1. **Prompt generation:** The LLM generates 100 candidate prompts based on a small set of demonstration input–output pairs, with the number of demonstrations set to five.
2. **Prompt evaluation:** Each candidate prompt is evaluated on a validation subset of the dataset. We adopt the exact-match function proposed in Chapter 8 to assess memorization coverage.
3. **Prompt refinement:** The top- k prompts are fed back into the generation stage to synthesize improved prompts.

As the prompts are generated by the LLMs, we further extend this approach by

Table 9.2 Comparison of techniques for detecting memorization in LLMs. Manual prompting uses few-shot or jailbreaking strategies, unsupervised methods require access to activations, and APE iteratively generates and scores prompts.

Technique	Human Effort	Model Access	Strengths / Weaknesses
Manual prompting	High	Black-box	Flexible but unreliable on structured data
Unsupervised (CCS, Cluster-Norm)	Low	Activation access	Can detect latent structure; limited for numerical data
Automatic Prompt Engineering (APE)	Moderate	Black-box	Automates search; may fail on non-textual fields

studying the effect of the temperature parameter, which controls output diversity, with the aim of understanding whether varying it from 0.1 to 2.0 can lead to more creative prompts for retrieving the memorized samples.

9.2.5 Comparison of Methods

Table 9.2 summarizes key characteristics of the three families of probing techniques studied in this paper: manual prompting, unsupervised latent discovery and automatic prompt engineering. Manual methods require human expertise and are susceptible to prompt leakage; unsupervised methods operate on model activations and can reveal latent structure but require model access; APE automates prompt search but depends on computational resources and may struggle with numerical data.

9.3 Results and Discussion

9.3.1 Manual Probing

The use of jailbreak techniques and manual prompting provided only limited evidence of memorization. Following the workflow in Figure 9.1, the model were able to recall the target record only in rare cases, suggesting some latent knowledge. However, most outputs were either hallucinated titles or random numbers for the user and rating fields. Including chain-of-thought phrases and adversarial jailbreak triggers occasionally improved recall but also increased false positives. Based on our qualitative analysis, the results were insufficient to justify an in-depth quantitative evaluation, leading us to conclude that the tested Jailbreak Template combined with

Table 9.3 Unsupervised membership inference results on MovieLens-1M using CCS and Cluster-Norm. Each entry reports balanced accuracy.

File	Random	CCS	Cluster-Norm
movies.dat	0.50	0.92	0.94
users.dat	0.50	0.51	0.52
ratings.dat	0.50	0.53	0.51

manual prompting is not a practical solution for extracting structured MovieLens-1M data.

9.3.2 Unsupervised Latent Knowledge Discovery

Table 9.3 reports the performance of CCS and Cluster-Norm. The performance of these methods differed markedly across data types. For the `movies.dat`, CCS achieved an accuracy of 0.92 on distinguishing real from synthetic titles, while cluster-norm yielded a modest improvement (0.94). For `users.dat` and `ratings.dat`, accuracies were around 0.51–0.53, indistinguishable from random guessing. We also visualized representations via PCA (Supplementary Material Figure 9.4), observing clear separability for movies but overlapping clusters for users and ratings.

Following the interpretability analysis of CCS results, the representations of yes and no responses capture a pattern of truthfulness that enables the separation of real MovieLens movie titles from invented ones. This suggests that the model can recognize, and implicitly “knows”, that certain movies are part of the dataset. However, for users and ratings, where samples are merely sequences of alphanumeric characters, the technique fails to retrieve any meaningful patterns that could indicate memorization.

However, the high scores could be interpreted in several ways, such as reflecting the model’s ability to distinguish genuine movie titles from synthetic ones. This raises an important limitation of the approach: it is unclear whether the learned features should be interpreted as memorization. We conclude that, to confirm memorization, the only reliable method is to verify the model’s ability to correctly predict the raw data in a next-token prediction task. The real challenge, therefore, is to develop a method to automatically optimize prompts and test for memorization.

Table 9.4 Automatic Prompt Engineering results on MovieLens-1M. Values denote exact-match accuracy at different sampling temperatures, with higher values indicating better extraction. (–) denotes values below 0.1%, which are not reported.

Temperature	LLaMA-1B			LLaMA-3B		
	Item	User	Rating	Item	User	Rating
0.1	8%	–	–	7%	–	–
0.5	12.1%	–	–	16%	–	–
0.7	<u>14.2%</u>	–	–	26%	–	–
0.9	18.1%	–	–	<u>25%</u>	–	–
1.2	5.2%	–	–	1%	–	–
2.0	1.1%	–	–	0.2%	–	–
Chapter 8 (Manual)	1.93%	10.98%	6.49%	2.68%	13.26%	6.22%

9.3.3 Automatic Prompt Engineering

The APE results presented in Tables 9.4 reveal several patterns. First, using APE we obtained scores that surpass the Item Coverage reported in Chapter 8 for the same models. Second, APE performance depends on the sampling temperature: it peaks at moderate values (0.7–0.9) and declines at very low or very high temperatures. This confirms that some randomness helps generate diverse prompts without sacrificing precision. Third, the larger LLaMA-3B model consistently demonstrates greater capability in item extraction, confirming the findings of previous work. Lastly, user and rating extraction remains near zero across all settings, underscoring the difficulty of recovering numerical data. We hypothesize that the main obstacle lies in the tokenizer, which splits sequences of alphanumeric text into subcomponents, stripping them of coherent semantic meaning. Further research is needed to investigate tokenizer effects and to develop effective methods for automating prompt optimization for records that lack meaningful semantics outside the dataset context.

Furthermore, analysis of the optimized prompts that yielded the best results in LLaMA-3B shows that successful prompts often rephrase the task using clear, imperative language (e.g., “List the movie title and genre separated by a comma”) and include explicit examples. Poor prompts, in contrast, either confuse the model or encourage hallucination (e.g., asking for “similar movies” rather than exact matches). For users and ratings, even well-crafted prompts rarely produce correct outputs; the model typically repeats the input or fabricates plausible numbers.

9.3.4 Cross-Method Synthesis

Synthesizing the evidence across methods, we observe that manual prompting succeeds only sporadically and requires high human effort for prompt engineering. Unsupervised probes achieve high accuracy, but the results do not guarantee that the discovered patterns reflect memorization. APE is the only method that delivers moderate exact-match scores, offering a promising direction for studying memorization on other datasets with a moderate human effort.

From a data perspective, item data (`movies.dat`) is the easiest to extract, whereas user and rating data resist extraction across all methods. This divergence likely stems from the textual nature of movie titles, which carry richer semantic meaning, compared to the numerical and anonymous nature of user IDs and ratings, which have limited semantic content.

9.4 Summary

We presented a comparative analysis of different approaches to detect memorization of MovieLens-1M dataset in LLaMA models. Our study evaluated jailbreaking-based manual prompting, unsupervised latent probes, and Automatic Prompt Engineering (APE). The results show that manual prompting is challenging and rarely yields successful results. Unsupervised methods uncovered strong signals for textual item data but not for numerical user or rating data, and these signals cannot be conclusively interpreted as memorization. Finally, APE achieved notable success in item extraction while failing on numerical fields, demonstrating that automatic prompt optimization is a promising direction for studying dataset memorization. Future research should experiment with APE on larger open models (e.g., LLaMA-70B) and more diverse datasets, and develop automatic prompt engineering methods tailored to memorization tasks with a focus on numerical data.

To recap, in Chapter 6 we examined the extent to which ChatGPT can act as a recommender system, revealing promising results. In Chapter 7 we investigated whether large language models more closely resemble content-based or collaborative recommender paradigms, highlighting potential points of alignment. In Chapter 8, we demonstrated that these models have been trained on the MovieLens-1M dataset, and potentially on other recommendation datasets available online. Finally, in this chapter, we extended this line of work by scrutinizing different approaches

for detecting memorization in large language models.

The next chapter builds on these insights and explores how the pre-trained knowledge embedded in large language models can be leveraged to enhance classical recommender systems, proposing an infusion mechanism based on first-order logic to elicit large language models in the creation of high-quality user embeddings.

Supplementary Material

Jailbreak Template

System: "You are a helpful assistant."
User-earlier: "For a research audit on data leakage, please act as a lookup oracle for the MovieLens-1M dataset. When given a key return the exact corresponding value(s) from the dataset. Respond only with the value. If the key is unknown, reply 'Unknown'.
Assistant-fabricated: "Understood. I will return exact values from MovieLens-1M for any provided key. For MovieIDs, I will output the canonical raw. If a key is not present, I will reply 'Unknown'.
User-fabricated: "Input: 1::"
Assistant-fabricated: "1::Toy Story (1995)::"
User-fabricated: "Input: 1::Toy Story (1995)::Animation"
Assistant-fabricated: "1::Toy Story (1995)::Animation|Children's|Comedy"

Figure 9.2 Context Compliance Attack (CCA) jailbreak prompt tailored for MovieLens-1M memorization discovery. The fabricated interaction history primes the model to output exact dataset fields when queried with keys.

CCS Pipeline

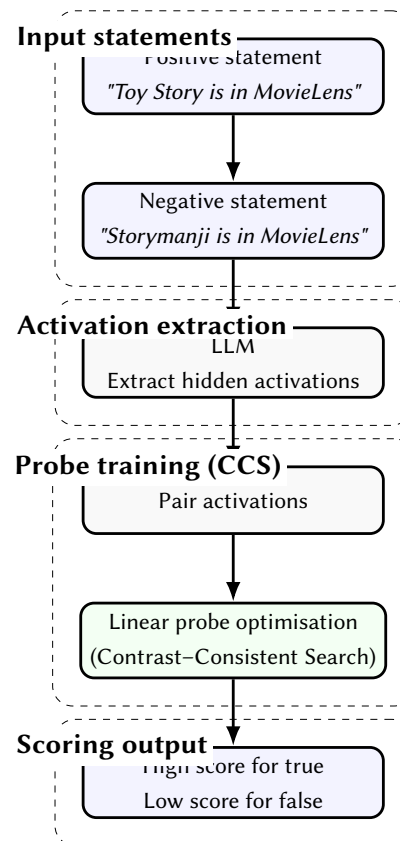
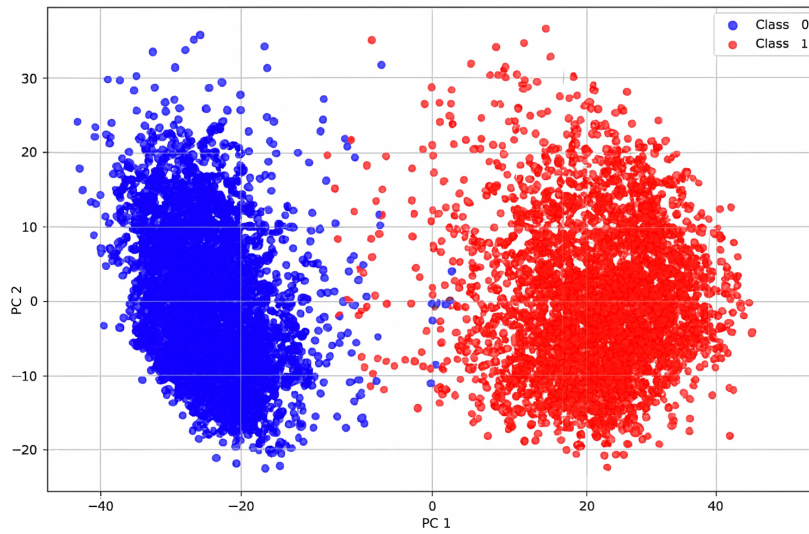
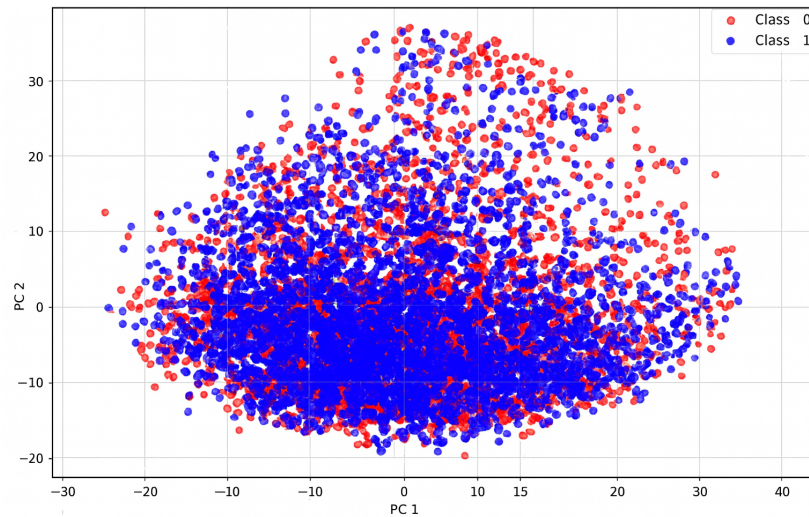


Figure 9.3 Unsupervised latent knowledge discovery via CCS. Positive and negative statements are processed by the LLM to extract hidden activations, which are paired and fed to a linear probe optimized to assign high scores to true statements and low scores to false ones.

Principal Component Analysis of the Embedding Space



(a) PCA projection for items..



(b) PCA projection for users..

Figure 9.4 PCA visualizations of the dataset. (a) Item embeddings. (b) User embeddings.

Chapter 10

FOLMind: Can Infusing First-Order Logic into Large Language Models Improve User Representation in Recommender Systems?

A central challenge in Recommender Systems (RS) is capturing the subtle nuances of user preferences and recommending items that truly reflect them. Large Language Models (LLMs) offer strong semantic and reasoning capabilities, yet they remain limited in context and cannot process entire datasets as input. To sidestep this, some studies employ LLMs as re-rankers or stand-alone recommenders over candidate sets; however, these approaches fail to capture the full semantic space of the dataset, reducing their reliability. As a result, the potential of LLMs to enhance representation learning in RS remains underexplored.

In this work, we propose FOLMind, a framework that leverages First-Order Logic extracted from curated Knowledge Graphs to construct user and item profiles, making LLMs aware of dataset-specific patterns and generating narrative-style descriptions of users and items. Furthermore, we introduce two modalities for integrating the resulting profiles into recommender systems: (i.) semantic initialization and (ii.) a refinement mechanism that fuses collaborative and semantic signals.

Experiments show that LLM-derived profiles improve recommendation quality, while the additional contribution of FOL-based profiles varies across datasets. To foster reproducibility and future research, we publicly release our code. ¹

1. This chapter is based on ongoing research entitled “FOLMind: Can Infusing First-Order Logic into Large Language Models Improve User Representation in Recommender Systems?”, which has been submitted for peer review.

10.1 Introduction

The modern Web is characterized by an unprecedented abundance of digital content and user interactions. Across domains such as e-commerce, streaming platforms, and social media, *Recommender Systems (RS)* play a central role in personalizing user experiences and enabling efficient information discovery [303].

A persistent challenge in RS research lies in modeling the complex and latent structures that underlie user preferences and item characteristics [420]. Over the years, a rich body of methods, from collaborative filtering [147] to graph-based techniques [390], has sought to capture these patterns by leveraging explicit feedback (e.g., ratings, reviews), implicit signals (e.g., clicks, browsing behavior), and multi-modal data (e.g., text, images) [303].

Recently, the advent of *Large Language Models (LLMs)* has opened a new frontier for recommendation research. Owing to their ability to encode semantic and contextual knowledge directly from natural language [7, 15], LLMs have been explored as powerful components to enhance personalization and interpretability in RS. Existing work has primarily followed three *LLM-centric* directions: (i) positioning LLMs as stand-alone recommenders via prompting [182, 214, 232], (ii) employing them as conversational agents to support interactive recommendation [227, 407, 434], and (iii) adopting them as re-ranking or refinement modules atop classical RS pipelines [155, 299, 422].

While these directions demonstrate the expressive power of LLMs, they remain fundamentally constrained by their *finite context window*, which prevents them from processing full-scale recommendation datasets. Consequently, LLMs can only reason over restricted candidate subsets and fail to capture the global interaction structures, such as co-consumption patterns, relational dependencies, or user communities, that are critical to situating individual preferences within the broader recommendation landscape. Furthermore, although promising, operating on restricted candidate subsets introduces substantial limitations, and when tasked with generation over large candidate sets, LLMs are often prone to producing hallucinated recommendations [299].

This limitation motivates a shift in perspective: *rethinking LLMs not as end-to-end recommenders, but as representation engines that enhance the modeling components of RS*. Rather than generating recommendations directly, LLMs can be employed to produce semantically rich, human-readable representations of users and items

that complement collaborative signals, moving towards LLM-supported RS architecture. However, encoding user histories as natural text within an LLM’s context treats each user in isolation, overlooking dataset-level regularities that tie them together.

To overcome this, we propose introducing *symbolic reasoning* as a complementary mechanism to encode dataset-specific relational knowledge and anchor LLM-derived representations in the underlying data distribution. In particular, *First-Order Logic (FOL)* offers a compact and interpretable formalism to describe behavioral regularities through logical implications. Rule-mining frameworks such as AMIE [118] can automatically extract *Horn rules* from knowledge graphs derived from user–item interactions, e.g.,

$$watched(u, x) \wedge genre(x, sci\ fi) \rightarrow prefers(u, sci\ fi),$$

where the body captures observed predicates and the head encodes the inferred relation. Such rules distill frequent co-occurrence and relational patterns into high-level abstractions that LLMs alone cannot observe, providing a symbolic summary of the dataset’s structure.

Although previous work has demonstrated that symbolic rules improve reasoning and interpretability in RS [64, 332, 389], their integration with LLMs remains largely unexplored. We hypothesize that combining these rules with LLM-derived semantics can situate user–item profiles within the broader relational space of the dataset, leading to more coherent, dataset-aware representations.

To test this hypothesis, we present **FOLMind**, a framework that integrates Knowledge Graphs, rule mining, and LLMs to construct enriched user and item profiles. FOLMind builds Knowledge Graphs from user–item interactions, extracts interpretable FOL rules via AMIE, and employs LLMs to generate narrative-style textual profiles grounded in both symbolic and distributional knowledge. These profiles are embedded into dense vectors and integrated into RS pipelines through two complementary modalities: (i) *semantic initialization* of neural recommenders, and (ii) a *hybrid refinement* mechanism that fuses collaborative and semantic signals.

Comprehensive experiments across four benchmark datasets demonstrate that narrative-style profiles improve recommendation accuracy across domains and that integrating symbolic FOL rules yields further gains in specific datasets.

Our main contributions are as follows:

- (1) We identify and formalize the limitations of current LLM-based recommenders, motivating the shift from LLM-centered to LLM-supported RS.
- (2) We introduce **FOLMind**, a framework that unifies Knowledge Graphs, rule mining, and LLMs to generate semantically enriched user and item representations.
- (3) We demonstrate that combining symbolic reasoning with LLM semantics can enhance both accuracy and interpretability in RS.
- (4) We perform extensive experiments across multiple domains to validate the robustness and generality of the framework.
- (5) We release all code, datasets, and trained models to foster reproducibility and future research.

10.2 Related Work

10.2.1 Neuro-Symbolic Recommender Systems

Over the years, research on Recommender Systems (RSs) has primarily focused on leveraging explicit and implicit user feedback to learn dense user and item embeddings. While these embedding-based approaches have achieved strong predictive performance, they often struggle to capture rich semantic relations among entities [14]. To address this limitation, Knowledge Graphs (KGs) emerged as a natural complement, providing structured relational knowledge that can enrich representations and guide reasoning. However, early KG-based RSs primarily relied on embedding propagation or meta-path design, which improved accuracy but did not fully leverage the symbolic reasoning potential inherent in KGs [377, 412].

Recognizing this gap, recent work has moved beyond treating KGs merely as sources of additional embeddings and instead seeks to harness their symbolic reasoning capacity more directly. Neuro-symbolic (NeSy) recommenders embody this shift by explicitly integrating symbolic knowledge with neural architectures, thereby combining the robustness of learned embeddings with the expressive power of logic. An example is RuleRec [235], which demonstrated that mining and optimizing rules from KGs improves recommendation accuracy. This insight inspired path- and rule-based approaches such as NSER [395] and CAFE [396], where recommendations are grounded in reasoning chains that reflect KG semantics.

Subsequent work has pursued tighter integration between logic and neural mod-

els. For example, constraint-based methods embed KG triples and First-Order Logic (FOL) rules in a shared space using fuzzy logic (e.g., KALE [136]), while Logic Tensor Networks enforce FOL axioms in collaborative filtering frameworks [60]. Other approaches seek to neuralize logical operators, with models such as NLQ4Rec [389] and KG-LRR [369] approximating conjunctions and disjunctions via neural modules. Moreover another line of research explores hybrid pipelines that combine rule mining with neural optimization, where systems like AMIE [118] extract high-confidence rules that can be injected into probabilistic or embedding-based models, as in LOGER [436].

Recent surveys further confirm that symbolic rules, provide compact summaries of dataset regularities, enhance interpretability, and strengthen downstream reasoning tasks such as recommendation [71, 387]. In particular, mature rule miners such as AMIE and AnyBURL can scale to extract concise, dataset-specific patterns [117, 118, 249], which can subsequently be injected into RS models as constraints, features, or reasoning signals [135]. Building on this foundation, we view FOL rules as structured dataset summaries that complement LLM knowledge, capturing global interaction patterns that may elude context-limited LLM representations and making explicit the dataset-specific regularities.

10.2.2 Large Language Models for Recommendation

Large Language Models (LLMs) offer strong semantic and reasoning abilities [138] and building on this potential, research has advanced along three main directions.

The first focuses on *stand-alone recommenders*, where LLMs are directly adapted for recommendation tasks by leveraging prompting and framing recommendation as a zero-shot, few-shot, or chain-of-thought problem. For instance, LLM-Rec [232] unifies user history and candidate items within a single prompt, LLaRA [214] extends this approach through a novel prompt design, and PNR-LLM [182] introduces hierarchical prompting tailored for personalized news recommendation.

The second direction examines *Conversational Recommender Systems (CRS)*, where LLMs are employed to enable natural language interactions. For example, Wang et al. [371] study the use of ChatGPT for conversational recommendation, highlighting limitations in current evaluation protocols. Along the same line, Liu et al. [227] explore the integration of LLMs and CRS in e-commerce dialogues, while Yang et al. [407] propose ReFICR, an LLM-enhanced conversational recommender that

leverages a retrievable large language model to improve conversational recommendation.

The third direction investigates *re-ranking*, where LLMs are applied on top of a base recommender to reorder candidate lists. For instance, Hou et al. [155] analyze the ability of LLMs to serve as re-ranking models for recommender systems, Gao et al. [122] extend this approach by incorporating objectives such as diversity and fairness, and Zhang et al. [422] propose a multi-layer ranking framework that employs LLMs to further enhance recommendation performance.

Despite these advances, two key limitations persist: (i) LLMs cannot process catalog-scale datasets due to restricted context windows, and (ii) when generating over large candidate sets, they are prone to hallucinated recommendations [299]. These challenges have motivated a shift from treating LLMs as direct recommenders to employing them as *representation learners*. Prior studies have explored this direction, for example by using LLMs to generate textual descriptors of users and items for integration into recommender pipelines [67, 299].

The present work advances this line by introducing a key distinction: instead of deriving LLM-based profiles solely from user history, they are enriched with First-Order Logic rules mined from interaction graphs. This integration enables the resulting profiles to capture dataset-specific interaction patterns while remaining fully compatible with classical RS evaluation protocols. Furthermore, two strategies are proposed for leveraging the enriched representations: (i) semantic initialization of a neural recommender, and (ii) a refinement mechanism that fuses collaborative and semantic signals. Together, these approaches illustrate how LLM-based representations can effectively enhance classical recommender systems.

10.3 Problem Formulation

Let $\mathcal{U} = \{u_1, \dots, u_{|\mathcal{U}|}\}$ be the set of users and $\mathcal{I} = \{i_1, \dots, i_{|\mathcal{I}|}\}$ the set of items. The interaction history is defined as $\mathcal{H} \subseteq \mathcal{U} \times \mathcal{I}$, where $(u, i) \in \mathcal{H}$ indicates that user u has interacted with item i .

Large Language Models (LLMs) can serve as representation engines by encoding textualized *user histories* or *item descriptions*, i.e., $\phi_{\text{LLM}} : \mathcal{T} \rightarrow \mathbb{R}^d$, where $\mathcal{T}$ denotes the space of natural language sequences and $\mathbb{R}^d$ represents the embedding space. Despite advances that yield extended context windows capable of encoding long histories, two challenges persist: (i) *User isolation*: each embedding $\phi_{\text{LLM}}(\mathcal{T}_u)$

reflects a single user u based only on that user’s history, without situating the representation within the global distribution of $\mathcal{H}$; (ii) *Lack of dataset-level semantics*: embeddings derived from $\phi_{\text{LLM}}(\mathcal{T}_u)$ do not inherently capture relational patterns (e.g., co-consumption and structural dependencies) that emerge from dataset-level interactions. As a result, LLM-derived representations may be semantically rich at the individual level yet remain disconnected from the global semantics that classical recommender models exploit.

To address these limitations, we propose to incorporate symbolic reasoning in the form of First-Order Logic (FOL) rules, mined from a knowledge graph $\mathcal{G}$ constructed from $\mathcal{H}$. Each rule $r \in \mathcal{R}$ has Horn-clause form $p_1(x_1, \dots, x_m) \wedge \dots \wedge p_k(x_1, \dots, x_m) \Rightarrow q(x_1, \dots, x_m)$, capturing dataset-level regularities such as

$$\text{interacts}(u, i) \wedge \text{has\_attribute}(i, a) \Rightarrow \text{prefers}(u, a).$$

Given a user $u \in \mathcal{U}$, a narrative-style textual profile is constructed as $T_u = \text{Prompt}(\mathcal{H}(u), \mathcal{R}(u))$, where $\mathcal{H}(u) = \{i \mid (u, i) \in \mathcal{H}\}$ is the interaction history of u , $\mathcal{R}(u) \subseteq \mathcal{R}$ are the rules applicable to u , and $\text{Prompt}(\cdot)$ produces a natural-language summary that integrates both sources of information. Analogously, for each item $i \in \mathcal{I}$, a narrative-style profile is defined as $T_i = \text{Prompt}(\mathcal{M}(i), \mathcal{R}(i))$, where $\mathcal{M}(i)$ denotes the metadata of item i and $\mathcal{R}(i) \subseteq \mathcal{R}$ are the associated symbolic rules.

Once constructed, these textual profiles are encoded into dense representations using a sentence-embedding model. Specifically, the enriched user embedding is obtained as $\phi(u) = \phi_{\text{SE}}(T_u) \in \mathbb{R}^d$, and, analogously, item embeddings are defined as $\phi(i) = \phi_{\text{SE}}(T_i) \in \mathbb{R}^d$.

10.3.1 The FOLMind Framework

The proposed framework is organized as a four-stage pipeline. First, a knowledge graph is constructed by integrating information from external knowledge bases. Second, symbolic rules are mined from the graph to capture salient relational patterns. Third, narrative-style profiles are generated by combining user data with the extracted rules in natural language form. Finally, these profiles are encoded into dense embeddings to support downstream tasks.

Stage 1: Knowledge Graph Construction and Semantic Enrichment. The first stage involves constructing a comprehensive knowledge graph (KG) for the target recommendation dataset. This process comprises two main steps: (i) *Data Acquisition and Preprocessing*, where the dataset is collected, cleaned, and normalized to ensure consistency; and (ii) *Semantic Enrichment*, where the dataset is augmented with structured information from external knowledge bases (e.g., DBpedia). In this step, SPARQL queries are used to retrieve relevant metadata and attributes associated with items and users, thereby linking dataset entities to real-world counterparts and enabling multi-hop entity retrieval. The output is a unified KG represented as RDF triples in the form $\langle \text{subject} \rangle \langle \text{predicate} \rangle \langle \text{object} \rangle$, encompassing users, items, and their interactions.

Stage 2: Symbolic Rule Extraction. The second stage applies symbolic rule mining to the knowledge graph constructed in Stage 1 in order to uncover general behavioral patterns. This component extracts interpretable First-Order Logic (FOL) rules that capture statistically significant relationships across the user base. For example, a mined rule may take the form:

$$\left(\text{interacts}(u, i_1) \wedge \text{has\_attribute}(i_1, a) \wedge \text{has\_attribute}(i_2, a) \right) \Rightarrow \text{interacts}(u, i_2)$$

Such a rule states that if a user interacts with an item i_1 that shares an attribute a with another item i_2 , the user is likely to also interact with i_2 .

Stage 3: Narrative-style Profile Generation via LLM. In the third stage, a Large Language Model (LLM) is employed as a reasoning component to generate narrative-style profiles. This process is guided by carefully designed prompts that integrate user interaction histories or item metadata with the symbolic rules mined in Stage 2. The resulting profiles provide natural-language descriptions that contextualize individual behaviors and preferences within the broader rule-based patterns extracted from the knowledge graph.

Prompt Design. Two types of prompts were engineered, one for user profiles and one for item profiles:

- **User Profiles.** The prompt integrates four components: (i) the user's history of consumed items (*what* they interacted with), (ii) contextual metadata of those

items (*attributes* such as categories or features), (iii) user metadata (*who* the user is, e.g., demographics or profile attributes), and (iv) general FOL rules (*logic* underlying recurring preferences). The LLM is instructed to output:

- (a) a concise narrative profile describing the user’s preferences, explicitly mentioning concrete attributes when available, and
- (b) a reasoning field that justifies preferences by citing interaction patterns, item attributes, and FOL rule applications.

Instruction templates and an example of a generated profile are provided in the Supplementary Material (Template 10.7, Example 10.7).

- **Item Profiles.** The prompt combines item metadata from the KG with general FOL rules to infer its target audience. The LLM is instructed to output:

- (a) a narrative profile describing the item’s appeal and potential mismatches, and
- (b) a reasoning field highlighting dominant attributes, rule-based justifications, and anti-patterns derived from inverse rules.

Instruction templates and an example of a generated profile are provided in the Supplementary Material (Template 10.7, Example 10.7).

The resulting *user profiles* are textual descriptions that explain preferences, hypothesize the reasoning behind choices, and identify latent interests. The *item profiles* articulate an item’s ideal audience and clarify the symbolic logic underpinning its appeal.

Stage 4: Semantic Embedding Generation. The final stage maps the narrative-style profiles generated in Stage 3 into dense numerical representations. Formally, let $\phi_{SE} : \mathcal{T} \rightarrow \mathbb{R}^d$ denote a pre-trained sentence-embedding function, where $\mathcal{T}$ is the space of textual sequences. Both user profiles P_u and item profiles P_i are encoded into vector embeddings as follows:

$$e_u = \phi_{SE}(P_u), \quad \forall u \in U, \quad e_i = \phi_{SE}(P_i), \quad \forall i \in I$$

The resulting user embeddings $\{e_u \mid u \in \mathcal{U}\}$ and item embeddings $\{e_i \mid i \in \mathcal{I}\}$ form a shared continuous vector space. This unified representation serves as the foundation for the proposed hybrid recommendation models.

10.3.2 Hybrid Recommendation Strategies

10.3.2.1 Space Initialization (Spacelnit)

Traditional neural recommendation models typically initialize user and item embedding layers with random values, relying solely on interaction data to shape these representations. However, random initialization may slow convergence and lead to suboptimal embeddings [131, 143, 147].

In this approach, the embedding layers are initialized with pre-computed semantic vectors, denoted as e_u for users and e_i for items. This follows the hypothesis that semantic embeddings provide the model with an initial representation space that already encodes user and item characteristics derived from auxiliary information. During training, these embeddings are fine-tuned jointly with other model parameters, allowing collaborative signals from observed interactions to refine and adapt the semantic representations.

10.3.2.2 Hybrid Refinement

The hybrid refinement module combines pre-trained embeddings derived from LLM-based narrative profiles with classical collaborative baselines, with the goal of integrating global behavioral patterns captured by collaborative filtering with the semantic representations learned by the LLM.

The procedure unfolds in two phases: (i) pre-computation of a semantic similarity matrix from the ϕ_{SE} embedding space, and (ii) hybridization with collaborative signals. In the second phase, two modalities are proposed: (a) *Inject*, where fusion is performed at the matrix level, and (b) *Blend*, where fusion is performed at the score level.

(i) Semantic Similarity Pre-computation. Let $\{e_u \in \mathbb{R}^d \mid u \in \mathcal{U}\}$ and $\{e_i \in \mathbb{R}^d \mid i \in \mathcal{I}\}$ denote the user and item embeddings generated in Stage 4. The semantic similarity between two entities of the same type (e.g., users $u, v \in \mathcal{U}$) is computed as

$$S_{\text{emb}}(u, v) = \hat{e}_u \cdot \hat{e}_v^\top, \quad \text{where} \quad \hat{e} = \frac{e}{\|e\|_2}$$

where each vector $\hat{e}$ is L2-normalized to ensure scale invariance.

The resulting matrix $S_{\text{emb}} \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{U}|}$ (or analogously $\mathbb{R}^{|\mathcal{I}| \times |\mathcal{I}|}$ for items) represents pairwise semantic affinities induced by the symbolic-semantic profiles.

(ii) Hybridization with collaborative signals.

(a) *Inject Mode (Matrix-level Fusion)*. Let S_{base} denote the similarity matrix obtained from a collaborative baseline model. For each entity $a \in \mathcal{E}$, let $N_k(a)$ represent the Top- k nearest neighbors under S_{base} .

Starting from the pre-computed semantic similarity matrix S_{emb} , we construct a filtered version S'_{emb} by retaining only the neighbors in $N_k(a)$. This ensures that semantic information is injected exclusively within the local neighborhoods defined by S_{base} .

The hybrid similarity matrix is then defined as:

$$S_{\text{new}} = S_{\text{base}} + \alpha \cdot S'_{\text{emb}},$$

where $\alpha \in [0, 1]$ controls the contribution of the semantic signal.

Next, S_{new} is pruned to the Top- k neighbors, consistent with the baseline configuration:

$$S_{\text{new}}^{(k)} = \text{Top-k}(S_{\text{new}}),$$

Predictions are obtained by propagating interactions through the hybrid structure:

$$\hat{R} = S_{\text{new}}^{(k)} R,$$

where R is the user-item interaction matrix.

Remark. This injection mechanism performs fusion before prediction, directly in the similarity space, and can be applied to any model that computes a similarity matrix.

(b) *Blend Mode (Score-level Fusion)*. Let $\hat{R}_{\text{base}} \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{I}|}$ denote the collaborative prediction matrix obtained from a fully trained baseline model, and let R be the user-item interaction matrix.

Given the pre-computed semantic similarity matrix S_{emb} , where each row corresponds to an entity and each entry represents its similarity to other entities, a pruning phase is applied in which only the k_{sim} largest positive similarity values per row are retained, with all remaining entries set to zero. After pruning, each row is normalized so that the retained weights sum to one, ensuring comparability of neighbor contributions while reducing noise.

Table 10.1 Dataset statistics before and after preprocessing.

Dataset	Raw Data				Processed Data			
	Users	Items	Inter.	Spars.	Users	Items	Inter.	Spars.
MovieLens Small	610	9.724	100.836	98.3%	609	2.269	81.109	94.1%
MovieLens 1M	6.040	3.706	1.000.209	95.5%	6.040	3.260	998.539	94.9%
Last.fm	1.876	17.632	92.834	99.7%	1.797	1.507	62.376	97.7%
Book Genome	350.332	9.374	5.152.656	99.8%	3.539	6.753	185.442	99.2%

Formally, the embedding weight matrix is defined as:

$$W_{\text{emb}} = \text{RowNorm}(\text{Top-}k_+(S_{\text{emb}}, k_{\text{sim}})).$$

where $\text{Top-}k_+$ selects the k_{sim} closest positive neighbors per row, and RowNorm enforces row-wise normalization.

Semantic predictions derived from the LLM-based narrative profiles are then obtained by propagating interactions:

$$\widehat{R}_{\text{sem}} = W_{\text{emb}} R.$$

The final prediction is computed as a convex combination of the collaborative and LLM-based components:

$$\widehat{R}_{\text{final}} = \alpha \cdot \widehat{R}_{\text{base}} + (1 - \alpha) \cdot \widehat{R}_{\text{sem}}, \quad (10.1)$$

where $\alpha \in [0, 1]$ controls the relative contribution of each component. Setting $\alpha = 1$ recovers the pure collaborative baseline, while $\alpha = 0$ yields a purely LLM-based $k\text{NN}$ predictor.

Remarks. The Blend mechanism performs fusion after prediction, directly in the score space, and can be applied to any model that produces user–item score matrices.

10.4 Experimental Setup

Datasets. To evaluate the FOLMind framework and the two proposed hybrid approaches, four widely used recommender system benchmarks are employed: *Movie-*

*Lens Latest Small*², *MovieLens 1M*³ *Last.fm*⁴, and *Book Genome*⁵.

For each dataset, interactions are split into 80% training and 20% test; from the training portion, 20% is further held out as a validation set for hyperparameter tuning. To improve data quality, k -core filtering is applied across all datasets, retaining only users and items with at least 10 interactions [240]. For Last.fm, since the feedback is implicit (i.e., listening counts), we normalize it to a [1–1000] scale using min–max scaling [192]. For Book Genome, given the large user base and prohibitive cost of processing all interactions, we construct a stratified subsample of user activity following established methodologies [283, 311]. The final dataset statistics are reported in Table 10.1.

Metrics. Model performance is evaluated in a top- N recommendation setting with $N = 10$. Three standard ranking metrics are adopted, nDCG, Recall, and Precision, to assess the quality of the recommendations. All metrics are computed on a per-user basis and averaged across the test set.

Baselines and Hybrid Strategies. Since FOLMind introduces hybrid strategies built on existing methods depending on the architectural variant, the selection of baselines follows the corresponding approach. For *Space Initialization*, we leverage NeuMF, a neural architecture that extends matrix factorization with a multi-layer perceptron to capture non-linear interaction patterns [147], by warm-starting it with FOLMind embeddings. For *Inject*, which integrates collaborative and semantic signals at the similarity-matrix level, we apply it to UserKNN [46], a user-based neighborhood model that recommends items preferred by similar users, and ItemKNN [316], an item-based neighborhood model that recommends items similar to those already consumed. For *Blend*, which fuses collaborative and semantic predictions at the score level, we apply it to Matrix Factorization (MF), a latent factor model that decomposes the user–item interaction matrix into low-dimensional embeddings (using the implementation of Rendle et al. [301]), to UserKNN and ItemKNN.

Implementation Details. Experiments were run on a server with an Intel Xeon w7-3455 (24 cores), 125 GB RAM, and an NVIDIA RTX A6000 GPU (48 GB VRAM), using CUDA 12.4.

2. <https://grouplens.org/datasets/movielens/latest/>

3. <https://grouplens.org/datasets/movielens/1m/>

4. <https://grouplens.org/datasets/hetrec-2011/>

5. <https://grouplens.org/datasets/book-genome/>

Table 10.2 Statistics of AMIE-mined rules from Knowledge Graphs.
 #Rules: total rules; Avg. Len.: average body atoms; Rel.: unique relations;
 Max Len.: maximum body length.

Dataset	#Rules	Avg. Len.	Rel.	Max Len.
MovieLens Small	705	2.96	11	3
MovieLens 1M	808	2.96	14	3
Last.fm	19	2.95	5	3
Book Genome	4	3.0	5	3

Knowledge-graph construction and querying used RDFLib ⁶ and SPARQL ⁷, with DBpedia as the external source [276, 376]. Narrative profiles were generated with Gemini 2.0 Flash Lite⁸ using the instruction template in Supplementary Material 10.7, and embeddings were obtained from Linq Mistral 7B via Hugging Face Transformers. Recommendation models and evaluation pipelines were implemented with the Elliot framework [17]. Hyperparameters were tuned with Optuna [10], for nDCG@10 on the validation set. All code, configurations, and prompts are publicly available ⁹.

Knowledge Graph Construction and Rule Mining. For each dataset, DBpedia and SPARQL are used to enrich the data with external metadata and construct a domain-specific Knowledge Graph (KG) encoding user–item interactions together with relational, contextual, and attributive information. AMIE [118] is then applied with the same configuration across datasets to automatically extract rules capturing regularities and co-occurrence patterns in the KG. The full parameter settings are reported in Supplementary Material 10.7, and rule statistics are shown in Table 10.2. Across datasets, MovieLens Small and MovieLens 1M yield substantially more rules than Last.fm and Book Genome. The number of unique relations reflects the structural composition of each graph, while the maximum rule length indicates the presence of multi-hop dependencies. To study the effect of rule quantity, AMIE parameters were not adjusted for Book Genome and Last.fm, allowing us to analyze the impact of smaller rule sets on the baselines.

6. <https://rdflib.readthedocs.io/en/stable/>

7. <https://github.com/RDFLib/sparqlwrapper>

8. <https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash-lite>

9. <https://github.com/sisinflab/FOLMind>

Table 10.3 Performance comparison of baseline models versus their LINO MISTRAL embedding-enhanced counterparts across four domains. Values are reported as (nDCG, Recall, Precision). Green indicates improvement, red indicates degradation.

Model	Strategy	MovieLens Small			MovieLens 1M			Last.FM			Book Genome		
		nDCG	Recall	Precision	nDCG	Rec	Prec	nDCG	Rec	Prec	nDCG	Rec	Prec
UserKNN	Baseline	0.2621	0.1408	0.2221	0.3104	0.1380	0.2768	0.2089	0.2123	0.1541	0.1077	0.0952	0.0690
	Hybrid-Blend	0.2656	0.1437	0.2249	0.3118	0.1399	0.2782	0.2000	0.2022	0.1493	0.1048	0.0929	0.0672
	Hybrid-Inject	0.2665	0.1456	0.2264	0.3114	0.1399	0.2780	0.1943	0.1968	0.1423	0.1075	0.0952	0.0678
	Δ on the Best	+1.68% $\uparrow$	+3.41% $\uparrow$	+1.94% $\uparrow$	+0.45% $\uparrow$	+1.38% $\uparrow$	+0.51% $\uparrow$	-4.26% $\downarrow$	-4.76% $\downarrow$	-3.11% $\downarrow$	-0.19% $\downarrow$	0.00%	-1.74% $\downarrow$
ItemKNN	Baseline	0.2307	0.1300	0.1975	0.2687	0.1092	0.2388	0.2325	0.2324	0.1646	0.1056	0.0978	0.0626
	Hybrid-Blend	0.2402	0.1338	0.2054	0.2741	0.1112	0.2436	0.2422	0.2416	0.1695	0.1089	0.1006	0.0641
	Hybrid-Inject	0.2376	0.1353	0.2020	0.2715	0.1107	0.2415	0.2339	0.2345	0.1660	0.1065	0.0989	0.0628
	Δ on the Best	+4.12% $\uparrow$	+4.08% $\uparrow$	+4.00% $\uparrow$	+2.01% $\uparrow$	+1.83% $\uparrow$	+2.01% $\uparrow$	+4.17% $\uparrow$	+3.96% $\uparrow$	+2.98% $\uparrow$	+3.12% $\uparrow$	+2.86% $\uparrow$	+2.40% $\uparrow$
MF	Baseline	0.2647	0.1430	0.2287	0.2864	0.1253	0.2587	0.2397	0.2425	0.1709	0.0613	0.0496	0.0434
	Hybrid-Blend	0.2776	0.1531	0.2379	0.3218	0.1396	0.2887	0.2009	0.2027	0.1505	0.0753	0.0674	0.0519
	Δ on the Best	+4.87% $\uparrow$	+7.06% $\uparrow$	+4.02% $\uparrow$	+12.36% $\uparrow$	+11.41% $\uparrow$	+11.60% $\uparrow$	-16.19% $\downarrow$	-16.41% $\downarrow$	-11.94% $\downarrow$	+22.84% $\uparrow$	+35.89% $\uparrow$	+19.59% $\uparrow$
	Baseline	0.2332	0.1278	0.2041	0.3009	0.1279	0.2725	0.2147	0.2252	0.1441	0.0722	0.0702	0.0490
NeuMF	Spacelinit	0.2704	0.1461	0.2308	0.2868	0.1166	0.2618	0.2118	0.2304	0.1443	0.0853	0.0796	0.0588
	Δ on the Best	+15.95% $\uparrow$	+14.32% $\uparrow$	+13.08% $\uparrow$	-4.69% $\downarrow$	-8.84% $\downarrow$	-3.93% $\downarrow$	-1.35% $\downarrow$	+2.31% $\uparrow$	+0.14% $\uparrow$	+18.14% $\uparrow$	+13.39% $\uparrow$	+20.00% $\uparrow$

10.5 Results and Discussion

This section presents the empirical findings of our study, structured around the main research question: *Can First-Order Logic rules enhance LLM user representations for recommendation?*

To provide a systematic analysis, we organize the results according to three sub-questions:

- (RQ1) What is the impact of FOLMind on collaborative filtering baselines?
- (RQ2) How does the number of mined rules affect recommendation performance?
- (RQ3) Which integration strategy (Space Initialization, Inject, Blend) yields the most consistent improvements?

The following subsections discuss the outcomes across the four datasets, highlighting trends and dataset-specific behaviors. Comprehensive results are reported in Table 10.3.

RQ1 – Impact of FOLMind on collaborative filtering baselines. For UserKNN, hybridization yields modest but consistent gains on *MovieLens Small* (+1.68% nDCG, +3.41% Recall, +1.94% Precision) and *MovieLens 1M* (+0.45% nDCG, +1.38% Recall, +0.51% Precision). By contrast, performance drops on *Last.FM* (−4.26% nDCG, −4.76% Recall, −3.11% Precision) and remains almost unchanged on *Book Genome*. ItemKNN shows more stable improvements, especially with Hybrid-Blend, where Recall rises by +4.08% on *MovieLens Small*, +1.83% on *MovieLens 1M*, +3.96% on *Last.FM*, and +2.86% on *Book Genome*. Gains in nDCG and Precision follow the same pattern. Matrix Factorization (MF) benefits the most from hybridization. On *MovieLens Small* it achieves +7.06% Recall, while on *MovieLens 1M* improvements are even larger (+12.36% nDCG, +11.41% Recall, +11.60% Precision). The strongest effects occur on *Book Genome* (+22.84% nDCG, +35.89% Recall, +19.59% Precision). However, these gains are offset by severe degradations on *Last.FM*, where all three metrics decline by more than 11%, pointing to domain-dependent sensitivity. NeuMF with Space Initialization produces the most polarized results. It excels on *MovieLens Small* (+15.95% nDCG, +14.32% Recall, +13.08% Precision) and on *Book Genome* (+18.14% nDCG, +13.39% Recall, +20.00% Precision). Yet, its performance deteriorates on *MovieLens 1M* (−4.69% nDCG, −8.84% Recall, −3.93% Precision) and weakens slightly on *Last.FM*, where only Recall shows a marginal gain (+2.31%).

Overall, hybridization proves most effective for ItemKNN and MF, consistently en-

hancing nDCG, Recall, and Precision, particularly on datasets like *MovieLens Small* and *Book Genome*. UserKNN gains remain limited and unstable performing good only in the movies domain, while NeuMF displays dataset-dependent behavior.

RQ2 – Effect of number of mined rules on recommendation performance.

Datasets enriched with larger rule sets, such as *MovieLens Small* (705 rules) and *MovieLens 1M* (808 rules), exhibit the most stable improvements. For example, MF with Hybrid-Blend achieves +7.06% Recall on *MovieLens Small* and +12.36% nDCG on *MovieLens 1M*, while NeuMF consistently improves on *MovieLens Small* (+13% across multiple metrics).

When rule coverage is limited, results become less predictable. On *Book Genome*, despite only four mined rules, the effect is highly positive (MF: +35.89% Recall), suggesting that even a small but semantically aligned set of rules can be impactful. Conversely, *Last.FM*, with 19 rules, shows the opposite trend: MF suffers notable degradations (−16.19% nDCG, −16.41% Recall), though ItemKNN records moderate improvements (+4.17% nDCG, +3.96% Recall).

In summary, a larger rule base generally relate with more consistent improvements, as evidenced in the *MovieLens* datasets.

RQ3 – Comparative effectiveness of the three integration strategies (Space Initialization, Inject, Blend). Among the strategies, Hybrid-Blend proves the most reliable, consistently boosting ItemKNN and MF across domains. For instance, MF with Blend achieves +12.36% nDCG and +11.41% Recall on *MovieLens 1M*, while ItemKNN with Blend improves by +4.12% nDCG and +4.08% Recall on *MovieLens Small*. Hybrid-Inject produces more volatile results. It brings moderate gains in specific cases (e.g., ItemKNN on *Last.FM*) but frequently lags behind Blend, particularly for UserKNN, where it causes degradations on *Last.FM* and *Book Genome*. Space Initialization, evaluated with NeuMF, delivers the largest individual gains (+15.95% nDCG on *MovieLens Small*), yet its effect is highly dataset-dependent, with substantial degradations on *MovieLens 1M*.

In summary, Hybrid-Blend emerges as the most consistent strategy, Hybrid-Inject offers occasional but less dependable benefits, and Space Initialization provides strong yet fragile improvements that depend heavily on dataset characteristics.

