



Politecnico di Bari

Repository Istituzionale dei Prodotti della Ricerca del Politecnico di Bari

Models and methods for eXplainable Artificial Intelligence in pervasive Cyber-Physical Systems

This is a PhD Thesis

Original Citation:

Models and methods for eXplainable Artificial Intelligence in pervasive Cyber-Physical Systems / Loconte, Davide. - ELETTRONICO. - (2026).

Availability:

This version is available at <http://hdl.handle.net/11589/296360> since: 2026-01-27

Published version

DOI:

Publisher: Politecnico di Bari

Terms of use:

(Article begins on next page)

23 April 2026



POLITECNICO DI BARI

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

Electrical and Information Engineering Ph.D. Program

SSD: ING-INF/05 – INFORMATION PROCESSING SYSTEMS

Final dissertation

Models and methods for eXplainable Artificial Intelligence in pervasive Cyber-Physical Systems

by Davide Loconte

Ph.D. program

Coordinator:

Prof. Ing. Mario Carpentieri

Supervisors:

Prof. Ing. Michele Ruta

Prof. Ing. Floriano
Scioscia

38TH COURSE, 01/11/2022 – 31/10/2025



LIBERATORIA PER L'ARCHIVIAZIONE DELLA TESI DI DOTTORATO

Al Magnifico Rettore
del Politecnico di Bari

Il/la sottoscritto/a Loconte Davide nato/a a Acquaviva delle Fonti (BA) il 21/09/1995

residente a Bari in via Isonzo 231, cap 70125 e-mail davide.loconte@poliba.it

iscritto al III° anno di Corso di Dottorato di Ricerca in Ingegneria Elettrica e dell'Informazione ciclo 38°

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

Models and methods for eXplainable Artificial Intelligence in pervasive Cyber-Physical Systems

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 ricorreranno 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 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**/inviolate/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, 19/01/2026

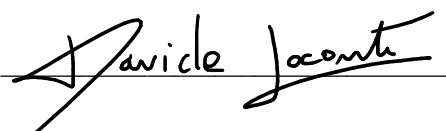
Firma 

Il/La 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, 19/01/2026

Firma 



POLITECNICO DI BARI

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

Electrical and Information Engineering Ph.D. Program

SSD: ING-INF/05 – INFORMATION PROCESSING SYSTEMS

Final dissertation

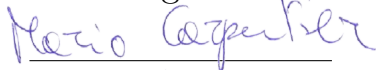
**Models and methods for eXplainable
Artificial Intelligence in pervasive
Cyber-Physical Systems**

by Davide Loconte 

Ph.D. program

Coordinator:

Prof. Ing. Mario Carpentieri



Referees:

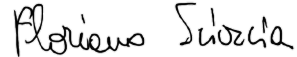
Prof. Ing. Antonio Puliafito

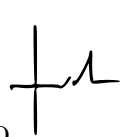
Dott. Andrea Pazienza

Supervisors:

Prof. Ing. Michele Ruta

Prof. Ing. Floriano

Scioscia 

38TH COURSE, 01/11/2022 – 31/10/2025

There's some good in this world, Mr. Frodo. And it's worth fighting for.

- Samwise Gamgee, *The Lord of the Rings: The Two Towers* (2002), directed by Peter Jackson

Abstract

Cyber-Physical Systems (CPSs) integrate sensing, computation, communication, and actuation to monitor and control physical processes in real time through Internet of Things (IoT) devices and protocols. Artificial Intelligence (AI) models and methods are increasingly integrated in CPSs to enhance data-driven autonomous decision-making capabilities. Despite improvements in embedded hardware, however, most devices still operate as passive data sources, relying on cloud infrastructures for computation and storage. This centralized approach introduces latency, bandwidth, and privacy concerns. Exploiting edge computing paradigms for data mining and Machine Learning (ML) tasks moves computation closer to data sources, allowing to mitigate these issues, but increases complexity due to resource constraints of edge devices and distributed task coordination. These problems are exacerbated when pushing computation and distributed decision-making further toward the IoT devices in the field. In a *Trustworthy AI* perspective, interpretability and auditability of ML are also critical for pervasive CPS in key sectors, but are rarely considered in edge intelligence. This dissertation describes the work done to address these limitations by combining knowledge representation and reasoning with distributed, incremental, and federated learning across the Cloud-to-Thing continuum.

The first major contribution is a general-purpose serverless framework for distributed CPSs, enabling orchestration of AI workflows, including data collection, training, and inference, across heterogeneous nodes. The second is MAFALDA 2.0, a lightweight and explainable on-device classifier that integrates symbolic Knowledge Representation and Reasoning (KRR) with subsymbolic statistical ML, supporting incremental and federated learning with logic-based justification of results.