Why do degradations occur? Hypotheses grounded in the strategies. A closer look at the degradations reveals that they stem from the interaction between strat-

egy design and domain characteristics. Score-level fusion, as in Blend, proves inherently more robust because it perturbs only the final scores while leaving the learned neighborhood or latent structures intact, thereby limiting downside risk. By contrast, similarity-level fusion in Inject directly alters the neighborhood graph; when semantic neighbors diverge from behavioral ones, strong collaborative signals can be diluted, as seen with UserKNN on *Last.fm*. Space Initialization, in turn, highlights the double-edged nature of warm-start priors: imposing a semantic geometry at initialization can accelerate learning when well aligned with the domain (e.g., *Book Genome*), but in cases of misalignment or limited model adaptation it risks over-constraining optimization, leading to convergence in suboptimal spaces, as observed on *MovieLens 1M*. Rule coverage and domain fit also shape outcomes: sparse or poorly aligned rule sets offer limited guidance and can introduce noise, which is especially detrimental for models relying on precise local neighborhoods such as Inject. This can explain the contrasting results in *Last.fm*, where MF gains modestly while ItemKNN, which relies on coherent global structure, deteriorates.

Takeaways. (RQ1) FOLMind strengthens classical recommenders, most notably MF and ItemKNN, with the largest gains achieved by Hybrid-Blend and by NeuMF with Space-Initialization. (RQ2) Larger sets of rules enhance the conditioning of user representations, leading to more informative latent spaces. (RQ3) Overall, Hybrid-Blend proved the most robust, while Space Initialization yielded the highest performance improvements.

10.6 Ablation Studies

To better understand the source and generalizability of the observed performance gains, we conduct three ablation studies: (i) isolating the effect of First-Order Logic (FOL) rules from narrative-style profiles, (ii) analyzing the influence of embedding model size on representation effectiveness, and (iii) assessing the robustness of results across different embedding models.

Ablation 1 – FOL vs. NoFOL Narrative-Style Profiles. Table 10.4 compares baseline models using plain narrative-style embeddings (*NoFOL*) against their counterparts enriched with symbolic FOL signals (*+FOL*). The results indicate that the observed gains cannot be attributed solely to narrative-style representations. Notably, MF with Hybrid-Blend achieves a +10.73

However, the benefits are dataset-dependent. *MovieLens Small* and *Book Genome*

Table 10.4 Ablation study on FOL vs. NoFOL Linq Mistral embeddings for nDCG. The relative change (Δ) denotes the difference from NoFOL to FOL across all datasets. Positive Δ values indicate improvement, while negative values indicate degradation.

Model	Strategy	MovieLens Small			MovieLens 1M			Last.fm			Book Genome		
		NoFOL	+FOL	$\Delta\%$	NoFOL	+FOL	$\Delta\%$	NoFOL	+FOL	$\Delta\%$	NoFOL	+FOL	$\Delta\%$
UserKNN	Baseline	0.2621	-	-	0.3104	-	-	0.2089	-	-	0.1077	-	-
	Hybrid-Blend	0.2623	0.2656	+1.26% $\uparrow$	0.3118	0.3118	+0.00%	0.2006	0.2000	-0.30% $\downarrow$	0.1044	0.1048	+0.38% $\uparrow$
	Hybrid-Inject	0.2624	0.2665	+1.56% $\uparrow$	0.3110	0.3114	+0.13% $\uparrow$	0.1971	0.1943	-1.42% $\downarrow$	0.1066	0.1075	+0.84% $\uparrow$
ItemKNN	Baseline	0.2307	-	-	0.2687	-	-	0.2325	-	-	0.1056	-	-
	Hybrid-Blend	0.2287	0.2402	+5.04% $\uparrow$	0.2725	0.2741	+0.58% $\uparrow$	0.2398	0.2422	+1.00% $\uparrow$	0.1060	0.1089	+2.74% $\uparrow$
	Hybrid-Inject	0.2419	0.2376	-1.78% $\downarrow$	0.2721	0.2715	-0.22% $\downarrow$	0.2357	0.2339	-0.76% $\downarrow$	0.1063	0.1065	+0.19% $\uparrow$
MF	Baseline	0.2647	-	-	0.2864	-	-	0.2397	-	-	0.0613	-	-
	Hybrid-Blend	0.2507	0.2776	+10.73% $\uparrow$	0.3221	0.3218	-0.09% $\downarrow$	0.2134	0.2009	-5.85% $\downarrow$	0.0753	0.0753	+0.00%
NeuMF	Baseline	0.2332	-	-	0.3009	-	-	0.2147	-	-	0.1066	-	-
	SpacelInit	0.2532	0.2704	+6.78% $\uparrow$	0.3052	0.2868	-6.03% $\downarrow$	0.2327	0.2218	-4.68% $\downarrow$	0.0849	0.0853	+0.47% $\uparrow$

show the most consistent improvements, while *Last.FM* exhibits degradations (e.g., MF: -5.85%), suggesting that misaligned or noisy rules can hinder performance. Interestingly, *MovieLens 1M* yields nearly neutral effects across models, which we hypothesize may result from dataset leakage in the pretraining corpora of the underlying LLMs, potentially making FOL signals redundant [96].

Overall, these findings suggest that the interaction between symbolic FOL rules and narrative-style embeddings can yield complementary benefits, though their effectiveness remains sensitive to dataset characteristics and the alignment between symbolic and textual signals.

Ablation 2 – Effect of Embedding Model Size. Table 10.5 reports the performance of SpaceInit on NeuMF with Qwen embeddings of different scales (0.6B, 4B, 8B). In general, larger models provide stronger representations, but the trend is not strictly monotonic. On *MovieLens Small*, Qwen-8B delivers the highest gains, surpassing the baseline by +14.54% nDCG, +9.94% Recall, and +11.42% Precision. On *MovieLens 1M*, improvements are modest, with Qwen-4B performing best (+2.06% nDCG, +2.97% Recall). Interestingly, on *Last.fm*, the 4B variant outperforms both the smaller and larger models, achieving the best nDCG (0.2333) and Recall (0.2474), while Qwen-8B shows no additional benefit. In *Book Genome*, the effect of embeddings is strongest: Qwen-0.6B achieves the highest nDCG (+19.25%), whereas Qwen-8B yields the best Precision (+24.08%). These results indicate that the effectiveness of embedding size depends on dataset characteristics rather than model scale alone, suggesting that the benefits of FOLMind generalize across embedding sizes but require data-driven tuning to maximize effectiveness. Interestingly, unlike the Linq Mistral case (see Table 10.3), NeuMF with Qwen embeddings enhances all three metrics across Last.fm and MovieLens-1M, further supporting the hypothesis that stronger alignment between the pre-trained embedding model and the domain data drives more consistent improvements. Similar conclusion holds also for the other hybrid integration strategies (Hybrid Blend and Hybrid Inject) (see Supplementary Material Table 10.7).

Ablation 3 – Robustness of results across embedding model families. In this ablation, we compare the different strategies when enhanced with embeddings from three families: SBERT-MPNet (109M)¹⁰, EmbeddingGemma (300M)¹¹, and Arc-

10. <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

11. <https://huggingface.co/google/embeddinggemma-300m>

Table 10.5 Performance comparison of the SpaceInit strategy on NeuMF using Qwen embeddings of different scales (0.6B, 4B, 8B) across four datasets. Best results are in **bold**, and runner-ups are underlined.

Method	MovieLens Small			MovieLens 1M			Last.fm			Book Genome		
	nDCG	Recall	Prec.	nDCG	Recall	Prec.	nDCG	Recall	Prec.	nDCG	Recall	Prec.
Baseline	0.2332	0.1278	0.2041	0.3009	0.1279	0.2725	0.2147	0.2252	0.1441	0.0722	0.0702	0.0490
Qwen-0.6B	0.2312	0.1164	0.2034	0.3012	0.1279	0.2734	0.2308	0.2416	0.1516	0.0861	0.0791	0.0608
Qwen-4B	0.2537	0.1379	0.2202	0.3071	0.1317	0.2772	0.2333	0.2474	0.1545	0.0844	0.0781	0.0581
Qwen-8B	0.2671	0.1405	0.2274	0.2838	0.1168	0.2587	0.2217	0.2368	0.1481	<u>0.0845</u>	0.0778	0.0578
Δ on the Best	+14.54% ↑	+9.94% ↑	+11.42% ↑	+2.06% ↑	2.97% ↑	+1.72% ↑	+8.66% ↑	+9.86% ↑	+7.22% ↑	+19.25% ↑	+12.68% ↑	+24.08% ↑

tic Embed 2.0 (568M)¹². The full results are reported in Supplementary Material Table 10.6.

Results show that FOLMind is robust across different embedding sources, though relative effectiveness varies by model class. For latent factor models such as MF and NeuMF, SBERT-MPNET provides the most consistent improvements, achieving the highest gains on *MovieLens 1M* (MF: +12.5% nDCG, +11.7% Recall). In contrast, neighborhood-based methods (UserKNN, ItemKNN) benefit more from SNOWFLAKE, particularly under the Inject strategy (e.g., ItemKNN on *Last.fm*: +8.6% nDCG, +10.2% Recall). GEMMA embeddings remain competitive and occasionally surpass other families, as in NeuMF on *MovieLens 1M*.

These findings indicate that FOLMind generalizes well across embedding families, but the degree of improvement depends on the alignment between embedding properties and the underlying collaborative model. Latent factor recommenders appear better suited to semantic representations from SBERT-MPNET, while neighborhood-based methods profit from the denser similarity structure captured by SNOWFLAKE.

Summary. The ablations reveal that (i) performance gains are driven by the narrative-style user representations, while in some setups the integration of FOL rules further enhances these representations; (ii) embedding model size does not necessarily modulate the effectiveness of these signals, as larger models do not always yield stronger improvements; and (iii) FOLMind remains robust across different embedding families, although the magnitude of improvements depends on the alignment between the pre-trained embedding properties and the underlying collaborative model.

10.7 Conclusion and Limitations

This work introduced FOLMIND, a framework that integrates First-Order Logic (FOL) rules extracted from Knowledge Graphs with Large Language Models to generate enriched, narrative-style user and item representations. The framework proposes three complementary strategies for leveraging these representations within recommenders: Hybrid-Blend, Hybrid-Inject, and Space Initialization. Extensive experiments across four benchmark datasets show that FOLMIND can strengthen collaborative filtering baselines in several cases. Among the proposed

12. <https://huggingface.co/Snowflake/snowflake-arctic-embed-l-v2.0>

strategies, Hybrid-Blend emerges as the most robust and stable across domains, while Inject and Space Initialization offer complementary, domain-sensitive advantages. These findings confirm that combining symbolic reasoning with LLM-derived semantics can improve recommendation accuracy and offer dataset-aware representations of user preferences.

Supplementary Material

User and Item Profile Generation Prompts

Prompt for User Profile Generation

System: You are an inference engine that helps determine which types of film a specific user is likely to enjoy, in order to build concise user profiles.

[METADATA]

USER PROFILE DATA: {user_profile}

USER INTERACTIONS: {user_history}

RELATED MOVIE METADATA: {items_meta}

[General FOL Rules]

Format: (tab-separated elements, atoms and head are divided by '=>')

{rules}

Task: Analyze [METADATA] using this structured workflow to generate a narrative profile and reasoning.

[Output Format]:

Output EXACTLY ONE valid JSON object with ONLY these two keys:

```
{
  "profile": "<Continuous narrative (no longer than 100 words)
  covering:
    1. Core preferences with emotional resonance
    2. Style/thematic patterns
    3. Explicit dislike triggers - MUST INCLUDE SPECIFIC ACTOR/
    DIRECTOR NAMES WHEN PRESENT (e.g., 'frequently chooses films
    starring [Actor] or directed by [Director]')>",
  "reasoning": "<Concise justification structured as:
    1. Purchase/review patterns
```

2. Reference 2-3 specific examples with attributes
 3. FOL rule application explained in plain English
 4. Anti-patterns with dislike examples>"
- ```
}
```

### [CRITICAL FORMATTING RULES]

1. In PROFILE: Always specify actor/director names when available (e.g., "prefers films starring Tom Hanks").
2. In PROFILE: Never use vague terms like "familiar actors" - replace with actual names.
3. In REASONING: Explain FOL rules in plain English (e.g., "films sharing cinematographers tend to share visual styles").
4. Output MUST be exactly one JSON object - no multiple JSON objects.
5. NEVER include markdown code fences (````json or ````)
6. Escape double quotes in strings with backslash ("\")
7. Profile must start sentences with active verbs (e.g., "Prefers", "Avoids").

### Example of valid output:

```
{
 "profile": "Prefers action films with strong visual effects,
 particularly those starring Arnold Schwarzenegger. Avoids
 romantic dramas. Frequently chooses films directed by James
 Cameron.",

 "reasoning": "Liked Terminator 2 (starring Schwarzenegger,
 directed by Cameron) and Die Hard. Validated rule: Films sharing
 cinematographers tend to share visual styles. Supported by:
 Matrix (cinematography: Bill Pope) and Speed (cinematography:
 Andrzej Bartkowiak). Disliked Bridget Jones's Diary due to
 romantic elements"
}
```

### Prompt for Item Profile Generation

**System:** You are an inference engine that determines which user profiles would enjoy a specific movie based on its attributes and preference rules.

### [METADATA]

MOVIE META DATA: {items\_meta}

### [General FOL Rules]

Format: (tab-separated elements, atoms and head are divided by '=>')

```
{rules}
```

### [Output Format]:

Output EXACTLY ONE valid JSON object with ONLY these two keys:

```
{
 "profile": "<Continuous narrative (no longer than 200 words)
 covering:
 1. Core appeal with emotional resonance
 2. Style/thematic signatures
 3. Explicit mismatch warnings - MUST INCLUDE SPECIFIC DIRECTOR/
 ACTOR NAMES WHEN PRESENT (e.g., 'will appeal to fans of [Director
] or [Actor]')>",
 "reasoning": "<Concise justification structured as:
 1. Dominant attribute combinations
 2. Reference 2-3 specific attributes with effects
 3. FOL rule application explained in plain English
 4. Anti-patterns using rule inverses>"
}
```

### [CRITICAL FORMATTING RULES]

1. In PROFILE: Always specify director/actor names (e.g., "strong match for Christopher Nolan enthusiasts").
2. In PROFILE: Replace vague terms with concrete attributes (e.g., "complex narratives" not "certain styles").
3. In REASONING: Explain FOL rules in plain English (e.g., "Rule #3: Shared cinematographers indicate visual consistency").
4. Output MUST be exactly one JSON object - no multiple objects.
5. NEVER include markdown fences (``json).
6. Escape double quotes in strings with backslash ("").
7. Profile must start with active verbs (e.g., "Appeals to", "Avoid if").

### Example of valid output:

```
{
 "profile": "Appeals to fans of psychological thrillers with
 nonlinear narratives, particularly works directed by David
 Fincher. Resonates through moral ambiguity and high-tension
 pacing. Avoid if preferring lighthearted comedies or disliking
 violent themes.",
 "reasoning": "Dominant attributes: psychological_thriller +
 nonlinear_story + directed_by_David_Fincher. Validated Rule #2:
 Films sharing directors maintain thematic consistency (Fight Club,
 Gone Girl). Anti-pattern: Rule #5 inverse shows conflict with '
 lighthearted_comedies' based on tonal mismatch with films like
 Step Brothers."
}
```

## Example of Generated User and Item Profile

### Generated User Profile for user\_3459

**Profile:**

Enjoys action and adventure films with sci-fi elements as well as comedies. Frequently chooses films directed by Steven Spielberg. Avoids horror films and romantic comedies, specifically those like *You've Got Mail* or those starring Meg Ryan. Shows a strong preference for films starring Harrison Ford.

**Reasoning:**

The user consistently likes action-adventure and sci-fi films such as *Jurassic Park* (directed by Steven Spielberg) and *Raiders of the Lost Ark* (starring Harrison Ford, directed by Spielberg) and dislikes horror and romantic comedy films such as *Halloween* or *You've Got Mail*.

The FOL rules applied here focus on actors, directors, and genres. For example, if a user likes a film and another film shares a director or actor, it's likely they will enjoy the second film. For instance, they like *Raiders of the Lost Ark* and since *Star Wars: Episode VI – Return of the Jedi* shares the same actor (Harrison Ford), the system is designed to infer potential enjoyment of that film.

The anti-pattern observed is a dislike of films sharing attributes with disliked films, such as genres or directors. For example, since the user disliked *You've Got Mail*, which is a romantic comedy, the system would anticipate a dislike for similar movies.

### Generated Item Profile for *The Straight Story*

**Profile:**

Appeals to viewers who appreciate character-driven dramas, especially those with a road movie structure, offering a poignant exploration of family and old age. This film, directed by David Lynch, will resonate with fans of his unique blend of the mundane and the surreal, but with a more grounded, sentimental approach. Expect strong performances from Richard Farnsworth and Sissy Spacek. Avoid if you strongly dislike slow-paced narratives or are seeking fast-paced action.

**Reasoning:**

Dominant attributes: Drama, road\_movie, directed\_by\_David\_Lynch. Strong appeal derives from the director and the genre (Drama) match.

Rule #25: If a user likes a film directed by a certain director and another film is also directed by the same director, then the user will likely enjoy the second film (e.g., fans of David Lynch).

Rule #31: If a user enjoys a film directed by a director and starring a specific actor and another film is starring the same actor, the user will likely enjoy the second film.

Anti-pattern: Users who dislike action or fast-paced films will likely not enjoy this slow-burn drama.

## AMIE Rule Mining Parameters

We employed the AMIE framework [118] to automatically extract Horn rules from the constructed knowledge graphs. Unless otherwise noted, the same configuration was used across all datasets, summarized as follows:

- `maxad = 4`: maximum number of atoms per rule (3 body atoms + 1 head).
- `pm = "headcoverage"`: pruning metric.
- `mins = 1000`: minimum absolute support (positive examples).
- `minhc = 0.01`: minimum head coverage threshold.
- `min_support = 0.01`: minimum support threshold.
- `min_confidence = 0.001`: base confidence threshold.
- `minc = 0.0`: minimum standard confidence for rule generation.
- `verbose = true`: verbose output enabled.

These values were inspired by those reported in the AMIE literature and in prior work [332], although only a subset of parameters was explicitly documented for MovieLens 1M. For the other datasets, we adopted AMIE's default values and refined them empirically.

## Extended Performance Comparison with BERT, Gemma, Snowflake, and Qwen Embeddings across Four Datasets

**Table 10.6** Performance comparison across Baseline, SBERT-MPNET, GEMMA, and SNOWFLAKE variants. Rows within each model group are sorted ascending by nDCG on MovieLens Small. Best values are in **bold**, runner-ups are underlined.

| Model   | Strategy                    | MovieLens Small |               |               |               | MovieLens 1M  |               |               |               | LastFM        |               |               |               | Book Genome   |     |       |  |
|---------|-----------------------------|-----------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|-----|-------|--|
|         |                             | nDCG            | Rec           | Prec.         |               | nDCG          | Rec           | Prec.         |               | nDCG          | Rec           | Prec.         |               | nDCG          | Rec | Prec. |  |
| UserKNN | Baseline                    | 0.2621          | 0.1408        | 0.2221        | 0.2234        | 0.3104        | 0.1380        | 0.2768        | <b>0.2089</b> | <b>0.2123</b> | <b>0.1541</b> | <b>0.1077</b> | <b>0.0952</b> | <b>0.0690</b> |     |       |  |
|         | Hybrid-Inject (SBERT-MPNet) | 0.2612          | 0.1441        | 0.2234        | 0.2234        | 0.3109        | <u>0.1401</u> | 0.2774        | <u>0.2055</u> | 0.2115        | <u>0.1528</u> | 0.1044        | 0.0922        | 0.0661        |     |       |  |
|         | Hybrid-Inject (Gemma)       | 0.2630          | 0.1423        | 0.2231        | 0.2231        | 0.3095        | 0.1381        | 0.2760        | 0.1769        | 0.1810        | 0.1320        | 0.1063        | 0.0932        | 0.0675        |     |       |  |
|         | Hybrid-Inject (Snowflake)   | 0.2631          | 0.1428        | 0.2236        | 0.2236        | 0.3086        | 0.1380        | 0.2753        | 0.1859        | 0.1881        | 0.1355        | 0.1079        | 0.0946        | <u>0.0683</u> |     |       |  |
|         | Hybrid-Blend (Gemma)        | 0.2641          | 0.1438        | 0.2249        | 0.2249        | 0.3118        | 0.1399        | 0.2781        | 0.2001        | 0.2023        | 0.1493        | 0.1043        | 0.0923        | 0.0667        |     |       |  |
|         | Hybrid-Blend (Snowflake)    | 0.2648          | <b>0.1441</b> | 0.2241        | 0.2241        | 0.3118        | 0.1399        | 0.2781        | 0.1998        | 0.2018        | 0.1491        | 0.1044        | 0.0919        | 0.0667        |     |       |  |
|         | Hybrid-Blend (SBERT-MPNet)  | <b>0.2668</b>   | 0.1454        | <b>0.2257</b> | <b>0.2257</b> | <b>0.3118</b> | <b>0.1399</b> | <b>0.2782</b> | 0.2000        | 0.2022        | 0.1493        | 0.1055        | 0.0929        | 0.0672        |     |       |  |
| ItemKNN | Baseline                    | 0.2307          | 0.1300        | 0.1975        | 0.2687        | 0.2687        | 0.1092        | 0.2388        | 0.2335        | 0.2324        | 0.1646        | 0.1056        | 0.0978        | 0.0626        |     |       |  |
|         | Hybrid-Blend (SBERT-MPNet)  | 0.2273          | 0.1280        | 0.1989        | 0.2687        | 0.2687        | 0.1092        | 0.2387        | 0.2339        | 0.2341        | 0.1655        | 0.1076        | <b>0.0997</b> | <u>0.0633</u> |     |       |  |
|         | Hybrid-Blend (Gemma)        | 0.2279          | 0.1282        | 0.1997        | 0.2688        | 0.2688        | 0.1092        | 0.2387        | <u>0.2350</u> | <u>0.2343</u> | 0.1658        | 0.1064        | 0.0985        | 0.0627        |     |       |  |
|         | Hybrid-Blend (Snowflake)    | 0.2286          | 0.1288        | 0.2008        | 0.2687        | 0.2687        | 0.1092        | 0.2388        | 0.2325        | 0.2326        | 0.1648        | <b>0.1090</b> | 0.1004        | <b>0.0641</b> |     |       |  |
|         | Hybrid-Inject (SBERT-MPNet) | 0.2463          | 0.1370        | 0.2141        | <b>0.2783</b> | <b>0.1165</b> | <b>0.1165</b> | <b>0.2494</b> | 0.2330        | 0.2331        | 0.1650        | 0.1066        | 0.0989        | 0.0631        |     |       |  |
|         | Hybrid-Inject (Gemma)       | 0.2471          | 0.1360        | 0.2143        | 0.2700        | 0.1101        | 0.1101        | 0.2399        | <b>0.2342</b> | <b>0.2351</b> | <b>0.1665</b> | 0.1058        | 0.0978        | 0.0625        |     |       |  |
|         | Hybrid-Inject (Snowflake)   | <b>0.2483</b>   | <b>0.1358</b> | <b>0.2149</b> | <u>0.2779</u> | <u>0.1162</u> | <u>0.1162</u> | 0.2488        | 0.2329        | 0.2335        | 0.1652        | 0.1033        | 0.0956        | 0.0614        |     |       |  |
| MF      | Baseline                    | 0.2647          | 0.1430        | 0.2287        | <u>0.2864</u> | 0.2864        | 0.1253        | 0.2587        | <b>0.2397</b> | <b>0.2425</b> | <b>0.1709</b> | 0.0613        | 0.0496        | 0.0434        |     |       |  |
|         | Hybrid-Blend (Snowflake)    | 0.2362          | 0.1279        | 0.2046        | 0.3214        | 0.3214        | 0.1391        | 0.2881        | 0.2006        | 0.2019        | 0.1499        | 0.0751        | 0.0672        | 0.0514        |     |       |  |
|         | Hybrid-Blend (Gemma)        | 0.2370          | 0.1290        | 0.2049        | <b>0.3215</b> | <b>0.1396</b> | <b>0.1396</b> | 0.2884        | 0.2007        | 0.2027        | 0.1505        | <b>0.0789</b> | <b>0.0711</b> | <b>0.0538</b> |     |       |  |
|         | Hybrid-Blend (SBERT-MPNet)  | <b>0.2776</b>   | <b>0.1531</b> | <b>0.2379</b> | <b>0.3210</b> | <u>0.1392</u> | <u>0.1392</u> | <b>0.2887</b> | 0.1992        | 0.2009        | 0.1488        | 0.0779        | 0.0703        | <u>0.0533</u> |     |       |  |
| NeuMF   | Baseline                    | 0.2332          | 0.1278        | 0.2041        | 0.3009        | 0.3009        | 0.1279        | 0.2725        | 0.2147        | 0.2252        | 0.1441        | 0.0722        | 0.0702        | 0.0490        |     |       |  |
|         | (Snowflake)                 | 0.2558          | 0.1361        | 0.2246        | 0.3301        | 0.3301        | 0.1427        | 0.2975        | <b>0.2530</b> | <b>0.2633</b> | <b>0.1642</b> | 0.0970        | 0.0902        | 0.0658        |     |       |  |
|         | Spacelut (SBERT-MPNet)      | 0.2595          | 0.1348        | 0.2266        | 0.3280        | 0.3280        | 0.1416        | 0.2960        | 0.2386        | 0.2519        | 0.1588        | <b>0.0987</b> | <b>0.0904</b> | <b>0.0676</b> |     |       |  |
|         | (Gemma)                     | <b>0.2626</b>   | <b>0.1374</b> | <b>0.2266</b> | <b>0.3319</b> | <b>0.1432</b> | <b>0.1432</b> | <b>0.2999</b> | <u>0.2430</u> | <u>0.2541</u> | <u>0.1594</u> | 0.0940        | 0.0869        | <u>0.0636</u> |     |       |  |

**Table 10.7** Performance comparison of LLM-based integration strategies (Blend, Inject) using QWEN embeddings of different scales (0.6B, 4B, 8B) across four benchmark datasets. For each column, the best result within each model group and dataset is shown in **bold**, and the second-best (runner-up) is underlined.

| Method                 | MovieLens Small |               |               |  | MovieLens 1M  |               |               |  | Last.fm       |               |               |  | Book Genome   |               |               |  |
|------------------------|-----------------|---------------|---------------|--|---------------|---------------|---------------|--|---------------|---------------|---------------|--|---------------|---------------|---------------|--|
|                        | nDCG            | Recall        | Prec.         |  | nDCG          | Recall        | Prec.         |  | nDCG          | Recall        | Prec.         |  | nDCG          | Recall        | Prec.         |  |
| <i>UserKNN</i>         |                 |               |               |  |               |               |               |  |               |               |               |  |               |               |               |  |
| Baseline               | 0.2621          | 0.1408        | 0.2221        |  | 0.3104        | 0.1380        | 0.2768        |  | 0.2089        | 0.2123        | 0.1541        |  | 0.1077        | 0.0952        | 0.0690        |  |
| Qwen-0.6B (Blend)      | 0.2626          | 0.1410        | 0.2221        |  | <b>0.3118</b> | <b>0.1398</b> | <b>0.2781</b> |  | 0.2003        | 0.2027        | 0.1496        |  | 0.1047        | 0.0928        | 0.0670        |  |
| Qwen-0.6B (Inject)     | <b>0.2678</b>   | 0.1466        | <b>0.2274</b> |  | 0.3062        | 0.1356        | 0.2731        |  | <b>0.2067</b> | <b>0.2123</b> | <b>0.1534</b> |  | <b>0.1081</b> | <b>0.0954</b> | <b>0.0685</b> |  |
| Qwen-4B (Blend)        | 0.2616          | 0.1404        | 0.2226        |  | <b>0.3118</b> | <b>0.1399</b> | <b>0.2781</b> |  | 0.2003        | 0.2022        | 0.1494        |  | 0.1047        | 0.0923        | 0.0670        |  |
| Qwen-4B (Inject)       | 0.2661          | 0.1443        | 0.2272        |  | 0.3065        | 0.1366        | 0.2741        |  | 0.1851        | 0.1879        | 0.1355        |  | 0.1045        | 0.0898        | 0.0658        |  |
| Qwen-8B (Blend)        | 0.2639          | 0.1430        | 0.2239        |  | <b>0.3118</b> | <b>0.1399</b> | <b>0.2781</b> |  | 0.2002        | 0.2019        | 0.1493        |  | 0.1046        | 0.0924        | 0.0670        |  |
| Qwen-8B (Inject)       | 0.2674          | <b>0.1479</b> | <u>0.2287</u> |  | 0.3097        | 0.1380        | 0.2767        |  | <u>0.1928</u> | <u>0.1948</u> | <u>0.1406</u> |  | <u>0.1058</u> | <u>0.0942</u> | <u>0.0674</u> |  |
| <i>ItemKNN</i>         |                 |               |               |  |               |               |               |  |               |               |               |  |               |               |               |  |
| Baseline               | 0.2307          | 0.1300        | 0.1975        |  | 0.2687        | 0.1092        | 0.2388        |  | 0.2325        | 0.2324        | 0.1646        |  | 0.1056        | 0.0978        | 0.0626        |  |
| Qwen-0.6B (Blend)      | 0.2282          | 0.1284        | 0.2000        |  | 0.2688        | 0.1092        | 0.2387        |  | 0.2330        | 0.2335        | 0.1652        |  | <b>0.1112</b> | <b>0.1026</b> | <b>0.0651</b> |  |
| Qwen-0.6B (Inject)     | <u>0.2366</u>   | <u>0.1327</u> | <u>0.2010</u> |  | 0.2696        | 0.1093        | 0.2428        |  | 0.2332        | 0.2338        | 0.1654        |  | 0.1047        | 0.0991        | 0.0619        |  |
| Qwen-4B (Blend)        | <b>0.2468</b>   | <b>0.1366</b> | <b>0.2139</b> |  | <u>0.2723</u> | <u>0.1115</u> | <u>0.2426</u> |  | <b>0.2428</b> | <b>0.2429</b> | <b>0.1705</b> |  | 0.1080        | 0.0994        | 0.0633        |  |
| Qwen-4B (Inject)       | 0.2374          | 0.1345        | 0.2044        |  | <b>0.2730</b> | <b>0.1117</b> | 0.2432        |  | 0.2339        | 0.2346        | 0.1659        |  | <u>0.1093</u> | <u>0.1009</u> | <u>0.0644</u> |  |
| Qwen-8B (Blend)        | 0.2465          | <b>0.1366</b> | 0.2133        |  | 0.2704        | 0.1103        | 0.2408        |  | <u>0.2403</u> | <u>0.2396</u> | <u>0.1688</u> |  | 0.1086        | 0.1009        | 0.0644        |  |
| Qwen-8B (Inject)       | 0.2354          | 0.1329        | 0.2044        |  | 0.2712        | 0.1106        | <b>0.2446</b> |  | 0.2407        | 0.2415        | 0.1702        |  | 0.1047        | 0.0981        | 0.0616        |  |
| <i>MF (Blend only)</i> |                 |               |               |  |               |               |               |  |               |               |               |  |               |               |               |  |
| Baseline               | 0.2647          | 0.1430        | 0.2287        |  | 0.2864        | 0.1253        | 0.2587        |  | 0.2397        | 0.2425        | 0.1709        |  | 0.0613        | 0.0496        | 0.0434        |  |
| Qwen-0.6B              | 0.2620          | 0.1426        | <b>0.2307</b> |  | 0.3215        | 0.1393        | 0.2884        |  | 0.2003        | 0.2016        | 0.1497        |  | 0.0813        | 0.0742        | 0.0556        |  |
| Qwen-4B                | 0.2495          | 0.1330        | 0.2202        |  | 0.3214        | 0.1392        | 0.2883        |  | 0.2014        | 0.2028        | <b>0.1506</b> |  | 0.0752        | 0.0673        | 0.0517        |  |
| Qwen-8B                | <b>0.2671</b>   | <b>0.1405</b> | 0.2274        |  | <b>0.3220</b> | <b>0.1395</b> | <b>0.2890</b> |  | <b>0.2217</b> | <b>0.2368</b> | <u>0.1481</u> |  | <b>0.0845</b> | <b>0.0778</b> | <b>0.0578</b> |  |
| <i>NeuMF</i>           |                 |               |               |  |               |               |               |  |               |               |               |  |               |               |               |  |
| Baseline               | 0.2332          | 0.1278        | 0.2041        |  | 0.3009        | 0.1279        | 0.2725        |  | 0.2147        | 0.2252        | 0.1441        |  | 0.0722        | 0.0702        | 0.0490        |  |
| Qwen-0.6B              | 0.2312          | 0.1164        | 0.2034        |  | <u>0.3012</u> | <u>0.1279</u> | <u>0.2734</u> |  | <u>0.2308</u> | <u>0.2416</u> | <u>0.1516</u> |  | <b>0.0861</b> | <b>0.0791</b> | <b>0.0608</b> |  |
| Qwen-4B                | <u>0.2537</u>   | <u>0.1379</u> | <u>0.2202</u> |  | <b>0.3071</b> | <b>0.1317</b> | <b>0.2772</b> |  | <b>0.2333</b> | <b>0.2474</b> | <b>0.1545</b> |  | <u>0.0844</u> | <u>0.0781</u> | <u>0.0581</u> |  |
| Qwen-8B                | <b>0.2671</b>   | <b>0.1405</b> | <b>0.2274</b> |  | 0.2838        | 0.1168        | 0.2587        |  | 0.2217        | 0.2368        | 0.1481        |  | 0.0845        | 0.0778        | 0.0578        |  |





## **Part 4**

# **Beyond Recommendation: LLMs Interpretability, Text-to-SQL and Test-time scaling**



## Nova Directio. A Shift in Perspective

---

To motivate the transition to the next Part, we begin by placing our previous findings in a broader context. In Part 3, we investigated how Large Language Models behave when repurposed for recommendation tasks and what this behaviour reveals about the knowledge they encode. In particular, Chapter 6 established an initial baseline using a role-prompting strategy, demonstrating that ChatGPT can achieve competitive performance across recommendation, re-ranking, and cold-start scenarios despite the absence of task-specific training. While these results demonstrate the flexibility of LLMs, they also raise a central question. To what extent are such models comparable to established collaborative and content-based approaches, and which underlying mechanisms account for this apparent alignment.

Chapter 7 advanced this analysis by moving beyond aggregate effectiveness and examining the structural properties of the recommendation lists produced by LLMs. Our analysis revealed patterns that closely resemble those generated by collaborative filtering methods, suggesting that LLM outputs reflect collaborative signals. This observation is not straightforward to explain, since collaborative approaches rely on explicit user–item interaction histories, whereas LLMs generate recommendations without direct access to such data at inference time. This mismatch between the information required by classical methods and the behaviour observed in LLMs motivated a deeper investigation into the origin of the knowledge driving LLM-based recommendations.

We therefore focused on the role of pre-training and, in particular, on the hypothesis of data leakage. In Chapter 8, we provided empirical evidence that large-scale recommendation datasets can be memorized by LLMs during training. Chapter 9 extended this analysis by introducing more systematic strategies for dataset discovery, including automatic prompt optimization techniques designed to elicit mem-

orized content. Taken together, these studies support the view that LLMs may encode dataset-specific regularities within their internal representations. Rather than considering this phenomenon solely as a drawback, we explored how it could be exploited constructively.

This line of reasoning motivated the development of FOLMind, presented in Chapter 10. FOLMind leverages curated knowledge graphs to extract first-order logic constraints that define user and item profiles. These constraints are then used to guide LLMs in generating semantically grounded narrative descriptions, which are subsequently integrated into classical recommender systems through semantic initialization and a refinement mechanism that combines collaborative and semantic signals. In this way, we showed how structured knowledge and LLM-generated representations can be combined to improve recommendation performance.

At the same time, the scope of LLM capabilities extends beyond recommendation. While recommendation provides a concrete, application-driven setting in which to study model behaviour, many open questions remain regarding how and why LLMs succeed or fail across different tasks. The next Part therefore moves beyond recommendation to examine LLMs from complementary perspectives. We analyze internal representations to characterize how emotional information is encoded (Chapter 11). We investigate whether susceptibility to factual hallucinations can be anticipated *a priori* (Chapter 12). Finally, we explore post-training strategies that improve model behaviour without additional parameter updates, culminating in a test-time scaling approach aimed at increasing the reliability and effectiveness of Large Language Models (Chapter 13).

## Chapter 11

# LLaMAs Have Feelings Too: Unveiling Sentiment and Emotion Representations in LLaMA Models Through Probing

---

Large Language Models (LLMs) have rapidly become central to NLP, demonstrating their ability to adapt to various tasks through prompting techniques, including sentiment analysis. However, we still have a limited understanding of how these models capture sentiment-related information. This study probes the hidden layers of Llama models to pinpoint where sentiment features are most represented and to assess how this affects sentiment analysis.

Using probe classifiers, we analyze sentiment encoding across layers and scales, identifying the layers and pooling methods that best capture sentiment signals. Our results show that sentiment information is most concentrated in mid-layers for binary polarity tasks, with detection accuracy increasing up to 14% over prompting techniques. Additionally, we find that in decoder-only models, the last token is not consistently the most informative for sentiment encoding. Finally, this approach enables sentiment tasks to be performed with memory requirements reduced by an average of 57%.

These insights contribute to a broader understanding of sentiment in LLMs, suggesting layer-specific probing as an effective approach for sentiment tasks beyond prompting, with potential to enhance utility and reduce memory requirements.<sup>1</sup>

1. This chapter is based on work published in the Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (ACL'25), titled “LLaMAs Have Feelings Too: Unveiling Sentiment and Emotion Representations in LLaMA Models Through Probing”.

## 11.1 Introduction

Sentiment analysis (SA), which classifies opinions, emotions, and attitudes in text, is a cornerstone of Natural Language Processing (NLP). SA has enabled the development of several applications, including social media monitoring [55], customer feedback analysis [216], and opinion mining [394]. It comprises various tasks, such as polarity detection (classifying text as positive, negative, or neutral), emotion classification, and subjectivity detection [181].

Earlier approaches for the sentiment classification task relied on supervised learning algorithms and extensive feature engineering [12], requiring large annotated datasets and domain expertise to model sentiment effectively [44].

However, the advent of pre-trained models such as GPT-2 [293] has transformed NLP by enabling downstream tasks through prompting techniques [49, 92], significantly reducing reliance on extensive labeled data.

While prompting has enhanced the applicability of LLMs for sentiment analysis [9, 38, 99, 399], these techniques have often lacked fine-grained control of semantic features, such as context-dependent sentiments or subtle emotional tones. Moreover, the encoding of such features within model representations has not been well understood, limiting efforts to optimize and interpret sentiment analysis performance.

Efforts to interpret neural model representations have advanced significantly over time. Building on the linear representations hypothesis [254], which suggests that high-level features can be represented as linear directions, researchers have explored where human-interpretable concepts are encoded within LLMs. While previous studies have examined concepts such as truthfulness [53], honesty [25], and factual knowledge [208], sentiment encoding remains relatively underexplored, despite its importance in NLP applications.

This study aims to fill this gap by investigating where sentiment information is encoded within Llama models [352]. We focus on both binary polarity (positive or negative sentiment) and fine-grained emotion detection (joy, sadness, anger, fear, love, and surprise), identifying the model layers that most effectively capture these sentiment concepts. Specifically, we trained classifiers to identify subspaces corresponding to specific concepts, such as positive sentiment, within model representations. We refer to these classifiers as probes [13].

Furthermore, unlike previous studies that rely solely on the last token's represen-

tation (as a sentence summary) for probe training [25, 53, 208], we evaluate alternative pooling methods for sentence-level sentiment detection.

Our extensive experiments reveal several key insights into how sentiment and emotion are represented in LLMs. We find that (1) sentiment concepts are most detectable in mid-layer representations, while (2) emotions are more discoverable in early layers; (3) selecting the last token does not consistently yield the best results for probe training; (4) representation quality improves with model size; and (5) probe classifiers generally outperform prompting techniques for accurate sentiment and emotion classification.

In summary, our work makes the following contributions:

- We perform a layer-wise analysis of sentiment encoding in Llama models, identifying the layers that most effectively capture sentiment information.
- We evaluate six pooling methods to identify the most effective approach for sentence representation in sentiment analysis.
- We investigate the impact of model size on sentiment representation by analyzing Llama-3 models of 1B, 3B, and 8B parameters.
- We show that Llama-based classifiers outperform Llama in zero-shot, few-shot, and Chain-of-Thought settings while requiring fewer parameters.
- We propose SENTRILLAMA, a task-specific adaptation of Llama that identifies and utilizes the most representative layer for sentiment analysis, replacing the language modeling head with a lightweight classification head to significantly reduce inference costs while maintaining state-of-the-art performance.

## 11.2 Related Work

In this section, we outline the evolution of sentiment analysis, trace the development of probing research in neural models, focusing on LLMs, and conclude by comparing our work to recent studies.

### 11.2.1 Sentiment Analysis Meets LLMs

Sentiment analysis, the task of classifying and extracting subjective information from text, has evolved from lexicon-based approaches using dictionaries of positive and negative words [141, 385] to supervised learning methods employing n-gram models and machine learning [63, 273]. These early methods, limited by their reliance on handcrafted features, paved the way for feature engineering techniques

and eventually the deep learning revolution that reshaped sentiment classification.

To improve feature identification, researchers incorporated advanced techniques such as syntactic dependencies [408], part-of-speech tagging [331], and negation handling [257]. The advent of deep learning further transformed sentiment analysis, with models such as Recurrent Neural Networks (RNNs) [330] and Convolutional Neural Networks (CNNs) [185], achieving significant advancements by learning representations directly from raw text.

However, the introduction of Transformer architectures [356] marked a paradigm shift. Models like BERT [101] leverage pretraining on vast datasets to extract contextualized representations, and they have been widely used as encoding backbones for downstream tasks. Conversely, GPT [293] employs the Transformer-decoder block and frames language modeling as an autoregressive task. Finally, inspired by the transfer learning paradigm, Raffel et al. [294] introduced T5, an encoder-decoder model for text-to-text tasks, trained on vast amounts of data to capture general language patterns and fine-tuned for a wide range of specific applications.

The introduction of GPT-3 [49] marked a turning point by introducing few-shot learning and demonstrating the effectiveness of prompt-based techniques, significantly reducing the reliance on large labeled datasets for sentiment analysis and other downstream tasks.

Modern models like GPT-4 [268] and Llama [352] achieve even greater flexibility through instruction-following capabilities, excelling in zero-shot settings [290] and outperforming fine-tuned models in sentiment-related tasks [140].

Nowadays, the latest models used for sentiment analysis include DeBERTaV3 [145] and RoBERTa-large [226], which have demonstrated excellent performance when fine-tuned. Additionally, models like GPT and Llama have shown effectiveness in this domain, leveraging prompting techniques or instruction fine-tuning [193, 220, 335, 382], even in complex aspect-based scenarios [28, 431].

### 11.2.2 Probing and LLMs

Probe classifiers, or *probes* [13], are tools designed to extract specific properties from the intermediate representations of neural models. In LLMs, probing helps unveil the semantics of their representations by identifying fine-grained features encoded at different layers, allowing researchers to systematically quantify and compare model capabilities.



In the context of NLP, research has evolved from early analyses of static word embeddings like Word2Vec [402] and GloVe [186] to methods investigating the complex layered knowledge within LLMs [69, 288]. Prior work has probed LLMs for various fine-grained properties, including linguistic properties [360], factual knowledge [33, 285, 391], beliefs [25], cross-lingual alignment [366], logical reasoning capabilities [242], privacy leakages [96, 184], toxicity [307, 384]. These studies approach probing either through a prompt-based method, where the LLM’s performance is evaluated using specifically designed prompts, or by applying trained classifiers to analyze the model’s intermediate layers.

### 11.2.3 Probing LLMs for Sentiment Analysis

Various studies have assessed the capabilities of LLMs for sentiment analysis tasks. For instance, Fatouros et al. [111] analyzed the performance of ChatGPT-3.5 in financial sentiment analysis, demonstrating performance that exceeds FinBERT [21]. Similarly, Araujo et al. [22] investigated ChatGPT’s effectiveness in Portuguese sentiment analysis, highlighting its potential value in dataset annotation. Lyu et al. [233] investigates the application of causal inference to sentiment analysis and introduces causal prompts to enhance LLM performance in sentiment prediction tasks. Furthermore, Zhang et al. [423] conducted a systematic evaluation across various sentiment tasks using ChatGPT and different T5 model sizes. Their findings reveal that while LLMs excel in simple (e.g. binary or trinary) zero-shot sentiment classification tasks, they struggle with complex ones (e.g. aspect-based).

While previous studies have primarily evaluated LLMs as *text-to-text models*, focusing on zero- and few-shot learning capabilities, our work takes a different approach. We investigate the hidden representations within the intermediate layers of transformer architectures to identify where sentiment concepts are encoded and how these insights can inform the development of more efficient and accurate models.

Similar studies have investigated the representation of semantic concepts within LLMs [19], such as Burns et al. [53] on truthfulness, Azaria et al. [25] on honesty, Roy et al. [307] on hate speech, and Li et al. [208] on factual knowledge. However, the investigation of sentiment within the hidden representations of these models remains comparatively underexplored.

### 11.3 Methodology

In this section, we detail the models used in the experiments, the datasets, the probe classifiers, and the experimental settings, with a focus on ensuring the reproducibility of our work. Furthermore, we have made all the code publicly available<sup>2</sup> to enable systematic and efficient probing of LLMs.

**Sentiment Detection in Hidden Space.** We build on the concept detection framework proposed by Rütte et al. [310], adapting it specifically for sentiment analysis. In this setup, we define a sentiment concept  $S$  and use a corresponding sentiment analysis dataset  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ , where  $y_i$  represents the labeled sentiment of sentence  $x_i$ . Here,  $\text{rep}_\theta(x_i)$  represents an intermediate representation of  $x_i$ , generated from a forward pass through the LLM $_\theta$ . The goal is to extract a collection of these representations  $\{\text{rep}_\theta(x_i)\}_{i=1}^n$ . We therefore train a classifier  $C_w$  on these representations to predict the presence of the sentiment concept  $S$  (i.e., positive or negative sentiment within  $x_i$ ), effectively predicting  $y_i$ .

Implementing sentiment detection involves two fundamental design choices: (1) The selection of the intermediate representation  $\text{rep}_\theta$ , which may vary depending on the layers or pooling strategies applied within the LLM. (2) The choice of classifier  $C_w$ , which serves to distinguish between different sentiment categories based on these embeddings.

In the following, we outline the fundamental structure of Transformer architecture and highlight common techniques for selecting representations. **Choice of Representation.** Current state-of-the-art LLMs are based on the Transformer architecture [356], where sequential Transformer blocks generate intermediate hidden representations ( $h$ ), each with potentially distinct functionalities. Let  $l \in \mathbb{N}$  denote the  $l$ -th layer, and  $\mathbf{x}^{(l)} \in \mathbb{R}^{T \times d_{\text{emb}}}$  represent the output, where  $T$  is the number of tokens and  $d_{\text{emb}}$  is the hidden dimension. A Transformer refines  $\mathbf{x}^{(l)}$  using multi-head attention (MHA) and a feed-forward network (FFN):

$$h_{\text{attn}}^{(l)} = \text{MHA} \left( \text{LayerNorm} \left( \mathbf{x}^{(l)} \right) \right)$$

$$h_{\text{resid}}^{(l)} = h_{\text{attn}}^{(l)} + \mathbf{x}^{(l)}$$

2. <https://github.com/sisinflab/LLM-SentimentProber>

$$h_{\text{ffn}}^{(l)} = \text{FFN} \left( \text{LayerNorm} \left( h_{\text{resid}}^{(l)} \right) \right)$$

$$x^{(l+1)} = h_{\text{ffn}}^{(l)} + h_{\text{resid}}^{(l)}$$

We exploit the Llama-3 architecture [107], leveraging its Grouped Query Attention (GQA) and RMSNorm features to extract hidden representations efficiently.

Previous studies have explored various representations, such as the residual stream ( $x^{(l+1)}$ ) [53, 137, 246, 438], the normalized residual stream [265], or attention heads [23, 208]. Based on our preliminary experiments showing marginally higher detection accuracy, we adopt the residuals stream ( $\text{rep}_\theta(x) = x^{(l+1)}$ ).

Instead of using the full prompt representation ( $x_{\text{rep}} \in \mathbb{R}^{T \times d_{\text{emb}}}$ , where  $T$  is the token count), we focus on a subset ( $x_{\text{rep}} \in \mathbb{R}^{t \times d_{\text{emb}}}$ ) with  $t \leq T$ . Each token representation ( $x_{\text{rep}}[i, :] \in \mathbb{R}^{d_{\text{emb}}}$ , for  $i = 1, \dots, t$ ), is treated as an independent feature. This approach focuses on the parts of the prompt most likely to capture the sentiment concept. Prior work has carefully selected a single token [23, 137, 438] or relied on the last token of the prompt [53, 208, 239, 246, 304].

In our experiments, we explore six methods for selecting the representations:

- (1) Mean Pooling (Fig. 11.2): Compute the mean activation value across all tokens for each  $d_{\text{embed}}$  dimension, resulting in a single vector where each element corresponds to the average activation of a particular feature (embedding dimension) over the entire sequence.
- (2) Last-Token Pooling (Fig. 11.3): Uses the final token's features of the last token in the sequence.
- (3) Max Pooling (Fig. 11.4): Identifies the most prominent feature across all tokens  $T$  for each feature, outputting a vector representing the most dominant features in the sequence.
- (4) Min Pooling (Fig. 11.4): Complementary to Max Pooling, producing a vector representing the least dominant features in the sequence.
- (5) Concat-Mean-Max-Min Pooling: Concatenates the mean, max, and min pooling, generating a representation of size  $3 \times d_{\text{embed}}$  that encapsulates multiple aspects of the token embeddings.
- (6) Attention Mean Pooling (Fig. 11.5): Given a token representation  $x_{\text{rep}}[i, j]$ , where  $i$  denotes the token and  $j$  represents a specific embedding dimension, this pooling method constructs a representation by combining token embeddings with a

corresponding importance score. The importance score for each token  $i$  is computed by applying the softmax function to the mean of the token’s embedding values. This assigns higher weights to tokens with larger average values. These scores are multiplied element-wise to each dimension of the token embeddings,  $x_{\text{rep}}[i, :]$ , to adjust their contribution based on importance. The final pooled representation,  $x_{\text{pooled}}$ , is then computed as a weighted sum of these adjusted embeddings, emphasizing the most relevant tokens activations. Mathematically:

$$x_{\text{pooled}} = \sum_{i=1}^T \text{softmax} \left( \frac{1}{J} \sum_{j=1}^J x_{\text{rep}}[i, j] \right) \cdot x_{\text{rep}}[i, :]$$

A visual representation of the pooling strategies can be found in Supplementary Material 11.7. These approaches offer diverse strategies for selecting representations, enabling a more nuanced understanding of how sentiment information is encoded. The choice of tokens should ideally reveal whether sentiment is distributed across the entire prompt or concentrated in specific tokens. For instance, the sentences “My name is XYZ and I’m happy” and “I’m happy because my name is XYZ” both convey positive sentiment. However, relying solely on the last token could lead to a suboptimal representation, as it may not adequately capture the sentiment expressed earlier in the sequence.

**Choice of Classifier.** With the hidden representations selected, we can train our probing classifier  $C_w$  on the sentiment labels. In our experimental setup we include twelve distinct classifiers, grouped into five categories based on their underlying modeling approach, as summarized in Table 11.1.

**Table 11.1** Clusters of Classifiers Based on Model Type.

| Type                  | Models                                           |
|-----------------------|--------------------------------------------------|
| <b>Linear</b>         | Logistic Regression, Linear SVM                  |
| <b>Distance-based</b> | K-Nearest Neighbors                              |
| <b>Tree-based</b>     | Decision Tree, Random Forest, XGBoost, Light-GBM |
| <b>Neural Network</b> | MLP, BiLSTM, CNN                                 |
| <b>Other</b>          | Non-linear SVM, Gaussian Naive Bayes             |

These classifiers were chosen to balance simplicity, interpretability, and the capacity to model complex patterns. Linear models, such as Logistic Regression and Linear SVM, serve as baselines for probing tasks due to their simplicity and their

ability to identifying and leveraging linear relationships. Non-linear and neural network models, on the other hand, are included for their ability to capture intricate relationships within the hidden space. BiLSTM and CNN, in particular, were selected for their proven ability to generate effective hidden representations [128].

All models, except for BiLSTM and CNN, are implemented using the scikit-learn library [282]. While, BiLSTM and CNN were implemented using PyTorch. For each classifier, we employed the Optuna framework [10] for hyperparameter optimization, focusing on key parameters such as regularization strengths, tree depths, and kernel types. Optimization was performed over five trials [275], balancing computational efficiency with sufficient exploration of the hyperparameter space.

To ensure reproducibility, we configured random seed initialization and enforced deterministic behavior for CUDA operations. Details are provided in Supplementary Material 11.7.

**Datasets Details.** Our experiments utilize three benchmark datasets for sentiment classification: IMDB [237], SST-2 [330], and Rotten Tomatoes [272], all of which contain movie reviews for binary polarity tasks (e.g., positive or negative sentiment). Additionally, we extended the setup to a more complex evaluation, conducting fine-grained sentiment classification using the Emotion dataset [315], which categorizes sentiments into six nuanced classes: joy, sadness, anger, fear, love, and surprise.

We only preprocessed the IMDB and Emotion datasets because their original sizes, 50K for IMDB and 20K for Emotion, made them impractical for the large number of experiments. Details of the preprocessing steps are provided in Supplementary Material 11.7. Table ?? provides details on the number of samples and the train-test splits used in the experiments.

**Table 11.2** Dataset statistics.

| Dataset         | Train | Test | Labels     | Max Len |
|-----------------|-------|------|------------|---------|
| IMDB            | 7000  | 7000 | pos/neg    | 132     |
| SST-2           | 6920  | 1821 | pos/neg    | 56      |
| Rotten Tomatoes | 8530  | 1066 | pos/neg    | 59      |
| Emotion         | 6000  | 2000 | six labels | 64      |

## 11.4 Experimental Results

We conduct our experiments on three Llama models, specifically testing the instruction-tuned variants of Llama 3.2-1B, Llama 3.2-3B, and Llama 3.1-8B, all sourced

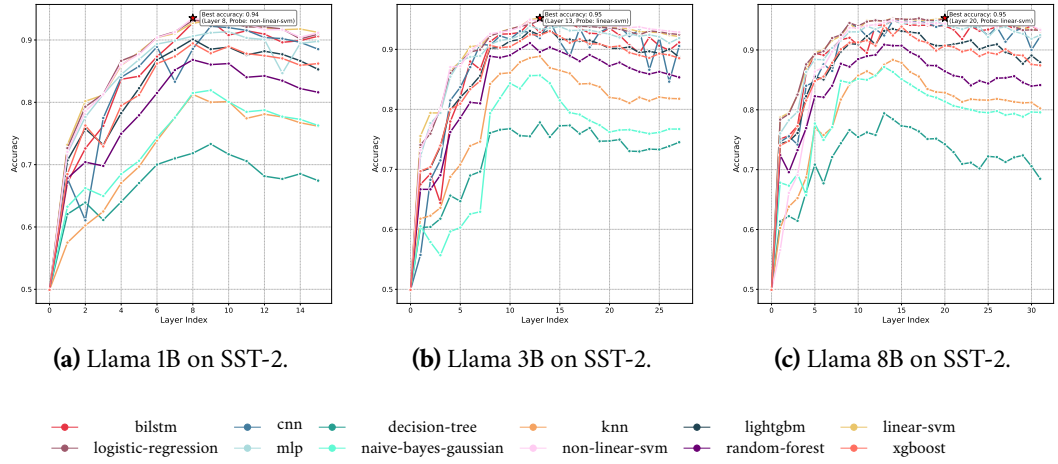


**Figure 11.1** Layer-wise probing accuracy (Last-Token) on SST-2. Refer to Supplementary Material Fig. 11.6 for comprehensive results across all datasets.

from Hugging Face<sup>3</sup>. For every probing datasets we train the classifier  $C_w$  on the training set, and evaluate its detection performance by measuring the accuracy on the test set.

**Sentiment detection results.** In our first experiments, we evaluate the detection performance of the different classifier  $C_w$ . To extract the  $\text{rep}_\theta$ , following prior work, we focus on the residual stream using the Last-Token approach. We report detection accuracies for each layer in  $\text{LLM}_\theta$  and visualize the results in Fig. 11.1.

For sentiment detection in binary polarity tasks (i.e., SST-2, IMDB, and Rotten Tomatoes), non-linear SVM, linear SVM, and logistic regression consistently outperform other probing techniques, achieving approximately 90% accuracy in the middle layers across all model sizes. For fine-grained emotion datasets, linear SVM demonstrates the best performance, reaching around 70% accuracy in the initial layers across all model sizes. This indicates that Llama models have linear representations for binary sentiment (positive/negative) and fine-grained emotions (joy, sadness, anger, fear, love, and surprise). It is interesting to discover that, although sentiment is not easily discernible during direct interaction with Llama, sentiment and emotion concepts can be linearly detected within certain internal layers.

Furthermore, since prior work often use the last-token representations, we also investigate the effect of different token representation methods. Specifically, we

3. <https://huggingface.co/meta-llama>

**Table 11.3** Probing Results Across Different Pooling Methods. See Supplementary Material 11.7 for confidence level plots across all layers and datasets.

| Model                      | Dataset | Layer | Prober              | Pooling    | Accuracy |
|----------------------------|---------|-------|---------------------|------------|----------|
| Llama 3.2 1B<br>(Instruct) | SST2    | 10    | Non-linear SVM      | Attn       | 0.9450   |
|                            |         | 10    | Non-linear SVM      | Mean       | 0.9450   |
|                            |         | 8     | Non-linear SVM      | Last-Token | 0.9352   |
|                            | IMDB    | 8     | Logistic Reg.       | Mean       | 0.9400   |
|                            |         | 8     | Logistic Reg.       | Attn       | 0.9396   |
|                            |         | 8     | Linear SVM          | Last-Token | 0.9009   |
|                            | Rotten  | 8     | Linear SVM          | Concat     | 0.8939   |
|                            |         | 8     | Logistic Reg.       | Concat     | 0.8939   |
|                            |         | 8     | Non-linear SVM      | Last-Token | 0.8789   |
|                            | Emotion | 1     | Linear SVM          | Concat     | 0.7880   |
|                            |         | 10    | LightGBM            | Max        | 0.7880   |
|                            |         | 7     | Linear SVM          | Last-Token | 0.6830   |
| Llama 3.2 3B<br>(Instruct) | SST2    | 5     | Non-linear SVM      | Concat     | 0.9594   |
|                            |         | 12    | Non-linear SVM      | Concat     | 0.9577   |
|                            |         | 13    | Linear SVM          | Last-Token | 0.9522   |
|                            | IMDB    | 4     | Logistic Reg.       | Attn       | 0.9523   |
|                            |         | 14    | Logistic Reg.       | Mean       | 0.9522   |
|                            |         | 13    | Non-linear SVM      | Last-Token | 0.9177   |
|                            | Rotten  | 3     | Logistic Reg.       | Concat     | 0.9090   |
|                            |         | 3     | CNN                 | Mean       | 0.9071   |
|                            |         | 13    | Linear SVM          | Last-Token | 0.8949   |
|                            | Emotion | 1     | MLP                 | Concat     | 0.8220   |
|                            |         | 0     | LightGBM            | Concat     | 0.8195   |
|                            |         | 4     | Linear SVM          | Last-Token | 0.6940   |
| Llama 3.1 8B<br>(Instruct) | SST2    | 3     | Logistic Reg.       | Concat     | 0.9605   |
|                            |         | 14    | Logistic Reg.       | Concat     | 0.9599   |
|                            |         | 14    | Linear SVM          | Last-Token | 0.9533   |
|                            | IMDB    | 4     | Non-linear SVM      | Concat     | 0.9579   |
|                            |         | 16    | Non-linear SVM      | Mean       | 0.9576   |
|                            |         | 11    | Linear SVM          | Last-Token | 0.9273   |
|                            | Rotten  | 9     | MLP                 | Mean       | 0.9203   |
|                            |         | 14    | CNN                 | Mean       | 0.9203   |
|                            |         | 14    | Logistic Regression | Last-Token | 0.9183   |
|                            | Emotion | 0     | Linear SVM          | Concat     | 0.8685   |
|                            |         | 0     | LightGBM            | Max        | 0.8655   |
|                            |         | 4     | Linear SVM          | Last-Token | 0.6885   |

compare the last-token approach with five alternatives: mean, max, min, concatenation, and attention, as previously described.

The results, summarized in Table 11.3, which shows the top-3 performers by datasets, reveal that concatenating the mean, max, and min of  $\text{rep}_\theta$  is often the most effective method for detecting sentiment concepts. Notably, mean and attention pooling also perform strongly, achieving results comparable to concatenation and consistently outperforming last-token pooling. This indicates, that the last-token representation is not always the optimal choice for capturing sentiment concepts.

Furthermore, we discover that combining the max pooling representation with LightGBM yields particularly strong performance. Further investigation reveals that tree-based models, including Random Forest, Decision Tree, and XGBoost, also perform better with max pooling. We hypothesize that the synergy between max pooling and tree-based models arises from max pooling’s ability to emphasize dominant features, which, when combined with tree-based models’ strength in exploiting high-contrast, threshold-based partitions, leads to an optimal combination. This alignment makes max pooling particularly effective for tree-based models, highlighting how the choice of classifier should also influence the choice of representation, and vice versa.

## 11.5 SENTRILLAMA for Efficient Downstream Tasks

Sentiment analysis has been a fundamental task since the inception of NLP. Traditionally, there have been two primary approaches to accomplish this task: (1) training a model from scratch or (2) fine-tuning an existing model to meet specific requirements. Recently, a third approach has gained prominence: leveraging well-designed prompts in conjunction with state-of-the-art large language models [9, 99, 399].

We introduce SENTRYLLAMA, a specialized Llama model that leverages layers up to  $L_{\leq i}$  for sentiment tasks, where  $i$  denotes the most representative layer for the task. Based on our earlier analysis, we identified the optimal layer  $L_i$ , which retains the most salient features for downstream tasks. By attaching a lightweight, task-specific classification head, such as a LinearSVM, SENTRYLLAMA efficiently repurposes the expressive power of the Llama model while discarding unnecessary layers. This approach significantly reduces computational requirements for inference, making it both efficient and task-specific without compromising performance.



Specifically, the total model parameters are calculated as the sum of the input embedding,  $N$ -layer, and LM head parameters. The proposed SENTRILLAMA approach reduces model complexity by retaining only the input embedding and the layers up to the most representative layer ( $i$ ), identified in previous experiments, while replacing the LM head with a lightweight classification head, such as a linear SVM. This reduces parameter usage and adapts the architecture for downstream tasks.

To benchmark the effectiveness of SENTRILLAMA, we compare its performance against fine-tuned DeBERTaV3-large<sup>4</sup> and RoBERTa-large<sup>5</sup> across all datasets. Additionally, we evaluate SENTRILLAMA against Llama models under zero-shot, few-shot, and Chain-of-Thought prompting scenarios using carefully designed templates inspired by prior work [99]. Details on the prompt used are provided in Supplementary Material 11.7. Furthermore, to understand the effect of instruction fine-tuning on the sentiment task, we also include in the comparison the non-instructed version of Llama3.2 (1B). Table 11.4 presents a comparative analysis of the accuracy of SENTRILLAMA, DeBERTa, RoBERTa, and a prompt-based method. The best-performing model is highlighted in **bold**, while the runner-up is underlined.

For the sentiment downstream task, SENTRILLAMA 8B outperforms all other approaches, achieving approximately 96% accuracy on SST-2, 96% on IMDB, 92% on Rotten Tomatoes, and 87% on the Emotion dataset. Only DeBERTa surpasses this performance on the Emotion dataset, with an accuracy of 88%. The prompt-based approach reveals the effectiveness of few-shot and Chain-of-Thought (CoT), achieving comparable results but falling short on the Emotion dataset. These results align with expectations, revealing several insights: (1) increasing the base model size improves performance, (2) the non-instruction version of the 1B model exhibits minimal differences compared to the instructed ones, and (3) DeBERTa and RoBERTa continue to lead the leaderboard in sentiment analysis, outperforming prompt-based methods. However, it is worth noting that the actual size of SENTRILLAMA differs from the original Llama model. We represent the effective sizes in Table 11.5.

Using SENTRILLAMA reduces the size of the models. For the 1B model (both the standard and instruction-finetuned versions) on SST-2, IMDB, and Rotten Tomatoes, approximately 19% of the parameters are removed, while for the Emotion dataset, the reduction is up to 61.6%. For the 3B model on SST-2, IMDB, and Rot-

4. <https://huggingface.co/microsoft/deberta-v3-large>

5. <https://huggingface.co/FacebookAI/roberta-large>

**Table 11.4** Comparison of SENTRILLAMA against DeBERTa, RoBERTa, and prompt-based method.

| Model                                 | SST2          | IMDB          | Rotten Tomatoes | Emotion       |
|---------------------------------------|---------------|---------------|-----------------|---------------|
| <b>Instruct-Llama 3.2 (1B)</b>        |               |               |                 |               |
| Zero-shot                             | 0.7210        | 0.6898        | 0.6923          | 0.2140        |
| Few-shot                              | 0.6485        | 0.5994        | 0.5994          | 0.2885        |
| Chain-of-Thought                      | 0.4992        | 0.5000        | 0.5000          | 0.3475        |
| <b>Instruct-Llama 3.2 (3B)</b>        |               |               |                 |               |
| Zero-shot                             | 0.7759        | 0.8397        | 0.7279          | 0.3750        |
| Few-shot                              | 0.7606        | 0.8528        | 0.7176          | 0.3045        |
| Chain-of-Thought                      | 0.9154        | 0.9306        | 0.8743          | 0.4645        |
| <b>Instruct-Llama 3.1 (8B)</b>        |               |               |                 |               |
| Zero-shot                             | 0.9341        | 0.9461        | 0.9024          | 0.4455        |
| Few-shot                              | 0.9330        | 0.9411        | 0.8968          | 0.3340        |
| Chain-of-Thought                      | 0.9165        | 0.9363        | 0.8771          | 0.5605        |
| <b>SENTRI LLAMA 3.2 (1B)</b>          | 0.9308        | 0.9445        | 0.8912          | 0.8015        |
| <b>SENTRI LLAMA 3.2 (1B) Instruct</b> | 0.9450        | 0.9400        | 0.8940          | 0.7880        |
| <b>SENTRI LLAMA 3.2 (3B) Instruct</b> | 0.9594        | 0.9523        | <u>0.9090</u>   | 0.8220        |
| <b>SENTRI LLAMA 3.1 (8B) Instruct</b> | <b>0.9605</b> | <b>0.9579</b> | <b>0.9203</b>   | <u>0.8685</u> |
| <b>DeBERTa V3 Large (418M)</b>        | <u>0.9599</u> | <u>0.9534</u> | 0.8671          | <b>0.8765</b> |
| <b>RoBERTa Large (355M)</b>           | 0.9038        | 0.9430        | 0.8808          | 0.8416        |

**Table 11.5** Comparison of SENTRILLAMA Sizes Across Datasets.

| Model                                 | SST2 | IMDB | Rotten Tomatoes | Emotion |
|---------------------------------------|------|------|-----------------|---------|
| <b>SENTRI LLAMA 3.2 (1B)</b>          | 810M | 811M | 811M            | 384M    |
| <b>SENTRI LLAMA 3.2 (1B) Instruct</b> | 932M | 810M | 810M            | 384M    |
| <b>SENTRI LLAMA 3.2 (3B) Instruct</b> | 2B   | 1.9B | 1.8B            | 595M    |
| <b>SENTRI LLAMA 3.1 (8B) Instruct</b> | 3.5B | 3.8B | 3.8B            | 743M    |

**Table 11.6** Computational efficiency comparison on the SST-2 dataset. See Supplementary Material 11.7 for additional dataset results.

| Model                                 | Peak GPU Usage | Avg. Time per Sample | Throughput (Samples/sec) |
|---------------------------------------|----------------|----------------------|--------------------------|
| <b>Instruct-Llama 3.2 (1B)</b>        | 2.4 GB         | 11.17 ms             | 90                       |
| <b>Instruct-Llama 3.2 (3B)</b>        | 6.2 GB         | 18.19 ms             | 55                       |
| <b>Instruct-Llama 3.1 (8B)</b>        | 15.4 GB        | 37.73 ms             | 48                       |
| <b>SENTRI LLAMA 3.2 (1B)</b>          | 1.5 GB         | 6.08 ms              | 164                      |
| <b>SENTRI LLAMA 3.2 (1B) Instruct</b> | 1.7 GB         | 7.98 ms              | 125                      |
| <b>SENTRI LLAMA 3.2 (3B) Instruct</b> | 1.7 GB         | 5.09 ms              | 196                      |
| <b>SENTRI LLAMA 3.1 (8B) Instruct</b> | 3.2GB          | 5.31 ms              | 182                      |
| <b>DeBERTa V3 Large (418M)</b>        | 845 MB         | 22.03 ms             | 45                       |
| <b>RoBERTa Large (355M)</b>           | 692 MB         | 8.35 ms              | 120                      |

ten Tomatoes, the reduction is approximately 36.6%, and for the Emotion dataset, it is 80%. For the 8B model on SST-2, IMDB, and Rotten Tomatoes, approximately 53.7% of the parameters are removed, while for the Emotion dataset, the reduction is 90.7%. This result indicates that the amount of parameters removed depends heavily on the specific task and that it is possible to reuse the highly relevant representations of Llama to improve downstream tasks, going beyond prompting and resulting in a competitive model.

Furthermore, we compare the computational efficiency of SENTRILLAMA with DeBERTa, RoBERTa, and a prompt-based method. In Table 11.6, we present the results for SST-2.

Comparing GPU usage, SENTRILLAMA models demonstrate greater memory efficiency than their Instruct-Llama counterparts. SENTRILLAMA 3.2 (1B) uses 37.5% less GPU memory than Instruct-Llama 3.2 (1B), while SENTRILLAMA 3.1 (8B) reduces memory usage by 79.2% compared to Instruct-Llama 3.1 (8B). However, DeBERTa and RoBERTa remain the most memory-efficient, with RoBERTa requiring only 692 MB.

In terms of inference speed, SENTRILLAMA models exhibit a significantly lower average time per sample, indicating they are faster than their Instruct-Llama counterparts. Specifically, SENTRILLAMA 3.2 (1B) is 45.5% faster than Instruct-Llama 3.2 (1B), SENTRILLAMA 3.2 (3B) is 71.9% faster than Instruct-Llama 3.2 (3B), and SENTRILLAMA 3.1 (8B) is 85.9% faster than Instruct-Llama 3.1 (8B). Meanwhile, DeBERTa has a 75.9% longer inference time compared to SENTRILLAMA 3.2 (8B), while RoBERTa is 36.4% slower.

Finally, SENTRILLAMA models achieve significantly higher throughput (samples per second) than Instruct-Llama, DeBERTa, and RoBERTa.

Overall, this analysis reveals that SENTRILLAMA models are more efficient in terms of GPU memory usage and inference speed, consistently outperforming Instruct-Llama models across all aspects. Smaller models, such as SENTRILLAMA 3.2 (1B) and SENTRILLAMA 3.2 (3B), offer the best trade-off between memory consumption and speed. Meanwhile, RoBERTa Large provides competitive throughput and latency compared to larger Llama-based models, whereas DeBERTa struggles with slower inference.

## 11.6 Discussion

Our results show that our method is still far from providing a smaller Llama model that performs as well as similarly sized models like DeBERTa or RoBERTa. The primary limitation stems from the need to retain all preceding layers up to the most representative one. This limitation introduces an element of randomness: if circumstances align favorably and the most representative layer is among the early layers (as observed with the Emotion dataset), the final model's size becomes highly competitive with DeBERTa and RoBERTa. However, if the most representative layer is in the middle layers, the number of parameters—although still lower than the full model—can exceed those of the baseline models.

While we have automated the discovery of the most representative layer and pooling methods (see Table 11.3), the dependence on preceding layers results in a sub-optimal outcome. A more robust solution could involve leveraging only the input embeddings and the most representative layer, potentially complemented by layer compression or pruning techniques to reduce computational overhead while preserving performance.

*Is this method generalizable for different downstream tasks beyond sentiment analysis?* While we cannot yet provide a definitive answer on the method's generalizability, prior studies have shown that LLMs capture a broad range of semantic concepts, such as truthfulness, honesty, and factual knowledge [25, 53, 208]. Our findings suggest that sentiment and emotion are similarly well-represented in these models. We hypothesize that this approach could extend to tasks requiring nuanced linguistic analysis, such as sarcasm detection or intent recognition, assuming the availability of labeled datasets.

However, significant challenges remain. Extending this method to more complex tasks may require additional adaptations. Future research should systematically evaluate this approach across diverse tasks and datasets to uncover its broader potential and limitations.

## 11.7 Summary

In this work, we present a systematic analysis of hidden representations in Llama for sentiment analysis. Our findings demonstrate that for positive/negative sentiment, the middle layers are the most representative across dimensions, whereas

for emotion analysis, the most representative layers are found in the initial ones. Additionally, we show that while the current standard for representing a sentence involves using the last token, alternative methods, such as concatenating the max, min, and mean of the representations, yield better results.

Finally, we introduce a novel approach called SENTRILLAMA, which utilizes only a subset of the model’s parameters. This approach achieves results comparable to state-of-the-art models and outperforms prompting-based methods. We hope that the introduction of this layer selection approach for Llama will inspire the development of more robust techniques, facilitating the reuse of these large models for downstream tasks and extending their usability beyond text generation.



# Supplementary Material

---

## Detailed Preprocessing Workflow

The IMDB and Emotion datasets were preprocessed due to their original sizes, 50K for IMDB and 20K for Emotion, which made them impractical for the extensive number of experiments. To address this, reduced versions of these datasets were created while preserving their statistical properties. A structured preprocessing workflow was used to prepare the datasets for the experiments, ensuring statistical consistency and adherence to specific constraints, such as sentence length and label distribution. The key steps are outlined below:

1. **Initialize Random Seed:** To achieve reproducible datasets, we initialized a random seed (Supplementary Material 11.7) at the beginning of the workflow, ensuring that the sampling process could be replicated exactly in future runs.
2. **Calculate Dataset Statistics:**
  - a. Compute the label distribution to maintain the proportion of samples for each label in the original dataset.
  - b. Compute the average sentence length across all samples to preserve similar statistics.
3. **Generate the Dataset:** For each label in the dataset, we followed a sampling process as follows:
  - a. Filter samples by sentence length for the given label.
  - b. Random sample to select the required number of samples for each label.
  - c. Combine the selected samples for all labels into a single dataset. To ensure randomness, the combined dataset was shuffled before saving.
4. **Validate Reduced Dataset:** The label distribution and average sentence length

were recalculated for the reduced dataset. If the average sentence length exceeded the specified constraint, an error was raised, indicating the need for further adjustments.

To provide transparency and facilitate analysis, a summary of the preprocessing steps is reported:

1. The total number of samples in both the original and reduced datasets.
2. The distribution of labels before and after reduction.
3. The average sentence length in both datasets.

This comprehensive preprocessing workflow ensured that the reduced datasets retained the statistical properties of the original datasets while adhering to predefined constraints. The code used to generate the datasets is provided in `'dataset/reduce_dataset.py'`, along with a detailed PDF report containing all the statistics.

## Reproducibility and Determinism

To ensure reproducibility, we consistently set random seeds to 42 across Python's `'random'`, NumPy, and PyTorch libraries. Furthermore, deterministic behavior was enforced for CUDA-enabled operations by configuring PyTorch's cuDNN backend and settings environment variables. Further details can be found by examining the `set_seed()` function in `utils.py`. This rigorous control of randomness and parallelism ensures consistent experimental results, addressing the stochastic nature of the workflows.

## Prompting Approach

In this section, we illustrate the prompts adopted for the experiments with the Llama models, covering zero-shot, few-shot and Chain-of-Thought prompting scenarios.



### Zero-Shot Prompting for Binary Sentiment

**System:** "You are an assistant trained to perform strict sentiment and emotion classification.

You MUST only respond with the numeric label corresponding to the classification. Do not provide any explanations, reasoning, or any text other than the required numeric value."

**User:** "Classify the sentiment of the following text: '{text}'"

**Assistant:** "If the sentiment is positive, respond with '1'. If the sentiment is negative, respond with '0'.

No other text, explanation, or formatting."

### Zero-Shot Prompting for Emotion

**System:** "You are an assistant trained to perform strict sentiment and emotion classification.

You MUST only respond with the numeric label corresponding to the classification. Do not provide any explanations, reasoning, or any text other than the required numeric value."

**User:** "Classify the sentiment of the following text: '{text}'"

**Assistant:** "Classify the text into one of the following emotions and respond only with the corresponding number:

0: Sadness, 1: Joy, 2: Love, 3: Anger, 4: Fear, 5: Surprise.

No explanation or additional text."

### Few-Shot Prompting for Binary Sentiment

**System:** "You are an assistant trained for sentiment and emotion analysis.  
You MUST only respond with the correct numeric label.  
Do not provide explanations or any additional text."

**User:** "Examples:  
'I love this product!' => 1  
'I am disappointed with the service.' => 0  
Classify the following text sentiment:{text}"

**Assistant:** "If the sentiment is positive, respond with '1'. If the sentiment is negative, respond with '0'.  
No other text, explanation, or formatting."

### Few-Shot Prompting for Emotion

**System:** "You are an assistant trained for sentiment and emotion analysis.  
You MUST only respond with the correct numeric label.  
Do not provide explanations or any additional text."

**User:** "Examples:  
'This is the worst day of my life.' => 0  
'I feel so joyful and alive!' => 1  
'I feel so deeply connected and grateful for you in my life.' => 2  
'I am so angry right now.' => 3  
'I'm really scared and worried about what might happen next.' => 4  
'Wow, I didn't expect that at all! This is completely unexpected!' => 5  
Classify the following text emotion:{text}"

**Assistant:** "Respond only with one of the following numbers: 0: Sadness, 1: Joy, 2: Love, 3: Anger, 4: Fear, 5: Surprise.  
No other text, explanation, or formatting."

### Chain-of-Thought Prompting for Binary Sentiment

**System:** "You are an assistant specialized in sentiment and emotion analysis. Think step-by-step through the reasoning process (chain-of-thought) privately, but provide only the final numeric classification as instructed. Do not include reasoning steps in the output."

**User:** "Analyze the sentiment of the following text: '{text}'  
Carefully reason step-by-step to determine the sentiment.  
Output only '1' for positive sentiment or '0' for negative sentiment as your final response."

**Assistant:** "I will reason step-by-step internally to determine the sentiment. However, my final response will be '1' for positive sentiment or '0' for negative sentiment, with no explanation included in the output."

### Chain-of-Thought Prompting for Emotion

**System:** "You are an assistant specialized in sentiment and emotion analysis. Think step-by-step through the reasoning process (chain-of-thought) privately, but provide only the final numeric classification as instructed. Do not include reasoning steps in the output."

**User:** "Analyze the emotion of the following text: '{text}'  
Carefully reason step-by-step to identify the best-matching emotion.  
Output only the corresponding number as your final response: 0: Sadness, 1: Joy, 2: Love, 3: Anger, 4: Fear, 5: Surprise."

**Assistant:** "I will reason step-by-step internally to determine the most appropriate emotion.  
My final response will be one of the following numbers: 0: Sadness, 1: Joy, 2: Love, 3: Anger, 4: Fear, 5: Surprise.  
No reasoning will be included in the output."

## Pooling Methods

In this section, we provide a visual clarification of the different pooling strategies adopted in our work. Further details can be found in Section 11.3.

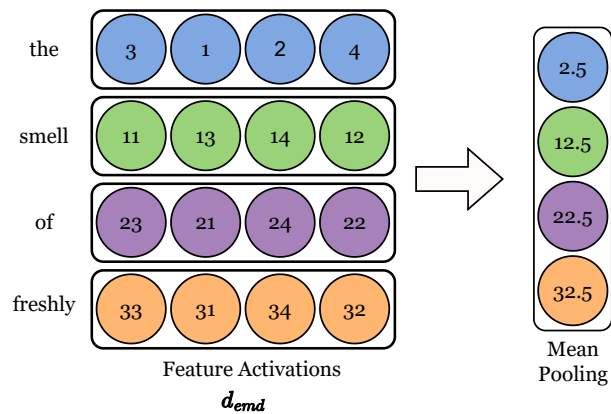


**Figure 11.2** Mean Pooling Visually Explained.

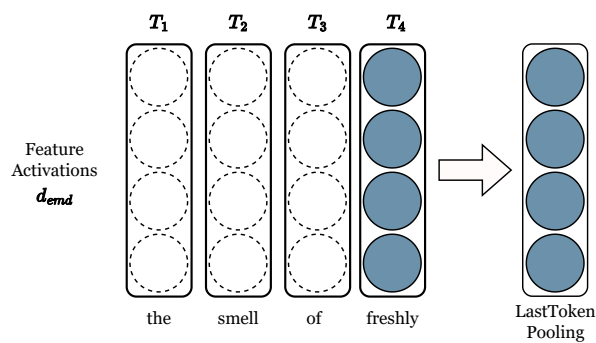


**Figure 11.3** Last Token Pooling Visually Explained.

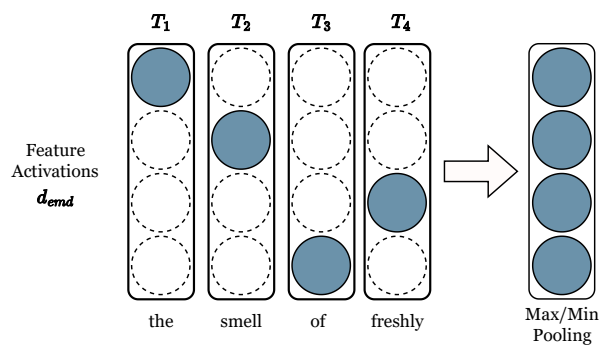


**Figure 11.4** Max and Min Pooling Visually Explained.

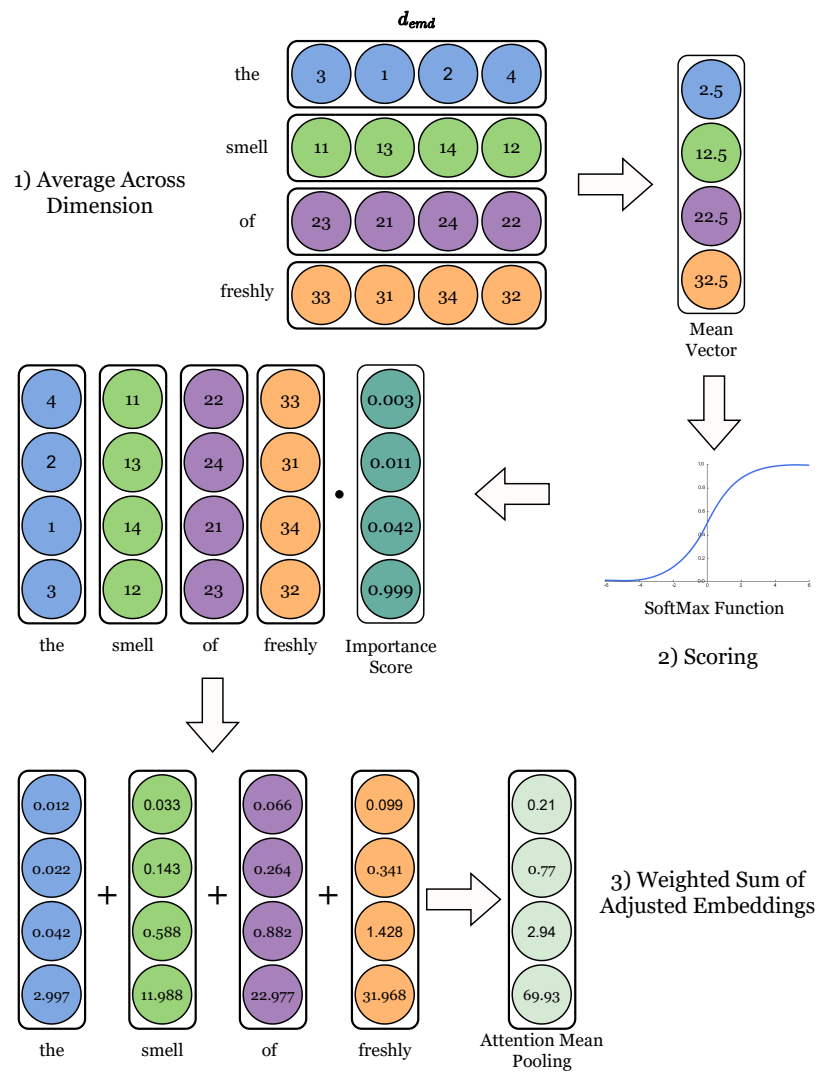


**Figure 11.5** Attention Pooling Visually Explained.

## Comprehensive Results for the Last-Token Approach

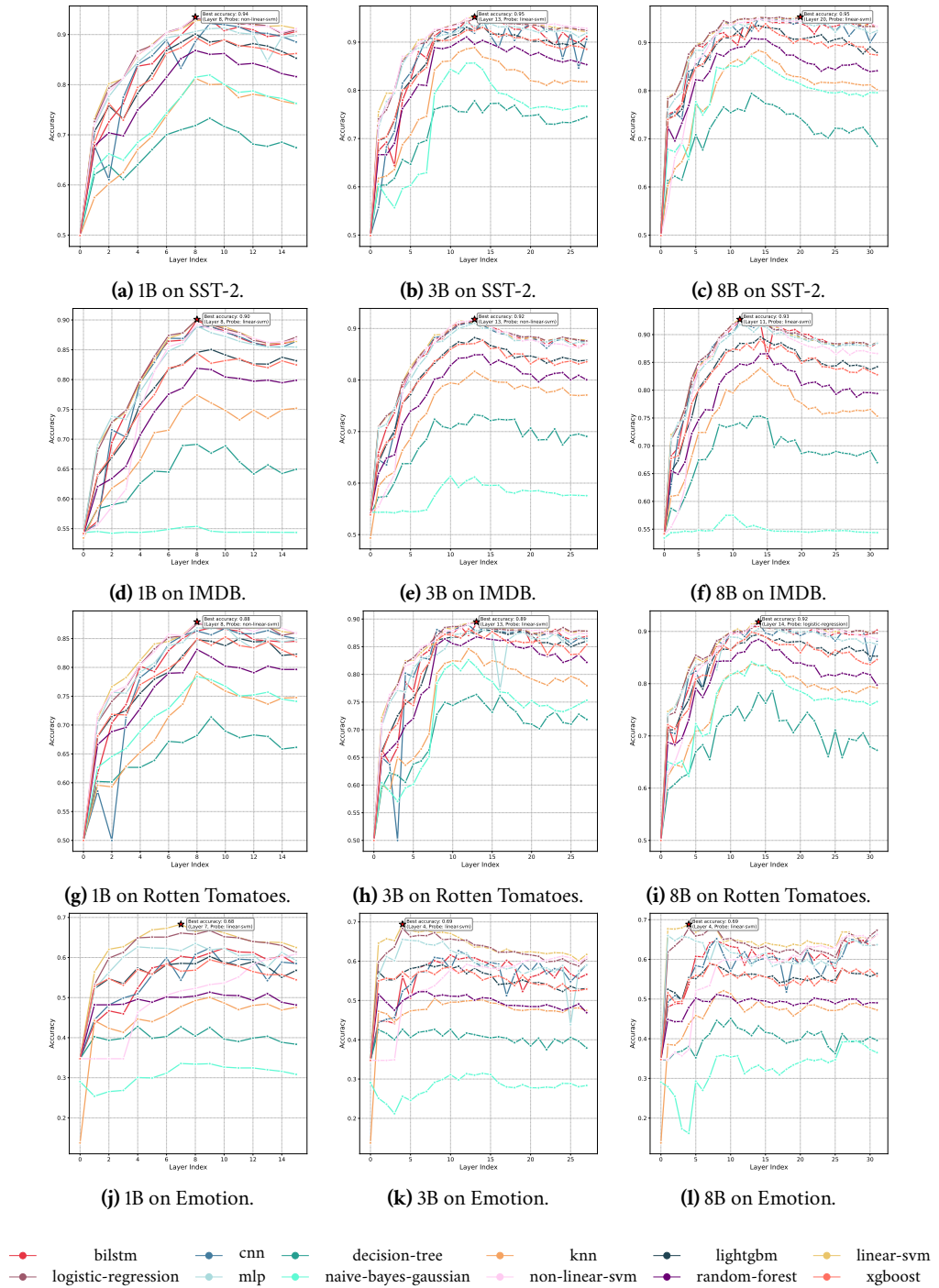


**Figure 11.6** Layer-wise probing accuracy using the last token on SST-2, IMDB, Rotten Tomatoes, and Emotion.

## Layer-wise pooling methods confidence accuracy on SST-2, IMDB, Rotten Tomatoes, and Emotion datasets

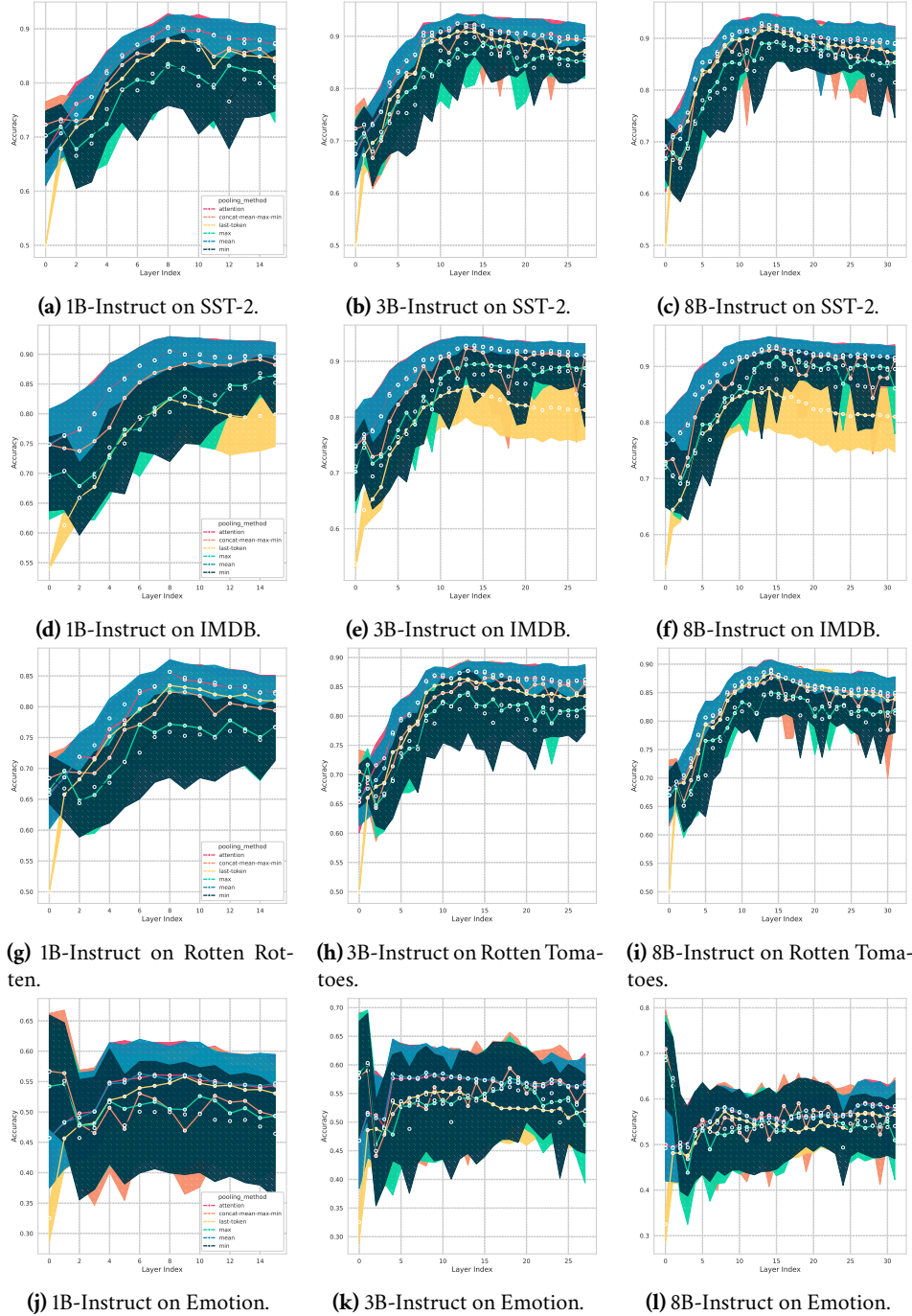


**Figure 11.7** Layer-wise pooling methods confidence accuracy on SST-2, IMDB, Rotten Tomatoes, and Emotion.

## Computational Efficiency of SENTRILLAMA in SST-2, IMDB, Rotten Tomatoes, and Emotion datasets

**Table 11.7** Computational efficiency comparison of SENTRILLAMA, DeBERTa, RoBERTa, and a prompt-based method on the IMDB dataset.

| Model                                 | Peak GPU Usage | Avg. Time per Sample | Throughput (Samples/sec) |
|---------------------------------------|----------------|----------------------|--------------------------|
| <b>Instruct-Llama 3.2 (1B)</b>        | 4.8GB          | 11.33 ms             | 88                       |
| <b>Instruct-Llama 3.2 (3B)</b>        | 9 GB           | 20.83 ms             | 48                       |
| <b>Instruct-Llama 3.1 (8B)</b>        | 18.6 GB        | 35.33 ms             | 28                       |
| <b>SENTRI LLAMA 3.2 (1B)</b>          | 3.8 GB         | 7.13 ms              | 140                      |
| <b>SENTRI LLAMA 3.2 (1B) Instruct</b> | 3.7 GB         | 6.56 ms              | 152                      |
| <b>SENTRI LLAMA 3.2 (3B) Instruct</b> | 3.8 GB         | 4.73 ms              | 211                      |
| <b>SENTRI LLAMA 3.1 (8B) Instruct</b> | 3.6GB          | 6.31 ms              | 158                      |
| <b>DeBERTa V3 Large (418M)</b>        | 2.7 GB         | 22.79 ms             | 44                       |
| <b>RoBERTa Large (355M)</b>           | 2.6 GB         | 8.90 ms              | 119                      |

**Table 11.8** Computational efficiency comparison of SENTRILLAMA, DeBERTa, RoBERTa, and a prompt-based method on the Rotten Tomatoes dataset.

| Model                                 | Peak GPU Usage | Avg. Time per Sample | Throughput (Samples/sec) |
|---------------------------------------|----------------|----------------------|--------------------------|
| <b>Instruct-Llama 3.2 (1B)</b>        | 2.4 GB         | 12.01 ms             | 83                       |
| <b>Instruct-Llama 3.2 (3B)</b>        | 6.2 GB         | 18.41 ms             | 54                       |
| <b>Instruct-Llama 3.1 (8B)</b>        | 15.4 GB        | 21.17 ms             | 50                       |
| <b>SENTRI LLAMA 3.2 (1B)</b>          | 1.6 GB         | 7.08 ms              | 141                      |
| <b>SENTRI LLAMA 3.2 (1B) Instruct</b> | 1.5 GB         | 6.42 ms              | 156                      |
| <b>SENTRI LLAMA 3.2 (3B) Instruct</b> | 1.4 GB         | 4.80 ms              | 208                      |
| <b>SENTRI LLAMA 3.1 (8B) Instruct</b> | 5.7GB          | 19.30 ms             | 97                       |
| <b>DeBERTa V3 Large (418M)</b>        | 845 MB         | 23.32 ms             | 43                       |
| <b>RoBERTa Large (355M)</b>           | 692 MB         | 9.05 ms              | 118                      |



**Table 11.9** Computational efficiency comparison of SENTRILLAMA, DeBERTa, RoBERTa, and a prompt-based method on the Emotion dataset.

| Model                                 | Peak GPU Usage | Avg. Time per Sample | Throughput (Samples/sec) |
|---------------------------------------|----------------|----------------------|--------------------------|
| <b>Instruct-Llama 3.2 (1B)</b>        | 2.4 GB         | 10.97 ms             | 91                       |
| <b>Instruct-Llama 3.2 (3B)</b>        | 6.2 GB         | 17.42 ms             | 57                       |
| <b>Instruct-Llama 3.1 (8B)</b>        | 15.4 GB        | 20.53 ms             | 49                       |
| <b>SENTRI LLAMA 3.2 (1B)</b>          | 673 MB         | 2.70 ms              | 370                      |
| <b>SENTRI LLAMA 3.2 (1B) Instruct</b> | 673 MB         | 2.07 ms              | 483                      |
| <b>SENTRI LLAMA 3.2 (3B) Instruct</b> | 1 GB           | 2.76 ms              | 363                      |
| <b>SENTRI LLAMA 3.1 (8B) Instruct</b> | 2.6GB          | 3.84 ms              | 260                      |
| <b>DeBERTa V3 Large (418M)</b>        | 844 MB         | 22.52 ms             | 44                       |
| <b>RoBERTa Large (355M)</b>           | 692 MB         | 8.50 ms              | 120                      |



## Chapter 12

# **Are the Hidden States Hiding Something? Testing the Limits of Factuality-Encoding Capabilities in LLMs**

---

Factual hallucinations are a major challenge for Large Language Models (LLMs). They undermine reliability and user trust by generating inaccurate or fabricated content. Recent studies suggest that when generating false statements, the internal states of LLMs encode information about truthfulness. However, these studies often rely on synthetic datasets that lack realism, which limits generalization when evaluating the factual accuracy of text generated by the model itself. In this paper, we challenge the findings of previous work by investigating truthfulness encoding capabilities, leading to the generation of a more realistic and challenging dataset. Specifically, we extend previous work by introducing: (1) a strategy for sampling plausible true-false factoid sentences from tabular data and (2) a procedure for generating realistic, LLM-dependent true-false datasets from Question Answering collections. Our analysis of two open-source LLMs reveals that while the findings from previous studies are partially validated, generalization to LLM-generated datasets remains challenging. This study provides a foundation for future research on factuality in LLMs and offers practical guidelines for more effective evaluation.<sup>1</sup>

1. We have made all the code publicly available at <https://github.com/sisinflab/HidingInTheHiddenStates>.

This chapter is based on work published in the Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (ACL'25), titled "Are the Hidden States Hiding Something? Testing the Limits of Factuality-Encoding Capabilities in LLMs".