Experimental validation shows that the proposed AI approach achieves accuracy comparable to state-of-the-art on-device learning systems while offering explainability through semantic reasoning. The two contributions

converge into a unified case study combining MAFALDA 2.0 with the Cloud-to-Thing framework, demonstrating the feasibility of explainable federated learning across heterogeneous computing infrastructures.

Future work includes dynamic ontology generation, expanding test coverage in large-scale deployments, and optimizing performance for highly constrained nodes to enhance applicability in real-world IoT environments. A further research gap concerns systematic qualitative and quantitative evaluation of eXplainable Artificial Intelligence (XAI) methods, incorporating standard interpretability metrics and user-centered studies to objectively assess the clarity and reliability of explanations.

Contents

Introduction	1
1 Towards Pervasive Intelligent Cyber-Physical Systems	6
1.1 Concepts and Architecture of Cyber-Physical Systems	7
1.2 Edge Computing and the Internet of Things	9
1.2.1 The Internet of Things in Cyber-Physical Systems	9
1.2.2 The Edge Computing Paradigm	11
1.3 Elements of Artificial Intelligence and Machine Learning	14
1.3.1 Generative AI and Large Language Models	20
1.4 Methods for eXplainable Artificial Intelligence	22
1.4.1 A Taxonomy of Explainability Methods in Artificial Intelligence	25
1.4.2 Approaches to Explainable Artificial Intelligence	28
1.5 Knowledge Representation and Reasoning for explainability	31
1.5.1 Description Logics	32
1.5.2 Web Ontology Language	35
1.5.3 Reasoning Services	37
1.6 State of the Art in Distributed Architectures for Cyber-Physical Systems	40
1.7 Semantic-based Enhancements of Pervasive Cyber-Physical Systems	43
1.7.1 Connection Layer: Semantic-Oriented Infrastructure and Connectivity Management	44
1.7.2 Conversion Layer: Semantic Reasoning and Data-to- Knowledge Mapping for CPS	48

1.7.3	Cyber Layer: Local Semantic Reasoning at the Edge for Autonomous Decision Support	51
1.7.4	Cognition Layer: Retrieval-Augmented Generation for Digital Twins	56
2	Expanding pervasive Cyber-Physical Systems to the Server- less Cloud-to-Thing continuum	61
2.1	Design Motivation and System Requirements	61
2.2	Abstract Architecture for Cloud-to-Thing Orchestration	65
2.3	Functional Components and System Workflow	70
2.4	Prototype Deployment of Serverless Machine Learning in a Cloud-to-Thing Architecture	73
2.4.1	Portability	77
2.4.2	Experimental Evaluation and Results	79
3	Explainable Federated On-Device Learning for the Semantic Web of Everything	82
3.1	On-Device Federated and Incremental Learning for Pervasive Cyber-Physical Systems	83
3.2	Design of an Explainable On-Device Learning Framework	86
3.2.1	Ontology schema and data annotation	89
3.2.2	Incremental and federated learning	93
3.2.3	Inference and explainability	100
3.3	Performance and Accuracy Evaluation	108
3.4	Case Study: Activity Recognition in Cloud-Edge-IoT Scenarios	119
3.4.1	Prototype Workflow	120
3.4.2	Experiments	122
	Conclusions and Perspectives	131
	Bibliography	135

List of Figures

1.1	The 5C architecture of Cyber-Physical Systems.	8
1.2	Cloud computing paradigm.	11
1.3	Edge computing paradigm.	11
1.4	Classification of Cloud-Edge Intelligence approaches [203]. . .	13
1.5	Decision tree example [18].	16
1.6	Retrieval augmented generation architecture [69].	22
1.7	The MARGIN XAI method [8]	29
1.8	Semantic matchmaking framework	40
1.9	Semantic-enabled network management architecture.	45
1.10	yang2OWL architecture.	46
1.11	Example of the OWL ontology generated from YANG module. . .	46
1.12	Example of ABox generation from JSON document.	48
1.13	Semantic Web of Everything Architecture.	49
1.14	Porting workflow and high-level architecture of Tiny-ME Wasm. .	52
1.15	Low-level architecture of Tiny-ME Wasm.	53
1.16	Tiny-ME Wasm ontology classification results.	54
1.17	Tiny-ME Wasm semantic matchmaking results.	55
1.18	Smart Grid Architecture Model [69].	56
1.19	Reference graph-based data model [69].	58
1.20	Examples of digital-twin visualizations: (a) energy grid repre- sentation; (b) 2D model of plant assets [69].	59
1.21	Screenshots of the mobile application. [69]	60
2.1	Minimal Computing Node.	66
2.2	Complete Computing Node.	68
2.3	Sequence diagram of a generic operation dispatch.	69