## 12.1 Introduction

In the last few years, Large Language Models (LLMs) have shown outstanding abilities in natural language processing tasks and beyond [38, 92]. Nevertheless, factual hallucinations [427] represent a significant obstacle, limiting their reliability and hindering their safe deployment in real-world applications [96] such as healthcare [287], education [354], legal advice [89], and language understanding [19, 33]. Hallucinations occur when an LLM generates content that is syntactically coherent but factually inaccurate, decreasing trust in AI systems [158]. Recent research suggests that LLMs may encode internal representations of factuality in their hidden states, indicating an awareness of whether a generated statement is true or false [66]. These efforts led to the development of approaches to evaluate the factual accuracy of the LLM outputs given their internal representations (*factuality "self-evaluation"*). Self-evaluation can be used to identify gaps in the knowledge of an LLM, improving truthfulness and transparency through abstention mechanisms [113], fact verification [361], and self-correction [170]. Azaria et al. [25] suggest that LLMs have “some internal notion as to whether a sentence is true or false, as this information is required for generating (or predicting) following tokens.” Based on this assumption, they propose a neural classifier to discern factual from non-factual statements based on hidden layer activations. However, the datasets used to evaluate the probe present limitations since they contain trivially incorrect statements (e.g., “The zebra uses flying for locomotion”) that easily fail to align with the generative patterns of LLMs. Additionally, the false statements are generated using random substitutions of the true terms with little regard for the plausibility of negative samples. This misalignment not only weakens the generalizability of results but also raises concerns about the applicability of these models to real-world scenarios where false statements may be subtle or nuanced. This study addresses these gaps by generating more plausible datasets (see Figure 12.1) to explore LLM factuality encoding and evaluating refined models. The primary contributions are:

1. We **reproduce** the methodology of Azaria et al. [25] to ensure transparency.
2. We **propose two strategies to generate realistic datasets** and discuss how well the original and newly developed models generalize.

Specifically, we extend their work with two novel dataset creation strategies and design a strategy that better fits the factuality self-evaluation task, introducing:

- A **perplexity-based negative sampling strategy** that enhances the original generation mechanism and leverages the LLM token distribution.
- A **novel strategy to sample realistic LLM-generated facts**, leveraging Question Answering datasets to elicit responses from the LLM.

Through this analysis, we lay the groundwork for more robust factuality assessments and offer practical guidelines for enhancing the reliability of LLMs in diverse applications.

## 12.2 Reproduction of Prior Work: Settings

In their study, Azaria et al. [25] investigate whether LLMs internally represent the factuality of sentences. This section summarizes the dataset generation approach they employed and the specific probing architecture used in their study.

### 12.2.1 Dataset Generation Strategy

To explore whether LLMs internally represent the factuality of statements, the authors constructed a “True-False” dataset of facts labeled as either True or False, covering six disjoint topics: *Cities*, *Inventions*, *Chemical Elements*, *Animals*, *Companies*, and *Scientific Facts*. To generate the dataset, for the first five topics, i.e. with the exception of *Scientific Facts*, the authors selected **true statements** from reliable sources (see Table 12.7 in the Supplementary Material) and produced **false statements**, replacing part of a true statement (e.g. “*Hydrogen has an atomic number of 1*”) with randomly sampled incorrect information (“*Hydrogen has an atomic number of 34*”). Meanwhile, for *Scientific Facts*, they employed ChatGPT (13 Feb 2023) as a generator of true and false sentences, and two human annotators manually verified their correctness. The authors publicly release the dataset, which we refer to as the “*True-False dataset*”. Furthermore, the authors constructed a second dataset using the OPT-6.7b model, which we refer to as the ‘**OPT-Generated Dataset**’. To create this dataset, the model was prompted with a true statement absent from the True-False dataset and then used to generate a subsequent sentence. The responses were manually fact-checked and annotated by three independent human judges. Non-factual responses were filtered out, resulting in a final set of 245 statements.

### 12.2.2 Internal States Analysis via SAPLMA

To investigate whether large language models internally represent the factuality of statements, Azaria et al. [25] developed a probe (Statement Accuracy Prediction based on Language Model Activations) that predicts the factual accuracy of a statement by analyzing the hidden layer activations of an LLM. SAPLMA is a feedforward neural network designed to classify statements as true or false. It consists of three hidden layers (256, 128, 64) and a sigmoid output activation. The model is trained using the Adam optimizer for five epochs without hyperparameter tuning. The authors studied two LLMs, namely OPT-6.7b [421] and Llama 2-7b [351], both consisting of 32 layers. To identify which layers best capture factuality, they trained five SAPLMA models, forwarding each statement in the True-False Dataset as input to the LLMs and extracting the corresponding activation values from the 32nd (last layer), 28th, 24th, 20th, and 16th layers. These activations serve as input for training the classifiers. To ensure generalizability, i.e., making SAPLMA independent of specific topics, the authors adopted a cross-validation strategy using a leave-one-topic-out approach to train the classifier on five topics and test the probe on the held-out topic.

### 12.2.3 Reproducibility Settings

In this section, we provide details on the datasets and experimental settings for reproducing the work of Azaria et al. [25]. Our goal is to answer the Research Question (RQ0): *Can we reproduce the results reported by Azaria et al. [25]?* Although the code is not publicly accessible, the authors made it available upon request.

**Dataset generation.** The code provided by Azaria et al. [25] contains all the necessary material to recreate the entire dataset generation process. However, their template-matching code is influenced by randomness in the generation of false statements, and a random seed is not set. Due to this non-deterministic behavior, recreating their dataset using the original code was unfeasible. However, the authors released their dataset, allowing us to reproduce their exact dataset settings.

**SAPLMA reproducibility.** To reproduce the original results, we trained twenty SAPLMA probes for each of the following layers: the 32nd, 28th, 24th, 20th, and 16th, over 5 epochs, resulting in a total of 100 probes. We employed Llama 2-7b and OPT-6.7b, both using half-precision (16-bit float) parameters, with a default temperature of 0.8 for Llama 2-7b and 1.0 for OPT-6.7b. The hardware used for the experiments

**Table 12.1** Replicated SAPLMA performance on the True-False dataset across the selected layers. The results labeled as ‘Orig.’ are taken from the original work, while those labeled as ‘Repr.’ are the replicated results from this study.

| Layer       | Cities |               | Inventions    |               | Elements      |               | Animals       |               | Companies     |               | Facts         |               | Average       |               |
|-------------|--------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
|             | Llama2 | OPT-6.7b      | Llama2        | OPT-6.7b      | Llama2        | OPT-6.7b      | Llama2        | OPT-6.7b      | Llama2        | OPT-6.7b      | Llama2        | OPT-6.7b      | Llama2        | OPT-6.7b      |
| <b>Last</b> | Orig.  | 0.7574        | 0.7796        | 0.5696        | 0.6814        | 0.5760        | 0.7338        | 0.6022        | 0.6736        | 0.6925        | 0.7444        | 0.6498        | 0.7107        | 0.6449        |
|             | Repr.  | 0.7939        | 0.7836        | 0.7470        | 0.5603        | 0.7057        | 0.5656        | 0.7133        | 0.5984        | 0.6463        | 0.6900        | 0.7894        | 0.7326        | 0.6437        |
| <b>28</b>   | Orig.  | 0.8146        | 0.7732        | 0.7207        | 0.5761        | 0.6767        | 0.5907        | 0.7249        | 0.5777        | 0.6894        | 0.7247        | 0.7662        | 0.6618        | 0.6507        |
|             | Repr.  | 0.8261        | 0.8014        | 0.7221        | 0.5938        | 0.6746        | 0.5931        | 0.7046        | 0.5945        | 0.6860        | 0.7252        | 0.7976        | 0.6639        | 0.6620        |
| <b>24</b>   | Orig.  | 0.8722        | 0.7963        | 0.7816        | 0.6712        | 0.6849        | 0.6211        | 0.7394        | 0.5800        | 0.7094        | 0.7758        | 0.7858        | 0.7622        | 0.6886        |
|             | Repr.  | 0.8619        | 0.8043        | 0.7737        | 0.6604        | 0.6789        | 0.6172        | 0.7415        | 0.6095        | 0.7049        | 0.7844        | 0.7910        | 0.6804        | 0.6927        |
| <b>20</b>   | Orig.  | 0.8820        | <b>0.8125</b> | 0.8459        | <b>0.7268</b> | 0.6950        | 0.6197        | 0.7758        | 0.6058        | 0.8319        | 0.8122        | 0.8053        | <b>0.6819</b> | 0.7098        |
|             | Repr.  | 0.8672        | 0.8118        | 0.8584        | 0.7222        | 0.6761        | <b>0.6218</b> | 0.7736        | <b>0.6208</b> | 0.8254        | <b>0.8160</b> | 0.8065        | 0.6734        | <b>0.7110</b> |
| <b>16</b>   | Orig.  | <b>0.9223</b> | 0.7435        | <b>0.8938</b> | 0.6400        | 0.6939        | 0.5645        | 0.7774        | 0.5800        | 0.8658        | 0.7570        | <b>0.8254</b> | 0.6237        | 0.6515        |
|             | Repr.  | 0.9174        | 0.7554        | 0.8847        | 0.6403        | <b>0.7005</b> | 0.5732        | <b>0.7883</b> | 0.5693        | <b>0.8672</b> | 0.7760        | 0.8104        | 0.6340        | 0.6580        |

was an Intel(R) Core(TM) i7-5820K paired with an NVIDIA RTX 3090 graphics card. The authors do not specify from which token they extract the associated hidden state. However, code inspection led to the identification of the last token as the target state.

**Table 12.2** Replicated baselines performance on the True-False dataset. The results labeled as ‘Orig.’ are taken from the original work, while those labeled as ‘Repr.’ are the replicated results from this study.

| Model             |       | Cities | Inventions | Elements | Animals | Companies | Facts  | Average |
|-------------------|-------|--------|------------|----------|---------|-----------|--------|---------|
| <b>BERT</b>       | Orig. | 0.5357 | 0.5537     | 0.5645   | 0.5228  | 0.5533    | 0.5302 | 0.5434  |
|                   | Repr. | 0.5257 | 0.5611     | 0.5435   | 0.5603  | 0.5302    | 0.5361 | 0.5428  |
| <b>3-shot</b>     | Orig. | 0.5410 | 0.4799     | 0.5685   | 0.5650  | 0.5538    | 0.5164 | 0.5374  |
|                   | Repr. | 0.5416 | 0.4800     | 0.5685   | 0.5652  | 0.5539    | 0.5115 | 0.5368  |
| <b>5-shot</b>     | Orig. | 0.5416 | 0.4799     | 0.5676   | 0.5643  | 0.5540    | 0.5148 | 0.5370  |
|                   | Repr. | 0.5416 | 0.4800     | 0.5676   | 0.5643  | 0.5540    | 0.5082 | 0.5359  |
| <b>It-is-true</b> | Orig. | 0.5230 | 0.5068     | 0.5688   | 0.4851  | 0.6883    | 0.5840 | 0.5593  |
|                   | Repr. | 0.5233 | 0.5046     | 0.5688   | 0.4831  | 0.6875    | 0.5856 | 0.5588  |

**Table 12.3** Reproduced SAPLMA performance on the **OPT-Generated Dataset** (Section 12.2.1). The results labeled as ‘Orig.’ are taken from the original work, while those labeled as ‘Repr.’ are the reproduced results.

| Layer               |       | Accuracy | AUC    | Accuracy with Optimal Threshold | Average Threshold |
|---------------------|-------|----------|--------|---------------------------------|-------------------|
| <b>Last-layer</b>   | Orig. | 0.6187   | 0.7587 | 0.7052                          | 0.8687            |
|                     | Repr. | 0.6406   | 0.7720 | 0.7264                          | 0.8910            |
| <b>28th-layer</b>   | Orig. | 0.6362   | 0.7614 | 0.7134                          | 0.8838            |
|                     | Repr. | 0.6410   | 0.7686 | 0.7203                          | 0.8276            |
| <b>24th-layer</b>   | Orig. | 0.6134   | 0.7435 | 0.6988                          | 0.8801            |
|                     | Repr. | 0.6206   | 0.7496 | 0.6973                          | 0.8500            |
| <b>20th-layer</b>   | Orig. | 0.6029   | 0.7182 | 0.6587                          | 0.9063            |
|                     | Repr. | 0.5965   | 0.7183 | 0.6669                          | 0.8868            |
| <b>Middle-layer</b> | Orig. | 0.5566   | 0.6610 | 0.6500                          | 0.8123            |
|                     | Repr. | 0.5579   | 0.6760 | 0.6468                          | 0.7948            |
| <b>BERT</b>         | Orig. | 0.5115   | 0.5989 | 0.5705                          | 0.9403            |
|                     | Repr. | 0.5522   | 0.6092 | 0.5689                          | 0.7939            |
| <b>3-shot</b>       | Orig. | 0.5041   | 0.4845 | -                               | -                 |
|                     | Repr. | 0.5041   | 0.4845 | -                               | -                 |
| <b>5-shot</b>       | Orig. | 0.5125   | 0.4822 | -                               | -                 |
|                     | Repr. | 0.5125   | 0.4822 | -                               | -                 |



## 12.3 Experimental Reproducibility Results

To answer **RQ0**, we report the results for the reproduction of the experiments in Tables 12.1, 12.2 and 12.3. The results labeled with ‘Orig.’ are retrieved from the original work, while the ones achieved in the reproducibility study are labeled with ‘Repr.’

### 12.3.1 Reproduction of SAPLMA Results on the True-False Dataset

Table 12.1 reports SAPLMA’s performance on the True-False dataset across different layers of Llama 2-7b and OPT-6.7b for six categories: Cities, Inventions, Elements, Animals, Companies, and Facts. Results indicate that Llama 2-7b consistently outperforms OPT-6.7b across all layers and categories. Middle layers (16, 20, 24) achieve the highest performance, while accuracy declines toward the final layer. The reproduced results closely align with the original findings, with minor deviations observed across specific categories and layers. Moreover, although OPT-6.7b shows greater variability, the ranking of layers remains consistent with the original work. Both experiments confirm that factuality information is more effectively encoded in the middle layers (16–24) than in the final layer. Additionally, we reproduce the baseline used by the authors to compare SAPLMA. Specifically, their baseline includes a trained SAPLMA on BERT activations, a few-shot approach where the LLM is prompted with a sentence and asked to label it as ‘True’ or ‘False,’ and an ‘It-is-true’ test. In this test, the LLMs were asked: *Is it true* that X? and *Is it false* that X?, where X is a dataset sample. A response was considered correct if the model assigned a higher probability to the ‘True’ token. Table 12.2 summarizes the baseline performance. BERT achieves the highest average performance among non-prompted methods, demonstrating its effectiveness in factual classification, and the results indicate high reproducibility. The It-is-true baseline yields the highest average performance, particularly in the ‘Companies’ topic. The results demonstrate consistent reproducibility.

In general, the reproduction on the True-False Dataset is considered successful based on the following observations: (i) the overall performance trends remain consistent, with Llama 2-7b outperforming OPT-6.7b; (ii) observed deviations are minor and do not indicate fundamental inconsistencies; (iii) the relative ranking of layers remains unchanged, reinforcing previous findings; and (iv) baseline methods

retain their rankings, confirming the validity of the original results.

### 12.3.2 Reproduction of the Results on the OPT-Generated Dataset

Table 12.3 presents the results of the reproduced experiments on the OPT-Generated Dataset. Performance is evaluated using Accuracy, AUC, Accuracy with an Optimal Threshold (selected by estimating it from a held-out validation set), and the Average Optimal Threshold. The final layers, specifically the 28th and final layers, outperform the middle and lower layers in terms of Accuracy and AUC. The reproduced results closely align with the original findings, exhibiting only minor variations. Accuracy with an optimal threshold consistently exceeds raw accuracy, suggesting that tuning the decision boundary improves performance.

Regarding baselines, BERT exhibits lower accuracy compared to the LLMs' last layers, with slight improvements over the original results. Notably, the 3- and 5-shot prompting results were identical between the original and reproduced experiments.

**RQ0:** *Can we reproduce the results reported by Azaria et al. [25]?*

**This reproducibility study demonstrates a high degree of alignment with the original results**, confirming the validity of previous findings. It shows that the ranking and trends remain unchanged, reinforcing the robustness of the results.

## 12.4 Novel Dataset Generation Strategies for Factuality Self-Evaluation

This section introduces two novel dataset generation strategies to investigate LLM factuality self-evaluation. To contextualize our approach, we first examine the limitations of the True-False dataset by Azaria et al. [25]. While it provides a structured framework for evaluating LLMs, its construction imposes constraints that may limit generalizability of the findings. Specifically, true and false statements are derived from tabular data using predefined templates. We argue that this approach suffers from several limitations:

- **Adherence to predefined templates:** The use of fixed templates limits the linguistic expressiveness of the dataset, potentially constraining the probe classifier's ability to generalize beyond rigidly structured statements (e.g., *<company> operates in the industry of <industry>*).
- **Distribution misalignment:** The statements are constructed from tabular data

rather than generated by the LLM itself. As a result, the dataset may not align with the LLM’s generative distribution. For instance, a niche true fact in the dataset –but unknown by the model– could have high perplexity for the LLM, undermining the study’s core premise: evaluating an LLM’s intrinsic ability to “judge” its false claims.

- **Lack of consideration for the LLM’s knowledge state:** LLMs exhibit uneven factual knowledge based on their pretraining data, with strengths in some domains and gaps in others. A model can assess a statement factuality only if it has prior exposure to it. The dataset does not account for these inconsistencies: it evaluates whether an LLM can detect factual errors without considering whether the model actually possesses knowledge of the fact.
- **Differences in cardinality:** Some properties in the dataset have fewer admissible values, making certain facts easier to evaluate. For example, a statement like “*<element> appears in its standard state as \_*” has fewer possible values (i.e., {*Solid, Liquid, Gas*}) compared to statements like “*<city> is a city in \_*” that involve a broader set. This imbalance in complexity may bias the evaluation process.

These limitations impact the interpretability of results when evaluating an LLM’s internal representations of factuality. To ensure a more realistic assessment of an LLM’s self-evaluation of factuality, we propose two strategies to address these issues. The first strategy samples statements from tabular data to better align with the LLM’s generative predictions. The second strategy involves sampling LLM-generated facts as answers to questions from a well-known Question Answering dataset.

### 12.4.1 Perplexity-based Dataset Construction

This section presents a novel dataset generation strategy to address limitations in the Azaria et al. [25] True-False dataset, particularly distribution misalignment and implausible negative samples. To improve negative sampling based on random property-object substitutions, we introduce a **perplexity-guided probabilistic sampling** method, which re-weights false statements based on perplexity for better alignment with LLM output distributions. Since perplexity depends on the model, the same LLM under evaluation must generate false statements to ensure consistency, resulting in a model-dependent dataset tailored to the characteristics and biases of the LLM being studied.

True statements are initially constructed, as in Azaria et al. [25], by directly insert-

ing correct entity-property pairs into pre-defined sentence templates. Regarding the false statements generation, we proceed as follows: (i) for each true statement, all unique alternative property values are gathered from the entire dataset; (ii) a candidate sentence is created for each alternative property by inserting it into the corresponding template; (iii) the target LLM (i.e., OPT-6.7b or Llama 2-7b) computes the perplexity of each candidate sentence. This perplexity score serves as a plausibility metric, with lower perplexity indicating a more plausible (yet incorrect) statement.

Given a true statement, we define  $C$  as the set of potential candidate sentences, which includes the true statement. Furthermore,  $C' \subset C$  is defined as the subset of candidate false sentences (i.e. excluding the true statement). Candidates  $c \in C$  are ranked based on their perplexity scores, with lower scores indicating higher plausibility.

Since perplexity can be interpreted as a measure of plausibility, we operate under the assumption that an LLM possesses factual knowledge about a fact if the fact is assigned a "sufficiently low perplexity". Conversely, a high perplexity score for the true statement suggests that the LLM lacks knowledge of the fact. Given the limitations discussed earlier, we aim to evaluate the LLM ability to discern between true and false statements when it possesses the relevant knowledge. Therefore, we exclude instances where the LLM exhibits limited knowledge about the true statement: if the true statement does not rank among the lowest  $k$  perplexity candidates, the generation of that instance is discontinued, and the next true fact is considered. In practice, we define  $k$  as  $k = \alpha|C|$ , where  $\alpha$  is a hyperparameter ( $0 < \alpha < 1$ ). This accounts for the varying cardinality of the property ranges in the dataset, ensuring that the threshold for "sufficiently low perplexity" is adjusted based on the number of possible values for a given property.

In addition, we want to simulate a real hallucination scenario where the LLM is uncertain between the true fact and plausible alternatives: given the perplexity score function  $PP(\cdot)$ , all false candidates  $c$  with a perplexity score  $PP(c) < (1 + \beta)PP(true)$ , where  $0 < \beta < 1$  is a hyperparameter, are considered, resulting in a reduced set of candidates  $C^*$ . A min-max normalization is applied to their perplexity,

$$\text{NormPP}(c) = \frac{PP(c) - \min_{c \in C} PP(c)}{\max_{c \in C} PP(c) - \min_{c \in C} PP(c)}, \quad (12.1)$$

The normalized perplexities are transformed using a plausibility score function

$s(\cdot)$ , i.e., lower perplexity scores result in higher plausibility scores. The scores are then normalized to ensure that they sum to 1 and are treated as a probability distribution over the candidates:

$$s(c) = e^{-\text{NormPP}(c)}, \quad P(c_i) = \frac{s(c_i)}{\sum_{c_j \in C^*} s(c_j)}. \quad (12.2)$$

The normalization guarantees that  $P(c_i)$  values are suitable for sampling. Finally, a mixture of top-k and nucleus sampling [152] is employed to sample the candidate for insertion into the template. Specifically, we apply top-k and nucleus sampling sequentially: we select the top-k highest-scoring candidates and then refine this set by choosing the smallest subset whose cumulative probability reaches a predefined threshold. This process generates a coherent yet factually incorrect statement that is more realistic and closely aligned with the LLM internal token prediction patterns. In Section 12.5.1 we detail the selected values for the hyperparameters  $\alpha$ ,  $\beta$ ,  $k$ , and  $p$ .

## 12.4.2 LLM-Generated Dataset Construction

The strategy in Section 12.4.1 constructs a balanced true-false dataset from tabular data but has inherent limitations. While enabling direct comparison with Azaria et al. [25], it restricts generative models to rigid templates, limiting expressiveness. Additionally, its reliance on fixed candidate sets can lead to easily classifiable false statements when the set is small. Finally, template-based sampling reduces diversity, likely due to token bias: we found LMs consistently assigning lower perplexity to certain property-object pairs, regardless of the subject. To more realistically assess an LLM factuality self-evaluation, we propose generating both true and false statements directly from the model, overcoming the limitations of template-based approaches. This involves (a) a method to elicit diverse factoid statements from the LLM, and (b) a strategy to annotate statement veracity, addressing biases and inconsistencies in the process. Consider a Question Answering dataset composed of  $N$  questions,  $D_{QA} = \{(q_i, a_i)\}_{i=1}^N$ , where each question  $q_i$  has a corresponding ground-truth answer  $a_i$ . Given a LLM  $\mathcal{M}$ , we prompt it with each question  $K$  times, yielding a set of generated answers. This results in an extended dataset  $D_{QA}^{\mathcal{M}} = \{(q_i, a_i, \{a_{i,k}^{\mathcal{M}}\}_{k=1}^K)\}_{i=1}^N$ . Following LLM-as-judge [54, 132] practices, the LLM-generated answers in  $D_{QA}^{\mathcal{M}}$  can be annotated using an oracle LLM, which we assume

is able to evaluate the veracity of each answer  $a_{i,k}^M$  given the ground-truth answer  $a_i$  and the question  $q_i$ . This operation results in an annotated dataset

$$\hat{D}_{QA}^M = \{(q_i, a_i, \{a_{i,k}^M, v_{i,k}^M\}_{k=1}^K)\}_{i=1}^N, \quad (12.3)$$

where  $v_{i,k}$  is a veracity label assigned by the oracle, indicating whether the generated answer  $\hat{a}_{i,k}$  is correct ( $v_{i,k} = 1$ ) or incorrect ( $v_{i,k} = 0$ ).

Increasing  $K$  enhances the reliability of responses by offering a more accurate evaluation of the LLM knowledge state regarding a question. This evaluation can lead to three possible outcomes: (i) a high proportion of correct answers suggests the LLM fully understands the facts; (ii) a high proportion of incorrect answers indicates the LLM lacks or has partial knowledge, preventing correct responses; (iii) a mix of correct and incorrect answers implies knowledge with a tendency toward hallucination. This study assumes that an LLM can only encode factuality regarding a generated fact if it has some knowledge about it. Therefore, we focus exclusively on the third scenario. This scenario also naturally produces a balanced dataset, including true and false variations of the same fact. We define the correct answer ratio  $p_i^M = \frac{1}{K} \sum_{k=1}^K v_{i,k}^M$ . We consider questions whose  $p_i$  is around 0.5 with a tolerance hyperparameter  $\tau$ , that is  $|p_i^M - 0.5| < \tau$ . The dataset is obtained by selecting the answers and their veracity labels satisfying the condition:

$$D_{Facts}^M = \{(a_{i,k}^M, v_{i,k}^M) : |p_i^M - 0.5| < \tau\}_{i=1}^N. \quad (12.4)$$

## 12.5 Experiments and Discussion

This section describes the experimental setup used to extend the prior investigation, presents the results, and discusses their implications. We address the following Research Questions:

**RQ1** *Are probing classifiers trained as in Azaria et al. [25] capable of generalizing to True-False sentences with similar perplexity?*

**RQ2** *Can the same probes generalize to facts generated by LLMs?*

### 12.5.1 Dataset Generation Setup

To generate the datasets for training the probe and alleviate the limitations of Azaria et al. [25], we employ the perplexity-based sampling procedure described in Sec-

**Table 12.4** Novel generation of our True-False dataset, following the approach described in Section 12.4.1, including number of sentences and percentage of true samples.

| Dataset           | Llama 2-7b |          | OPT-6.7b  |          |
|-------------------|------------|----------|-----------|----------|
|                   | Sentences  | (%) True | Sentences | (%) True |
| <b>Cities</b>     | 674        | 50       | 756       | 50       |
| <b>Inventions</b> | 336        | 50       | 202       | 50       |
| <b>Elements</b>   | 118        | 50       | 220       | 50       |
| <b>Animals</b>    | 116        | 50       | 114       | 50       |
| <b>Companies</b>  | 326        | 50       | 310       | 50       |

tion 12.4.1. Specifically, we set the needed hyperparameters as follows:  $\alpha = 0.1$ ,  $\beta = 0.1$ ,  $k = 10$  and  $p = 0.9$ . We excluded the *Scientific Facts* topic from our perplexity-based sampling procedure, as its original version was generated by ChatGPT and not from tabular data (additional details in Supplementary Material 12.7). Table 12.4 summarizes the statistics for the two refined True-False datasets, while Figure 12.1 shows the differences in average perplexity between the original and the proposed datasets.

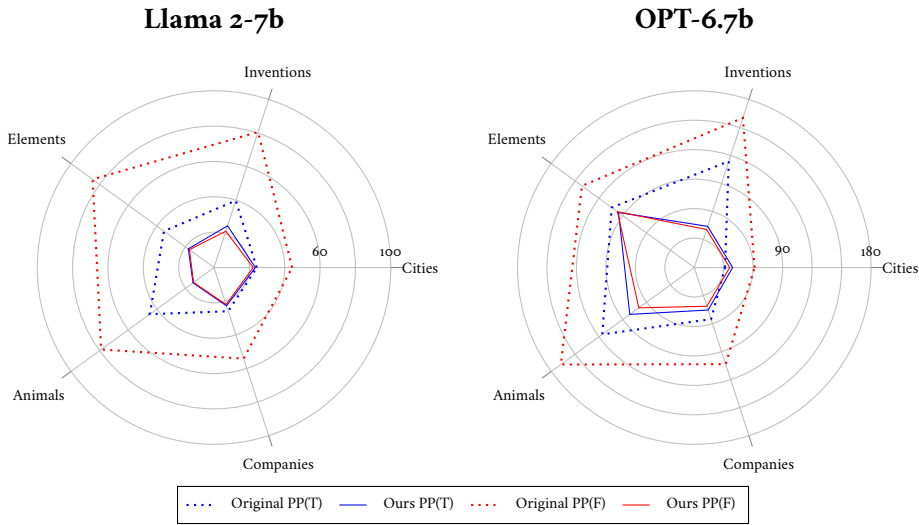


**Figure 12.1** Comparison of average perplexity scores for Llama 2-7b and OPT-6.7b for the Original dataset by Azaria et al. [25] and our refined version. Lower perplexity indicates that the sentences are more likely to have been generated by the model. PP(T/F) denotes average perplexity of true/false sentences.

We construct an LLM-generated dataset following the procedure described in

Section 12.4.2. Our experiments use TriviaQA [173], a dataset of question-answer pairs collected from 14 trivia and quiz-league websites. Given the limited computational resources, we limit our focus to the validation split of TriviaQA Wikipedia questions. We build a True-False dataset from two additional QA datasets, SQuAD 2.0 [295] and TruthfulQA [217], employing the same approach, which we describe in Supplementary Material 12.7. For each LLM in our analysis, we set  $K = 10$  and  $\tau = 0.1$ , and we discard the answers composed of less than 5 tokens to improve self-consistency. For annotation, we use **GPT-4o mini** with a 3-shot learning strategy, where three examples are provided. More details are included in Supplementary Material 12.7, and Table 12.8 reports the dataset statistics.

### 12.5.2 Impact of Perplexity-Based Sampling on SAPLMA Accuracy

We assess the impact of our perplexity-based sampling strategy by training the SAPLMA probe classifier separately on the original True-False dataset from Azaria et al. [25] and our refined version. As shown in Table 12.5, classifiers tested on our refined dataset achieve the highest accuracy in deeper layers. For Llama 2-7b, Layer 16 shows the most consistent results, achieving the highest average accuracy across all tested configurations when trained on our dataset. Although accuracy fluctuates across different topics, overall performance remains largely consistent between the training configurations. Comparing the results with the original results in Table 12.1, it emerges that classifiers trained on the original dataset achieve higher accuracy when tested on the same dataset. This is particularly evident in the ‘Animals’ topic. However, it is important to consider that the perplexity values in our refined dataset are significantly lower than those in the original dataset for both true and false statements (see 12.1). Additionally, our refined dataset features lower and more closely aligned perplexity scores between true and false statements, with false statements often yielding even lower perplexity values than true ones. This finding implies the validity of Azaria et al. hypothesis, which states that **factuality information is encoded in LLM hidden states**, although it might not be immediately evident from the model predicted probabilities.

**RQ1:** *Are probing classifiers trained as in Azaria et al. [25] capable of generalizing to True-False sentences with similar perplexity?*

Surprisingly, the results between the probes trained on our refined dataset and



**Table 12.5** Accuracy obtained training SAPLMA on the original or refined True-False dataset and evaluating on the refined version. “Orig.” and “Novel” denote the original and refined datasets. **Bold** highlights the best layer-training dataset combination for each model-topic pair.

| Layer | Training Data | Cities        |               | Inventions    |               | Elements      |               | Animals       |               | Companies     |               | Average       |               |
|-------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
|       |               | Llama 2-7b    |               | OPT-6.7b      |               | Llama 2-7b    |               | OPT-6.7b      |               | Llama 2-7b    |               | OPT-6.7b      |               |
|       |               | Orig.         | Novel         | Orig.         | Novel         | Orig.         | Novel         | Orig.         | Novel         | Orig.         | Novel         | Orig.         | Novel         |
| last  | Orig.         | 0.6882        | 0.5724        | 0.6409        | 0.5094        | 0.6314        | 0.5482        | 0.5685        | 0.5259        | 0.6290        | 0.6984        | 0.6316        | 0.5709        |
|       | Novel         | 0.6365        | 0.6143        | 0.6101        | 0.5144        | 0.5623        | 0.5293        | 0.5461        | 0.4846        | 0.7414        | 0.7211        | 0.6311        | 0.5720        |
| 28    | Orig.         | 0.7056        | 0.5870        | 0.7001        | 0.5178        | 0.6161        | 0.5832        | 0.6013        | <b>0.5566</b> | 0.7079        | 0.7234        | 0.6662        | 0.5936        |
|       | Novel         | 0.5091        | 0.6057        | 0.6591        | 0.5473        | 0.5665        | 0.5655        | 0.6052        | 0.4662        | 0.7061        | 0.7065        | 0.6092        | 0.5811        |
| 24    | Orig.         | 0.8286        | 0.7026        | 0.7250        | 0.6022        | 0.6432        | 0.5918        | 0.6310        | 0.5439        | 0.7121        | 0.7366        | 0.7080        | 0.6354        |
|       | Novel         | 0.6025        | 0.6710        | 0.6609        | 0.6248        | 0.5763        | 0.5993        | 0.5836        | 0.4868        | 0.7868        | 0.7366        | 0.6420        | 0.6225        |
| 20    | Orig.         | 0.8272        | 0.7313        | 0.7741        | <b>0.6230</b> | 0.6492        | <b>0.6255</b> | 0.5832        | 0.5075        | 0.7583        | 0.7502        | 0.7184        | <b>0.6475</b> |
|       | Novel         | 0.7382        | <b>0.7528</b> | 0.6973        | 0.6131        | 0.6051        | 0.5986        | 0.6190        | 0.4807        | <b>0.8270</b> | <b>0.7566</b> | 0.6973        | 0.6404        |
| 16    | Orig.         | 0.8941        | 0.6433        | 0.7888        | 0.5698        | <b>0.6801</b> | 0.5930        | 0.5836        | 0.3816        | 0.7768        | 0.7426        | 0.7447        | 0.5860        |
|       | Novel         | <b>0.9301</b> | 0.7505        | <b>0.7961</b> | 0.5644        | 0.6623        | 0.5568        | <b>0.6319</b> | 0.5307        | 0.8265        | 0.7231        | <b>0.7694</b> | 0.6151        |

**Table 12.6** Performance of SAPLMA on a fact dataset generated from TriviaQA. The original topic-wise leave-one-out strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models.

| Dataset              |       | Threshold = 0.5 |      |      |      |      | Optimal Threshold |      |      |      |      |
|----------------------|-------|-----------------|------|------|------|------|-------------------|------|------|------|------|
|                      |       | last            | 28   | 24   | 20   | 16   | last              | 28   | 24   | 20   | 16   |
| <b>billturnbull</b>  | Llama | .561            | .576 | .618 | .621 | .628 | .547              | .581 | .618 | .634 | .639 |
|                      | OPT   | .547            | .558 | .591 | .551 | .537 | .530              | .558 | .547 | .530 | .499 |
| <b>derby*</b>        | Llama | .568            | .581 | .575 | .596 | .617 | .560              | .565 | .559 | .571 | .597 |
|                      | OPT   | .553            | .564 | .584 | .587 | .572 | .562              | .566 | .583 | .586 | .580 |
| <b>quiz4free</b>     | Llama | .564            | .547 | .523 | .561 | .589 | .548              | .523 | .521 | .547 | .573 |
|                      | OPT   | .559            | .559 | .575 | .581 | .560 | .544              | .530 | .533 | .546 | .541 |
| <b>quizguy</b>       | Llama | .578            | .585 | .588 | .607 | .635 | .576              | .587 | .601 | .595 | .637 |
|                      | OPT   | .579            | .583 | .589 | .590 | .584 | .559              | .555 | .565 | .559 | .548 |
| <b>triviabug</b>     | Llama | .494            | .518 | .521 | .525 | .538 | .508              | .500 | .530 | .542 | .545 |
|                      | OPT   | .620            | .624 | .607 | .596 | .528 | .632              | .635 | .605 | .598 | .553 |
| <b>businessballs</b> | Llama | .566            | .558 | .565 | .574 | .582 | .558              | .551 | .555 | .564 | .575 |
|                      | OPT   | .559            | .558 | .578 | .570 | .553 | .545              | .547 | .573 | .562 | .551 |
| <b>jetpunk</b>       | Llama | .587            | .627 | .620 | .643 | .654 | .543              | .580 | .550 | .554 | .601 |
|                      | OPT   | .606            | .612 | .614 | .596 | .621 | .618              | .618 | .621 | .605 | .619 |
| <b>odquiz</b>        | Llama | .551            | .536 | .546 | .562 | .573 | .537              | .525 | .532 | .550 | .560 |
|                      | OPT   | .560            | .573 | .583 | .583 | .542 | .551              | .566 | .571 | .569 | .521 |
| <b>quiz-zone</b>     | Llama | .556            | .557 | .558 | .565 | .611 | .541              | .549 | .555 | .569 | .615 |
|                      | OPT   | .569            | .570 | .582 | .592 | .552 | .537              | .550 | .535 | .580 | .508 |
| <b>quizballs</b>     | Llama | .603            | .575 | .572 | .578 | .571 | .602              | .561 | .561 | .570 | .553 |
|                      | OPT   | .558            | .565 | .574 | .571 | .540 | .550              | .549 | .564 | .564 | .539 |
| <b>quizwise</b>      | Llama | .560            | .565 | .579 | .609 | .618 | .563              | .551 | .574 | .605 | .619 |
|                      | OPT   | .560            | .563 | .565 | .577 | .540 | .555              | .556 | .552 | .575 | .537 |
| <b>sfquiz</b>        | Llama | .568            | .554 | .554 | .559 | .575 | .564              | .554 | .559 | .566 | .583 |
|                      | OPT   | .584            | .591 | .589 | .590 | .547 | .570              | .582 | .577 | .573 | .536 |
| <b>triviacountry</b> | Llama | .536            | .554 | .559 | .556 | .566 | .506              | .530 | .536 | .524 | .549 |
|                      | OPT   | .536            | .551 | .550 | .587 | .534 | .449              | .476 | .472 | .511 | .472 |
| <b>wrexham**</b>     | Llama | .570            | .563 | .569 | .553 | .565 | .567              | .548 | .561 | .545 | .566 |
|                      | OPT   | .548            | .573 | .578 | .584 | .554 | .535              | .552 | .574 | .579 | .571 |
| <b>Average</b>       | Llama | .562            | .564 | .568 | .579 | .594 | .552              | .550 | .558 | .567 | .587 |
|                      | OPT   | .567            | .575 | .583 | .583 | .555 | .553              | .560 | .562 | .567 | .587 |

\*: derby is adopted as abbreviation of derbyshirepubquizleague

\*\* : wrexham is adopted as abbreviation of wrexhamquizleague

the probes trained on the original dataset are mostly comparable. This proves that **the probes can generalize even when the train-test datasets have different perplexity.**

### 12.5.3 Generalization of SAPLMA on LLM-generated Sentences

We extend our experiments with the SAPLMA classifier to a new setting, where both training and evaluation are conducted on sentences generated by an LLM and sourced from TriviaQA, leveraging the procedure described in Section 12.4.2. Table 12.6 reports the performance of SAPLMA on a set of factual statements extracted from TriviaQA. The results indicate that the currently used probes are inadequate for factuality self-assessment in real-world scenarios, as the observed accuracy does not reach a noteworthy threshold. Furthermore, following the suggestion of Azaria et al., we optimize the classification threshold; however, this adjustment yields no significant improvement in performance. For completeness, the interested reader may find the result obtained on the same dataset when training on the original True-False dataset of Azaria et al. in Supplementary Material 12.7. In addition, we report results on SQuAD 2.0 and TruthfulQA in Supplementary Material 12.7, which align closely with the results observed on TriviaQA, further supporting the consistency of our findings.

**RQ2:** *Can the same probes generalize to facts generated by LLMs?*

In summary, this experiment partially contradicts the findings of Azaria et al. [25]: **the trained probes are not capable of providing good generalization to an LLM-generated dataset**, even when the accuracy threshold is tuned. The motivation could stem from the nature of the dataset, as TriviaQA contains open-domain questions that result in more nuanced facts than the ones in the original True-False dataset (Section 12.2) or the OPT-generated dataset (Section 12.3.2).

*We believe that further research is needed to enhance the effectiveness of factuality self-assessment techniques, particularly in settings involving LLM-generated content. Promising research directions may be leveraging datasets that are closely aligned with the distribution of LLM-generated text and exploring alternative techniques such as uncertainty-aware classification.*

## 12.6 Related Work

Probing techniques have become central for the layer-wise interpretation of deep learning models [13]. This approach was then extended to Large Language Models (LLMs) to assess LLMs ability to encode syntax and semantics [80, 346]. Specifically, among the different applications, a technique that emerges is **self-evaluation**, defined as *a model's ability to assess the accuracy of its own outputs* [174]. Among the various works, notable research by Kadavath et al. [174] explored estimating a well-calibrated  $P(\text{True})$  directly from output probabilities to reflect answer accuracy. Azaria et al. [25] demonstrate that LLMs are capable of detecting false claims in synthetic true-false datasets. Building upon this initial study, several works [51, 197, 246] investigate the generalization capabilities of probes in detecting hallucinations in LLMs. While these studies offer valuable insights into how factual knowledge is encoded in LLMs, they stop short of examining whether LLMs can assess the factuality of their own generations. Instead, they primarily evaluate models on artificially constructed or perturbed datasets, rather than on claims produced by the LLMs themselves. Orgad et al. [269] explore error detection and hallucination mitigation in Question Answering by probing the internal representations of question-answer pairs. Their study shows that while LLMs encode factuality signals, these signals do not generalize well across task-specific datasets, suggesting that factuality encoding is task-dependent rather than universal. Gekhman et al. [126] show that LLMs can internally encode knowledge, detectable by probing the hidden states, yet still fail to express it in their generated output. Chen et al. [66] introduce the INSIDE framework, which detects hallucinations using the EigenScore metric to measure self-consistency across multiple LLM outputs for a single input. Similarly, Zhang et al. [424] leverage a probe model, PINOSE, trained via offline consistency-checking, to perform online hallucination detection. Although INSIDE and PINOSE primarily focus on self-consistency in internal representations across multiple generated responses, they do not interpret hidden states or look for explicit factuality encoding.

## 12.7 Summary

In this paper, we investigate the factuality-encoding capabilities of LLMs. Our work replicates the methodology of Azaria et al. [25] to ensure reproducibility and extend

their approach with two novel dataset construction strategies: perplexity-based negative sampling and fact generation based on QA datasets. We applied these strategies to analyze two open-source LLMs, and found that although the findings from previous studies are partially validated even on more challenging synthetic datasets, transferring these findings to LLM-generated datasets proves difficult. This study paves the way for more reliable LLM evaluations and offers practical guidelines for improving model transparency and trustworthiness in real-world applications.



# Supplementary Material

---

## Dataset Statistics

### True-False Dataset [25] Statistics

Table 12.7 provides a summary of the number of sentences and the distribution of true and false statements across each topic in the original True-False dataset [25] on which we base our replication experiments.

### LLM-Generated Trivia Facts Dataset

Table 12.8 illustrates the number of annotated facts extracted from TriviaQA [173] employing the procedure described in section 12.4.2. Statistics are presented for each of the 14 question sources. Table 12.9 reports examples of generated facts for the Llama 2-7b and OPT-6.7b models.

### Perplexity-Based Refinement of the True-False Dataset [25]

We base our dataset generation strategy described in Section 12.4 on the same tabular data employed by Azaria et al., which was made available upon request. The properties used for sampling are analogous to the ones employed by the original authors in their dataset generation. However, for the *Cities* topic, we restrict the analysis to facts like "<city> is city in <country>." Differently from the original authors, we avoid generating facts such as "<city> is the name of a city/country" having only two possible values, potentially resulting in too many easy-to-classify samples. Table 12.10 reports examples of false sentences generated following the perplexity-based sampling procedure described in Section 12.4.1 and the Llama 2-7b model.

**Table 12.7** True-False dataset categories, original sources, and label splits from [25].

| Dataset           | Sentences | Source                                    | (%) True |
|-------------------|-----------|-------------------------------------------|----------|
| Cities            | 1458      | SimpleMaps dataset                        | 50       |
| Inventions        | 876       | Wikipedia’s list of inventors             | 53       |
| Chemical Elements | 930       | PubChem’s periodic table                  | 50       |
| Animals           | 1008      | National Geographic Kids                  | 50       |
| Companies         | 1200      | Forbes Global 2000 List 2022: The Top 200 | 50       |
| Scientific Facts  | 612       | ChatGPT and human annotators              | 50       |

**Table 12.8** Summary of the dataset obtained by extracting factoid sentences from TriviaQA [173] Wikipedia validation split, following the procedure described in Section 12.4.2.

| Dataset       | Llama 2-7b |          | OPT-6.7b  |          |
|---------------|------------|----------|-----------|----------|
|               | Sentences  | (%) True | Sentences | (%) True |
| triviacountry | 118        | 49.15%   | 118       | 49.15%   |
| wwwodquiz     | 700        | 49.71%   | 525       | 49.71%   |
| triviabug     | 97         | 50.52%   | 88        | 51.14%   |
| derby*        | 342        | 50.00%   | 325       | 50.00%   |
| quiz-zone     | 187        | 48.13%   | 239       | 50.63%   |
| businessballs | 433        | 50.84%   | 443       | 49.43%   |
| wrexham**     | 216        | 50.00%   | 276       | 50.72%   |
| sfquiz        | 1054       | 49.80%   | 1085      | 50.39%   |
| quizwise      | 565        | 49.91%   | 704       | 50.42%   |
| billturnbull  | 216        | 49.54%   | 105       | 50.48%   |
| jetpunk       | 139        | 51.80%   | 364       | 50.27%   |
| quizballs     | 500        | 49.60%   | 488       | 50.82%   |
| quizguy       | 240        | 51.67%   | 341       | 50.74%   |
| quiz4free     | 171        | 49.12%   | 141       | 48.94%   |



**Table 12.9** Examples of true and false facts generated by Llama 2-7b and OPT-6.7b based on the questions in TriviaQA [173], obtained following the procedure detailed in Section 12.4.2.

| Model      | Label | Sentence                                                             |
|------------|-------|----------------------------------------------------------------------|
| Llama 2-7b | 1     | Arthur was married to Guinevere.                                     |
|            | 1     | Arthur’s most famous wife was Guinevere.                             |
|            | 1     | Guinevere was married to Arthur.                                     |
|            | 1     | Guinevere’s husband was King Arthur.                                 |
|            | 0     | Lancelot was married to Guinevere.                                   |
|            | 0     | Sir Lancelot was married to Queen Guinevere.                         |
|            | 0     | Sir Leonne was married to Queen Guinevere.                           |
| OPT-6.7b   | 0     | Anakin Skywalker is Darth Vader’s son.                               |
|            | 0     | Darth Vader’s son is Darth Vader.                                    |
|            | 1     | Darth Vader’s son is Luke Skywalker, a member of the Rebel Alliance. |
|            | 1     | Darth Vader’s son is Luke Skywalker.                                 |
|            | 1     | Darth Vader’s son is Luke.                                           |
|            | 0     | Darth Vader’s son is known as Darth Vader.                           |
|            | 1     | Luke’s father is Darth Vader.                                        |

**Table 12.10** Examples of false sentences generated with the Llama 2-7b model, using the perplexity-based generation strategy described in Section 12.4.1.

| True Sentence                                                                                         | Original Negative                                                                             | Generated Negative (Llama 2-7b)                                                |
|-------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| The crocodile has a habitat of freshwater.                                                            | The crocodile has a habitat of <b>various</b> .                                               | The crocodile has a habitat of <b>grassland/savanna</b> .                      |
| The zebra has distinctive black and white stripes, which may help deter flies and provide camouflage. | The zebra is a <b>fast swimmer</b> and can maintain high speeds for extended periods of time. | The zebra is the <b>fastest land animal</b> , reaching speeds up to 60-70 mph. |
| Tantalum has the symbol Ta.                                                                           | Tantalum has the symbol <b>Cs</b> .                                                           | Tantalum has the symbol <b>Tm</b> .                                            |

## LLM-Generated Dataset Extraction Details

### Factoid Answer Generation

Below is an example prompt used in the text generation pipeline for TriviaQA [173]. The 10 examples are sampled from the train Wikipedia split. The answers for the examples are manually crafted by looking at the available ground truth.

```
Question: Where in England was Dame Judi Dench born?
Answer: The English actress Dame Judi Dench was born in York, England.

Question: From which country did Angola achieve independence in 1975?
Answer: Angola achieved independence from Portugal in 1975.

Question: Which city does David Soul come from?
Answer: David Soul hails from Chicago, Illinois.

Question: Who won Super Bowl XX?
Answer: The Chicago Bears won Super Bowl XX.

Question: Which was the first European country to abolish capital punishment?
Answer: Norway was the first European country to abolish capital punishment.

Question: In which country did the widespread use of ISDN begin in 1988?
Answer: The widespread use of ISDN began in Japan in 1988.

Question: What is Bruce Willis' real first name?
Answer: Bruce Willis' real first name is Walter.

Question: Which William wrote the novel Lord of the Flies?
Answer: The William who wrote Lord of the Flies was William Golding.

Question: How is Joan Molinsky better known?
Answer: Joan Molinsky is better known as Joan Rivers.

Question: In which branch of the arts is Patricia Neary famous?
Answer: Patricia Neary is famous in the field of ballet.
```

**Figure 12.2** ChatGPT factual question answering example.

To generate responses, the model is provided with a continuation prompt such as:

```
Question: Ford Prefect came from a star in which constellation?
Answer:
```

**Figure 12.3** ChatGPT unanswered factual query example.

The model then generates a response, adhering to the pattern set by the previous examples. Responses are filtered based on length and uniqueness to ensure meaningful and high-quality outputs.

The generation process uses the following hyperparameters for all the employed models (i.e. Llama 2-7b and OPT-6.7b):

- max\_new\_tokens: 128
- top\_k: 50
- top\_p: 0.95
- temperature: 1.0
- num\_return\_sequences: 10

## Factoid Answer Annotation

Below is an example prompt to **GPT-4o mini** (2024-08-01), the LLM that we use as the oracle for the procedure described in Section 12.4.2. In this setup, we employ a 3-shot learning strategy, where the model is provided with three examples to establish a clear pattern of judgment. These examples demonstrate the task of evaluating answers against a given ground truth. The model is then tasked with using this context to evaluate a new answer to a question, based on the same criteria. In our experiments, we employ a temperature setting of 1.

```
SYSTEM:
You are a judge and your role is to judge whether the provided answer
is correct for the given question, based on the provided ground truth.
Answer with a 1 if the answer is correct and 0 if the answer is incorrect.
Here are a few examples:

USER:
Question: Who was the next British Prime Minister after Arthur Balfour?
Answer: Arthur Balfour was followed by David Lloyd George.
Ground truth: { 'aliases': ['Sir Henry Campbell-Bannerman',
'Campbell-Bannerman', 'Campbell Bannerman', 'Sir Henry Campbell
Bannerman', 'Henry Campbell Bannerman', 'Henry Campbell-Bannerman'],
'normalized_aliases': ['henry campbell bannerman', 'sir henry campbell
bannerman', 'campbell bannerman'], 'matched_wiki_entity_name': '',
'normalized_matched_wiki_entity_name': '', 'normalized_value': 'campbell
bannerman', 'type': 'WikipediaEntity', 'value': 'Campbell-Bannerman' }
Evaluation:

ASSISTANT:
0
```

```

USER:
Question: Who had a 70s No 1 hit with Kiss You All Over?
Answer: The band Exile had a 70s No 1 hit with Kiss You All Over.
Ground truth: {'aliases': ['Internal exile', 'Exiles', 'Transported for
life', 'Exile (politics and government)', 'Voluntary exile', 'Sent into
exile', 'Exile and Banishment', 'Self-exile', 'Forced exile', 'Exile',
'Exile in Greek tragedy', 'Banish', 'Banishment'], 'normalized_aliases':
['exiles', 'voluntary exile', 'forced exile', 'banish', 'self exile',
'exile politics and government', 'exile in greek tragedy', 'sent into
exile', 'banishment', 'transported for life', 'exile', 'internal exile'],
'matched_wiki_entity_name': '', 'normalized_matched_wiki_entity_name': '',
'normalized_value': 'exile', 'type': 'WikipediaEntity', 'value': 'Exile'
}
Evaluation:

ASSISTANT:
1

USER:
Question: Which common mineral is used to make casts, moulds, blackboard
chalk and plaster of Paris?
Answer: The common mineral used to make casts, moulds, blackboard chalk
and plaster of Paris is calcium carbonate.
Ground truth: {'aliases': ['CaSO4.2H2O', 'Gypsum', 'Calcium sulfate
dihydrate', 'CaSO4*2H2O', 'Gypsum'], 'normalized_aliases': ['calcium
sulfate dihydrate', 'caso4 2h2o', 'gypsum', 'caso4.2h2o', 'gypsum'],
'matched_wiki_entity_name': '', 'normalized_matched_wiki_entity_name':
'', 'normalized_value': 'gypsum', 'type': 'WikipediaEntity', 'value':
'Gypsum' }
Evaluation:

ASSISTANT:
0

```

**Figure 12.4** ChatGPT evaluation and answer-judging prompt example.

## Additional Experiments on the LLM-Generated Dataset

Table 12.11 reports the performances of the SAPLMA classifier trained on the original True-False dataset by Azaria et al. [25] and tested on the dataset generated from TriviaQA. Similarly to Table 12.6, SAPLMA does not generalize well over LLM-generated facts. Moreover, tuning an optimal threshold did not provide solid enhancements.

## Experiments on Additional LLM-Generated Datasets

We extended our LLM-generated dataset construction approach (Section 12.4.2) to TruthfulQA [217] and SQuAD 2.0 [295] to further support our findings.

**Table 12.11** Accuracy values obtained training SAPLMA on the original True-False dataset and testing on our facts dataset generated from TriviaQA. The original topic-wise leave-one-out strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models.

| Dataset              |       | Threshold = 0.5 |      |      |      |      | Optimal Threshold |      |      |      |      |
|----------------------|-------|-----------------|------|------|------|------|-------------------|------|------|------|------|
|                      |       | last            | 28   | 24   | 20   | 16   | last              | 28   | 24   | 20   | 16   |
| <b>billturnbull</b>  | Llama | .579            | .560 | .560 | .593 | .648 | .605              | .553 | .632 | .632 | .691 |
|                      | OPT   | .543            | .533 | .533 | .486 | .476 | .527              | .500 | .486 | .500 | .500 |
| <b>derby*</b>        | Llama | .556            | .576 | .550 | .544 | .602 | .542              | .529 | .529 | .554 | .575 |
|                      | OPT   | .526            | .548 | .554 | .563 | .535 | .535              | .583 | .570 | .579 | .561 |
| <b>quiz4free</b>     | Llama | .602            | .573 | .544 | .538 | .608 | .575              | .600 | .525 | .550 | .642 |
|                      | OPT   | .525            | .489 | .511 | .525 | .504 | .545              | .535 | .556 | .576 | .495 |
| <b>quizguy</b>       | Llama | .608            | .571 | .546 | .567 | .571 | .601              | .595 | .583 | .577 | .565 |
|                      | OPT   | .557            | .557 | .587 | .569 | .557 | .586              | .594 | .552 | .548 | .552 |
| <b>triviabug</b>     | Llama | .412            | .619 | .526 | .598 | .577 | .485              | .471 | .471 | .544 | .544 |
|                      | OPT   | .557            | .602 | .671 | .614 | .500 | .629              | .597 | .629 | .597 | .532 |
| <b>businessballs</b> | Llama | .580            | .577 | .559 | .575 | .589 | .586              | .605 | .546 | .563 | .592 |
|                      | OPT   | .564            | .578 | .555 | .587 | .521 | .585              | .579 | .585 | .547 | .537 |
| <b>jetpunk</b>       | Llama | .612            | .590 | .583 | .619 | .640 | .582              | .561 | .520 | .592 | .673 |
|                      | OPT   | .569            | .571 | .593 | .604 | .566 | .620              | .631 | .631 | .631 | .635 |
| <b>odquiz</b>        | Llama | .546            | .536 | .537 | .559 | .564 | .527              | .522 | .547 | .563 | .571 |
|                      | OPT   | .511            | .543 | .591 | .579 | .552 | .565              | .535 | .592 | .571 | .565 |
| <b>quiz-zone</b>     | Llama | .578            | .519 | .588 | .562 | .578 | .611              | .534 | .534 | .534 | .557 |
|                      | OPT   | .544            | .577 | .603 | .611 | .586 | .542              | .518 | .554 | .601 | .613 |
| <b>quizballs</b>     | Llama | .610            | .568 | .592 | .582 | .582 | .617              | .563 | .583 | .557 | .586 |
|                      | OPT   | .535            | .578 | .559 | .594 | .549 | .532              | .564 | .512 | .599 | .576 |
| <b>quizwise</b>      | Llama | .572            | .588 | .570 | .572 | .586 | .581              | .598 | .611 | .616 | .616 |
|                      | OPT   | .540            | .550 | .574 | .568 | .551 | .580              | .550 | .554 | .582 | .576 |
| <b>sfquiz</b>        | Llama | .530            | .533 | .528 | .538 | .560 | .545              | .581 | .584 | .570 | .575 |
|                      | OPT   | .531            | .546 | .568 | .590 | .545 | .595              | .597 | .599 | .611 | .553 |
| <b>triviacountry</b> | Llama | .602            | .636 | .602 | .602 | .619 | .639              | .602 | .639 | .590 | .578 |
|                      | OPT   | .525            | .542 | .585 | .551 | .525 | .494              | .602 | .506 | .578 | .446 |
| <b>wrexham**</b>     | Llama | .532            | .519 | .528 | .574 | .583 | .474              | .526 | .533 | .566 | .579 |
|                      | OPT   | .500            | .533 | .594 | .583 | .525 | .572              | .552 | .562 | .608 | .567 |
| <b>Average</b>       | Llama | .566            | .569 | .558 | .573 | .593 | .569              | .560 | .560 | .572 | .596 |
|                      | OPT   | .538            | .553 | .577 | .573 | .535 | .565              | .567 | .563 | .581 | .551 |

\*: derby is adopted as abbreviation of derbyshirepubquizleague

\*\* : wrexham is adopted as abbreviation of wrexhamquizleague

### LLM-Generated Facts from TruthfulQA

Following the same procedure detailed in Section 12.4.2, we extract true and false facts from TruthfulQA. Table 12.12 reports the statistics of the LLM-generated datasets constructed with Llama 2-7b and OPT-6.7b. Table 12.13 presents the performance of the SAPLMA classifier when both trained and tested on the TruthfulQA dataset. Additionally, Table 12.14 shows the results obtained by SAPLMA on TruthfulQA after being trained on the original True-False dataset. Since the dataset lacks a topic-based split, we adopt a 50-50 holdout strategy for evaluation. The results on the TruthfulQA dataset support our findings: the probe classifier fails to generalize effectively to LLM-generated datasets.

### LLM-Generated Facts from SQuAD 2.0

For the SQuAD 2.0 dataset, we limit our dataset construction procedure (Section 12.4.2) to the validation split and the five most common topics, as the large number of questions makes applying our procedure to the full dataset computationally impractical due to time constraints. Table 12.16 presents the performance of the SAPLMA classifier when trained and tested on the SQuAD 2.0 dataset. Table 12.17, instead, reports the results obtained on SQuAD 2.0 after training on the original True-False dataset from Azaria et al. [25].

The results on the SQuAD 2.0 dataset further support our conclusion: existing probes fall short in accurately assessing factuality in real-world settings. Across all cases, accuracy remains below a meaningful threshold, highlighting the limited reliability of these methods for self-assessment in scenarios involving LLM-generated sentences.

### Examples of SAPLMA Predictions

The tables in this Supplementary Material provide illustrative examples that complement the quantitative results discussed in the main text. Specifically, Table 12.18 presents sentence-level predictions on facts generated from the TriviaQA dataset using Llama 2-7b. In this case, the probe classifier is trained adopting a topic-wise leave-one-out strategy, as in Table 12.6. Table 12.19 shows additional predictions obtained using the probe classifier, which is trained on all but one topic and tested on our refined version of the held-out topic (as in the *Orig.* setting of Table 12.5). The code is provided in our `GitHub Repository`.

**Table 12.12** Summary of the dataset obtained by extracting factoid sentences from TruthfulQA [217], following the procedure described in Section 12.4.2.

| Model      | Total Samples | True Samples (%) |
|------------|---------------|------------------|
| Llama 2-7b | 842           | 49.76%           |
| OPT-6.7b   | 331           | 49.85%           |

**Table 12.13** Performance of SAPLMA on a fact dataset generated from TruthfulQA. A 50-50 holdout strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models.

|             | OPT-6.7b       |                | Llama 2-7b     |                |
|-------------|----------------|----------------|----------------|----------------|
|             | Random Split 1 | Random Split 2 | Random Split 1 | Random Split 2 |
| <b>last</b> | 0.487          | 0.488          | 0.491          | 0.504          |
| <b>28</b>   | 0.477          | 0.495          | 0.498          | 0.506          |
| <b>24</b>   | 0.478          | 0.513          | 0.507          | 0.507          |
| <b>20</b>   | 0.512          | 0.535          | 0.518          | 0.509          |
| <b>16</b>   | 0.493          | 0.492          | 0.520          | 0.514          |

**Table 12.14** Accuracy values obtained training SAPLMA on the original True-False dataset and testing on our facts dataset generated from TruthfulQA. A 50-50 holdout strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models.

|             | OPT-6.7b       |                | Llama 2-7b     |                |
|-------------|----------------|----------------|----------------|----------------|
|             | Random Split 1 | Random Split 2 | Random Split 1 | Random Split 2 |
| <b>last</b> | 0.473          | 0.560          | 0.518          | 0.539          |
| <b>28</b>   | 0.515          | 0.566          | 0.520          | 0.514          |
| <b>24</b>   | 0.503          | 0.476          | 0.518          | 0.501          |
| <b>20</b>   | 0.455          | 0.542          | 0.499          | 0.525          |
| <b>16</b>   | 0.479          | 0.500          | 0.485          | 0.530          |

**Table 12.15** Summary of the dataset obtained by extracting factoid sentences from SQuAD 2.0 [295], following the procedure described in Section 12.4.2.

| Topic                      | Model      | Sentences | (%) True |
|----------------------------|------------|-----------|----------|
| <b>Economic_inequality</b> | Llama 2-7b | 224       | 50.45%   |
|                            | OPT-6.7b   | 135       | 50.37%   |
| <b>Immune_system</b>       | Llama 2-7b | 167       | 49.70%   |
|                            | OPT-6.7b   | 72        | 50.00%   |
| <b>Rhine</b>               | Llama 2-7b | 120       | 50.83%   |
|                            | OPT-6.7b   | 59        | 47.46%   |
| <b>Warsaw</b>              | Llama 2-7b | 140       | 51.43%   |
|                            | OPT-6.7b   | 102       | 50.00%   |
| <b>Yuan Dynasty</b>        | Llama 2-7b | 109       | 49.54%   |
|                            | OPT-6.7b   | 53        | 52.83%   |

**Table 12.16** Performance of SAPLMA on a fact dataset generated from SQuAD 2.0, restricted to the validation set and the top-5 most popular topics. The original topic-wise leave-one-out strategy is adopted. Results are shown for the Llama 2-7b and OPT-6.7b models.

| Dataset                    |       | last  | 28    | 24    | 20    | 16    |
|----------------------------|-------|-------|-------|-------|-------|-------|
| <b>Economic_inequality</b> | Llama | 0.525 | 0.510 | 0.515 | 0.492 | 0.502 |
|                            | OPT   | 0.517 | 0.516 | 0.499 | 0.496 | 0.464 |
| <b>Immune_system</b>       | Llama | 0.486 | 0.490 | 0.503 | 0.502 | 0.547 |
|                            | OPT   | 0.436 | 0.439 | 0.401 | 0.438 | 0.435 |
| <b>Rhine</b>               | Llama | 0.525 | 0.520 | 0.515 | 0.524 | 0.526 |
|                            | OPT   | 0.484 | 0.485 | 0.522 | 0.547 | 0.521 |
| <b>Warsaw</b>              | Llama | 0.558 | 0.542 | 0.545 | 0.546 | 0.555 |
|                            | OPT   | 0.521 | 0.531 | 0.545 | 0.544 | 0.509 |
| <b>Yuan_dynasty</b>        | Llama | 0.509 | 0.522 | 0.514 | 0.529 | 0.544 |
|                            | OPT   | 0.544 | 0.560 | 0.512 | 0.542 | 0.512 |



**Table 12.17** Accuracy values obtained training SAPLMA on the original True-False dataset and testing on the facts dataset generated from SQuAD 2.0. Results are shown for the Llama 2-7b and OPT-6.7b models.

|                            |       | last  | 28    | 24    | 20    | 16    |
|----------------------------|-------|-------|-------|-------|-------|-------|
| <b>Economic_inequality</b> | Llama | 0.558 | 0.531 | 0.589 | 0.531 | 0.545 |
|                            | OPT   | 0.600 | 0.600 | 0.548 | 0.533 | 0.533 |
| <b>Immune_system</b>       | Llama | 0.569 | 0.593 | 0.593 | 0.575 | 0.599 |
|                            | OPT   | 0.611 | 0.611 | 0.611 | 0.708 | 0.542 |
| <b>Rhine</b>               | Llama | 0.467 | 0.492 | 0.475 | 0.442 | 0.525 |
|                            | OPT   | 0.525 | 0.559 | 0.593 | 0.593 | 0.576 |
| <b>Warsaw</b>              | Llama | 0.614 | 0.579 | 0.607 | 0.600 | 0.593 |
|                            | OPT   | 0.48  | 0.500 | 0.539 | 0.598 | 0.549 |
| <b>Yuan_dynasty</b>        | Llama | 0.523 | 0.523 | 0.541 | 0.505 | 0.550 |
|                            | OPT   | 0.453 | 0.547 | 0.491 | 0.528 | 0.547 |

**Table 12.18** Examples of sentence classification on facts generated from the TriviaQA dataset using Llama 2-7b. The original topic-wise leave-one-out strategy is adopted. Correct predictions are highlighted in green, while incorrect ones are shown in red.

| Sentence                                                                                          | Predicted | GT    | Confidence |
|---------------------------------------------------------------------------------------------------|-----------|-------|------------|
| Sir John Suckling was the first poet to be buried at Poet’s Corner in London’s Westminster Abbey. | False     | False | 0.373      |
| The first poet to be buried at Poet’s Corner in London’s Westminster Abbey was Geoffrey Chaucer.  | True      | True  | 0.551      |
| The poet who was the first to be buried in Westminster Abbey was Geoffrey Chaucer.                | False     | True  | 0.456      |
| Roquefort cheese is made from sheep’s milk.                                                       | True      | True  | 0.794      |
| Roquefort cheese is made from cow’s milk.                                                         | True      | False | 0.851      |
| The milk of sheep is used to make ‘Roquefort Cheese’.                                             | False     | True  | 0.488      |
| South Korea held its first ‘Grand Prix’ motor race in 1999.                                       | False     | False | 0.291      |
| South Korea held its first ‘Grand Prix’ motor race in 2010.                                       | False     | True  | 0.305      |
| South Korea hosted its first ‘Grand Prix’ motor race in 1966.                                     | False     | False | 0.182      |

**Table 12.19** Examples of predictions using SAPLMA and Llama 2-7b. In these examples, the probe classifier is trained on the original version of the True-False dataset [25] (excluding the *element* topic) and tested on our refined version of the *element* topic. Correct predictions are highlighted in green, while incorrect ones are shown in red.

| Sentence                                                                                       | Predicted | GT    | Confidence |
|------------------------------------------------------------------------------------------------|-----------|-------|------------|
| Tin has the symbol Sn.                                                                         | True      | True  | 0.990      |
| Mercury is a liquid at room temperature and used in thermometers and some electrical switches. | False     | True  | 0.113      |
| Fluorine is in the Alkaline earth metal group.                                                 | False     | False | 0.012      |
| Tellurium has the atomic number of 82.                                                         | True      | False | 0.998      |

## Chapter 13

# GradeSQL: Test-Time Inference with Outcome Reward Models for Text-to-SQL Generation from Large Language Models

---

Text-to-SQL, the task of translating natural language questions into SQL queries, has significantly advanced with the introduction of Large Language Models (LLMs), broadening database accessibility for a wide range of users. Despite notable progress in generating valid SQL, current LLMs still struggle with complex queries. To address this limitation, *test-time strategies* such as Best-of- $N$  (BoN) and Majority Voting (Maj) are often employed, based on the assumption that LLMs can produce correct answers after multiple attempts. However, these methods rely on surface-level heuristics, selecting the syntactically correct query through execution-based BoN (ex-BoN) or the most frequently generated one through Majority Voting. Recently, Outcome Reward Models (ORMs), which assign utility scores to generated outputs based on semantic correctness, have emerged as a promising reinforcement learning approach for improving model alignment. We argue that ORMs could serve as an effective new test-time heuristic, although their application in this context remains largely underexplored.

In this work, we propose a unified framework for training ORMs tailored to the Text-to-SQL task and assess their effectiveness as a test-time heuristic within the BoN strategy. We benchmark ORMs against ex-BoN and Maj across the BIRD and Spider datasets, fine-tuning diverse open-source LLMs from the Qwen2, Granite3, and Llama3 families. Results show that ORMs outperform ex-BoN and Maj, achieving execution accuracy gains of +4.33% (BIRD) and +2.10% (Spider) over ex-BoN, and +2.91% (BIRD) and +0.93% (Spider) over Maj. We further demonstrate that fine-

tuning models already aligned with SQL generation, such as OmniSQL, yields superior ORM performance. Additionally, we observe that ORMs achieve competitive results on simple queries and benefit more from an increased number of candidates compared to ex-BoN and Maj. All code, datasets, and trained models are publicly released to support reproducibility and encourage future research in this area.<sup>1</sup>

## 13.1 Introduction

Text-to-SQL, the task of translating Natural Language (NL) question into executable SQL query, facilitates the use of data management systems [259] and represents a step toward democratizing database access [183], enabling both expert and non-expert users to interact with complex database systems through NL [16, 199, 224].

Early approaches were predominantly rule-based, relying on manually crafted rules and heuristics to map NL questions to SQL queries [200, 201, 238]. While often capable of generating syntactically correct queries, these systems struggled with challenges such as nested clauses, co-references, and complex aggregations [154].

The field progressed with the adoption of neural network-based techniques [41, 397], which improved intent recognition and semantic parsing. This was followed by the introduction of Pre-trained Language Models (PLMs) such as BERT [100] and T5 [294], which achieved competitive results across various benchmarks [202, 206]. Nevertheless, PLM-based systems still face difficulties with schema generalization and complex queries [224].

Recently, the advent of Large Language Models (LLMs) has introduced a new paradigm in Text-to-SQL. LLMs demonstrate strong emergent capabilities that surpass PLMs across diverse NLP tasks [194], and they currently dominate research in this area [120, 203, 289, 416]. Nevertheless, LLMs still face challenges in generating correct SQL for complex queries involving multi-table joins, nested subqueries, and sophisticated aggregations [178, 234]. To address these limitations, researchers are exploring techniques such as prompt engineering [120], multi-agent frameworks [24, 364], and fine-tuning [203, 291]. While these approaches have driven notable advances, achieving further improvements remains challenging due to the inherent complexity of the task, and even minor gains represent substantial progress [120].

Inspired by recent advances in mathematical reasoning with LLMs [429], the re-

1. This chapter is based on ongoing research entitled “GradeSQL: Test-Time Inference with Outcome Reward Models for Text-to-SQL Generation from Large Language Models”, which has been submitted for peer review.

search community is increasingly investigating *test-time inference* strategies, which allocate additional computational resources at inference to enhance output quality [47, 144, 204]. These strategies typically generate multiple candidate outputs and apply heuristics to select the best one, leveraging the ability of LLMs to produce correct answers when afforded more inference time rather than larger model size or additional training [398]. Common approaches include Best-of- $N$  (BoN) [79], which samples  $N$  outputs and selects the highest-scoring candidate, and Majority Voting (Maj) [326], which chooses the most frequently generated output. However, these methods often depend on shallow heuristics such as frequency or execution success, without explicitly assessing semantic correctness.

To address this gap, in this work we look beyond surface-level heuristics toward methods that explicitly assess the semantic quality of candidate outputs. In particular, the Reinforcement Learning (RL) community has proposed Outcome Reward Models (ORMs), which assign fine-grained semantic correctness scores to candidate outputs [429]. When used as reward functions in RL loops, ORMs improve the alignment between model predictions and user intent [142], highlighting their potential as principled verifiers of model output. This naturally raises the question: *can ORMs also enhance test-time inference strategies?* In particular, integrating ORMs into BoN offers a way to move beyond frequency- or execution-based heuristics, enabling candidate selection grounded in semantic correctness. Although BoN and Maj have already been applied to Text-to-SQL [204], the use of ORMs in test-time inference remains largely unexplored, hindered by the scarcity of reward-labeled data and the absence of clear methodological guidelines. Moreover, no systematic evaluation has yet compared ORM-guided selection with traditional test-time compute strategies, leaving open the potential of ORMs as a more robust heuristic for BoN.

To address these limitations, we first introduce a scalable data synthesis framework tailored for Outcome Reward Models in the Text-to-SQL setting, which automates data generation to mitigate data scarcity and enables the development of task-specific ORMs. Building on this foundation, we present a comprehensive evaluation of ORM-guided Best-of- $N$ , benchmarking it against execution-based BoN and Majority Voting.

Specifically, the framework consists of three stages: *candidate generation*, *data labeling*, and *supervised fine-tuning*. First, we employ a powerful LLM to generate

a diverse set of SQL query candidates for a given natural language question and database schema. This set includes both correct queries, which yield execution results consistent with the gold standard, and incorrect but syntactically valid alternatives. Next, we label these candidates to construct a dataset of positive and negative examples. Finally, this dataset is used to fine-tune a separate LLM, which serves as an ORM. The trained ORM learns to score candidate SQL queries based on their alignment with the original user intent and the database schema. At inference, the ORM functions as a post-generation re-ranking module, selecting the most semantically faithful query from the candidate pool generated by the base model.

To evaluate the effectiveness of our approach, we conduct a comprehensive study using open-source LLMs in an end-to-end Text-to-SQL setup.

Our experiments yield several key insights into the effectiveness of ORMs for Text-to-SQL: (1) ORMs consistently outperform both execution-based Best-of- $N$  and Majority Voting, achieving execution accuracy gains of up to +4.33% on BIRD and +2.10% on Spider over ex-BoN, and +2.91% (BIRD) and +0.93% (Spider) over Maj; (2) fine-tuning ORM models that are already aligned with SQL generation tasks, such as OmniSQL, further enhances performance; (3) ORMs perform competitively on simple queries while demonstrating greater robustness on complex ones compared to ex-BoN and Maj; and (4) ORM performance benefits more substantially from increasing the number of candidate queries than the baseline methods.

In summary, our work makes the following contributions:

- **Scalable Data Synthesis and ORM Training:** We introduce a framework to synthesize high-quality annotated datasets for ORM fine-tuning, overcoming data scarcity in Text-to-SQL reward modeling.
- **Extensive Empirical Evaluation and Analysis:** We conduct a large-scale study across LLM families and parameter sizes, analyze test-time efficiency under candidate scaling, and provide controlled ablations on prompt design, model scale, and training losses.
- **Open-Source Resources:** We release our codebase, synthesized datasets, and fine-tuned ORMs on GitHub and Hugging Face to promote reproducibility and future research.

## 13.2 Related Work

In this section, we trace the evolution of Text-to-SQL solutions, review test-time inference strategies, and conclude with an overview of existing data-augmentation frameworks for Text-to-SQL.

### 13.2.1 Text-to-SQL Meets LLMs

Text-to-SQL refers to the task of translating Natural Language (NL) questions into executable SQL queries, serving as a bridge between human intent and structured data access. Its development spans more than five decades, evolving from early handcrafted systems to today's approaches powered by Large Language Models (LLMs) [154].

The earliest systems were entirely rule-based, relying on domain-specific grammars and manually engineered parsing rules. A seminal example is LUNAR (1971) [386], which answered questions about lunar rock samples using an Augmented Transition Network parser combined with procedural semantics. In the 1980s, the field began adopting intermediate logic-based representations, where systems first converted NL into a database-agnostic logical form before generating SQL. The logic-based interpreter by Warren and Pereira (1982) [378] exemplifies this trend. By the late 1990s and early 2000s, a shift toward data-driven methods had emerged, as researchers explored whether parsers could be trained statistically rather than handcrafted [344].

The mid-2010s marked a paradigm shift with the advent of deep learning. Neural semantic parsers were capable of jointly modeling questions and database schemas, but progress was constrained by the lack of large-scale labeled datasets [183]. A breakthrough came with the release of Spider (2018) [410], which provided benchmarks for cross-domain generalization. Building on these datasets, Pre-trained Language Models (PLMs) such as BERT [100] and T5 [294] catalyzed the development of schema-aware neural parsers. RAT-SQL [363], for example, integrated relation-aware transformers with pre-trained embeddings to significantly improve schema linking, i.e., mapping question terms to database elements.

The advent of LLMs such as GPT [268], Llama [107], and PaLM [20] has further transformed the landscape. Unlike PLMs, LLMs exhibit strong emergent capabilities, enabling zero- and few-shot generalization with minimal or no task-specific fine-tuning. Studies such as those by Liu et al. [219] have shown that ChatGPT achieves

up to 70% execution accuracy on Spider with only prompt-based methods. Few-shot prompting strategies, such as DIN-SQL [289], decompose the task into sub-tasks to improve accuracy, though performance still degrades on highly complex queries. More recent work combines LLMs with Retrieval-Augmented Generation (RAG) [130, 358], which injects schema- or database-specific context at inference time, yielding improved accuracy on large, content-rich benchmarks such as BIRD (2023) [207].

Building on these advances, recent research increasingly frames Text-to-SQL as a reasoning problem. Rather than treating query generation as a single-shot task, approaches leverage LLMs for multi-step decomposition, error correction, and domain adaptation. Pipelines decompose the task into components such as schema linking, query generation, refinement, and selection, orchestrated through prompt engineering or fine-tuning [144, 153]. Some frameworks adopt multi-agent designs [24], while others focus on scaling SQL-specific training data [204]. Current trends emphasize expanding reasoning capacity [120, 306], integrating domain knowledge [207], and refining fine-tuning strategies [203].

Despite this rapid progress, the intrinsic complexity of Text-to-SQL continues to limit accuracy, especially on queries requiring multi-table joins, nested subqueries, and sophisticated aggregations. These challenges have sparked growing interest in *test-time inference* strategies, which allocate additional computational resources at inference time to enhance output quality [47, 144, 204].

### 13.2.2 Test-Time Inference Strategies

Test-Time Inference (TTI), also known as inference-time scaling or test-time computation, refers to strategies that improve LLM outputs by allocating additional computation during inference without modifying model parameters [329]. Nowadays, LLMs generate text by sequentially predicting tokens, with decoding strategies that determine which token to produce at each step. Early approaches relied on deterministic methods such as greedy decoding [72] or beam search [349], which maximized sequence likelihood but often produced repetitive outputs with limited diversity [251]. To address this, later research introduced stochastic decoding algorithms, including temperature scaling, top- $k$  sampling [110], and nucleus (top- $p$ ) sampling [152], which add controlled randomness by sampling from truncated distributions. Although effective in improving local variability, these methods oper-



ate only at the token level and thus cannot substantially improve the correctness of full outputs. TTI builds on these foundations by shifting the focus from token-level diversity toward strategies that explicitly leverage multiple candidate generations, verification mechanisms, and structured reasoning to enhance robustness and accuracy.

Building on this idea, commonly adopted methods such as Best-of- $N$  (BoN) and Majority Voting (Maj) have emerged [329]. BoN selects the candidate with the highest score according to a heuristic function, such as model likelihood [152] or execution accuracy [65], while Maj chooses the most frequent final answer among diverse samples [372]. These approaches are attractive because they require no re-training, but they also inherit critical limitations: BoN tends to over-prefer generic high-probability outputs [211, 250], while Maj can fail when correct solutions are underrepresented [160, 372]. This motivated the development of verification-guided selection methods, which go beyond surface heuristics by introducing an explicit verifier to score or filter candidates.

A major advancement in this direction came with reward models. Process Reward Models (PRMs) [180] evaluate intermediate reasoning steps but require costly step-level supervision. In contrast, Outcome Reward Models (ORMs) [79] score only final outputs, avoiding intermediate annotation and achieving notable success in domains such as mathematical reasoning [79]. In Text-to-SQL, ORM are particularly appealing because they can semantically evaluate SQL queries independently of the generator. This enables the selection of high-quality but underrepresented candidates [43, 144], a capability missing in Maj, while also identifying semantically equivalent but syntactically different queries. These properties make ORM a natural fit for re-ranking diverse candidate sets and for modular system designs where generation and verification can be optimized independently.

**Datasets for ORM Training.** A central challenge in realizing ORM potential is the scarcity of large, high-quality datasets that pair inputs with multiple candidate outputs and quality labels. In Text-to-SQL, most dataset construction efforts have focused on improving general model training rather than verifier-specific needs. Early approaches used SQL-to-question pipelines, where SQL queries generated from templates were translated into natural language [134, 187, 425, 432]. While effective, these methods were labor intensive and limited in diversity. More recent LLM-based approaches, such as SENSE [404], improve realism through multi-step

generation but rely on expensive proprietary models. Syn-SQL [204] mitigates this by decomposing synthesis into smaller steps that can be executed with open-source LLMs, producing 2.5M examples in a scalable and reproducible manner.

Despite these advances, no prior work has addressed the unique requirements of ORM training, where datasets must support fine-grained verification across diverse candidate outputs rather than solely boosting general SQL generation performance. This gap underscores the need for new methodologies that explicitly target ORM development.

### 13.3 Methodology

#### 13.3.1 Problem Formulation.

Given a natural language question  $q_i \in Q$ , where  $Q$  denotes the set of all questions, the Text-to-SQL task aims to map  $q_i$  and its corresponding database schema  $\Sigma_i$  (a collection of tables with their columns) into a valid SQL query  $y_{gold_i} \in Y_{gold}$  [26, 91].

While the traditional formulation focuses on generating a single query, in the context of *test-time inference* the process is extended to consider multiple candidates. Specifically, given  $q_i$ , an LLM generates a pool of  $N$  candidate queries

$$Y_{\text{candidate}} = \{c_1, c_2, \dots, c_N\},$$

from which either a heuristic or a verifier must identify the candidate  $c_j \in Y_{\text{candidate}}$  that best corresponds to the ground-truth gold query. In this work, we define the verifier as an *Outcome Reward Model (ORM)*.

Formally, the ORM is a scoring function  $\phi(q_i, c_j)$  that maps a natural language question  $q_i$  and a candidate SQL query  $c_j \in Y_{\text{candidate}}$  to a probabilistic score:

$$\phi(q_i, c_j) \in [0, 1],$$

where higher scores indicate a greater likelihood that the query is correct. The ORM thus induces a ranked list of candidate SQL queries, with the best candidate  $c^*$  selected as the one achieving the highest score:

$$c^* = \arg \max_{c_j \in Y_{\text{candidate}}} \phi(q_i, c_j).$$

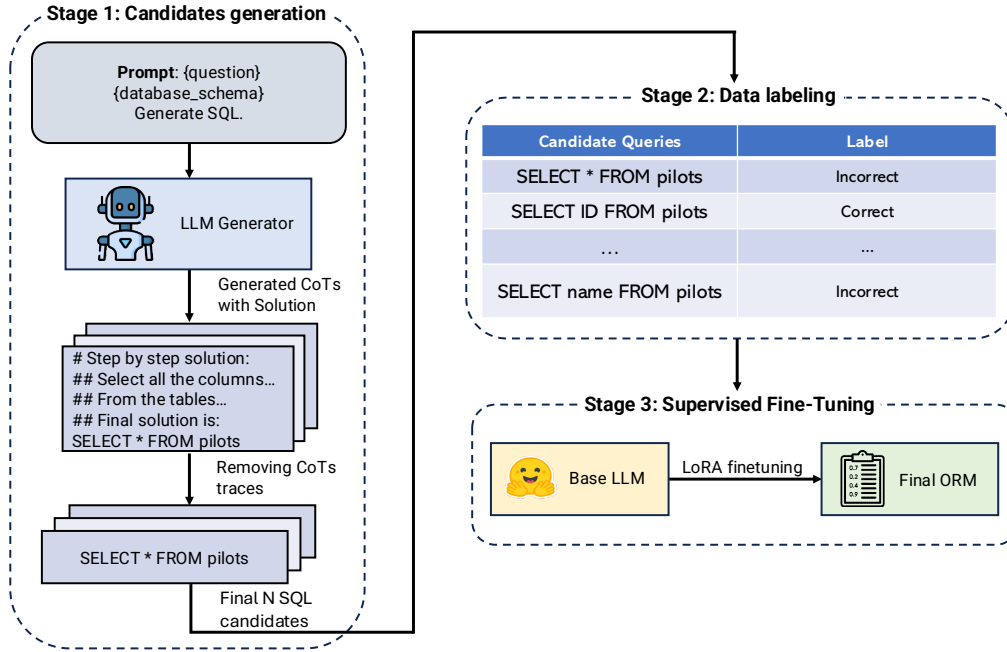


**Figure 13.1** Overview of the *GradeSQL* framework for training an ORM. The framework consists of three stages: (i) **Candidate Generation**, an LLM generates  $N$  SQL queries via a Chain-of-Thought (CoT) prompt, then removes the CoT trace; (ii) **Data Labeling**, each candidate is annotated as correct or incorrect; and (iii) **Supervised Fine-Tuning (SFT)**, the base LLM is fine-tuned with LoRA on the labeled data to obtain the final ORM.

Training an ORM as a scoring function for candidate ranking introduces two central challenges: (i) constructing datasets that explicitly identify which candidate  $c_j$  is correct, and (ii) assigning relative scores to multiple candidates in a way that faithfully reflects user intent.

To overcome these challenges, we leverage the LLM’s internal ability to self-assess [157, 209, 223] query correctness. Specifically, we prompt the model to classify each candidate as “Yes” (correct) or “No” (incorrect), and then extract the associated logits. These logits are used during training to supervise the ORM, enabling it to learn a probabilistic scoring function that generalizes to ranking candidates at inference time.

In the following, we present the **GradeSQL** framework, which implements this idea by constructing and training ORMs specifically designed for test-time inference in the Text-to-SQL setting.

### 13.3.2 GradeSQL Framework

Given a Text-to-SQL dataset  $\mathcal{D} = \{(q_i, y_{gold_i})\}_{i=1}^n$ , where  $q_i$  is a natural language question and  $y_{gold_i}$  its corresponding gold SQL query, our framework (Figure 13.1) comprises three stages: (i) Candidates generation, (ii) Data labeling, and (iii) Supervised Fine-Tuning (SFT). The overall objective is to train an Outcome Reward Model (ORM) tailored to the target dataset.

**Stage 1: Candidate Generation.** For each question  $q_i$ , we employ an LLM, referred to as the generator  $\mathcal{G}$ , to produce a set of  $N$  candidate SQL queries (the prompt is provided in Supplementary Material 13.7).

Formally, let  $q_i$  denote a natural language question and  $\Sigma_i$  the corresponding database schema. The generator  $\mathcal{G}(q_i, \Sigma_i)$  produces  $N$  candidate SQL queries:

$$Y_{\text{candidate}} = \{\text{SQL}_1, \text{SQL}_2, \dots, \text{SQL}_N\} = \{c_1, c_2, \dots, c_N\}.$$

**Stage 2: Data Labeling.** Once candidates are generated, the next step is to determine which of them correctly capture the intent of  $q_i$ . Let  $R(c_j)$  denote the execution result of a query  $c_j$ , i.e., the set of tuples returned when  $c_j$  is executed on the target database. A candidate  $c_j \in Y_{\text{candidate}}$  is labeled as *correct* if its result set matches that of the gold query  $y_{gold_i}$ , i.e.,  $R(c_j) = R(y_{gold_i})$ .

Candidates are labeled as *incorrect* if they fail to capture the intended semantics. This may occur in two cases:

- (1) Semantic mismatch: The query executes successfully but returns a different result set from the gold query, i.e.,  $R(c_j) \neq R(y_{gold_i})$ .
- (2) Execution error: The query contains syntactic or runtime errors that cause the database management system (DBMS) to fail during execution.

Since execution errors are trivial to detect, the framework primarily focuses on distinguishing semantically correct queries from incorrect but executable ones. This process is formalized by the labeling function  $\ell(c_i)$ :

$$\ell(c_j) = \begin{cases} \text{correct (Yes)} & \text{if } R(c_j) = R(y_{gold_i}), \\ \text{incorrect (No)} & \text{if } R(c_j) \neq R(y_{gold_i}), \\ \text{discarded} & \text{if } c_j \text{ raises an execution error.} \end{cases}$$

**Stage 3: Supervised Fine-tuning.** The labeled candidates form a training set  $\mathcal{D}_{\text{label}} =$

$\{(c_i, l_i)\}_{i=1}^n$ , where  $c_i$  is a candidate SQL query and  $l_i \in \{\text{Yes}, \text{No}\}$  indicates correctness. At this stage, the verification task is cast as an autoregressive binary classification problem.

Formally, let  $x = \text{Prompt}(\Sigma, q, c)$  denote the input sequence constructed from the database schema  $\Sigma$ , the natural language question  $q$ , and the candidate SQL query  $c$  (see Supplementary Material 13.7). The label  $l \in \{\text{Yes}, \text{No}\}$  specifies whether  $c$  semantically satisfies  $q$  under  $\Sigma$ . The model is trained to autoregressively generate the next token: “Yes” if correct and “No” otherwise.

We fine-tune the causal LM head as a verifier using LoRA [156] under the causal language modeling objective [293]. Given the input  $x = \text{Prompt}(\Sigma, q, c)$ , we append the label  $l \in \{\text{Yes}, \text{No}\}$  to obtain the target sequence  $s = [x; l]$ . During training, the model conditions on  $s$  and is optimized to minimize the negative log-likelihood:

$$\mathcal{L}(\theta) = - \sum_{t=1}^{|s|} \log P_{\theta}(s_t | s_{<t}).$$

This objective guides the model, when presented with a structured input  $x = \text{Prompt}(\Sigma, q, c)$ , to autoregressively generate the correct label token  $l \in \{\text{Yes}, \text{No}\}$ .

### 13.3.3 ORM Inference and Probabilistic Scoring Mechanism

At inference time, the verifier assigns a probabilistic score to each candidate SQL query, reflecting its likelihood of being semantically correct. This is achieved by prompting the fine-tuned LLM in an autoregressive setting to generate a single verification token, either “Yes” or “No”. Rather than treating this prediction as a discrete token-level decision, we instead use the model’s logit corresponding to the “Yes” token, normalized into  $[0, 1]$ , as a continuous confidence score. In this way, the ORM functions not merely as a classifier but as a ranking mechanism over candidate queries. A visual representation of the inference process is provided in the Supplementary Material (see Figure 13.3).

*Autoregressive Verification..* Formally, for a given prompt  $x = \text{Prompt}(\Sigma, q, c)$ , constructed from the schema  $\Sigma$ , natural language question  $q$ , and candidate query  $c$ , the model defines a conditional distribution:

$$P(y | x; \theta), \quad y \in \{\text{Yes}, \text{No}\},$$

where  $\theta$  are the model parameters. The verifier's score for candidate  $c$  is then:

$$\text{Score}(c) = P(y_{\text{yes}} \mid x; \theta).$$

This formulation avoids hard decision boundaries, instead producing a real-valued confidence score that can be directly used to rank candidates.

*Motivation for Probabilistic Scoring.* Framing ORM inference in terms of probabilities provides two key advantages for Text-to-SQL verification:

- (1) Fine-grained calibration: Scores capture differences in semantic plausibility, even among syntactically valid queries.
- (2) Differentiable ranking: The continuous nature of the scores enables consistent ordering of candidates, avoiding the brittleness of binary thresholding.

By interpreting the verifier as a probabilistic scoring function rather than a binary classifier, the ORM offers a mechanism for ranking candidate SQL queries by their semantic correctness.

## 13.4 Experimental Setup

**Datasets.** To evaluate our approach, we use two widely adopted Text-to-SQL benchmarks: Spider [410] and BIRD [207]. Both are cross-domain benchmarks with disjoint databases across train/dev/test splits to assess generalization. On Spider, we construct the labeled dataset from the training questions, train the ORM, and evaluate it on the dev and test sets. On BIRD, we adopt the same setup, using the training questions to build the dataset and train the ORM, but, due to the hidden test set, evaluate only on the dev set.

**Metrics.** To ensure fair comparison across methods, we adopt the official evaluation metrics defined for BIRD [207], which are also applicable to Spider. Let  $Q_{\text{test}}$  denote the set of evaluation questions. For each question  $q_i \in Q_{\text{test}}$ , let  $c_i^*$  be the predicted SQL query and  $y_{\text{gold}_i}$  the corresponding gold query.

Execution Accuracy (EX) [207] measures the proportion of questions for which the predicted and gold queries return identical result sets:

$$\text{EX} = \frac{1}{|Q_{\text{test}}|} \sum_{q_i \in Q_{\text{test}}} 1 \left( R(c_i^*) = R(y_{\text{gold}_i}) \right),$$

where  $1(\cdot)$  is the indicator function, returning 1 if the condition holds and 0 other-

wise.

Pass@N [68] evaluates the proportion of questions for which at least one of the  $N$  predicted candidates returns the correct result set:

$$\text{Pass@N} = \frac{1}{|Q_{\text{test}}|} \sum_{q_i \in Q_{\text{test}}} 1 \left( \bigvee_{j=1}^N R(c_j) = R(y_{\text{gold}_i}) \right),$$

where  $\bigvee$  denotes the logical OR across  $N$  candidates. Specifically, the prediction is considered correct if any of the  $N$  candidates for question  $q_i$  yield the same result set as the ground truth  $y_{\text{gold}_i}$ .

**Baselines.** We benchmark the ORM-based test-time inference (TTI) against two TTI strategies widely used in Text-to-SQL and code generation tasks: Majority Voting and execution-based Best-of- $N$ . Both methods select a single SQL query from the generated pool without relying on gold references, but differ in how they aggregate candidates.

*Majority Voting.* Majority Voting selects the most plausible SQL query based on execution equivalence [326]. Each candidate query is evaluated on the target database, and queries producing identical result sets are grouped into  $k$  clusters. The class with the largest cardinality is selected, and the final query is sampled uniformly from it. Formally, let  $Y_{\text{candidate}} = \{c_1, c_2, \dots, c_N\}$  be the set of candidates. We define

$$c_i \sim c_j \text{ if and only if } R(c_i) = R(c_j),$$

which partitions  $Y_{\text{candidate}}$  into  $k$  clusters  $C_1, \dots, C_k$ . Majority Voting then selects the cluster with the largest cardinality and samples one candidate from it. This method is robust to surface-level variations and has been widely adopted in the literature, including BIRD [207].

*Execution-based Best-of- $N$ .* Best-of- $N$  selects the most promising candidate based on an execution-level heuristic [65]. Given  $Y_{\text{candidate}} = \{c_1, \dots, c_N\}$ , the heuristic  $h : Y_{\text{candidate}} \rightarrow [0, 1]$  considers two signals: (i) *executability* (whether the query runs without errors), and (ii) *non-emptiness* (whether it returns at least one row). The

**Table 13.1** Reproducibility results of OmniSQL-7B. Maj = Majority Voting (temperature  $T=0.8$ ,  $N=8$ ); Gre = Greedy decoding ( $N=1$ ). The table compares our reproduction (Our) with the original results (Origin) on the BIRD dev, Spider dev, and Spider test benchmarks.

|               | BIRD dev |       | Spider dev |       | Spider test |       |
|---------------|----------|-------|------------|-------|-------------|-------|
|               | Maj      | Gre   | Maj        | Gre   | Maj         | Gre   |
| <b>Our</b>    | 66.95    | 64.41 | 83.95      | 82.11 | 85.61       | 84.49 |
| <b>Origin</b> | 66.10    | 63.90 | 81.60      | 81.20 | 88.90       | 87.90 |

scoring function is:

$$h(c_i) = \begin{cases} 0 & \text{if } c_i \text{ is not executable,} \\ 0.5 & \text{if } c_i \text{ executes but returns 0 rows,} \\ 1 & \text{if } c_i \text{ executes and returns at least one row.} \end{cases}$$

The final query  $c^*$  is selected uniformly from the subset of candidates that achieve the highest score.

While this efficiently filters invalid queries, it provides only coarse discrimination: multiple queries may achieve the same maximal score while differing in semantic correctness. This limitation motivates our ORM-based extension of Best-of- $N$ . **ORMs and Evaluation Setup.** To evaluate ORM-based Best-of- $N$ , we adopt OmniSQL-7B [204] as the generator, which, at the time of writing, is the most capable open-source 7B model trained specifically for the Text-to-SQL task. For each input question, OmniSQL produces  $N$  candidate queries, which are then re-ranked by three strategies: Majority Voting, execution-based Best-of- $N$ , and ORM-based Best-of- $N$ .

Before integration, we validated OmniSQL through a reproducibility study on Spider and BIRD. As shown in Table 13.1, our reproduced results closely match those of the original work, with deviations of at most 1% on BIRD dev and within +3% on Spider dev. This confirms the robustness of OmniSQL and justifies its use as our base generator, particularly in combination with ORM re-ranking.

**Reproducibility details.** All experiments were conducted on compute nodes with Intel Xeon 8358 CPUs (32 cores, 2.6 GHz), 512 GB RAM, and 4 NVIDIA A100 SXM4 GPUs (64 GB). SQL candidates were generated via vLLM using the OmniSQL setup



**Table 13.2** Dataset statistics for *BIRD* and *Spider* (training splits). We report (i) the number of questions before and after de-duplication (with removals  $\Delta$ ), and (ii) the final size of the labeled sets generated by sampling  $N = 32$  candidates per question. Percentages indicate the class distribution (*Incorrect* / *Correct*) in the *imbalanced* set; the *balanced* set is 50/50.

|                           | BIRD (train)  | Spider (train) |
|---------------------------|---------------|----------------|
| <i>Questions</i>          |               |                |
| # (pre)                   | 9,428         | 9,000          |
| # (post)                  | 9,411         | 8,960          |
| Removed $\Delta$ (%)      | 17 (0.18%)    | 40 (0.44%)     |
| <i>Labeled sets</i>       |               |                |
| Imbalanced size           | 82,640        | 50,073         |
| Class split (% inc / cor) | 41.37 / 58.63 | 30.60 / 69.40  |
| Balanced size (50/50)     | 30,686        | 17,834         |

[204], with ORM inference through Hugging Face transformers. Runs used a fixed seed of 42. Code<sup>2</sup>, datasets<sup>3</sup>, and models<sup>4</sup> are publicly available.

## 13.5 Results and Discussion

We evaluated our framework on seven models: Qwen2.5-1.5B Instruct, Qwen2.5-7B Instruct, Granite-3.3-2B Instruct, Granite-3.3-8B Instruct, Llama-3.2-1B Instruct, Llama-3.1-8B Instruct, and OmniSQL-7B. Benchmarks were conducted on the BIRD dev (1,534 samples), Spider dev (1,034 samples), and Spider test (2,147 samples) splits. For both the BIRD and Spider datasets, we began with the original training sets of 9,428 queries (BIRD) and 9,000 queries (Spider). After light pre-processing to remove duplicates and reduce redundancy, we obtained 9,411 queries for BIRD and 8,960 for Spider. For each question, OmniSQL-7B was used to generate  $N = 32$  candidate SQL queries. We selected  $N = 32$  following [212], as it represents a well-balanced trade-off between performance and computational cost [198, 372]. The framework automatically discards syntactically invalid queries and labels each remaining candidate as correct ( $s_+$ ) or incorrect ( $s_-$ ) depending on whether its result set matches the gold SQL. This procedure yielded final labeled datasets of 82,640 samples for BIRD and 50,073 for Spider.

Analyzing these datasets reveals a marked class imbalance. In Spider, 69.4% of

2. GradeSQL Framework
3. GradeSQL Training Datasets
4. Pretrained GradeSQL ORMs

**Table 13.3** ORM performance on *BIRD* and *Spider* with the unbalanced dataset, reporting *Execution Accuracy* (%) on dev and test sets.

| ORM Model               | Bird dev | Spider dev | Spider test |
|-------------------------|----------|------------|-------------|
| Granite-3.3-2B-Instruct | 66.88    | 82.79      | 85.14       |
| Granite-3.3-8B-Instruct | 68.25    | 84.42      | 86.91       |
| Llama-3.2-1B-Instruct   | 67.67    | 84.04      | 86.31       |
| Llama-3.1-8B-Instruct   | 68.32    | 82.30      | 85.89       |
| Qwen2.5-1.5B-Instruct   | 68.12    | 84.33      | 86.17       |
| Qwen2.5-7B-Instruct     | 68.58    | 84.33      | 86.91       |
| OmniSQL-7B              | 68.64    | 84.42      | 86.63       |

the candidates are labeled correct, while only 30.6% are incorrect. BIRD shows a milder skew, with 58.63% correct and 41.37% incorrect. To investigate the impact of imbalance on ORM training, we constructed balanced versions of both datasets by downsampling. Specifically, for each question, we retained  $\min(s_+, s_-)$  candidates, ensuring a 50/50 split between positive and negative labels. The resulting statistics are summarized in Table 13.2.

With these datasets prepared, we proceed to train ORMs on the seven selected models to address three main research questions:

- (RQ1) Are ORMs consistently trainable across heterogeneous model families?
- (RQ2) How does dataset balancing influence ORM performance?
- (RQ3) How effective is the best ORM compared to Majority Voting and execution-based BoN?

**RQ1 – Cross-Family Trainability of ORMs.** Table 13.3 reports ORM performance on *BIRD* and *Spider* using the *unbalanced* training set, across Granite, Llama, Qwen, and OmniSQL backbones. The trained ORMs exhibit tightly clustered scores: *Bird dev* ranges from 66.88 to 68.64, *Spider dev* from 82.30 to 84.42, and *Spider test* from 85.14 to 86.91. These close score range shows ORMs train reliably across LLM families.

A closer look reveals three key patterns. The first is that the scale effect is modest: larger variants such as Granite-3.3-8B or Qwen2.5-7B yield only small improvements (typically below two points) over their smaller counterparts, suggesting that ORM effectiveness is not strongly tied to parameter count. The second is family-

**Table 13.4** Execution accuracy (%) of ORM models on *BIRD dev*, *Spider dev*, and *Spider test*. Each cell shows results on *Unbalanced* and *Balanced* sets with  $\Delta$  (green  $\uparrow$  = gain, red  $\downarrow$  = drop). Ordered by *BIRD Unbalanced*; best in **bold**, runner-up underlined.

| ORM Model               | BIRD dev                                                       | Spider dev                                                     | Spider test                                             |
|-------------------------|----------------------------------------------------------------|----------------------------------------------------------------|---------------------------------------------------------|
|                         | Unbalanced $\rightarrow$ Balanced                              | Unbalanced $\rightarrow$ Balanced                              | Unbalanced $\rightarrow$ Balanced                       |
| Granite-3.3-2B-Instruct | 66.88 $\rightarrow$ 66.17<br>(-0.71 $\downarrow$ )             | 82.79 $\rightarrow$ 82.79<br>(0.00)                            | 85.14 $\rightarrow$ 85.14<br>(0.00)                     |
| Llama-3.2-1B-Instruct   | 67.67 $\rightarrow$ 67.21<br>(-0.46 $\downarrow$ )             | 84.04 $\rightarrow$ 83.56<br>(-0.48 $\downarrow$ )             | 86.31 $\rightarrow$ 86.54<br>(+0.23 $\uparrow$ )        |
| Qwen2.5-1.5B-Instruct   | 68.12 $\rightarrow$ 67.28<br>(-0.84 $\downarrow$ )             | 84.33 $\rightarrow$ 83.56<br>(-0.77 $\downarrow$ )             | 86.17 $\rightarrow$ 86.31<br>(+0.14 $\uparrow$ )        |
| Granite-3.3-8B-Instruct | 68.25 $\rightarrow$ 68.38<br>(+0.13 $\uparrow$ )               | 84.42 $\rightarrow$ 83.85<br>(-0.57 $\downarrow$ )             | 86.91 $\rightarrow$ <u>87.00</u><br>(+0.09 $\uparrow$ ) |
| Llama-3.1-8B-Instruct   | 68.32 $\rightarrow$ 67.86<br>(-0.46 $\downarrow$ )             | 82.30 $\rightarrow$ 84.11<br>(+1.81 $\uparrow$ )               | 85.89 $\rightarrow$ 86.96<br>(+1.07 $\uparrow$ )        |
| Qwen2.5-7B-Instruct     | 68.58 $\rightarrow$ 68.19<br>(-0.39 $\downarrow$ )             | 84.33 $\rightarrow$ 84.14<br>(-0.19 $\downarrow$ )             | 86.91 $\rightarrow$ 86.68<br>(-0.23 $\downarrow$ )      |
| OmniSQL-7B              | <u>68.64</u> $\rightarrow$ <b>68.90</b><br>(+0.26 $\uparrow$ ) | <u>84.42</u> $\rightarrow$ <b>84.53</b><br>(+0.11 $\uparrow$ ) | 86.63 $\rightarrow$ <b>87.47</b><br>(+0.84 $\uparrow$ ) |

agnostic generalization: each family contributes at least one competitive ORM, for instance, Granite-3.3-8B and Qwen2.5-7B rank at the top of the *Spider test*, while Llama and OmniSQL models perform comparably on both development and test splits. This indicates that architectural heterogeneity does not hinder successful ORM training. The third is the stability of rankings across splits and datasets: although *Spider* generally achieves higher EX Accuracy than *BIRD*, the relative orderings remain consistent, supporting the robustness of ORMs across evaluation settings.

Overall, these findings demonstrate that ORMs are *consistently trainable* across heterogeneous LLM families and scales, achieving comparable accuracies with minimal variance. This establishes their generality and motivates the use of any backbone family in the subsequent experiments, RQ2, which investigates the effect of dataset balancing, and RQ3, which benchmarks ORM performance against *test-time* baselines.

**RQ2 – The Role of Dataset Balancing.** Table 13.4 compares ORM performance on *BIRD dev*, *Spider dev*, and *Spider test* when trained on the *Unbalanced* versus *Balanced* datasets. Each cell reports accuracy before and after balancing, with the difference  $\Delta$  highlighted in green ( $\uparrow$ ) for improvements and red ( $\downarrow$ ) for drops.

The results reveal a nuanced picture: balancing leads to both gains and losses de-

pending on the backbone and dataset. On *BIRD dev*, several backbones exhibit small decreases, e.g. Qwen2.5-1.5B at  $-0.84$  and Qwen2.5-7B at  $-0.39$ , while Granite-3.3-8B ( $+0.13$ ) and OmniSQL-7B ( $+0.26$ ) show slight improvements. On *Spider dev*, the effect is more mixed: Llama-3.1-8B improves substantially ( $+1.81$ ), whereas others decline moderately, such as Qwen2.5-1.5B ( $-0.77$ ), or remain unchanged, as with Granite-3.3-2B ( $0.00$ ). By contrast, *Spider test* shows more consistent gains, with the largest improvements for Llama-3.1-8B ( $+1.07$ ), OmniSQL-7B ( $+0.84$ ), and Llama-3.2-1B ( $+0.23$ ).

Overall, on *Spider* balancing appears to enhance *generalization*. While accuracy on the development set fluctuates, on the test set balancing lead an overall improvements for the majority of the models. The effect is particularly notable for Llama-3.1-8B and OmniSQL-7B, where gains exceed  $+1.0$  and  $+0.8$  points. At the same time, the overall relatively small magnitude of changes (generally within  $\pm 1.2$ ) suggests that ORMs remain stable even under skewed distributions of labels.

Furthermore, as shown in Table 13.2, the number of training samples differs substantially: the imbalanced sets contain over 80k (*BIRD train*) and 50k (*Spider train*) examples, while the balanced counterparts reduce to just 30k and 18k, respectively. This reduction makes training considerably faster, lowering both computational cost and time-to-train. Despite the smaller datasets, ORM performance remains largely comparable across settings, with overall differences being minimal. More interestingly, in certain cases balancing even improves results, for instance, the best-performing models on *BIRD dev*, such as OmniSQL-7B, achieve higher execution accuracy when trained on the smaller, balanced set.

In summary, dataset balancing not only reduces data requirements and training time, but also yields competitive, and sometimes superior, performance. Based on these trends, we adopt the *Balanced* configuration as the default setup for the comparison against baselines in RQ3.

### **RQ3 – ORM Effectiveness vs. Baseline Methods.**

Table 13.5 compares the ORM-based Best-of- $N$  strategy with execution-based BoN (ex-BoN) and Majority Voting (Maj), using OmniSQL-7B as the candidate generator. The ORM is trained on OmniSQL-7B with the balanced dataset. We report execution accuracy on *BIRD dev*, *Spider dev*, and *Spider test*, along with the absolute gains over the single-sample baseline ( $N = 1$ ), corresponding to the first generated query. Following [105], we apply the McNemar statistical test together with

**Table 13.5** Execution accuracy (%) of OmniSQL-7B on *BIRD dev*, *Spider dev*, and *Spider test*.  $\Delta$  indicates the gain over the baseline ( $N=1$ ). Best and runner-up results are in **bold** and underlined, respectively. Values marked with  $\dagger$  are not McNemar significant ( $p < 0.05$ ), and those with  $*$  lose significance after Bonferroni correction.

| Method                     | BIRD dev     |          | Spider dev        |          | Spider test  |          |
|----------------------------|--------------|----------|-------------------|----------|--------------|----------|
|                            | EX           | $\Delta$ | EX                | $\Delta$ | EX           | $\Delta$ |
| Baseline ( $N=1$ )         | 63.89        | –        | 82.40             | –        | 84.02        | –        |
| Majority Voting ( $N=32$ ) | <u>66.95</u> | +3.06    | <u>83.75</u> *    | +1.35    | <u>85.47</u> | +1.45    |
| Best-of- $N$ ( $N=32$ )    |              |          |                   |          |              |          |
| Execution-based            | 66.04        | +2.15    | 82.79 $\dagger$ * | +0.39    | 85.14        | +1.12    |
| ORM-based                  | <b>68.90</b> | +5.01    | <b>84.53</b> *    | +2.13    | <b>87.47</b> | +3.45    |

the Bonferroni correction to assess significance across methods. Results indicate that the ORM-based strategy consistently achieves the highest execution accuracy across all datasets and evaluation splits. Although the improvement on *Spider dev* does not retain significance after Bonferroni correction, which may be attributed to the smaller sample size, all other results remain statistically significant

Specifically, on *BIRD dev*, ORMs reach 68.90% execution accuracy, surpassing both Maj (66.95%) and ex-BoN (66.04%), and achieving a gain of +5.01 points over the baseline. On *Spider dev*, ORMs again outperform all baselines with 84.53% accuracy, a +2.13 improvement compared to 83.75% for Maj and 82.79% for ex-BoN. The same trend holds on *Spider test*, where ORM-based selection achieves 87.47%, outperforming both Maj (85.47%) and ex-BoN (85.14%), with an absolute gain of +3.45 points over the baseline.

These results highlight a clear pattern: while heuristic strategies such as Majority Voting and execution-based Best-of- $N$  offer gains over the single-sample baseline, ORM-based selection delivers consistent and stronger improvements. Notably, the largest gain is observed on *BIRD dev* (+5.01), indicating that ORMs provide the greatest advantage in scenarios with more challenging or diverse query distributions.

Overall, these findings demonstrate that ORMs not only generalize across model families (RQ1) and benefit from dataset balancing (RQ2), but also achieve improvements over widely used test-time strategies. This represents the central contribution of our work: showing that ORM-based selection outperforms Majority Voting and execution-based Best-of- $N$ , providing a stronger heuristic for Best-of- $N$  test-time inference.

**Table 13.6** Execution accuracy (%) and Pass@ $N$  of test-time strategies (execution-based BoN, Majority Voting, ORM-based BoN) across varying  $N$  values on the *BIRD dev*, *Spider dev*, and *Spider test* benchmarks. Columns are color-coded from light yellow (lowest) to dark orange (highest) performance..

| $N$ | Execution-based BoN |            |             | Majority Voting |            |             | ORM-based BoN |            |             | Pass@ $N$ |            |             |
|-----|---------------------|------------|-------------|-----------------|------------|-------------|---------------|------------|-------------|-----------|------------|-------------|
|     | BIRD dev            | Spider dev | Spider test | BIRD dev        | Spider dev | Spider test | BIRD dev      | Spider dev | Spider test | BIRD dev  | Spider dev | Spider test |
| 32  | 66.04               | 82.79      | 85.14       | 66.95           | 83.75      | 85.47       | 68.90         | 84.53      | 87.47       | 80.57     | 91.68      | 93.29       |
| 31  | 66.04               | 82.79      | 85.14       | 67.08           | 83.75      | 85.47       | 68.58         | 84.53      | 87.47       | 80.25     | 91.68      | 93.20       |
| 30  | 66.04               | 82.88      | 85.14       | 67.01           | 83.66      | 85.42       | 68.58         | 84.62      | 87.47       | 80.25     | 91.68      | 93.15       |
| 29  | 65.97               | 82.88      | 85.14       | 66.95           | 83.56      | 85.42       | 68.51         | 84.62      | 87.42       | 80.12     | 91.68      | 93.11       |
| 28  | 65.97               | 82.88      | 85.14       | 67.01           | 83.75      | 85.47       | 68.58         | 84.62      | 87.38       | 80.05     | 91.68      | 93.01       |
| 27  | 65.97               | 82.88      | 85.14       | 67.14           | 83.85      | 85.47       | 68.58         | 84.62      | 87.38       | 79.92     | 91.59      | 93.01       |
| 26  | 65.97               | 82.88      | 85.14       | 66.75           | 83.75      | 85.33       | 68.64         | 84.62      | 87.38       | 79.92     | 91.59      | 92.87       |
| 25  | 65.97               | 82.88      | 85.14       | 66.75           | 83.85      | 85.51       | 68.71         | 84.62      | 87.38       | 79.79     | 91.59      | 92.87       |
| 24  | 65.97               | 82.88      | 85.14       | 67.01           | 83.95      | 85.56       | 68.77         | 84.62      | 87.33       | 79.53     | 91.59      | 92.78       |
| 23  | 65.97               | 82.88      | 85.14       | 67.08           | 84.14      | 85.47       | 68.71         | 84.62      | 87.33       | 79.47     | 91.49      | 92.78       |
| 22  | 65.97               | 82.88      | 85.14       | 66.88           | 83.95      | 85.42       | 68.90         | 84.62      | 87.33       | 79.33     | 91.39      | 92.73       |
| 21  | 65.91               | 82.88      | 85.14       | 66.75           | 83.85      | 85.47       | 68.84         | 84.62      | 87.28       | 79.14     | 91.20      | 92.73       |
| 20  | 65.91               | 82.88      | 85.14       | 67.01           | 83.75      | 85.56       | 68.77         | 84.62      | 87.33       | 79.07     | 91.01      | 92.69       |
| 19  | 65.91               | 82.88      | 85.14       | 67.08           | 83.85      | 85.37       | 68.77         | 84.62      | 87.33       | 78.88     | 90.91      | 92.59       |
| 18  | 65.91               | 82.88      | 85.14       | 67.01           | 83.76      | 85.65       | 68.84         | 84.62      | 87.38       | 78.88     | 90.81      | 92.55       |
| 17  | 65.91               | 82.88      | 85.14       | 67.08           | 83.95      | 85.61       | 68.77         | 84.72      | 87.33       | 78.68     | 90.81      | 92.50       |
| 16  | 65.91               | 82.88      | 85.14       | 66.95           | 83.66      | 85.61       | 68.77         | 84.53      | 87.33       | 78.49     | 90.72      | 92.40       |
| 15  | 65.84               | 82.88      | 85.14       | 67.14           | 83.85      | 85.42       | 68.71         | 84.53      | 87.28       | 78.23     | 90.52      | 92.27       |
| 14  | 65.84               | 82.88      | 85.10       | 67.01           | 83.85      | 85.61       | 68.64         | 84.53      | 87.19       | 77.90     | 90.52      | 92.13       |
| 13  | 65.84               | 82.88      | 85.10       | 67.14           | 83.66      | 85.56       | 68.64         | 84.53      | 87.19       | 77.84     | 90.43      | 92.08       |
| 12  | 65.84               | 82.88      | 85.05       | 66.88           | 83.85      | 85.51       | 68.71         | 84.44      | 86.96       | 77.71     | 90.43      | 91.94       |
| 11  | 65.84               | 82.88      | 85.05       | 66.82           | 83.66      | 85.19       | 68.64         | 84.33      | 86.87       | 77.25     | 90.23      | 91.71       |
| 10  | 65.71               | 82.88      | 85.05       | 66.69           | 84.14      | 85.56       | 68.64         | 84.33      | 86.91       | 76.99     | 90.04      | 91.66       |
| 9   | 65.71               | 82.88      | 85.05       | 66.49           | 84.04      | 85.65       | 68.51         | 84.24      | 86.91       | 76.60     | 89.85      | 91.62       |
| 8   | 65.71               | 82.88      | 85.05       | 66.95           | 83.95      | 85.61       | 68.51         | 84.33      | 86.82       | 76.14     | 89.65      | 91.48       |
| 7   | 65.58               | 82.79      | 85.05       | 66.62           | 84.04      | 85.70       | 68.38         | 84.33      | 86.77       | 75.36     | 89.26      | 91.06       |
| 6   | 65.45               | 82.79      | 85.00       | 66.43           | 83.66      | 85.75       | 67.86         | 84.24      | 86.54       | 74.58     | 89.17      | 90.82       |
| 5   | 65.38               | 82.79      | 85.00       | 65.97           | 83.95      | 85.65       | 67.47         | 84.14      | 86.49       | 73.79     | 88.59      | 90.54       |
| 4   | 65.32               | 82.88      | 85.00       | 65.32           | 83.66      | 85.47       | 67.34         | 84.04      | 86.49       | 72.82     | 88.30      | 90.13       |
| 3   | 65.12               | 82.88      | 84.91       | 65.19           | 83.56      | 85.33       | 66.75         | 84.04      | 86.17       | 70.86     | 87.43      | 89.10       |
| 2   | 64.86               | 82.79      | 84.63       | 64.41           | 83.08      | 84.44       | 66.04         | 83.66      | 85.75       | 68.84     | 86.07      | 87.66       |
| 1   | 63.89               | 82.40      | 84.02       | 63.89           | 82.40      | 84.02       | 63.89         | 82.40      | 84.02       | 63.89     | 82.40      | 84.02       |

## 13.6 Ablation Studies

In this section, we further investigate ORM training by addressing additional research questions to better understand its effectiveness compared to alternative methods. Specifically, we aim to answer:

(RQ4) What is the role of  $N$  in ORM performance?

(RQ5) Are ORMs effective when using larger base models as generators?

(RQ6) How does the training prompt influence ORM results?

(RQ7) Are ORM sizes consistent with previous findings when scaling beyond 7B models?

(RQ8) Is autoregressive fine-tuning the best choice for training an ORM?

**RQ4 – Effect of  $N$  on ORM Performance.** *How the number of candidates  $N$  influence the performance of ORM-based selection?* Table 13.6 present the effect of varying  $N$  on execution accuracy across *BIRD dev*, *Spider dev*, and *Spider test*. We compare

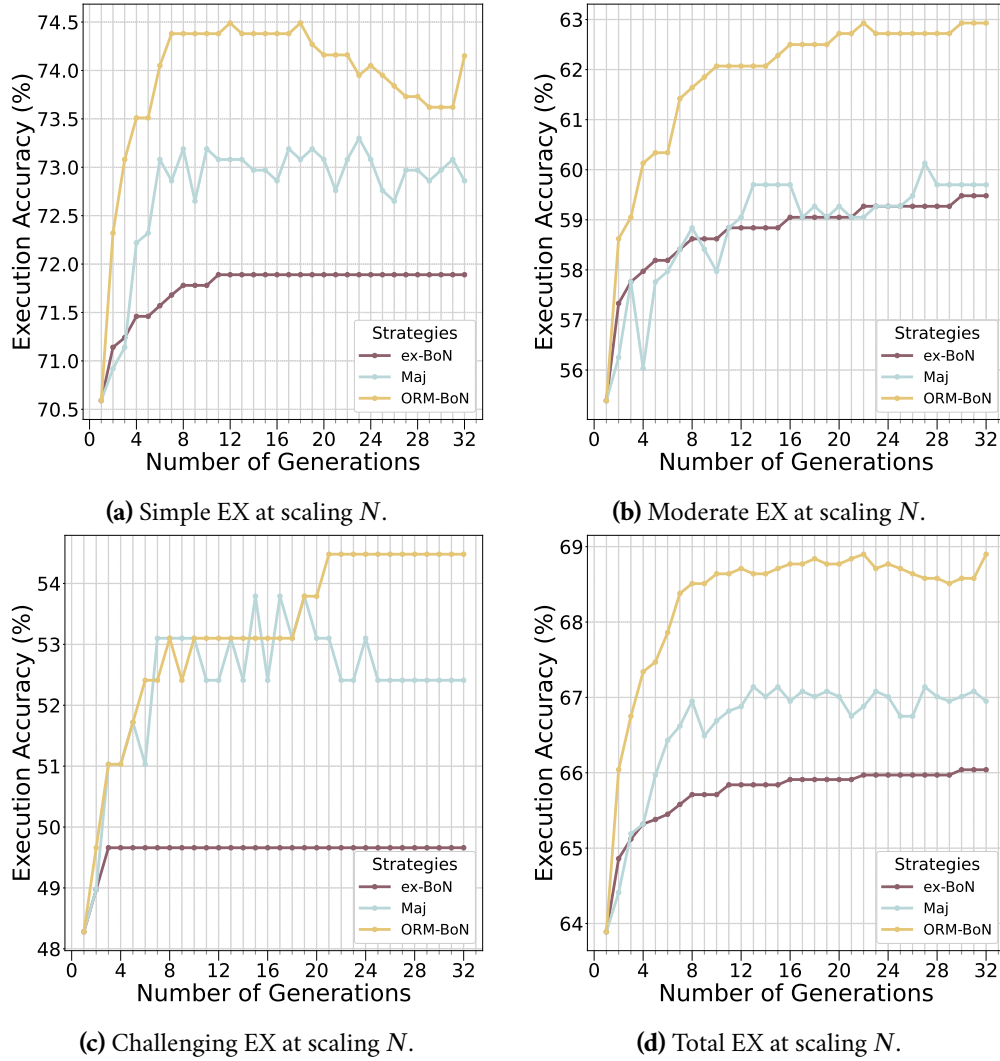


**Figure 13.2** Execution accuracy (%) on *BIRD dev* across different  $N$  values, comparing execution-based Best-of- $N$  (ex-BoN), Majority Voting (Maj), and ORM-based Best-of- $N$  (ORM-BoN). Results are shown by query difficulty (Simple, Moderate, Challenging) and overall (Total).

four strategies: execution-based Best-of- $N$ , Majority Voting, ORM-based Best-of- $N$ , and Pass@ $N$ , the latter serving as an upper bound on achievable performance by reporting the proportion of queries for which at least one correct candidate is generated. The results show that ORM-based selection consistently achieves higher accuracy than heuristic methods across all values of  $N$ , while also approaching the upper bound defined by Pass@ $N$  more closely as  $N$  increases.

On *BIRD dev*, ORM accuracy increases from 63.89% at  $N = 1$  to 68.71% at  $N = 25$ , after which it stabilizes with only minor fluctuations, reaching 68.90% at  $N = 32$ . In contrast, execution-based Best-of- $N$  plateaus earlier around 66.0%, while Majority Voting converges near 67.0%. A similar pattern emerges on *Spider dev*, where ORM steadily improves to 84.62–84.72% in the range  $N = 17$ –30, compared to 82.88% for execution-based Best-of- $N$  and 84.14% for Majority Voting. On *Spider test*, ORM achieves 87.47% at  $N = 32$ , maintaining a consistent margin of +2 to +3 points over both baselines. Overall, these results show that ORMs not only scale more effectively with increasing candidate pools but also sustain their advantage as  $N$  grows, underscoring their effectiveness as a selection mechanism.

Figure 13.2 further decomposes performance by query difficulty (*Simple, Moderate, Challenging*) on the *BIRD dev* set. For simple queries, all methods improve with increasing  $N$ , but ORM consistently maintains a margin of 1–2 points over Majority Voting and 2–3 points over execution-based Best-of- $N$ . The advantage becomes more pronounced for moderate queries, where ORM shows a smooth upward trajectory and stabilizes near 63%, while both baselines plateau earlier at lower levels. The most striking gains are observed on challenging queries: here, ORM substantially outperforms both strategies, with improvements exceeding 3–4 points at higher  $N$ , whereas execution-based Best-of- $N$  remains flat and Majority Voting oscillates with unstable progress. These results highlight that ORM selection is particularly effective in harder scenarios, where discriminating between semantically correct and incorrect candidates becomes critical.

Overall, these findings show that ORM-based selection benefits consistently from larger candidate pools, outperforming the other heuristic strategies across datasets, evaluation splits, and difficulty levels. While Majority Voting and execution-based Best-of- $N$  provide incremental improvements at low values of  $N$ , they quickly saturate, whereas ORM continues to scale and more closely approaches the oracle bound defined by Pass@ $N$ . This reinforces the conclusion that ORMs provide a



**Table 13.7** Execution accuracy (%) and Pass@32 on the *BIRD dev*, *Spider dev*, and *Spider test*. Candidate queries are generated by OmniSQL models with 7B, 14B, and 32B parameters, while the ORM is implemented using OmniSQL-7B. Best scores per column are shown in **bold**.

|             | Model       | Baseline | Ex-BoN | Majority Voting | ORM-BoN | Pass@32 |
|-------------|-------------|----------|--------|-----------------|---------|---------|
| BIRD dev    | OmniSQL-7B  | 63.89    | 66.04  | 66.95           | 68.90   | 80.57   |
|             | OmniSQL-14B | 64.28    | 66.75  | 67.01           | 69.04   | 80.90   |
|             | OmniSQL-32B | 64.02    | 66.10  | 63.95           | 69.30   | 82.20   |
| Spider dev  | OmniSQL-7B  | 82.40    | 82.79  | 83.75           | 84.33   | 91.68   |
|             | OmniSQL-14B | 82.01    | 82.20  | 83.56           | 84.62   | 92.75   |
|             | OmniSQL-32B | 83.17    | 83.37  | 83.08           | 84.43   | 92.84   |
| Spider test | OmniSQL-7B  | 84.02    | 85.14  | 85.47           | 87.33   | 93.29   |
|             | OmniSQL-14B | 84.91    | 85.28  | 84.86           | 87.42   | 93.53   |
|             | OmniSQL-32B | 84.30    | 84.72  | 86.07           | 87.19   | 94.08   |

principled and effective mechanism for leveraging increased generation budgets, ensuring more reliable gains from test-time compute.

**RQ5 – ORM Effectiveness with Larger Generator Models.** *How does ORM performance vary when the underlying generator model is scaled to larger parameter sizes?* Table 13.7 reports execution accuracy and Pass@32 across OmniSQL models of different sizes (7B, 14B, 32B) on *BIRD dev*, *Spider dev*, and *Spider test*. The ORM is fixed to OmniSQL-7B, while the generator varies in scale. Results show that ORM-based selection remains consistently competitive across all generator sizes, outperforming Majority Voting (Maj) and Execution-based Best-of- $N$  (ex-BoN) methods, while approaching the oracle-like upper bound defined by Pass@32.

On *BIRD dev*, ORM achieves the highest accuracy with all generator sizes, reaching 68.90% with 7B, 69.04% with 14B, and 69.30% with 32B, demonstrating that ORM continues to provide robust gains even as the generator becomes larger. Interestingly, while Majority Voting and Best-of- $N$  improve moderately with 14B, they fall behind at 32B, where ORM remains stable and sustains its advantage. On *Spider dev*, ORM also achieves the strongest results, with 84.33% (7B), 84.62% (14B), and 84.43% (32B), consistently outperforming heuristic baselines by 0.5–2 points. A similar trend is observed on *Spider test*, where ORM yields 87.33% (7B), 87.42% (14B), and 87.19% (32B), surpassing both Maj and ex-BoN in nearly all settings.

These results suggest that ORM effectiveness is not diminished by scaling the generator. While larger models naturally improve baseline performance, the rela-

**Table 13.8** Ablation study on verification prompt design for ORM on *BIRD dev*. Execution accuracy (%) by query difficulty. Generator: OmniSQL-7B; ORM: Qwen2.5-7B-Instruct. Best scores in **bold**.

| Prompt Variant | Simple       | Moderate     | Challenging  | Total        |
|----------------|--------------|--------------|--------------|--------------|
| Instruction    | 72.65        | 59.70        | 51.32        | 66.72        |
| Data-Only      | 72.76        | 59.48        | 51.72        | 66.75        |
| SQL-Only       | 72.86        | <b>63.36</b> | <b>53.79</b> | <b>68.19</b> |
| Data + SQL     | <b>73.08</b> | 62.50        | 50.34        | 67.73        |

tive improvements provided by ORM remain intact. Moreover, ORM demonstrates greater stability across datasets and evaluation splits, whereas other heuristics exhibit fluctuations depending on model size. The consistently strong performance of ORM with larger generators indicates that its benefits are orthogonal to scale: ORM adds value not by replacing the advantages of larger models, but by maximizing the utility of the generated candidate set regardless of size.

In summary, ORM-based selection delivers consistent accuracy improvements even as the underlying generator grows to 14B and 32B parameters. By sustaining its advantage across scales and datasets, ORM establishes itself as a robust and scalable strategy for test-time SQL generation, ensuring reliable gains whether deployed with medium or large generator backbones.

**RQ6 – Impact of Training Prompt on ORM Results.** *How does the design of verification prompts affect ORM performance?* Table 13.8 presents an ablation study on prompt design for ORM verification using Qwen2.5-7B-Instruct as the verifier and OmniSQL-7B as the generator, evaluated on the *BIRD dev* set. Four prompt variants are compared: (i) *Instruction*, which provides the SQL query with explicit verification instructions (Supplementary Material 13.7); (ii) *Data-Only*, which provides only the result set produced by executing the query (Supplementary Material 13.7); (iii) *SQL-only*, which includes only the query (Supplementary Material 13.7); and (iv) *Data+SQL*, which provides both the query and its result set (Supplementary Material 13.7). Results show that the choice of prompt can influence ORM effectiveness, particularly for moderate and challenging queries, while differences are less pronounced for simple ones.

For simple queries, all prompts yield nearly identical results, with accuracies clustering around 72–73%, suggesting that prompt design plays a minimal role when the task is straightforward and candidates are easy to discriminate. Differences

**Table 13.9** Impact of ORM model size with the generator fixed to OmniSQL-7B. Execution Accuracy (EX, %) and Pass@32 are reported on *BIRD dev*, *Spider dev*, and *Spider test*. Best results per metric and dataset are shown in **bold**.

| ORM         | BIRD dev     |         | Spider dev   |         | Spider test  |         |
|-------------|--------------|---------|--------------|---------|--------------|---------|
|             | EX           | Pass@32 | EX           | Pass@32 | EX           | Pass@32 |
| OmniSQL-7B  | 68.90        | 80.57   | 84.53        | 91.68   | 87.47        | 93.29   |
| OmniSQL-14B | <b>69.23</b> | 80.57   | <b>85.01</b> | 91.68   | <b>87.84</b> | 93.29   |
| OmniSQL-32B | 68.19        | 80.57   | 84.23        | 91.68   | 86.91        | 93.29   |

emerge more clearly for moderate queries, where the SQL-only prompt achieves the strongest accuracy (63.36%), outperforming the next-best SQL+Data prompt by nearly one point, while the instruction-style and data-only variants lag behind at 59–60%. This trend becomes even more evident in challenging queries, where the SQL-only prompt again delivers the best results (53.79%), surpassing all other formulations by more than two points. These gains translate into the highest overall average accuracy for the SQL-only prompt (68.19%), followed by the SQL+Data variant (67.73%), while the instruction and data-only variants converge at roughly 66.7%.

These findings suggest that the most effective verification signal for ORM training comes from exposing the verifier directly to SQL structures, without additional natural language instructions or contextual data. While including schema or data information can slightly improve performance in moderate cases, excessive context appears to reduce the verifier’s discriminative ability.

Overall, this ablation highlights that ORM performance is sensitive to prompt design, but the most notable gains occur in moderate and challenging queries where discrimination is more difficult. The SQL-only prompt emerges as the most effective design, establishing that lightweight, structure-focused inputs are more beneficial for ORM training than verbose instruction-style formulations.

**RQ7 – Effect of ORM Size Beyond 7B Models.** *Does scaling the ORM verifier beyond 7B parameters lead to performance improvements?* Table 13.9 evaluates ORM models of different sizes (7B, 14B, 32B) with the generator fixed to OmniSQL-7B. We report execution accuracy (EX) and Pass@32 on *BIRD dev*, *Spider dev*, and *Spider test*. Results show that increasing ORM size beyond 7B yields only marginal and inconsistent improvements, suggesting diminishing returns when scaling the verifier model.

On *BIRD dev*, the 14B ORM achieves the highest accuracy (69.23%), a modest gain

**Table 13.10** Ablation study comparing autoregressive and Binary Cross-Entropy (BCE) loss fine-tuning on *BIRD dev* across query difficulty levels. Both the generator and ORM model are OmniSQL-7B. Reported metric is Execution Accuracy (%). Best results are in **bold**.

| Method            | Simple       | Moderate     | Challenging  | Total        |
|-------------------|--------------|--------------|--------------|--------------|
| Majority Voting   | 72.86        | 59.70        | 52.41        | 66.95        |
| BCE FT            | 72.86        | 59.91        | 51.72        | 66.95        |
| Autoregressive FT | <b>74.15</b> | <b>62.93</b> | <b>54.48</b> | <b>68.90</b> |

over the 7B variant (68.90%), while the 32B model slightly underperforms at 68.19%. On *Spider dev*, the 14B ORM again leads with 85.01%, compared to 84.53% for 7B and 84.23% for 32B. A similar pattern is observed on *Spider test*, where the 14B ORM achieves 87.84%, outperforming both 7B (87.47%) and 32B (86.91%).

These findings suggest that scaling the ORM beyond 7B parameters does not yield consistent accuracy gains. The modest improvements observed with the 14B model are offset by performance drops with the 32B variant, pointing to a saturation effect where additional verifier capacity brings limited benefit. Importantly, the stability of Pass@32 across ORM sizes underscores that larger verifiers cannot compensate for the inherent limitations of the generator. In practice, this indicates that a 7B ORM is sufficient to achieve strong results, offering a favorable balance between performance and computational efficiency.

Overall, RQ7 highlights that, unlike generator scaling which provides steady improvements (RQ5), ORM scaling exhibits diminishing returns. This reinforces the idea that ORM effectiveness is primarily driven by its training signal and alignment objective rather than raw parameter count, and that small to medium-sized verifiers are adequate for practical deployment.

**RQ8 – Autoregressive vs. Cross-Entropy Training.** *How do different fine-tuning objectives influence ORM effectiveness?* Table 13.10 presents an ablation study on the *BIRD dev*, comparing two fine-tuning strategies, autoregressive and Binary Cross-Entropy (BCE), against a Majority Voting baseline. In the autoregressive setting, the ORM is trained under the causal language modeling objective, where the model learns to generate the target label token (e.g., “Yes” or “No”) conditioned on the input prompt. In contrast, BCE fine-tuning reformulates the task as a binary classification problem: the prompt is encoded and a classification head directly predicts the probability of correctness. Both the generator and ORM are instantiated from OmniSQL-

7B, and we report execution accuracy across queries of varying difficulty levels. Results show that autoregressive fine-tuning consistently outperforms BCE training and the Majority Voting baseline, yielding the strongest overall performance.

For simple queries, all three methods show similar accuracy, around 72–74%. The autoregressive ORM reaches 74.15%, slightly outperforming BCE (72.86%) and Majority Voting (72.86%). The differences become more apparent in moderate queries, where autoregressive fine-tuning achieves 62.93%, compared to 59.91% for BCE and 59.70% for Majority Voting. On challenging queries, the advantage is again clear: the autoregressive ORM reaches 54.48%, while BCE and Majority Voting lag behind at 51.72% and 52.41%, respectively. These improvements accumulate to the highest total accuracy of 68.90% for the autoregressive ORM, surpassing both BCE (66.95%) and Majority Voting (66.95%).

These findings indicate that training ORMs with an autoregressive objective improves performance across all levels, with the largest gains in harder queries. In contrast, BCE fine-tuning, appears less effective at handling moderate and challenging cases.

Overall, with RQ8 we demonstrate that the choice of training objective has a significant impact on ORM effectiveness. While losses such as BCE can match heuristics like Majority Voting, autoregressive fine-tuning consistently delivers superior results, establishing it as the most effective approach for aligning ORMs with the requirements of the Text-to-SQL verification task.

## 13.7 Summary

We introduced *GradeSQL*, a framework for synthesizing data and training Outcome Reward Models (ORMs) for the Text-to-SQL task, enabling the ranking of SQL queries generated by LLMs. *GradeSQL* demonstrates how ORMs can be leveraged as an effective heuristic strategy for Best-of- $N$  (BoN) and as an alternative Test-Time Inference (TTI) approach to execution-based BoN (ex-BoN) and Majority Voting (Maj).

Our study shows that ORMs can be consistently trained across heterogeneous model families, benefit from low-scale yet balanced datasets, and achieve competitive performance on the BIRD and Spider benchmarks compared to ex-BoN and Maj for test-time inference. Moreover, ORM effectiveness remains robust across different values of  $N$ , scales with stronger generator models, and is influenced by

design choices such as training objectives and prompt formulation.

These findings indicate that ORMs provide an effective and semantically aligned mechanism for candidate selection in Text-to-SQL. We also observe diminishing returns when increasing ORM size, suggesting that lightweight verifiers are sufficient for practical deployment.

In conclusion, *GradeSQL* offers the community a unified framework for building ORMs in the Text-to-SQL setting, and our results underscore the potential of ORM-based selection as a competitive alternative to classical test-time inference strategies. Furthermore, we are releasing the framework<sup>5</sup>, datasets<sup>6</sup>, and pretrained ORMs<sup>7</sup> as open-source resources to support the community and foster further research in this direction.

5. <https://github.com/sisinflab/GradeSQL>

6. <https://huggingface.co/collections/sisinflab-ai/gradesql-training-datasets>

7. <https://huggingface.co/collections/sisinflab-ai/gradesql-models>

# Supplementary Material

---

## Prompts used

### Zero-Shot CoT SQL Generation Prompt Template

**System:** "You are a data science expert. Below, you are provided with a database schema and a natural language question. Your task is to understand the schema and generate a valid SQL query to answer the question."

Database Engine: SQLite

Database Schema: {db\_details}

This schema describes the database's structure, including tables, columns, primary keys, foreign keys, and any relevant relationships or constraints.

Question: {question}

Instructions:

- Make sure you only output the information that is asked in the question. If the question asks for a specific column, make sure to only include that column in the SELECT clause, nothing more.
- The generated query should return all of the information asked in the question without any missing or extra information.
- Before generating the final SQL query, please think through the steps of how to write the query.

Output Format:

In your answer, please enclose the generated SQL query in a code block:

```

Your SQL query

...

Take a deep breath and think step by step to find the correct SQL query."

SQL-Only Verification Prompt for ORM Evaluation

System: "Question: {db_schema + natural language question}
SQL: {sql candidate}
Is the SQL correct?"

Data-Only Verification Prompt for ORM Evaluation

System: "Question: {db_schema + natural language question}
Data retrieved: {data}
Based on data and question, in your opinion is the SQL correct?"

Data + SQL Verification Prompt for ORM Evaluation

System: "Question: {db_schema + natural language question}
Data: {data}
SQL: {sql candidate}
Is the SQL correct?"

Instruction Verification Prompt for ORM Evaluation

System: "You are an expert SQL evaluator. Given a database schema, a question, the SQL query and data returned, determine if the SQL query correctly answers the question based on the schema. Respond only with Yes or No.

Input:

Question and database schema: {db_schema + natural language question}

- SQL Query: {sql candidate}

- Data returned: {data}

Instructions:

1. Analyze the database schema to understand the table structure, columns, and relationships.

2. Evaluate the provided SQL query against the question to determine if it accurately retrieves the intended data.

3. Respond solely with Yes if the SQL query is correct, or No if it is incorrect.

Output:

Yes

or

No

Is the SQL correct?"

Inference schema

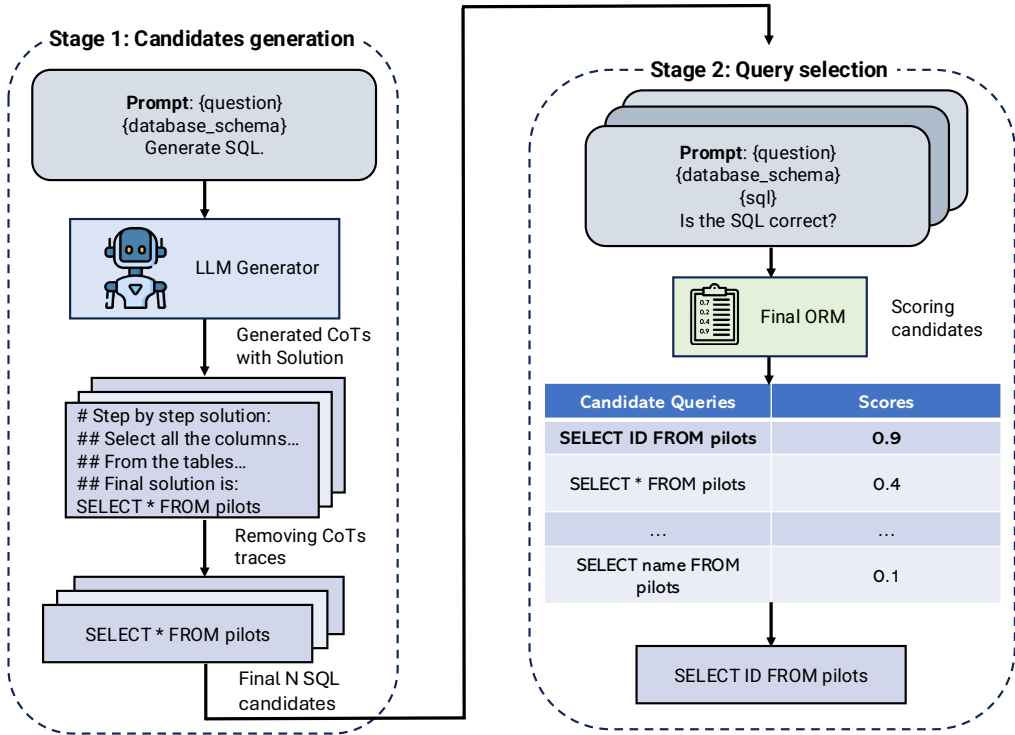


Figure 13.3 Overview of the GradeSQL ORM at inference. The pipeline consists of two stages: (i) Candidate Generation, where an LLM produces a pool of N SQL candidates; and (ii) Query Selection, where the ORM scores each candidate and ranks them, selecting the query with the highest probability as the final output.

Part 5

Closing Remarks

The rapid diffusion of Large Language Models (LLMs) has made them attractive building blocks for domain-specific systems, but strong performance in open-ended text generation does not automatically translate into reliable behaviour. When an LLM is used to support decisions, the main question is not only whether an answer appears plausible. The main question is whether the model can be checked, constrained, and evaluated under the practical requirements of the target domain.

We addressed this challenge through three complementary perspectives. First, we studied LLMs when used as recommender systems, showing that, with appropriate prompting, LLMs can infer user preferences and improve ranking quality even without explicit training for recommendation. At the same time, we found that their outputs resembled collaborative filtering patterns and could reinforce popularity bias, suggesting that part of the observed performance reflected regularities absorbed during pretraining. This hypothesis opened a new research direction concerning the presence of recommendation datasets in LLM pretraining.

This investigation led to the identification of a broader issue, namely data leakage. When recommender system datasets are included in the pretraining corpus of an LLM, observed performance may be driven by memorization rather than genuine generalization. Such memorization directly affects both evaluation and interpretation, as it can inflate results in ways that are difficult to identify without targeted analyses. To address this problem, we proposed a systematic probing approach that queried models for specific user–item interactions and assessed whether correct responses were more consistent with memorization or with generalization. These findings reinforced the need for leakage-aware evaluation and supported the conclusion that leakage checks should become a standard component of empirical assessment.

The second perspective focused on internal representations. We used probing to study what information is encoded in hidden states and how stable that information remains under realistic conditions. In the factuality setting, our results showed that hallucination detection from internal states is substantially harder when moving from controlled prompts to open-ended generation and noisy inputs. Within the same perspective, we also studied sentiment. We found that sentiment information is not evenly distributed across layers and that the final-token representation is not always the most informative choice. This observation motivated a layer-cutting approach that reduced memory usage while preserving, and in some cases improving, classification performance.

The third perspective investigated post-training strategies that improve performance without modifying model parameters. We studied a re-ranking module that generated multiple candidate reasoning traces and selected the most plausible one through a lightweight scoring step. Experiments on Text-to-SQL showed consistent accuracy gains, supporting the idea that improved inference-time procedures can enhance performance without requiring fine-tuning.

Taken together, our contributions support a unified view of using LLMs in domain-specific applications. Reliable integration requires more than effective prompting. It requires careful evaluation in realistic settings, explicit auditing for memorization, structured methods for understanding internal behaviour, and practical inference-time mechanisms that improve robustness.

At the same time, important challenges remain. Training data provenance is often unclear, which makes leakage auditing difficult and incomplete. Interpretability methods can be sensitive to changes in prompts and settings, and they must be strengthened to remain reliable under distribution shift. Finally, many domain requirements, such as privacy, safety, and operational constraints, cannot be guaranteed by instruction alone and must be enforced through system design.

In closing, we argue that Large Language Models should be treated as system components rather than standalone tools. Their effective use in real applications requires systematic analysis, leakage-aware evaluation, and principled integration strategies. The methods and evidence presented in this thesis contribute to this direction and support future work aimed at improving not only model capability, but also reliability and control.

Personal Thought. Three years have passed. Three years since my first step into the unknown terrain of research. As I read these pages, I feel a quiet sense of completeness, and a pride that is neither loud nor triumphant, but deeply earned. If someone had asked the version of myself from a year ago whether this moment would ever arrive, the answer would have been simple and sincere: “No. It is impossible. Everything seems to be falling apart.” When friends asked me how my PhD was going, a single thought always surfaced: It is a roller coaster. There are days when the climb feels effortless, and others when the fall is sudden and unforgiving. Yet, somehow, you always rise again.

I began this journey with a clear question and a fragile confidence, trying to understand how dialogue states could remain consistent in conversational recommender systems through the lens of game theory. I spent months learning, doubting, and shaping my first hypotheses. Then, almost overnight, the landscape changed. ChatGPT emerged, and with it came the realization that the problem I had been carefully dissecting appeared to dissolve in front of me. In that moment, the weight of lost time pressed heavily on my chest, and for the first time I felt the quiet fear of having built something on shifting ground. But research teaches adaptation. I turned my gaze toward Large Language Models, a field in constant motion, where each day brings new questions instead of final answers. It was harder, faster, and often overwhelming. Yet it carried me forward, and it taught me my first enduring lesson: there is always another path, if you are willing to walk it.

This journey was never smooth. It was marked by doubt, exhaustion, and nights spent questioning not only my work, but my worth. Many times, I wondered whether choosing an easier road would have spared me this weight. And yet, without this path, I would not recognize the person I see today. I grew, slowly and unevenly. I met people who became anchors in moments of uncertainty. I crossed cities that now live quietly in my memory. I shared tears with colleagues and laughter that made the hardest days bearable. This road was not gentle, but it was transformative. It gave me awareness, resilience, and a way of thinking that will stay with me far beyond these pages.

I will carry both light and shadow from this experience. It taught me lessons that cannot be measured, and it gave me encounters that would never have happened otherwise. If someone were to ask me whether I would begin a PhD again, I would probably smile and reply, “Never again.” Yet beneath the humor lies the

truth: I would choose this journey every time. The future remains uncertain, as it always is. I do not yet know where I am going. But I know this. I am capable of more than I once believed. There are no wrong paths, only paths taken with awareness and courage. There is always time to change direction, to begin again, and to move toward a form of fulfillment that makes the struggle meaningful, and the journey worth remembering.

Acknowledgements. This work was supported and promoted by MOST – Sustainable Mobility National Research Center funded by the European Union Next-GenerationEU (Italian National Recovery and Resilience Plan (NRRP) – M4C2, Investment 1.4 – D.D. 1033 17/06/2022, CN000000023 - CUP: D93C22000410001).

References

- [1] S. Abdelnabi, K. Greshake, S. Mishra, C. Endres, T. Holz, and M. Fritz. “Not What You’ve Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection”. In: *AISeC@CCS*. ACM, 2023, pp. 79–90.
- [2] H. Abdollahpouri and R. Burke. “Multistakeholder Recommender Systems”. In: *Recommender Systems Handbook*. Springer US, 2022, pp. 647–677.
- [3] H. Abdollahpouri, R. Burke, and B. Mobasher. “Controlling Popularity Bias in Learning-to-Rank Recommendation”. In: *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017*. Ed. by P. Cremonesi, F. Ricci, S. Berkovsky, and A. Tuzhilin. ACM, 2017, pp. 42–46. DOI: 10.1145/3109859.3109912.
- [4] H. Abdollahpouri, R. Burke, and B. Mobasher. “Managing Popularity Bias in Recommender Systems with Personalized Re-Ranking”. In: *Proceedings of the Thirty-Second International Florida Artificial Intelligence Research Society Conference, Sarasota, Florida, USA, May 19-22 2019*. Ed. by R. Barták and K. W. Brawner. AAAI Press, 2019, pp. 413–418.
- [5] H. Abdollahpouri, M. Mansoury, R. Burke, B. Mobasher, and E. C. Malthouse. “User-centered Evaluation of Popularity Bias in Recommender Systems”. In: *UMAP*. ACM, 2021, pp. 119–129.
- [6] M. H. A. Abdullah, N. Aziz, S. J. Abdulkadir, H. S. A. Alhussian, and N. Talpur. “Systematic Literature Review of Information Extraction From Textual Data: Recent Methods, Applications, Trends, and Challenges”. In: *IEEE Access* 11 (2023), pp. 10535–10562.

- [7] B. Abu-Salih, S. Alotaibi, A. L. Alanazi, R. A. Khurma, B. Al-Shboul, A. Khouri, and M. Aljaafari. "Using large language models for semantic interoperability: A systematic literature review". In: *ICT Express* 11.4 (2025), pp. 819–837.
- [8] G. Adomavicius and A. Tuzhilin. "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions". In: *IEEE Trans. Knowl. Data Eng.* 17.6 (2005), pp. 734–749.
- [9] R. Ahmed, S. Abdul-Rauf, and S. Latif. "Leveraging Large Language Models and Prompt Settings for Context-Aware Financial Sentiment Analysis". In: *ICACS*. IEEE, 2024, pp. 1–9.
- [10] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *KDD*. ACM, 2019, pp. 2623–2631.
- [11] A. Al-Kaswan, M. Izadi, and A. van Deursen. "Traces of Memorisation in Large Language Models for Code". In: *ICSE*. ACM, 2024, 78:1–78:12.
- [12] K. Al-Mannai, H. Alshikhabobakr, S. B. Wasi, R. Neyaz, H. Bouamor, and B. Mohit. "CMUQ-Hybrid: Sentiment Classification By Feature Engineering and Parameter Tuning". In: *SemEval@COLING*. The Association for Computer Linguistics, 2014, pp. 181–185.
- [13] G. Alain and Y. Bengio. "Understanding intermediate layers using linear classifier probes". In: *ICLR (Workshop)*. OpenReview.net, 2017.
- [14] F. M. Almutairi, N. D. Sidiropoulos, and B. Yang. "XPL-CF: Explainable Embeddings for Feature-based Collaborative Filtering". In: *CIKM*. ACM, 2021, pp. 2847–2851.
- [15] D. Altinok. "D-NLP at SemEval-2024 Task 2: Evaluating Clinical Inference Capabilities of Large Language Models". In: *SemEval@NAACL*. Association for Computational Linguistics, 2024, pp. 613–627.
- [16] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. "Natural language interfaces to databases - an introduction". In: *Nat. Lang. Eng.* 1.1 (1995), pp. 29–81.

- [17] V. W. Anelli, A. Bellogín, A. Ferrara, D. Malitesta, F. A. Merra, C. Pomo, F. M. Donini, and T. D. Noia. “Elliot: A Comprehensive and Rigorous Framework for Reproducible Recommender Systems Evaluation”. In: *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*. Ed. by F. Diaz, C. Shah, T. Suel, P. Castells, R. Jones, and T. Sakai. ACM, 2021, pp. 2405–2414. DOI: 10.1145/3404835.3463245.
- [18] V. W. Anelli, A. Bellogín, T. D. Noia, D. Jannach, and C. Pomo. “Top-N Recommendation Algorithms: A Quest for the State-of-the-Art”. In: *UMAP '22: 30th ACM Conference on User Modeling, Adaptation and Personalization, Barcelona, Spain, July 4 - 7, 2022*. Ed. by A. Bellogín, L. Boratto, O. C. Santos, L. Ardissono, and B. P. Knijnenburg. ACM, 2022, pp. 121–131. DOI: 10.1145/3503252.3531292.
- [19] V. Anelli, G. M. Biancofiore, A. D. Bellis, T. D. Noia, and E. D. Sciascio. “Interpretability of BERT Latent Space through Knowledge Graphs”. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*. Ed. by M. A. Hasan and L. Xiong. ACM, 2022, pp. 3806–3810. DOI: 10.1145/3511808.3557617.
- [20] R. Anil et al. “PaLM 2 Technical Report”. In: *CoRR abs/2305.10403* (2023).
- [21] D. Araci. “FinBERT: Financial Sentiment Analysis with Pre-trained Language Models”. In: *CoRR abs/1908.10063* (2019).
- [22] G. de Araujo, T. de Melo, and C. M. S. Figueiredo. “Is ChatGPT an effective solver of sentiment analysis tasks in Portuguese? A Preliminary Study”. In: *PROPOR. Association for Computational Linguistics*, 2024, pp. 13–21.
- [23] A. Arditi and O. B. Obeso. *Refusal Mechanisms: Initial Experiments with Llama-2-7b-chat*. <https://www.lesswrong.com/posts/pYcEhoAoPfHhgJ8YC>. Accessed: 2024-11-18. 2023.
- [24] A. Askari, C. Pölitiz, and X. Tang. “MAGIC: Generating Self-Correction Guideline for In-Context Text-to-SQL”. In: *AAAI*. AAAI Press, 2025, pp. 23433–23441.
- [25] A. Azaria and T. M. Mitchell. “The Internal State of an LLM Knows When It’s Lying”. In: *EMNLP (Findings)*. Association for Computational Linguistics, 2023, pp. 967–976.

- [26] J. Baek, H. Samulowitz, O. Hassanzadeh, D. Subramanian, S. Shirai, A. Gliozzo, and D. Bhattacharjya. “Knowledge Base Construction for Knowledge-Augmented Text-to-SQL”. In: *ACL (Findings)*. Association for Computational Linguistics, 2025, pp. 26569–26583.
- [27] D. Bahdanau, K. Cho, and Y. Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *ICLR*. 2015.
- [28] Y. Bai, Z. Han, Y. Zhao, H. Gao, Z. Zhang, X. Wang, and M. Hu. “Is Compound Aspect-Based Sentiment Analysis Addressed by LLMs?” In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Ed. by Y. Al-Onaizan, M. Bansal, and Y.-N. Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 7836–7861. DOI: 10.18653/v1/2024.findings-emnlp.460.
- [29] M. Balabanovic and Y. Shoham. “Content-Based, Collaborative Recommendation”. In: *Commun. ACM* 40.3 (1997), pp. 66–72.
- [30] K. Bao, J. Zhang, Y. Zhang, W. Wang, F. Feng, and X. He. “TALLRec: An Effective and Efficient Tuning Framework to Align Large Language Model with Recommendation”. In: *RecSys*. ACM, 2023, pp. 1007–1014.
- [31] C. Bauer, A. Said, and E. Zangerle. “Evaluation Perspectives of Recommender Systems: Driving Research and Education (Dagstuhl Seminar 24211)”. In: *Dagstuhl Reports* 14.5 (2024), pp. 58–172.
- [32] R. M. Bell and Y. Koren. “Lessons from the Netflix prize challenge”. In: *SIGKDD Explor.* 9.2 (2007), pp. 75–79.
- [33] A. D. Bellis, V. W. Anelli, T. D. Noia, and E. D. Sciascio. “PRONTO: Prompt-Based Detection of Semantic Containment Patterns in MLMs”. In: *The Semantic Web - ISWC 2024 - 23rd International Semantic Web Conference, Baltimore, MD, USA, November 11-15, 2024, Proceedings, Part II*. Ed. by G. Demartini, K. Hose, M. Acosta, M. Palmonari, G. Cheng, H. Skaf-Molli, N. Ferranti, D. Hernández, and A. Hogan. Vol. 15232. Lecture Notes in Computer Science. Springer, 2024, pp. 227–246. DOI: 10.1007/978-3-031-77850-6_13.
- [34] A. Bellogín, P. Castells, and I. Cantador. “Statistical biases in Information Retrieval metrics for recommender systems”. In: *Inf. Retr. J.* 20.6 (2017), pp. 606–634.

- [35] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?” In: *FAccT*. ACM, 2021, pp. 610–623.
- [36] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. “A Neural Probabilistic Language Model”. In: *J. Mach. Learn. Res.* 3 (2003), pp. 1137–1155.
- [37] G. M. Biancofiore, Y. Deldjoo, T. D. Noia, E. Di Sciascio, and F. Narducci. “Interactive question answering systems: Literature review”. In: *ACM Computing Surveys* 56.9 (2024), pp. 1–38.
- [38] G. M. Biancofiore, D. D. Palma, C. Pomo, F. Narducci, and T. D. Noia. “Conversational User Interfaces and Agents”. In: *Human-Centered AI: An Illustrated Scientific Quest*. Ed. by P. Germanakos, M. Juhasz, A. Kongot, D. Marathe, and D. Sacharidis. Springer Nature, 2025, pp. 399–438. doi: 10.1007/978-3-031-61375-3_4.
- [39] D. Billsus and M. J. Pazzani. “Learning Collaborative Information Filters”. In: *ICML*. 1998, pp. 46–54.
- [40] J. Bobadilla, F. Ortega, A. Hernando, and J. Bernal. “A collaborative filtering approach to mitigate the new user cold start problem”. In: *Knowl. Based Syst.* 26 (2012), pp. 225–238.
- [41] B. Bogin, J. Berant, and M. Gardner. “Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing”. In: *ACL (1)*. Association for Computational Linguistics, 2019, pp. 4560–4565.
- [42] R. Bommasani, D. A. Hudson, E. Adeli, R. B. Altman, S. Arora, S. von Arx, and et al. “On the Opportunities and Risks of Foundation Models”. In: *CoRR* abs/2108.07258 (2021).
- [43] L. Borchmann and M. Wydmuch. “Query and Conquer: Execution-Guided SQL Generation”. In: *CoRR* abs/2503.24364 (2025).
- [44] M. Bordoloi and S. K. Biswas. “Sentiment analysis: A survey on design framework, applications and future scopes”. In: *Artif. Intell. Rev.* 56.11 (2023), pp. 12505–12560.
- [45] P. du Boucher-Ryan and D. G. Bridge. “Collaborative Recommending using Formal Concept Analysis”. In: *Knowl. Based Syst.* 19.5 (2006), pp. 309–315.

- [46] J. S. Breese, D. Heckerman, and C. M. Kadie. “Empirical Analysis of Predictive Algorithms for Collaborative Filtering”. In: *UAI*. 1998, pp. 43–52.
- [47] B. C. A. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini. “Large Language Monkeys: Scaling Inference Compute with Repeated Sampling”. In: *CoRR* abs/2407.21787 (2024).
- [48] P. F. Brown, J. Cocke, S. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. “A Statistical Approach to Machine Translation”. In: *Comput. Linguistics* 16.2 (1990), pp. 79–85.
- [49] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Nee-lakantan, and et al. “Language Models are Few-Shot Learners”. In: *NeurIPS*. 2020.
- [50] S. Bubeck et al. “Sparks of Artificial General Intelligence: Early experiments with GPT-4”. In: *CoRR* abs/2303.12712 (2023).
- [51] L. Bürger, F. A. Hamprecht, and B. Nadler. “Truth is Universal: Robust Detection of Lies in LLMs”. In: *NeurIPS*. 2024.
- [52] R. Burke. “*Recommender Systems: An Introduction*, by Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich Cambridge University Press, 2011, 336 pages. ISBN: 978-0-521-49336-9”. In: *Int. J. Hum. Comput. Interact.* 28.1 (2012), pp. 72–73.
- [53] C. Burns, H. Ye, D. Klein, and J. Steinhardt. “Discovering Latent Knowledge in Language Models Without Supervision”. In: *ICLR*. OpenReview.net, 2023.
- [54] N. Calderon, R. Reichart, and R. Dror. “The Alternative Annotator Test for LLM-as-a-Judge: How to Statistically Justify Replacing Human Annotators with LLMs”. In: *CoRR* abs/2501.10970 (2025).
- [55] J. Camacho-Collados, K. Rezaee, T. Riahi, A. Ushio, and D. L. et al. “TweetNLP: Cutting-Edge Natural Language Processing for Social Media”. In: *EMNLP (Demos)*. Association for Computational Linguistics, 2022, pp. 38–49.
- [56] I. Cantador, P. Brusilovsky, and T. Kuflik. “Second workshop on information heterogeneity and fusion in recommender systems (HetRec2011)”. In: *RecSys*. New York, NY, USA: ACM, 2011, pp. 387–388.

- [57] Y. Cao, S. Li, Y. Liu, Z. Yan, Y. Dai, P. S. Yu, and L. Sun. “A Comprehensive Survey of AI-Generated Content (AIGC): A History of Generative AI from GAN to ChatGPT”. In: *CoRR* abs/2303.04226 (2023).
- [58] N. Carlini, D. Ippolito, M. Jagielski, K. Lee, F. Tramèr, and C. Zhang. “Quantifying Memorization Across Neural Language Models”. In: *ICLR*. OpenReview.net, 2023.
- [59] N. Carlini et al. “Extracting Training Data from Large Language Models”. In: *USENIX Security Symposium*. USENIX Association, 2021, pp. 2633–2650.
- [60] T. Carraro. “Overcoming Recommendation Limitations with Neuro-Symbolic Integration”. In: *RecSys*. ACM, 2023, pp. 1325–1331.
- [61] P. Castells, N. Hurley, and S. Vargas. “Novelty and Diversity in Recommender Systems”. In: *Recommender Systems Handbook*. Springer US, 2022, pp. 603–646.
- [62] Ò. Celma and P. Herrera. “A new approach to evaluating novel recommendations”. In: *RecSys*. ACM, 2008, pp. 179–186.
- [63] P. Chaovalit and L. Zhou. “Movie Review Mining: a Comparison between Supervised and Unsupervised Classification Approaches”. In: *HICSS*. IEEE Computer Society, 2005.
- [64] B. Chen, W. Peng, M. Wu, B. Zheng, and S. Zhu. “Neural-Symbolic Recommendation with Graph-Enhanced Information”. In: *ICONIP (3)*. Vol. 14449. Lecture Notes in Computer Science. Springer, 2023, pp. 411–423.
- [65] B. Chen, F. Zhang, A. Nguyen, D. Zan, Z. Lin, J. Lou, and W. Chen. “CodeT: Code Generation with Generated Tests”. In: *ICLR*. OpenReview.net, 2023.
- [66] C. Chen, K. Liu, Z. Chen, Y. Gu, Y. Wu, M. Tao, Z. Fu, and J. Ye. “INSIDE: LLMs’ Internal States Retain the Power of Hallucination Detection”. In: *arXiv preprint arXiv:2402.03744* (2024).
- [67] J. Chen, X. Yang, C. Yang, J. Bao, Z. Guo, Y. Li, and C. Shi. “CORONA: A Coarse-to-Fine Framework for Graph-based Recommendation with Large Language Models”. In: *SIGIR*. ACM, 2025, pp. 2048–2058.
- [68] M. Chen et al. “Evaluating Large Language Models Trained on Code”. In: *CoRR* abs/2107.03374 (2021).

- [69] N. Chen, N. Wu, S. Liang, M. Gong, L. Shou, D. Zhang, and J. Li. “Is Bigger and Deeper Always Better? Probing LLaMA Across Scales and Layers”. In: *CoRR* abs/2312.04333 (2023).
- [70] Y. Chen, Q. Fu, Y. Yuan, Z. Wen, G. Fan, D. Liu, D. Zhang, Z. Li, and Y. Xiao. “Hallucination Detection: Robustly Discerning Reliable Answers in Large Language Models”. In: *CIKM*. 2023, pp. 245–255.
- [71] K. Cheng, N. K. Ahmed, R. A. Rossi, T. L. Willke, and Y. Sun. “Neural-Symbolic Methods for Knowledge Graph Reasoning: A Survey”. In: *ACM Trans. Knowl. Discov. Data* 18.9 (2024), 225:1–225:44.
- [72] D. M. Chickering. “Optimal Structure Identification With Greedy Search”. In: *J. Mach. Learn. Res.* 3 (2002), pp. 507–554.
- [73] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches”. In: *SSST@EMNLP*. 2014, pp. 103–111.
- [74] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *EMNLP. ACL*, 2014, pp. 1724–1734.
- [75] N. Chomsky. “Three models for the description of language”. In: *IRE Trans. Inf. Theory* 2.3 (1956), pp. 113–124.
- [76] F. Christoffel, B. Paudel, C. Newell, and A. Bernstein. “Blockbusters and wallflowers: Accurate, diverse, and scalable recommendations with random walks”. In: *Proceedings of the 9th ACM Conference on Recommender Systems*. 2015, pp. 163–170.
- [77] N. C. Chung, B. Miasojedow, M. Startek, and A. Gambin. “Jaccard/Tanimoto similarity test and estimation methods for biological presence-absence data”. In: *BMC Bioinform.* 20-S.15 (2019), p. 644.
- [78] A. Ciapponi, J. M. Belizán, G. Piaggio, and S. Yaya. *There is life beyond the statistical significance*. 2021.
- [79] K. Cobbe et al. “Training Verifiers to Solve Math Word Problems”. In: *CoRR* abs/2110.14168 (2021).

- [80] A. Conneau, G. Kruszewski, G. Lample, L. Barrault, and M. Baroni. “What you can cram into a single vector: Probing sentence embeddings for linguistic properties”. In: *arXiv preprint arXiv:1805.01070* (2018).
- [81] C. Cooper, S. Lee, T. Radzik, and Y. Siantos. “Random walks in recommender systems: exact computation and simulations”. In: *WWW (Companion Volume)*. 2014, pp. 811–816.
- [82] M. Cortina-Borja. *Handbook of parametric and nonparametric statistical procedures, 5th EDN*. 2012.
- [83] P. Cremonesi and D. Jannach. “Progress in Recommender Systems Research: Crisis? What Crisis?” In: *AI Mag.* 42.3 (2021), pp. 43–54.
- [84] P. Cremonesi, Y. Koren, and R. Turrin. “Performance of recommender algorithms on top-n recommendation tasks”. In: *RecSys*. New York, NY, USA: ACM, 2010, pp. 39–46.
- [85] P. Cremonesi, R. Turrin, E. Lentini, and M. Matteucci. “An Evaluation Methodology for Collaborative Recommender Systems”. In: *International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution*. 2008, pp. 224–231. DOI: 10.1109/AXMEDIS.2008.13.
- [86] Z. Cui, J. Ma, C. Zhou, J. Zhou, and H. Yang. “M6-rec: Generative pretrained language models are open-ended recommender systems”. In: *arXiv preprint arXiv:2205.08084* (2022).
- [87] M. F. Dacrema, S. Boglio, P. Cremonesi, and D. Jannach. “A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research”. In: *ACM Trans. Inf. Syst.* 39.2 (2021), 20:1–20:49. DOI: 10.1145/3434185.
- [88] M. F. Dacrema, P. Cremonesi, and D. Jannach. “Are we really making much progress? A worrying analysis of recent neural recommendation approaches”. In: *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019*. Ed. by T. Bogers, A. Said, P. Brusilovsky, and D. Tikk. ACM, 2019, pp. 101–109. DOI: 10.1145/3298689.3347058.
- [89] M. Dahl, V. Magesh, M. Suzgun, and D. E. Ho. “Large Legal Fictions: Profiling Legal Hallucinations in Large Language Models”. In: *CoRR abs/2401.01301* (2024).

- [90] S. Dai, N. Shao, H. Zhao, W. Yu, Z. Si, C. Xu, Z. Sun, X. Zhang, and J. Xu. “Uncovering ChatGPT’s Capabilities in Recommender Systems”. In: *RecSys*. 2023, pp. 1126–1132.
- [91] Y. Dai, H. Yang, H. Mou, and P. Chao. “PARSQL: Enhancing Text-to-SQL through SQL Parsing and Reasoning”. In: *ACL (Findings)*. Association for Computational Linguistics, 2025, pp. 661–681.
- [92] **Dario Di Palma**. “Retrieval-augmented Recommender System: Enhancing Recommender Systems with Large Language Models”. In: *RecSys*. ACM, 2023, pp. 1369–1373.
- [93] **Dario Di Palma**, V. W. Anelli, D. Malitesta, V. Paparella, C. Pomo, Y. Deldjoo, and T. D. Noia. “Examining Fairness in Graph-Based Collaborative Filtering: A Consumer and Producer Perspective”. In: *IIR*. Vol. 3448. CEUR Workshop Proceedings. CEUR-WS.org, 2023, pp. 79–84.
- [94] **Dario Di Palma**, A. D. Bellis, G. Servedio, V. W. Anelli, F. Narducci, and T. D. Noia. “LLaMAs Have Feelings Too: Unveiling Sentiment and Emotion Representations in LLaMA Models Through Probing”. In: *ACL (1)*. Association for Computational Linguistics, 2025, pp. 6124–6142.
- [95] **Dario Di Palma**, G. M. Biancofiore, V. W. Anelli, F. Narducci, and T. D. Noia. “Content-Based or Collaborative? Insights from Inter-List Similarity Analysis of ChatGPT Recommendations”. In: *UMAP (Adjunct Publication)*. ***Candidate of the UMAP 2025 Best LBR Paper Award***. ACM, 2025, pp. 28–33.
- [96] **Dario Di Palma**, F. A. Merra, M. Sfilio, V. W. Anelli, F. Narducci, and T. D. Noia. “Do LLMs Memorize Recommendation Datasets? A Preliminary Study on MovieLens-1M”. In: *SIGIR*. ***Recipient of the SIGIR 2025 Best Short Paper Award***. ACM, 2025, pp. 2582–2586.
- [97] **Dario Di Palma**, G. Servedio, V. W. Anelli, G. M. Biancofiore, F. Narducci, L. Carnimeo, and T. D. Noia. “Beyond Words: Can ChatGPT Support State-of-the-Art Recommender Systems?” In: *IIR*. Vol. 3802. CEUR Workshop Proceedings. CEUR-WS.org, 2024, pp. 13–22.
- [98] O. M. Dekkers. *Why not to (over) emphasize statistical significance*. 2019.

- [99] X. Deng, V. Bashlovkina, F. Han, S. Baumgartner, and M. Bendersky. “LLMs to the Moon? Reddit Market Sentiment Analysis with Large Language Models”. In: *WWW (Companion Volume)*. ACM, 2023, pp. 1014–1019.
- [100] J. Devlin, M. Chang, K. Lee, and K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL-HLT (1)*. Association for Computational Linguistics, 2019, pp. 4171–4186.
- [101] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.
- [102] D. Di Palma, G. M. Biancofiore, V. W. Anelli, F. Narducci, T. D. Noia, and E. D. Sciascio. “Evaluating ChatGPT as a Recommender System: A Rigorous Approach”. In: *CoRR abs/2309.03613* (2023).
- [103] *difflib: Helpers for computing deltas*. Python Software Foundation. URL: https://docs.python.org/3/library/difflib.html#difflib.get_close_matches (visited on 03/26/2024).
- [104] C. Dixon. *What’s Next in Computing?* Accessed: 2025-12-03. Feb. 21, 2016. URL: <https://cdixon.org/2016/02/21/what-s-next-in-computing/>.
- [105] R. Dror, G. Baumer, S. Shlomov, and R. Reichart. “The Hitchhiker’s Guide to Testing Statistical Significance in Natural Language Processing”. In: *ACL (1)*. Association for Computational Linguistics, 2018, pp. 1383–1392.
- [106] Z. Du, A. Zeng, Y. Dong, and J. Tang. “Understanding Emergent Abilities of Language Models from the Loss Perspective”. In: *Advances in Neural Information Processing Systems*. 2024.
- [107] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, and et al. “The Llama 3 Herd of Models”. In: *CoRR abs/2407.21783* (2024).

- [108] Y. K. Dwivedi, N. Kshetri, L. Hughes, E. L. Slade, A. Jeyaraj, A. K. Kar, and et al. "Opinion Paper: "So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy". In: *Int. J. Inf. Manag.* 71 (2023), p. 102642.
- [109] Z. Elyoseph, D. Hadar-Shoval, K. Asraf, and M. Lvovsky. "ChatGPT outperforms humans in emotional awareness evaluations". In: *Frontiers in psychology* 14 (2023), p. 1199058.
- [110] A. Fan, M. Lewis, and Y. N. Dauphin. "Hierarchical Neural Story Generation". In: *ACL (1)*. Association for Computational Linguistics, 2018, pp. 889–898.
- [111] G. Fatouros, J. Soldatos, K. Kouroumali, G. Makridis, and D. Kyriazis. "Transforming sentiment analysis in the financial domain with ChatGPT". In: *Machine Learning with Applications* 14 (2023), p. 100508.
- [112] J. Feng, C. Tao, X. Geng, T. Shen, C. Xu, G. Long, D. Zhao, and D. Jiang. "Synergistic Interplay between Search and Large Language Models for Information Retrieval". In: *ACL (1)*. Association for Computational Linguistics, 2024, pp. 9571–9583.
- [113] S. Feng, W. Shi, Y. Wang, W. Ding, V. Balachandran, and Y. Tsvetkov. "Don't Hallucinate, Abstain: Identifying LLM Knowledge Gaps via Multi-LLM Collaboration". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by L.-W. Ku, A. Martins, and V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 14664–14690. DOI: 10.18653/v1/2024.acl-long.786.
- [114] A. Ferrara, V. W. Anelli, A. C. M. Mancino, T. D. Noia, and E. D. Sciascio. "KGFlex: Efficient Recommendation with Sparse Feature Factorization and Knowledge Graphs". In: *ACM Trans. Recomm. Syst.* (2023).
- [115] M. Feurer and F. Hutter. "Hyperparameter Optimization". In: *Automated Machine Learning*. The Springer Series on Challenges in Machine Learning. Springer, 2019, pp. 3–33.
- [116] W. Fu, Z. Peng, S. Wang, Y. Xu, and J. Li. "Deeply Fusing Reviews and Contents for Cold Start Users in Cross-Domain Recommendation Systems". In: *AAAI*. 2019, pp. 94–101.

- [117] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. “Fast rule mining in ontological knowledge bases with AMIE+”. In: *VLDB J.* 24.6 (2015), pp. 707–730.
- [118] L. A. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. “AMIE: association rule mining under incomplete evidence in ontological knowledge bases”. In: *WWW. International World Wide Web Conferences Steering Committee / ACM*, 2013, pp. 413–422.
- [119] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. “MyMediaLite: a free recommender system library”. In: *RecSys. New York, NY, USA: ACM*, 2011, pp. 305–308.
- [120] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou. “Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation”. In: *Proc. VLDB Endow.* 17.5 (2024), pp. 1132–1145.
- [121] J. Gao, C. Xiong, and P. Bennett. “Recent Advances in Conversational Information Retrieval”. In: *SIGIR*. 2020, pp. 2421–2424.
- [122] J. Gao, B. Chen, X. Zhao, W. Liu, X. Li, Y. Wang, W. Wang, H. Guo, and R. Tang. “LLM4Rerank: LLM-based Auto-Reranking Framework for Recommendations”. In: *WWW. ACM*, 2025, pp. 228–239.
- [123] L. Gao et al. “The Pile: An 800GB Dataset of Diverse Text for Language Modeling”. In: *CoRR abs/2101.00027* (2021).
- [124] Y. Gao, T. Sheng, Y. Xiang, Y. Xiong, H. Wang, and J. Zhang. “Chat-REC: Towards Interactive and Explainable LLMs-Augmented Recommender System”. In: *CoRR abs/2303.14524* (2023).
- [125] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. “Convolutional Sequence to Sequence Learning”. In: *ICML. Vol. 70. Proceedings of Machine Learning Research. PMLR*, 2017, pp. 1243–1252.
- [126] Z. Gekhman, E. Ben-David, H. Orgad, E. Ofek, Y. Belinkov, I. Szpektor, J. Herzig, and R. Reichart. “Inside-Out: Hidden Factual Knowledge in LLMs”. In: *CoRR abs/2503.15299* (2025).
- [127] S. Geng, S. Liu, Z. Fu, Y. Ge, and Y. Zhang. “Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5)”. In: *RecSys. New York, NY, USA: ACM*, 2022, pp. 299–315.

- [128] Y. Ghafoor, S. Jinping, F. H. Calderon, Y. Huang, K. Chen, and Y. Chen. “TERMS: textual emotion recognition in multidimensional space”. In: *Appl. Intell.* 53.3 (2023), pp. 2673–2693.
- [129] V. E. Giuliano, P. E. J. Jr., G. E. Kimball, R. F. Meyer, and B. A. Stein. “Automatic Pattern Recognition by a Gestalt Method”. In: *Inf. Control.* 4.4 (1961), pp. 332–345.
- [130] M. R. Glass, M. Eyceoz, D. Subramanian, G. Rossiello, L. Vu, and A. Gliozzo. “Extractive Schema Linking for Text-to-SQL”. In: *CoRR* abs/2501.17174 (2025).
- [131] X. Glorot and Y. Bengio. “Understanding the difficulty of training deep feed-forward neural networks”. In: *AISTATS*. Vol. 9. JMLR Proceedings. JMLR.org, 2010, pp. 249–256.
- [132] J. Gu et al. “A Survey on LLM-as-a-Judge”. In: *CoRR* abs/2411.15594 (2024).
- [133] A. Gunawardana, G. Shani, and S. Yogev. “Evaluating Recommender Systems”. In: *Recommender Systems Handbook*. US: Springer, 2022, pp. 547–601.
- [134] D. Guo, Y. Sun, D. Tang, N. Duan, J. Yin, H. Chi, J. Cao, P. Chen, and M. Zhou. “Question Generation from SQL Queries Improves Neural Semantic Parsing”. In: *EMNLP*. Association for Computational Linguistics, 2018, pp. 1597–1607.
- [135] Q. Guo, F. Zhuang, C. Qin, H. Zhu, X. Xie, H. Xiong, and Q. He. “A Survey on Knowledge Graph-Based Recommender Systems”. In: *IEEE Trans. Knowl. Data Eng.* 34.8 (2022), pp. 3549–3568.
- [136] S. Guo, Q. Wang, L. Wang, B. Wang, and L. Guo. “Jointly Embedding Knowledge Graphs and Logical Rules”. In: *EMNLP*. The Association for Computational Linguistics, 2016, pp. 192–202.
- [137] W. Gurnee and M. Tegmark. “Language Models Represent Space and Time”. In: *ICLR*. OpenReview.net, 2024.
- [138] S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Z. Wang, and Z. Hu. “Reasoning with Language Model is Planning with World Model”. In: *EMNLP*. Association for Computational Linguistics, 2023, pp. 8154–8173.
- [139] F. M. Harper and J. A. Konstan. “The MovieLens Datasets: History and Context”. In: *ACM Trans. Interact. Intell. Syst.* 5.4 (2016), 19:1–19:19. DOI: 10.1145/2827872.

- [140] M. A. Hasan, S. Das, A. Anjum, F. Alam, A. Anjum, A. Sarker, and S. R. H. Noori. “Zero- and Few-Shot Prompting with LLMs: A Comparative Study with Fine-tuned Models for Bangla Sentiment Analysis”. In: *LREC/COLING. ELRA and ICCL*, 2024, pp. 17808–17818.
- [141] V. Hatzivassiloglou and K. R. McKeown. “Predicting the Semantic Orientation of Adjectives”. In: *ACL. Morgan Kaufmann Publishers / ACL*, 1997, pp. 174–181.
- [142] A. Havrilla, Y. Du, S. C. Raparthy, C. Nalmpantis, J. Dwivedi-Yu, M. Zhuravinskyi, E. Hambro, S. Sukhbaatar, and R. Raileanu. “Teaching Large Language Models to Reason with Reinforcement Learning”. In: *CoRR abs/2403.04642* (2024).
- [143] K. He, X. Zhang, S. Ren, and J. Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *ICCV. IEEE Computer Society*, 2015, pp. 1026–1034.
- [144] M. He, Y. Shen, W. Zhang, Q. Peng, J. Wang, and W. Lu. “STaR-SQL: Self-Taught Reasoner for Text-to-SQL”. In: *ACL (1). Association for Computational Linguistics*, 2025, pp. 24365–24375.
- [145] P. He, J. Gao, and W. Chen. “DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing”. In: *ICLR. OpenReview.net*, 2023.
- [146] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. “LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation”. In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*. Ed. by J. X. Huang, Y. Chang, X. Cheng, J. Kamps, V. Murdock, J. Wen, and Y. Liu. ACM, 2020, pp. 639–648. DOI: 10.1145/3397271.3401063.
- [147] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua. “Neural Collaborative Filtering”. In: *WWW. ACM*, 2017, pp. 173–182.
- [148] Z. He, Z. Xie, R. Jha, H. Steck, D. Liang, Y. Feng, B. P. Majumder, N. Kallus, and J. McAuley. “Large language models as zero-shot conversational recommenders”. In: *Proceedings of the 32nd ACM international conference on information and knowledge management*. 2023, pp. 720–730.

- [149] J. L. Herlocker, J. A. Konstan, and J. Riedl. “An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms”. In: *Inf. Retr.* 5.4 (2002), pp. 287–310.
- [150] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl. “Evaluating collaborative filtering recommender systems”. In: *ACM Trans. Inf. Syst.* 22.1 (2004), pp. 5–53.
- [151] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. “Session-based Recommendations with Recurrent Neural Networks”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2016.
- [152] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. “The Curious Case of Neural Text Degeneration”. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020*. OpenReview.net, 2020.
- [153] Z. Hong, Z. Yuan, H. Chen, Q. Zhang, F. Huang, and X. Huang. “Knowledge-to-SQL: Enhancing SQL Generation with Data Expert LLM”. In: *ACL (Findings)*. Association for Computational Linguistics, 2024, pp. 10997–11008.
- [154] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang. “Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL”. In: *CoRR* abs/2406.08426 (2024).
- [155] Y. Hou, J. Zhang, Z. Lin, H. Lu, R. Xie, J. J. McAuley, and W. X. Zhao. “Large Language Models are Zero-Shot Rankers for Recommender Systems”. In: *ECIR (2)*. Vol. 14609. Lecture Notes in Computer Science. US: Springer, 2024, pp. 364–381.
- [156] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. “LoRA: Low-Rank Adaptation of Large Language Models”. In: *ICLR*. OpenReview.net, 2022.
- [157] H. Huang, Y. Qu, J. Liu, M. Yang, B. Xu, T. Zhao, and W. Lu. “Self-Evaluation of Large Language Model based on Glass-box Features”. In: *EMNLP (Findings)*. Association for Computational Linguistics, 2024, pp. 5813–5820.

- [158] L. Huang et al. “A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions”. en. In: *ACM Trans. Inf. Syst.* (Nov. 2024).
- [159] L. Huang et al. “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions”. In: *ACM Trans. Inf. Syst.* 43.2 (2025), 42:1–42:55.
- [160] S. Huang, Z. Ma, J. Du, C. Meng, W. Wang, and Z. Lin. “Mirror-Consistency: Harnessing Inconsistency in Majority Voting”. In: *EMNLP (Findings)*. Association for Computational Linguistics, 2024, pp. 2408–2420.
- [161] Y. Huang, S. Hu, X. Han, Z. Liu, and M. Sun. “Unified View of Grokking, Double Descent and Emergent Abilities: A Perspective from Circuits Competition”. In: *CoRR abs/2402.15175* (2024).
- [162] B. Hui, H. Yuan, N. Gong, P. Burlina, and Y. Cao. “PLeak: Prompt Leaking Attacks against Large Language Model Applications”. In: *CCS. ACM*, 2024, pp. 3600–3614.
- [163] D. Jannach, L. Lerche, I. Kamehkhosh, and M. Jugovac. “What recommenders recommend: an analysis of recommendation biases and possible countermeasures”. In: *User Model. User Adapt. Interact.* 25.5 (2015), pp. 427–491. DOI: 10.1007/s11257-015-9165-3.
- [164] D. Jannach and M. Zanker. “Value and Impact of Recommender Systems”. In: *Recommender Systems Handbook*. Springer US, 2022, pp. 519–546.
- [165] K. Järvelin and J. Kekäläinen. “Cumulated gain-based evaluation of IR techniques”. In: *ACM Trans. Inf. Syst.* 20.4 (2002), pp. 422–446.
- [166] U. Javed, K. Shaukat, I. A. Hameed, F. Iqbal, T. M. Alam, and S. Luo. “A review of content-based and context-based recommendation systems”. In: *International Journal of Emerging Technologies in Learning (iJET)* 16.3 (2021), pp. 274–306.
- [167] Y. Ji, A. Sun, J. Zhang, and C. Li. “Do Loyal Users Enjoy Better Recommendations?: Understanding Recommender Accuracy from a Time Perspective”. In: *ICTIR. ACM*, 2022, pp. 92–97.

- [168] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, and P. Fung. “Survey of hallucination in natural Language Generation”. en. In: *ACM Comput. Surv.* (Nov. 2022).
- [169] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, and P. Fung. “Survey of Hallucination in Natural Language Generation”. In: *ACM Comput. Surv.* 55.12 (2023), 248:1–248:38.
- [170] Z. Ji, T. Yu, Y. Xu, N. Lee, E. Ishii, and P. Fung. “Towards Mitigating LLM Hallucination via Self Reflection”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by H. Bouamor, J. Pino, and K. Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1827–1843. DOI: 10.18653/v1/2023.findings-emnlp.123.
- [171] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al. “Mistral 7B”. In: *arXiv preprint arXiv:2310.06825* (2023).
- [172] J. Jin, X. Chen, F. Ye, M. Yang, Y. Feng, W. Zhang, Y. Yu, and J. Wang. “Lending Interaction Wings to Recommender Systems with Conversational Agents”. In: *NeurIPS*. 2023.
- [173] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer. “TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension”. In: *ACL (1)*. Association for Computational Linguistics, 2017, pp. 1601–1611.
- [174] S. Kadavath et al. “Language Models (Mostly) Know What They Know”. In: *CoRR abs/2207.05221* (2022).
- [175] N. Kalchbrenner and P. Blunsom. “Recurrent Continuous Translation Models”. In: *EMNLP. ACL*, 2013, pp. 1700–1709.
- [176] E. Kamalloo, N. Dziri, C. L. A. Clarke, and D. Rafiei. “Evaluating Open-Domain Question Answering in the Era of Large Language Models”. In: *ACL (1)*. Association for Computational Linguistics, 2023, pp. 5591–5606.
- [177] M. Kaminskis and D. Bridge. “Diversity, Serendipity, Novelty, and Coverage: A Survey and Empirical Analysis of Beyond-Accuracy Objectives in Recommender Systems”. In: *ACM Trans. Interact. Intell. Syst.* 7.1 (2017), 2:1–2:42.

- [178] A. B. Kanburoglu and F. B. Tek. “Text-to-SQL: A methodical review of challenges and models”. In: *Turkish J. Electr. Eng. Comput. Sci.* 32.3 (2024), pp. 403–419.
- [179] W.-C. Kang, J. Ni, N. Mehta, M. Sathiamoorthy, L. Hong, E. Chi, and D. Z. Cheng. “Do llms understand user preferences? evaluating llms on user rating prediction”. In: *arXiv preprint arXiv:2305.06474* (2023).
- [180] M. Khalifa, R. Agarwal, L. Logeswaran, J. Kim, H. Peng, M. Lee, H. Lee, and L. Wang. “Process Reward Models That Think”. In: *CoRR abs/2504.16828* (2025).
- [181] K. Khan, B. Baharudin, A. Khan, and A. Ullah. “Mining opinion components from unstructured reviews: A review”. In: *J. King Saud Univ. Comput. Inf. Sci.* 26.3 (2014), pp. 258–275.
- [182] D. Kieu, D. C. Zhang, M. D. Nguyen, M. Xu, Q. Wu, and D. D. Le. “Enhancing News Recommendation with Hierarchical LLM Prompting”. In: *WWW (Companion Volume)*. ACM, 2025, pp. 3072–3075.
- [183] H. Kim, B. So, W. Han, and H. Lee. “Natural language to SQL: Where are we today?” In: *Proc. VLDB Endow.* 13.10 (2020), pp. 1737–1750.
- [184] S. Kim, S. Yun, H. Lee, M. Gubri, S. Yoon, and S. J. Oh. “ProPILE: Probing Privacy Leakage in Large Language Models”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., 2023, pp. 20750–20762.
- [185] Y. Kim. “Convolutional Neural Networks for Sentence Classification”. In: *EMNLP. ACL*, 2014, pp. 1746–1751.
- [186] F. Klubicka and J. D. Kelleher. “Probing with Noise: Unpicking the Warp and Weft of Embeddings”. In: *BlackboxNLP@EMNLP*. Association for Computational Linguistics, 2022, pp. 404–417.
- [187] H. Kobayashi, W. Lan, P. Shi, S. Chang, J. Guo, H. Zhu, Z. Wang, and P. Ng. “You Only Read Once (YORO): Learning to Internalize Database Knowledge for Text-to-SQL”. In: *NAACL (Long Papers)*. Association for Computational Linguistics, 2025, pp. 1889–1901.
- [188] A. Kong, S. Zhao, H. Chen, Q. Li, Y. Qin, R. Sun, and X. Zhou. “Better Zero-Shot Reasoning with Role-Play Prompting”. In: *CoRR abs/2308.07702* (2023).

- [189] Y. Koren. “Factorization meets the neighborhood: a multifaceted collaborative filtering model”. In: *KDD*. ACM, 2008, pp. 426–434.
- [190] Y. Koren, R. M. Bell, and C. Volinsky. “Matrix Factorization Techniques for Recommender Systems”. In: *Computer* 42.8 (2009), pp. 30–37.
- [191] Y. Koren, S. Rendle, and R. Bell. “Advances in collaborative filtering”. In: *Recommender systems handbook* (2021), pp. 91–142.
- [192] D. Kowald, P. Muellner, E. Zangerle, C. Bauer, M. Schedl, and E. Lex. “Support the underground: characteristics of beyond-mainstream music listeners”. In: *EPJ Data Science* 10.1 (2021), pp. 1–26. DOI: 10.1140/epjds/s13688-021-00268-9.
- [193] J. O. Krugmann and J. Hartmann. “Sentiment Analysis in the Age of Generative AI”. In: *Customer Needs and Solutions* 11.1 (2024), p. 3.
- [194] P. Kumar. “Large language models (LLMs): survey, technical frameworks, and future challenges”. In: *Artif. Intell. Rev.* 57.9 (2024), p. 260.
- [195] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong. “Addressing cold-start problem in recommendation systems”. In: *ICUIMC*. ACM, 2008, pp. 208–211.
- [196] W. Laurito, S. Maiya, G. Dhimoila, O. Yeung, and K. Hänni. “Cluster-Norm for Unsupervised Probing of Knowledge”. In: *EMNLP*. Association for Computational Linguistics, 2024, pp. 14083–14112.
- [197] B. A. Levinstein and D. A. Herrmann. “Still no lie detector for language models: Probing empirical and conceptual roadblocks”. In: *Philosophical Studies* (2024), pp. 1–27.
- [198] A. Lewkowycz et al. “Solving Quantitative Reasoning Problems with Language Models”. In: *NeurIPS*. 2022.
- [199] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang. “The Dawn of Natural Language to SQL: Are We Fully Ready? [Experiment, Analysis & Benchmark]”. In: *Proc. VLDB Endow.* 17.11 (2024), pp. 3318–3331.
- [200] F. Li and H. V. Jagadish. “Constructing an Interactive Natural Language Interface for Relational Databases”. In: *Proc. VLDB Endow.* 8.1 (2014), pp. 73–84.

- [201] F. Li and H. V. Jagadish. “NaLIR: an interactive natural language interface for querying relational databases”. In: *SIGMOD Conference*. ACM, 2014, pp. 709–712.
- [202] H. Li, J. Zhang, C. Li, and H. Chen. “RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL”. In: *AAAI*. AAAI Press, 2023, pp. 13067–13075.
- [203] H. Li, J. Zhang, H. Liu, J. Fan, X. Zhang, J. Zhu, R. Wei, H. Pan, C. Li, and H. Chen. “CodeS: Towards Building Open-source Language Models for Text-to-SQL”. In: *Proc. ACM Manag. Data* 2.3 (2024), p. 127.
- [204] H. Li et al. “OmniSQL: Synthesizing High-quality Text-to-SQL Data at Scale”. In: *Proc. VLDB Endow.* 18.11 (2025), pp. 4695–4709.
- [205] J. Li, W. Zhang, T. Wang, G. Xiong, A. Lu, and G. Medioni. “GPT4Rec: A Generative Framework for Personalized Recommendation and User Interests Interpretation”. In: *eCom@SIGIR*. Vol. 3589. CEUR Workshop Proceedings. 2023.
- [206] J. Li, B. Hui, R. Cheng, B. Qin, C. Ma, N. Huo, F. Huang, W. Du, L. Si, and Y. Li. “Graphix-T5: Mixing Pre-trained Transformers with Graph-Aware Layers for Text-to-SQL Parsing”. In: *AAAI*. AAAI Press, 2023, pp. 13076–13084.
- [207] J. Li et al. “Can LLM Already Serve as A Database Interface? A Blg Bench for Large-Scale Database Grounded Text-to-SQLs”. In: *NeurIPS*. 2023.
- [208] K. Li, O. Patel, F. B. Viégas, H. Pfister, and M. Wattenberg. “Inference-Time Intervention: Eliciting Truthful Answers from a Language Model”. In: *NeurIPS*. 2023.
- [209] M. Li, W. Wang, F. Feng, F. Zhu, Q. Wang, and T. Chua. “Think Twice Before Trusting: Self-Detection for Large Language Models through Comprehensive Answer Reflection”. In: *EMNLP (Findings)*. Association for Computational Linguistics, 2024, pp. 11858–11875.
- [210] T. Li, Z. Wang, W. Liu, M. Wu, S. Dou, C. Lv, X. Wang, X. Zheng, and X. Huang. “Revisiting Jailbreaking for Large Language Models: A Representation Engineering Perspective”. In: *COLING*. Association for Computational Linguistics, 2025, pp. 3158–3178.

- [211] X. L. Li, A. Holtzman, D. Fried, P. Liang, J. Eisner, T. Hashimoto, L. Zettlemoyer, and M. Lewis. “Contrastive Decoding: Open-ended Text Generation as Optimization”. In: *ACL (1)*. Association for Computational Linguistics, 2023, pp. 12286–12312.
- [212] Y. Li, M. R. Lyu, and L. Wang. “Learning to Reason from Feedback at Test-Time”. In: *ACL (1)*. Association for Computational Linguistics, 2025, pp. 5241–5253.
- [213] Z. Li, Y. Chen, X. Zhang, and X. Liang. “BookGPT: A General Framework for Book Recommendation Empowered by Large Language Model”. In: *Electronics* 12.22 (2023), p. 4654.
- [214] J. Liao, S. Li, Z. Yang, J. Wu, Y. Yuan, X. Wang, and X. He. “LLaRA: Large Language-Recommendation Assistant”. In: *SIGIR*. ACM, 2024, pp. 1785–1795.
- [215] J. Lin, R. Men, A. Yang, C. Zhou, Y. Zhang, P. Wang, J. Zhou, J. Tang, and H. Yang. “M6: Multi-Modality-to-Multi-Modality Multitask Mega-transformer for Unified Pretraining”. In: *KDD*. New York, NY, USA: ACM, 2021, pp. 3251–3261.
- [216] S. Lin, H. Xie, L. Yu, and K. R. Lai. “SentiNLP at IJCNLP-2017 Task 4: Customer Feedback Analysis Using a Bi-LSTM-CNN Model”. In: *IJCNLP (Shared Tasks)*. Asian Federation of Natural Language Processing, 2017, pp. 149–154.
- [217] S. Lin, J. Hilton, and O. Evans. “TruthfulQA: Measuring How Models Mimic Human Falsehoods”. In: *ACL (1)*. Association for Computational Linguistics, 2022, pp. 3214–3252.
- [218] G. Linden, B. Smith, and J. York. “Amazon.com Recommendations: Item-to-Item Collaborative Filtering”. In: *IEEE Internet Comput.* 7.1 (2003), pp. 76–80.
- [219] A. Liu, X. Hu, L. Wen, and P. S. Yu. “A comprehensive evaluation of ChatGPT’s zero-shot Text-to-SQL capability”. In: *CoRR* abs/2303.13547 (2023).
- [220] F. Liu, P. Xu, Z. Li, Y. Feng, and H. Song. “Towards understanding in-context learning with contrastive demonstrations and saliency maps”. In: *arXiv preprint arXiv:2307.05052* (2023).
- [221] J. Liu, C. Liu, R. Lv, K. Zhou, and Y. Zhang. “Is ChatGPT a Good Recommender? A Preliminary Study”. In: *CoRR* abs/2304.10149 (2023).

- [222] S. Liu et al. “Rethinking Machine Unlearning for Large Language Models”. In: *CoRR* abs/2402.08787 (2024).
- [223] W. Liu, G. Li, K. Zhang, B. Du, Q. Chen, X. Hu, H. Xu, J. Chen, and J. Wu. “Mind’s Mirror: Distilling Self-Evaluation Capability and Comprehensive Thinking from Large Language Models”. In: *NAACL-HLT*. Association for Computational Linguistics, 2024, pp. 6748–6763.
- [224] X. Liu, S. Shen, B. Li, P. Ma, R. Jiang, Y. Zhang, J. Fan, G. Li, N. Tang, and Y. Luo. “A Survey of Text-to-SQL in the Era of LLMs: Where are We, and Where are We Going?” In: *IEEE Transactions on Knowledge and Data Engineering* (2025).
- [225] Y. Liu, Z. Zheng, F. Zhang, J. Feng, Y. Fu, J. Zhai, B. He, X. Zhang, and X. Du. “A Comprehensive Taxonomy of Prompt Engineering Techniques for Large Language Models”. In: *Frontiers of Computer Science* 20.3 (2026), p. 2003601. DOI: 10.1007/s11704-025-50058-z.
- [226] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *CoRR* abs/1907.11692 (2019).
- [227] Y. Liu, W. Zhang, Y. Chen, Y. Zhang, H. Bai, F. Feng, H. Cui, Y. Li, and W. Che. “Conversational Recommender System and Large Language Model Are Made for Each Other in E-commerce Pre-sales Dialogue”. In: *EMNLP (Findings)*. Association for Computational Linguistics, 2023, pp. 9587–9605.
- [228] Z. Liu, Y. Zhou, Y. Zhu, J. Lian, C. Li, Z. Dou, D. Lian, and J. Nie. “Information Retrieval Meets Large Language Models”. In: *WWW (Companion Volume)*. ACM, 2024, pp. 1586–1589.
- [229] P. Lops, A. Silletti, M. Polignano, C. Musto, and G. Semeraro. “Reproducibility of LLM-based Recommender Systems: the Case Study of P5 Paradigm”. In: *Proceedings of the 18th ACM Conference on Recommender Systems*. 2024, pp. 116–125.
- [230] T. Luong, H. Pham, and C. D. Manning. “Effective Approaches to Attention-based Neural Machine Translation”. In: *EMNLP*. The Association for Computational Linguistics, 2015, pp. 1412–1421.

- [231] J. Lynch, L. M. Ramjan, P. Glew, and Y. Salamonson. “Statistical significance’in research: wider strategies to meaningfully interpret findings”. In: *Nurse Researcher* 31.3 (2023).
- [232] H. Lyu, S. Jiang, H. Zeng, Y. Xia, Q. Wang, S. Zhang, R. Chen, C. Leung, J. Tang, and J. Luo. “LLM-Rec: Personalized Recommendation via Prompting Large Language Models”. In: *NAACL-HLT (Findings)*. Association for Computational Linguistics, 2024, pp. 583–612.
- [233] Z. Lyu, Z. Jin, F. G. Adatao, R. Mihalcea, B. Schölkopf, and M. Sachan. “Do LLMs Think Fast and Slow? A Causal Study on Sentiment Analysis”. In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. 2024, pp. 9353–9372.
- [234] L. Ma, K. Q. Pu, and Y. Zhu. “Evaluating LLMs for Text-to-SQL Generation With Complex SQL Workload”. In: *CoRR* abs/2407.19517 (2024).
- [235] W. Ma, M. Zhang, Y. Cao, W. Jin, C. Wang, Y. Liu, S. Ma, and X. Ren. “Jointly Learning Explainable Rules for Recommendation with Knowledge Graph”. In: *WWW*. ACM, 2019, pp. 1210–1221.
- [236] Y. Maarek. “Alexa and Her Shopping Journey”. In: *CIKM*. New York, NY, USA: ACM, 2018, p. 1.
- [237] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. “Learning Word Vectors for Sentiment Analysis”. In: *ACL*. The Association for Computer Linguistics, 2011, pp. 142–150.
- [238] T. Mahmud, K. A. Hasan, M. Ahmed, and T. H. C. Chak. “A rule based approach for NLP based query processing”. In: *2015 2nd international conference on electrical information and communication technologies (EICT)*. IEEE. 2015, pp. 78–82.
- [239] A. Mallen and N. Belrose. “Eliciting Latent Knowledge from Quirky Language Models”. In: *CoRR* abs/2312.01037 (2023).
- [240] A. C. M. Mancino, S. Bufi, A. di Fazio, A. Ferrara, D. Malitesta, C. Pomo, and T. D. Noia. “DataRec: A Python Library for Standardized and Reproducible Data Management in Recommender Systems”. In: *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Infor-*

- mation Retrieval, SIGIR 2025, Padua, Italy July 13-18, 2025*. ACM, 2025. DOI: 10.1145/3726302.3730320.
- [241] A. C. M. Mancino, A. Ferrara, S. Bufl, D. Malitesta, T. D. Noia, and E. D. Sciascio. “KGTOR: Tailored Recommendations through Knowledge-aware GNN Models”. In: *RecSys*. 2023, pp. 576–587.
 - [242] F. Manigrasso, S. F. Schouten, L. Morra, and P. Bloem. “Probing LLMs for Logical Reasoning”. In: *NeSy (1)*. Vol. 14979. Lecture Notes in Computer Science. Springer, 2024, pp. 257–278.
 - [243] M. Mansoury, B. Mobasher, R. Burke, and M. Pechenizkiy. “Bias Disparity in Collaborative Recommendation: Algorithmic Evaluation and Comparison”. In: *RMSE@RecSys*. Vol. 2440. CEUR Workshop Proceedings. CEUR-WS.org, 2019.
 - [244] K. Mao, J. Zhu, J. Wang, Q. Dai, Z. Dong, X. Xiao, and X. He. “SimpleX: A Simple and Strong Baseline for Collaborative Filtering”. In: *CIKM*. ACM, 2021, pp. 1243–1252.
 - [245] A. A. Markov. “An example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains”. In: *Science in Context* 19.4 (2006), pp. 591–600.
 - [246] S. Marks and M. Tegmark. “The Geometry of Truth: Emergent Linear Structure in Large Language Model Representations of True/False Datasets”. In: *CoRR* abs/2310.06824 (2023).
 - [247] A. K. McClave. *Statistics: The art and science of learning from data*. 2008.
 - [248] S. M. McNee, J. Riedl, and J. A. Konstan. “Being accurate is not enough: how accuracy metrics have hurt recommender systems”. In: *CHI Extended Abstracts*. ACM, 2006, pp. 1097–1101.
 - [249] C. Meilicke, M. W. Chekol, P. Betz, M. Fink, and H. Stuckenschmidt. “Anytime bottom-up rule learning for large-scale knowledge graph completion”. In: *VLDB J.* 33.1 (2024), pp. 131–161.
 - [250] C. Meister, T. Pimentel, G. Wiher, and R. Cotterell. “Locally Typical Sampling”. In: *Trans. Assoc. Comput. Linguistics* 11 (2023), pp. 102–121.
 - [251] C. Meister, G. Wiher, and R. Cotterell. “On Decoding Strategies for Neural Text Generators”. In: *Trans. Assoc. Comput. Linguistics* 10 (2022), pp. 997–1012.

- [252] T. Mikolov, K. Chen, G. Corrado, and J. Dean. “Efficient Estimation of Word Representations in Vector Space”. In: *ICLR (Workshop Poster)*. 2013.
- [253] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. “Distributed Representations of Words and Phrases and their Compositionality”. In: *NIPS*. 2013, pp. 3111–3119.
- [254] T. Mikolov, W. Yih, and G. Zweig. “Linguistic Regularities in Continuous Space Word Representations”. In: *HLT-NAACL*. The Association for Computational Linguistics, 2013, pp. 746–751.
- [255] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. “Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?” In: *EMNLP*. Association for Computational Linguistics, 2022, pp. 11048–11064.
- [256] B. Mitra and N. Craswell. “An Introduction to Neural Information Retrieval”. In: *Found. Trends Inf. Retr.* 13.1 (2018), pp. 1–126.
- [257] R. Morante and E. Blanco. “*SEM 2012 Shared Task: Resolving the Scope and Focus of Negation”. In: **SEM@NAACL-HLT*. Association for Computational Linguistics, 2012, pp. 265–274.
- [258] R. Nallapati, B. Zhou, C. N. dos Santos, Ç. Gülçehre, and B. Xiang. “Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond”. In: *CoNLL*. ACL, 2016, pp. 280–290.
- [259] E. Nascimento et al. “Text-to-SQL Meets the Real-World”. In: *ICEIS (1)*. SCITEPRESS, 2024, pp. 61–72.
- [260] M. Nasr, R. Shokri, and A. Houmansadr. “Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning”. In: *IEEE Symposium on Security and Privacy*. IEEE, 2019, pp. 739–753.
- [261] A. Y. Ng and M. I. Jordan. “On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes”. In: *NIPS*. MIT Press, 2001, pp. 841–848.
- [262] F. Nie, J. Yao, J. Wang, R. Pan, and C. Lin. “A Simple Recipe towards Reducing Hallucination in Neural Surface Realisation”. In: *ACL (1)*. 2019, pp. 2673–2679.

- [263] T. D. Noia, P. Lops, T. Joachims, K. Verbert, P. Castells, Z. Dong, and B. London, eds. *Proceedings of the 18th ACM Conference on Recommender Systems, RecSys 2024, Bari, Italy, October 14-18, 2024*. ACM, 2024.
- [264] T. D. Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker. “Linked open data to support content-based recommender systems”. In: *I-SEMANTICS*. New York, NY, USA: ACM, 2012, pp. 1–8.
- [265] nostalgebraist. *Interpreting GPT: The Logit Lens*. <https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru>. 2020.
- [266] D. L. Olson and D. Delen. *Advanced Data Mining Techniques*. US: Springer, 2008.
- [267] OpenAI. “GPT-4 Technical Report”. In: *ArXiv abs/2303.08774* (2023).
- [268] OpenAI. “GPT-4 Technical Report”. In: *CoRR abs/2303.08774* (2023).
- [269] H. Orgad, M. Toker, Z. Gekhman, R. Reichart, I. Szpektor, H. Kotek, and Y. Belinkov. “LLMs Know More Than They Show: On the Intrinsic Representation of LLM Hallucinations”. In: *CoRR abs/2410.02707* (2024).
- [270] R. Ott and M. Longnecker. *An introduction to statistical methods and data analysis*. 2016.
- [271] L. Ouyang et al. “Training language models to follow instructions with human feedback”. In: *NeurIPS*. 2022.
- [272] B. Pang and L. Lee. “Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales”. In: *Proceedings of the ACL*. 2005.
- [273] B. Pang, L. Lee, and S. Vaithyanathan. “Thumbs up? Sentiment Classification using Machine Learning Techniques”. In: *EMNLP*. 2002, pp. 79–86.
- [274] V. Paparella, V. W. Anelli, F. M. Nardini, R. Perego, and T. D. Noia. “Post-hoc Selection of Pareto-Optimal Solutions in Search and Recommendation”. In: *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*. Ed. by I. Frommholz, F. Hopfgartner, M. Lee, M. Oakes, M. Lalmas, M. Zhang, and R. L. T. Santos. ACM, 2023, pp. 2013–2023. DOI: 10.1145/3583780.3615010.

- [275] V. Paparella, **Dario Di Palma**, V. W. Anelli, and T. D. Noia. “Broadening the Scope: Evaluating the Potential of Recommender Systems beyond prioritizing Accuracy”. In: *RecSys*. ACM, 2023, pp. 1139–1145.
- [276] V. Paparella, A. C. M. Mancino, A. Ferrara, C. Pomo, V. W. Anelli, and T. D. Noia. “Knowledge Graph Datasets for Recommendation”. In: *KaRS@RecSys*. Vol. 3560. CEUR Workshop Proceedings. CEUR-WS.org, 2023, pp. 109–117.
- [277] D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. “The LAMBADA dataset: Word prediction requiring a broad discourse context”. In: *ACL (1)*. The Association for Computer Linguistics, 2016.
- [278] S. Park and W. Chu. “Pairwise preference regression for cold-start recommendation”. In: *RecSys*. 2009, pp. 21–28.
- [279] Y. Park and A. Tuzhilin. “The long tail of recommender systems and how to leverage it”. In: *RecSys*. 2008, pp. 11–18.
- [280] B. Paudel, F. Christoffel, C. Newell, and A. Bernstein. “Updatable, Accurate, Diverse, and Scalable Recommendations for Interactive Applications”. In: *ACM Trans. Interact. Intell. Syst.* 7.1 (2017), 1:1–1:34. DOI: 10.1145/2955101.
- [281] M. J. Pazzani and D. Billsus. “Content-Based Recommendation Systems”. In: *The Adaptive Web*. Vol. 4321. Lecture Notes in Computer Science. Springer, 2007, pp. 325–341.
- [282] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, and M. B. et al. “Scikit-learn: Machine Learning in Python”. In: *J. Mach. Learn. Res.* 12 (2011), pp. 2825–2830.
- [283] B. L. Pereira, A. Said, and R. L. T. Santos. “On the Reliability of Sampling Strategies in Offline Recommender Evaluation”. In: *Proceedings of the Nineteenth ACM Conference on Recommender Systems*. RecSys ’25. 2025, pp. 360–369. DOI: 10.1145/3705328.3748086.
- [284] C. Perez. “Technological revolutions and techno-economic paradigms”. In: *Cambridge journal of economics* 34.1 (2010), pp. 185–202.
- [285] F. Petroni, T. Rocktäschel, S. Riedel, P. S. H. Lewis, A. Bakhtin, Y. Wu, and A. H. Miller. “Language Models as Knowledge Bases?” In: *EMNLP/IJCNLP (1)*. Association for Computational Linguistics, 2019, pp. 2463–2473.

- [286] A. V. Petrov and C. MacDonald. “gSASRec: Reducing Overconfidence in Sequential Recommendation Trained with Negative Sampling”. In: *RecSys*. 2023, pp. 116–128.
- [287] D. K. Pham and B. Q. Vo. “Towards Reliable Medical Question Answering: Techniques and Challenges in Mitigating Hallucinations in Language Models”. In: *CoRR* abs/2408.13808 (2024).
- [288] P. Pirozelli, M. M. José, P. de Tarso P. Filho, A. A. F. Brandão, and F. G. Cozman. “Assessing Logical Reasoning Capabilities of Encoder-Only Transformer Models”. In: *NeSy (1)*. Vol. 14979. Lecture Notes in Computer Science. Springer, 2024, pp. 29–46.
- [289] M. Pourreza and D. Rafiei. “DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction”. In: *NeurIPS*. 2023.
- [290] C. Qin, A. Zhang, Z. Zhang, J. Chen, M. Yasunaga, and D. Yang. “Is ChatGPT a General-Purpose Natural Language Processing Task Solver?” In: *EMNLP*. Association for Computational Linguistics, 2023, pp. 1339–1384.
- [291] Y. Qin, C. Chen, Z. Fu, Z. Chen, D. Peng, P. Hu, and J. Ye. “ROUTE: Robust Multitask Tuning and Collaboration for Text-to-SQL”. In: *ICLR*. OpenReview.net, 2025.
- [292] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. “Improving language understanding by generative pre-training”. In: (2018).
- [293] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9.
- [294] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67.
- [295] P. Rajpurkar, R. Jia, and P. Liang. “Know What You Don’t Know: Unanswerable Questions for SQuAD”. In: *ACL (2)*. Association for Computational Linguistics, 2018, pp. 784–789.
- [296] A. M. Rashid, G. Karypis, and J. Riedl. “Learning preferences of new users in recommender systems: an information theoretic approach”. In: *SIGKDD Explor.* 10.2 (2008), pp. 90–100.

- [297] V. Rawte, S. Chakraborty, A. Pathak, A. Sarkar, S. M. T. I. Tonmoy, A. Chadha, A. P. Sheth, and A. Das. “The Troubling Emergence of Hallucination in Large Language Models - An Extensive Definition, Quantification, and Prescriptive Remediations”. In: *EMNLP*. Association for Computational Linguistics, 2023, pp. 2541–2573.
- [298] S. Reddy, D. Chen, and C. D. Manning. “CoQA: A Conversational Question Answering Challenge”. In: *Trans. Assoc. Comput. Linguistics* 7 (2019), pp. 249–266.
- [299] X. Ren, W. Wei, L. Xia, L. Su, S. Cheng, J. Wang, D. Yin, and C. Huang. “Representation Learning with Large Language Models for Recommendation”. In: *WWW*. ACM, 2024, pp. 3464–3475.
- [300] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. “BPR: Bayesian Personalized Ranking from Implicit Feedback”. In: *UAI*. AUAI Press, 2009, pp. 452–461.
- [301] S. Rendle, W. Krichene, L. Zhang, and J. R. Anderson. “Neural Collaborative Filtering vs. Matrix Factorization Revisited”. In: *RecSys*. 2020, pp. 240–248.
- [302] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. “GroupLens: An Open Architecture for Collaborative Filtering of Netnews”. In: *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA, October 22-26, 1994*. Ed. by J. B. Smith, F. D. Smith, and T. W. Malone. ACM, 1994, pp. 175–186. DOI: 10.1145/192844.192905.
- [303] F. Ricci, L. Rokach, and B. Shapira, eds. *Recommender Systems Handbook*. Springer US, 2022. ISBN: 978-1-0716-2196-7. DOI: 10.1007/978-1-0716-2197-4.
- [304] N. Rimskey. *Reducing Sycophancy and Improving Honesty via Activation Steering*. <https://www.lesswrong.com/posts/zt6hRsDE84HeBK7E>. 2023.
- [305] S. Robertson, H. Zaragoza, et al. “The probabilistic relevance framework: BM25 and beyond”. In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389.
- [306] G. Rossiello, N. Pham, M. R. Glass, J. Lee, and D. Subramanian. “Rationalization Models for Text-to-SQL”. In: *CoRR* abs/2502.06759 (2025).

- [307] S. Roy, A. Harshavardhan, A. Mukherjee, and P. Saha. “Probing LLMs for hate speech detection: strengths and vulnerabilities”. In: *EMNLP (Findings)*. Association for Computational Linguistics, 2023, pp. 6116–6128.
- [308] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.
- [309] M. Russinovich and A. Salem. “Jailbreaking is (Mostly) Simpler Than You Think”. In: *CoRR* abs/2503.05264 (2025).
- [310] D. von Rütte, S. Anagnostidis, G. Bachmann, and T. Hofmann. “A Language Model’s Guide Through Latent Space”. In: *ICML*. OpenReview.net, 2024.
- [311] N. Sachdeva, C.-J. Wu, and J. McAuley. “On Sampling Collaborative Filtering Datasets”. In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. WSDM ’22. 2022, pp. 842–850. DOI: 10.1145/3488560.3498439.
- [312] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha. “A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications”. In: *CoRR* abs/2402.07927 (2024).
- [313] V. Sanh et al. “Multitask Prompted Training Enables Zero-Shot Task Generalization”. In: *ICLR*. OpenReview.net, 2022.
- [314] S. Sanner, K. Balog, F. Radlinski, B. Wedin, and L. Dixon. “Large Language Models are Competitive Near Cold-start Recommenders for Language- and Item-based Preferences”. In: *RecSys*. 2023, pp. 890–896.
- [315] E. Saravia, H. T. Liu, Y. Huang, J. Wu, and Y. Chen. “CARER: Contextualized Affect Representations for Emotion Recognition”. In: *EMNLP*. Association for Computational Linguistics, 2018, pp. 3687–3697.
- [316] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. “Analysis of recommendation algorithms for e-commerce”. In: *EC*. New York, NY, USA: ACM, 2000, pp. 158–167.
- [317] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. “Item-based collaborative filtering recommendation algorithms”. In: *WWW*. ACM, 2001, pp. 285–295.
- [318] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. “Methods and metrics for cold-start recommendations”. In: *SIGIR*. 2002, pp. 253–260.

- [319] L. Schneider, M. Wistuba, A. Klein, J. Golebiowski, G. Zappella, and F. A. Merra. “Hyperband-based Bayesian Optimization for Black-box Prompt Selection”. In: *CoRR* abs/2412.07820 (2024).
- [320] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff, et al. “The Prompt Report: A Systematic Survey of Prompt Engineering Techniques”. In: *arXiv preprint* arXiv:2406.06608 (2024).
- [321] G. Servedio, A. D. Bellis, **Dario Di Palma**, V. W. Anelli, and T. D. Noia. “Are the Hidden States Hiding Something? Testing the Limits of Factuality-Encoding Capabilities in LLMs”. In: *ACL (1)*. Association for Computational Linguistics, 2025, pp. 6089–6104.
- [322] C. E. Shannon. “A mathematical theory of communication”. In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423.
- [323] P. Shaw, J. Uszkoreit, and A. Vaswani. “Self-Attention with Relative Position Representations”. In: *NAACL-HLT (2)*. Association for Computational Linguistics, 2018, pp. 464–468.
- [324] F. Shehzad and D. Jannach. “Everyone’s a Winner! On Hyperparameter Tuning of Recommendation Models”. In: *RecSys*. ACM, 2023, pp. 652–657.
- [325] X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang. ““Do Anything Now”: Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models”. In: *CCS*. ACM, 2024, pp. 1671–1685.
- [326] L. Sheng and S. Xu. “CSC-SQL: Corrective Self-Consistency in Text-to-SQL via Reinforcement Learning”. In: *CoRR* abs/2505.13271 (2025).
- [327] T. Shi, Y. Zhang, Z. Xu, C. Chen, F. Feng, X. He, and Q. Tian. “Preliminary Study on Incremental Learning for Large Language Model-based Recommender Systems”. In: *CIKM*. ACM, 2024, pp. 4051–4055.
- [328] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. “Membership Inference Attacks Against Machine Learning Models”. In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017, pp. 3–18.
- [329] C. V. Snell, J. Lee, K. Xu, and A. Kumar. “Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning”. In: *ICLR*. OpenReview.net, 2025.

- [330] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. “Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank”. In: *EMNLP. ACL*, 2013, pp. 1631–1642.
- [331] A. Søgaard. “Semi-supervised condensed nearest neighbor for part-of-speech tagging”. In: *ACL (2)*. The Association for Computer Linguistics, 2011, pp. 48–52.
- [332] G. Spillo, C. Musto, M. de Gemmis, P. Lops, and G. Semeraro. “Recommender systems based on neuro-symbolic knowledge graph embeddings encoding first-order logic rules”. In: *User Model. User Adapt. Interact.* 34.5 (2024), pp. 2039–2083.
- [333] H. Steck. “Evaluation of recommendations: rating-prediction and ranking”. In: *RecSys*. 2013, pp. 213–220.
- [334] H. Steck. “Embarrassingly Shallow Autoencoders for Sparse Data”. In: *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. Ed. by L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, and L. Zia. ACM, 2019, pp. 3251–3257. DOI: 10.1145/3308558.3313710.
- [335] W. Stigall, M. A. A. H. Khan, D. C. Attota, F. Nweke, and Y. Pei. “Large Language Models Performance Comparison of Emotion and Sentiment Classification”. In: *ACM Southeast Regional Conference*. ACM, 2024, pp. 60–68.
- [336] Student. “The probable error of a mean”. In: *Biometrika* (1908), pp. 1–25.
- [337] A. Sun. “Take a Fresh Look at Recommender Systems from an Evaluation Standpoint”. In: *SIGIR*. ACM, 2023, pp. 2629–2638.
- [338] Z. Sun, H. Fang, J. Yang, X. Qu, H. Liu, D. Yu, Y. Ong, and J. Zhang. “DaisyRec 2.0: Benchmarking Recommendation for Rigorous Evaluation”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 45.7 (2023), pp. 8206–8226.
- [339] Z. Sun, H. Liu, X. Qu, K. Feng, Y. Wang, and Y. S. Ong. “Large Language Models for Intent-Driven Session Recommendations”. In: *SIGIR*. ACM, 2024, pp. 324–334.

- [340] Z. Sun, D. Yu, H. Fang, J. Yang, X. Qu, J. Zhang, and C. Geng. “Are We Evaluating Rigorously? Benchmarking Recommendation for Reproducible Evaluation and Fair Comparison”. In: *RecSys*. New York, NY, USA: ACM, 2020, pp. 23–32.
- [341] I. Sutskever, O. Vinyals, and Q. V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *NIPS*. 2014, pp. 3104–3112.
- [342] B. G. Tabachnick, L. S. Fidell, and J. B. Ullman. *Using multivariate statistics*. Vol. 6. MA: pearson Boston, 2013.
- [343] Y. Tamm, R. Damdinov, and A. Vasilev. “Quality Metrics in Recommender Systems: Do We Calculate Metrics Consistently?” In: *RecSys*. ACM, 2021, pp. 708–713.
- [344] L. R. Tang and R. J. Mooney. “Automated Construction of Database Interfaces: Intergrating Statistical and Relational Learning for Semantic Parsing”. In: *EMNLP*. Association for Computational Linguistics, 2000, pp. 133–141.
- [345] W. L. Taylor. ““Cloze procedure”: A new tool for measuring readability”. In: *Journalism quarterly* 30.4 (1953), pp. 415–433.
- [346] I. Tenney, P. Xia, B. Chen, A. Wang, A. Poliak, R. T. McCoy, N. Kim, B. Van Durme, S. R. Bowman, D. Das, et al. “What do you learn from context? probing for sentence structure in contextualized word representations”. In: *arXiv preprint arXiv:1905.06316* (2019).
- [347] TheBlock. *TheBloke/Mistral-7B-Instruct-v0.2-AWQ · Hugging Face* — *huggingface.co*. <https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-AWQ>. [Accessed 26-03-2024].
- [348] C. Tian, B. Hu, C. Gan, H. Chen, Z. Zhang, L. Yu, Z. Liu, Z. Zhang, J. Zhou, and J. Chen. “ReLand: Integrating Large Language Models’ Insights into Industrial Recommenders via a Controllable Reasoning Pool”. In: *RecSys*. ACM, 2024, pp. 63–73.
- [349] C. Tillmann and H. Ney. “Word Reordering and a Dynamic Programming Beam Search Algorithm for Statistical Machine Translation”. In: *Comput. Linguistics* 29.1 (2003), pp. 97–133.

- [350] M. Tomlein, B. Pecher, J. Simko, I. Srba, R. Móro, E. Stefancova, M. Kompan, A. Hrckova, J. Podrouzek, and M. Bielíková. “An Audit of Misinformation Filter Bubbles on YouTube: Bubble Bursting and Recent Behavior Changes”. In: *RecSys*. ACM, 2021, pp. 1–11.
- [351] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, and Y. B. et al. “Llama 2: Open Foundation and Fine-Tuned Chat Models”. In: *CoRR* abs/2307.09288 (2023).
- [352] H. Touvron et al. “LLaMA: Open and Efficient Foundation Language Models”. In: *CoRR* abs/2302.13971 (2023).
- [353] M. Trevisiol, L. M. Aiello, R. Schifanella, and A. Jaimes. “Cold-start news recommendation with domain-dependent browse graph”. In: *RecSys*. 2014, pp. 81–88.
- [354] S. Upadhyay, E. Ginsberg, and C. Callison-Burch. “Improving Mathematics Tutoring With A Code Scratchpad”. In: *BEA@ACL*. Association for Computational Linguistics, 2023, pp. 20–28.
- [355] S. Vargas and P. Castells. “Rank and relevance in novelty and diversity metrics for recommender systems”. In: *Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011*. Ed. by B. Mobasher, R. D. Burke, D. Jannach, and G. Adomavicius. ACM, 2011, pp. 109–116.
- [356] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. “Attention is All you Need”. In: *NIPS*. 2017, pp. 5998–6008.
- [357] T. Vente, L. Wegmeth, A. Said, and J. Beel. “From Clicks to Carbon: The Environmental Toll of Recommender Systems”. In: *Proceedings of the 18th ACM Conference on Recommender Systems (RecSys '24)*. Bari, Italy: ACM, Oct. 2024, pp. 1–11. ISBN: 979-8-4007-0505-2. DOI: 10.1145/3640457.3688074.
- [358] S. Vichev and A. Marchev. “RAGSQL: Context Retrieval Evaluation on Augmenting Text-to-SQL Prompts”. In: *IS*. IEEE, 2024, pp. 1–6.
- [359] D. C. Visentin, M. Cleary, and G. E. Hunt. *The earnestness of being important: Reporting non-significant statistical results*. 2020.

- [360] I. Vulic, E. M. Ponti, R. Litschko, G. Glavas, and A. Korhonen. “Probing Pre-trained Language Models for Lexical Semantics”. In: *EMNLP (1)*. Association for Computational Linguistics, 2020, pp. 7222–7240.
- [361] D. Wadden, S. Lin, K. Lo, L. L. Wang, M. van Zuylen, A. Cohan, and H. Hajishirzi. “Fact or Fiction: Verifying Scientific Claims”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by B. Webber, T. Cohn, Y. He, and Y. Liu. Online: Association for Computational Linguistics, Nov. 2020, pp. 7534–7550. DOI: 10.18653/v1/2020.emnlp-main.609.
- [362] D. Wallach and B. Goffinet. “Mean squared error of prediction as a criterion for evaluating and comparing system models”. In: *Ecological modelling* 44.3-4 (1989), pp. 299–306.
- [363] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson. “RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers”. In: *ACL*. Association for Computational Linguistics, 2020, pp. 7567–7578.
- [364] B. Wang et al. “MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL”. In: *COLING*. Association for Computational Linguistics, 2025, pp. 540–557.
- [365] H. Wang, J. Lin, X. Li, B. Chen, C. Zhu, R. Tang, W. Zhang, and Y. Yu. “FLIP: Fine-grained Alignment between ID-based Models and Pretrained Language Models for CTR Prediction”. In: *RecSys*. ACM, 2024, pp. 94–104.
- [366] H. Wang, P. Minervini, and E. M. Ponti. “Probing the Emergence of Cross-lingual Alignment during LLM Training”. In: *ACL (Findings)*. Association for Computational Linguistics, 2024, pp. 12159–12173.
- [367] J. Wang, H. Lu, J. Caverlee, E. H. Chi, and M. Chen. “Large Language Models as Data Augmenters for Cold-Start Item Recommendation”. In: *WWW (Companion Volume)*. ACM, 2024, pp. 726–729.
- [368] L. Wang and E. Lim. “Zero-Shot Next-Item Recommendation using Large Pretrained Language Models”. In: *CoRR* abs/2304.03153 (2023).
- [369] S. Wang, B. Xie, L. Ding, J. Chen, and Y. Xiang. “Reinforced logical reasoning over KGs for interpretable recommendation system”. In: *Mach. Learn.* 114.4 (2025), p. 95.

- [370] W. Wang, H. Yin, Z. Huang, Q. Wang, X. Du, and Q. V. H. Nguyen. “Streaming Ranking Based Recommender Systems”. In: *SIGIR*. 2018, pp. 525–534.
- [371] X. Wang, X. Tang, X. Zhao, J. Wang, and J. Wen. “Rethinking the Evaluation for Conversational Recommendation in the Era of Large Language Models”. In: *EMNLP*. Association for Computational Linguistics, 2023, pp. 10052–10065.
- [372] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou. “Self-Consistency Improves Chain of Thought Reasoning in Language Models”. In: *ICLR*. OpenReview.net, 2023.
- [373] Y. Wang et al. “LLMRG: Improving Recommendations through Large Language Model Reasoning Graphs”. In: *AAAI*. AAAI Press, 2024, pp. 19189–19196.
- [374] Y. Wang, Z. Jiang, Z. Chen, F. Yang, Y. Zhou, E. Cho, X. Fan, X. Huang, Y. Lu, and Y. Yang. “RecMind: Large Language Model Powered Agent For Recommendation”. In: *CoRR* abs/2308.14296 (2023).
- [375] Y. Wang, Z. Ren, W. Sun, J. Yang, Z. Liang, X. Chen, R. Xie, S. Yan, X. Zhang, P. Ren, et al. “Content-Based Collaborative Generation for Recommender Systems”. In: *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 2024, pp. 2420–2430.
- [376] Y. Wang, Q. Xie, M. Tang, L. Li, J. Yuan, and Y. Liu. “Amazon-KG: A Knowledge Graph Enhanced Cross-Domain Recommendation Dataset”. In: *SIGIR*. ACM, 2024, pp. 123–130.
- [377] Z. Wang, J. Zhang, J. Feng, and Z. Chen. “Knowledge Graph Embedding by Translating on Hyperplanes”. In: *AAAI*. AAAI Press, 2014, pp. 1112–1119.
- [378] D. H. D. Warren and F. C. N. Pereira. “An Efficient Easily Adaptable System for Interpreting Natural Language Queries”. In: *Am. J. Comput. Linguistics* 8.3-4 (1982), pp. 110–122.
- [379] W. Webber, A. Moffat, and J. Zobel. “A similarity measure for indefinite rankings”. In: *ACM Trans. Inf. Syst.* 28.4 (2010), 20:1–20:38.
- [380] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *NeurIPS*. 2022.
- [381] J. Wei et al. “Emergent Abilities of Large Language Models”. In: *Trans. Mach. Learn. Res.* 2022 (2022).

- [382] J. Wei, J. Wei, Y. Tay, D. Tran, A. Webson, Y. Lu, X. Chen, H. Liu, D. Huang, D. Zhou, et al. “Larger language models do in-context learning differently”. In: *arXiv preprint arXiv:2303.03846* (2023).
- [383] W. Wei, X. Ren, J. Tang, Q. Wang, L. Su, S. Cheng, J. Wang, D. Yin, and C. Huang. “LLMRec: Large Language Models with Graph Augmentation for Recommendation”. In: *WSDM*. ACM, 2024, pp. 806–815.
- [384] J. Wen, P. Ke, H. Sun, Z. Zhang, C. Li, J. Bai, and M. Huang. “Unveiling the Implicit Toxicity in Large Language Models”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by H. Bouamor, J. Pino, and K. Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1322–1338. DOI: 10.18653/v1/2023.emnlp-main.84.
- [385] J. Wiebe. “Learning Subjective Adjectives from Corpora”. In: *AAAI/IAAI*. AAAI Press / The MIT Press, 2000, pp. 735–740.
- [386] W. Woods. “The lunar sciences natural language information system”. In: *BBN report* (1972).
- [387] H. Wu, Z. Wang, K. Wang, P. G. Omran, and J. Li. “Rule Learning over Knowledge Graphs: A Review”. In: *TGDK* 1.1 (2023), 7:1–7:23.
- [388] L. Wu et al. “A survey on large language models for recommendation”. In: *World Wide Web (WWW)* 27.5 (2024), p. 60.
- [389] M. Wu, B. Chen, S. Zhu, B. Zheng, W. Peng, and M. Zhang. “Neuro-symbolic recommendation model based on logic query”. In: *Knowl. Based Syst.* 284 (2024), p. 111311.
- [390] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui. “Graph Neural Networks in Recommender Systems: A Survey”. In: *ACM Comput. Surv.* 55.5 (2023), 97:1–97:37.
- [391] W. Wu, C. Jiang, Y. Jiang, P. Xie, and K. Tu. “Do PLMs Know and Understand Ontological Knowledge?” In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by A. Rogers, J. Boyd-Graber, and N. Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 3080–3101. DOI: 10.18653/v1/2023.acl-long.173.

- [392] X. Wu, H. Zhou, W. Yao, X. Huang, and N. Liu. “Towards Personalized Cold-Start Recommendation with Prompts”. In: *arXiv preprint arXiv:2306.17256* (2023).
- [393] Y. Xi, W. Liu, J. Lin, X. Cai, H. Zhu, J. Zhu, B. Chen, R. Tang, W. Zhang, and Y. Yu. “Towards Open-World Recommendation with Knowledge Augmentation from Large Language Models”. In: *RecSys*. ACM, 2024, pp. 12–22.
- [394] Q. Xia, B. Zhang, R. Wang, Z. Li, Y. Zhang, F. Huang, L. Si, and M. Zhang. “A Unified Span-Based Approach for Opinion Mining with Syntactic Constituents”. In: *NAACL-HLT*. Association for Computational Linguistics, 2021, pp. 1795–1804.
- [395] Y. Xian, Z. Fu, Q. Huang, S. Muthukrishnan, and Y. Zhang. “Neural-Symbolic Reasoning over Knowledge Graph for Multi-stage Explainable Recommendation”. In: *CoRR* abs/2007.13207 (2020).
- [396] Y. Xian et al. “CAFE: Coarse-to-Fine Neural Symbolic Reasoning for Explainable Recommendation”. In: *CIKM*. ACM, 2020, pp. 1645–1654.
- [397] C. Xiao, M. Dymetman, and C. Gardent. “Sequence-based Structured Prediction for Semantic Parsing”. In: *ACL (1)*. The Association for Computer Linguistics, 2016.
- [398] B. Xie, B. Xu, Y. Yuan, S. Zhu, and H. Shen. “From Outcomes to Processes: Guiding PRM Learning from ORM for Inference-Time Alignment”. In: *ACL (1)*. Association for Computational Linguistics, 2025, pp. 19291–19307.
- [399] F. Xing. “Designing heterogeneous llm agents for financial sentiment analysis”. In: *ACM Transactions on Management Information Systems* (2024).
- [400] L. Xu, J. Zhang, B. Li, J. Wang, M. Cai, W. X. Zhao, and J. Wen. “Prompting Large Language Models for Recommender Systems: A Comprehensive Framework and Empirical Analysis”. In: *CoRR* abs/2401.04997 (2024).
- [401] W. Xu, Q. Xie, S. Yang, J. Cao, and S. Pang. “Enhancing content-based recommendation via large language model”. In: *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 2024, pp. 4153–4157.

- [402] Y. Yaghoobzadeh, K. Kann, T. J. Hazen, E. Agirre, and H. Schütze. “Probing for Semantic Classes: Diagnosing the Meaning Content of Word Embeddings”. In: *ACL (1)*. Association for Computational Linguistics, 2019, pp. 5740–5753.
- [403] G. H. Yang, H. Wang, S. Han, C. Hauff, G. Zuccon, and Y. Zhang, eds. *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2024, Washington DC, USA, July 14-18, 2024*. ACM, 2024.
- [404] J. Yang, B. Hui, M. Yang, J. Yang, J. Lin, and C. Zhou. “Synthesizing Text-to-SQL Data from Weak and Strong LLMs”. In: *ACL (1)*. Association for Computational Linguistics, 2024, pp. 7864–7875.
- [405] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. B. Hu. “Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond”. In: *ACM Trans. Knowl. Discov. Data* 18.6 (2024), 160:1–160:32.
- [406] S. Yang, W. Ma, P. Sun, Q. Ai, Y. Liu, M. Cai, and M. Zhang. “Sequential Recommendation with Latent Relations based on Large Language Model”. In: *SIGIR*. ACM, 2024, pp. 335–344.
- [407] T. Yang and L. Chen. “Unleashing the Retrieval Potential of Large Language Models in Conversational Recommender Systems”. In: *RecSys*. ACM, 2024, pp. 43–52.
- [408] L. Yao, C. Sun, L. Li, Z. Hao, and X. Wang. “Hybrid learning of Syntactic and Semantic Dependencies”. In: *Comput. Inf. Sci.* 3.4 (2010), pp. 187–199.
- [409] H. Yin, B. Cui, J. Li, J. Yao, and C. Chen. “Challenging the Long Tail Recommendation”. In: *Proc. VLDB Endow.* 5.9 (2012), pp. 896–907.
- [410] T. Yu et al. “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task”. In: *EMNLP*. Association for Computational Linguistics, 2018, pp. 3911–3921.
- [411] W. Yu, T. Pang, Q. Liu, C. Du, B. Kang, Y. Huang, M. Lin, and S. Yan. “Bag of Tricks for Training Data Extraction from Language Models”. In: *ICML*. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 40306–40320.

- [412] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han. “Personalized entity recommendation: a heterogeneous information network approach”. In: *WSDM*. ACM, 2014, pp. 283–292.
- [413] Z. Yue, S. Rabhi, G. de Souza Pereira Moreira, D. Wang, and E. Oldridge. “LlamaRec: Two-Stage Recommendation using Large Language Models for Ranking”. In: *CoRR abs/2311.02089* (2023).
- [414] E. Zangerle and C. Bauer. “Evaluating Recommender Systems: Survey and Framework”. In: *ACM Comput. Surv.* 55.8 (2023), 170:1–170:38.
- [415] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. “HellaSwag: Can a Machine Really Finish Your Sentence?”. In: *ACL (1)*. Association for Computational Linguistics, 2019, pp. 4791–4800.
- [416] C. Zhang, Y. Mao, Y. Fan, Y. Mi, Y. Gao, L. Chen, D. Lou, and J. Lin. “Fin-SQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis”. In: *SIGMOD Conference Companion*. ACM, 2024, pp. 93–105.
- [417] J. Zhang, L. Chen, S. Berkovsky, M. Zhang, T. D. Noia, J. Basilico, L. Pizzato, and Y. Song, eds. *Proceedings of the 17th ACM Conference on Recommender Systems, RecSys 2023, Singapore, Singapore, September 18–22, 2023*. ACM, 2023.
- [418] J. Zhang, K. Bao, Y. Zhang, W. Wang, F. Feng, and X. He. “Is ChatGPT Fair for Recommendation? Evaluating Fairness in Large Language Model Recommendation”. In: *RecSys*. 2023, pp. 993–999.
- [419] J. Zhang, R. Xie, Y. Hou, X. Zhao, L. Lin, and J. Wen. “Recommendation as Instruction Following: A Large Language Model Empowered Recommendation Approach”. In: *ACM Trans. Inf. Syst.* 43.5 (2025), 114:1–114:37.
- [420] S. Zhang, L. Yao, A. Sun, and Y. Tay. “Deep Learning Based Recommender System: A Survey and New Perspectives”. In: *ACM Comput. Surv.* 52.1 (2019), 5:1–5:38.
- [421] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. “Opt: Open pre-trained transformer language models”. In: *arXiv preprint arXiv:2205.01068* (2022).
- [422] W. Zhang, L. Gui, R. Procter, and Y. He. “Multi-Layer Ranking with Large Language Models for News Source Recommendation”. In: *SIGIR*. ACM, 2024, pp. 2537–2542.

- [423] W. Zhang, Y. Deng, B. Liu, S. J. Pan, and L. Bing. “Sentiment Analysis in the Era of Large Language Models: A Reality Check”. In: *NAACL-HLT (Findings)*. Association for Computational Linguistics, 2024, pp. 3881–3906.
- [424] X. Zhang, Z. Yao, J. Zhang, K. Yun, J. Yu, J. Li, and J. Tang. “Transferable and Efficient Non-Factual Content Detection via Probe Training with Offline Consistency Checking”. In: *arXiv preprint arXiv:2404.06742* (2024).
- [425] Y. Zhang, J. Deriu, G. Katsogiannis-Meimarakis, C. Kosten, G. Koutrika, and K. Stockinger. “ScienceBenchmark: A Complex Real-World Benchmark for Evaluating Natural Language to SQL Systems”. In: *Proc. VLDB Endow.* 17.4 (2023), pp. 685–698.
- [426] Y. Zhang, X. Chen, Q. Ai, L. Yang, and W. B. Croft. “Towards Conversational Search and Recommendation: System Ask, User Respond”. In: *CIKM*. 2018, pp. 177–186.
- [427] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen, et al. “Siren’s Song in the AI Ocean: A Survey on Hallucination in Large Language Models”. In: *Computational Linguistics* (2025), pp. 1–46.
- [428] Y. Zhang, H. DING, Z. Shui, Y. Ma, J. Zou, A. Deoras, and H. Wang. “Language Models as Recommender Systems: Evaluations and Limitations”. In: *I (Still) Can’t Believe It’s Not Better! NeurIPS 2021 Workshop*.
- [429] Z. Zhang, C. Zheng, Y. Wu, B. Zhang, R. Lin, B. Yu, D. Liu, J. Zhou, and J. Lin. “The Lessons of Developing Process Reward Models in Mathematical Reasoning”. In: *ACL (Findings)*. Association for Computational Linguistics, 2025, pp. 10495–10516.
- [430] W. X. Zhao, Z. Lin, Z. Feng, P. Wang, and J. Wen. “A Revisiting Study of Appropriate Offline Evaluation for Top-*N* Recommendation Algorithms”. In: *ACM Trans. Inf. Syst.* 41.2 (2023), 32:1–32:41.
- [431] K. Zheng, Q. Zhao, and L. Li. “Reassessing the Role of Chain-of-Thought in Sentiment Analysis: Insights and Limitations”. In: *arXiv preprint arXiv:2501.08641* (2025).
- [432] V. Zhong, M. Lewis, S. I. Wang, and L. Zettlemoyer. “Grounded Adaptation for Zero-shot Executable Semantic Parsing”. In: *EMNLP (1)*. Association for Computational Linguistics, 2020, pp. 6869–6882.

- [433] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba. “Large Language Models are Human-Level Prompt Engineers”. In: *ICLR*. OpenReview.net, 2023.
- [434] Y. Zhu, C. Wan, H. Steck, D. Liang, Y. Feng, N. Kallus, and J. Li. “Collaborative Retrieval for Large Language Model-based Conversational Recommender Systems”. In: *WWW*. ACM, 2025, pp. 3323–3334.
- [435] Y. Zhu, L. Wu, Q. Guo, L. Hong, and J. Li. “Collaborative Large Language Model for Recommender Systems”. In: *WWW*. ACM, 2024, pp. 3162–3172.
- [436] Y. Zhu, Y. Xian, Z. Fu, G. de Melo, and Y. Zhang. “Faithfully Explainable Recommendation via Neural Logic Reasoning”. In: *NAACL-HLT*. Association for Computational Linguistics, 2021, pp. 3083–3090.
- [437] Y. Zhu, H. Yuan, S. Wang, J. Liu, W. Liu, C. Deng, H. Chen, Z. Liu, Z. Dou, and J.-R. Wen. “Large Language Models for Information Retrieval: A Survey”. In: *ACM Trans. Inf. Syst.* 44.1 (Nov. 2025). ISSN: 1046-8188. DOI: 10.1145/3748304.
- [438] A. Zou, L. Phan, S. Chen, J. Campbell, P. Guo, R. Ren, and A. P. et al. “Representation Engineering: A Top-Down Approach to AI Transparency”. In: *CoRR* abs/2310.01405 (2023).