2.4	Reference framework architecture.	71
2.5	AWS-based prototype architecture.	74
2.6	IoT Field Device firmware.	76
2.7	Latency and processing time for edge and cloud nodes over message sequence.	79
3.1	MAFALDA 2.0 Architecture	87
3.2	Training Matrix structure.	94
3.3	Generated Ontology	110
3.4	Classification results: confusion matrices	113
3.5	MAFALDA 2.0 global explanation: learned semantic description of output class	117
3.6	MAFALDA 2.0 local explanation: classification outcome ex- plained by means of reasoning services.	118
3.7	Data upload sequence diagram	120
3.8	Model update sequence diagram	121
3.9	Inference workflow diagram	123
3.10	Confusion matrices: bootstrap model vs final model	128

List of Tables

1.1	Naming conventions for Description Logics	34
1.2	Syntax and semantics of $\mathcal{ALN}$ constructors and axiom types.	35
1.3	Mapping between OWL RDF/XML and DL syntax.	36
2.1	Reference development boards for Microcontroller Units. . . .	63
2.2	Reference development boards for Microprocessor Units. . . .	64
2.3	Framework operations grouped by functional category.	72
2.4	Average and standard deviation (in milliseconds) of communi- cation latency and processing time per node.	80
3.1	Example dataset with two features and one label.	92
3.2	Training matrix at the fourth sample of the example dataset .	97
3.3	Training matrix after	98
3.4	Training matrix at the fourth sample of the example dataset .	98
3.5	Normalized training matrix	104
3.6	Comparison of ontology generation strategies for numerical features.	111
3.7	Performance Metrics for Heart Disease and Breast Cancer Datasets	113
3.8	Comparison of Breast Cancer Wisconsin dataset results with related works	114
3.9	Comparison of Heart Disease Binary dataset results with re- lated works	115
3.10	Serialization and Deserialization Times	116
3.11	Accuracy for Breast Cancer Dataset of all devices	116
3.12	Comparison on the Handwritten Digits dataset.	119

3.13 Comparison on the HAR dataset between TFMicro and MAFALDA	
2.0	119
3.14 Communication latencies for edge- and cloud-connected field	
devices (ms)	125
3.15 AWS Lambda Pricing in eu-central-1 region	126
3.16 Cloud Node: execution Time and Cost	127
3.17 Mafalda functions training and prediction performance	127
3.18 Comparison of evaluation summaries for three models	129

Acronyms

AI Artificial Intelligence. 1–4, 8, 12, 14, 22, 23, 25, 27, 30, 31, 42, 43, 45, 56, 62, 70, 82, 85, 86, 108, 131, 134, I

ANN Artificial Neural Network. 17, 19

API Application Programming Interface. 47, 52, 55, 57

AWS Amazon Web Services. 73–80, 119, 125, 126, 131, 132

BERT Bidirectional Encoder Representations from Transformers. 20, 21

BSP Board Support Package. 76, 77

CEI Cloud-Edge Intelligence. 12, 13, 40, 41, 83

CLI Command Line Interface. 87

CMSIS Common Microcontroller Software Interface Standard. 76

CNF Conjunctive Normal Form. 39, 105

CNN Convolutional Neural Network. 19, 83

CoAP Constrained Application Protocol. 66, 69

COTS Commercial Off-the-Shelf. 73, 131

CPS Cyber-Physical System. 1, 3, 4, 6–10, 14, 20, 31, 37, 40, 44, 48, 50, 51, 56, 61, 62, 66–68, 82–84, 86, 107, 119, 128, 131, 134, I

CPU Central Processing Unit. 126

CSV Comma Separated Value. 54

CV Cross Validation. 124, 129

DL Description Logics. 31–33, 35–38

DNN Deep Neural Network. 19, 24, 25, 42, 119

DRAM Dynamic Random Access Memory. 64

DSS Decision Support System. 45, 47

DT Decision Tree. 17, 85

DTDL Digital Twins Definition Language. 57

ETSI European Telecommunications Standards Institute. 46

FaaS Function as a Service. 42

FL Federated Learning. 41–43, 65, 70, 77, 78, 82, 84–86, 89, 100, 116, 119, 132

FOL First Order Logic. 32

GPT Generative Pre-trained Transformer. 20

GPU Graphical Processing Unit. 19, 21

GRU Gated Recurrent Unit. 19

HAL Hardware Abstraction Layer. 76

IAM Identity and Access Management. 80

ICE Individual Conditional Expectation. 30

IL Incremental Learning. 78, 82, 84, 86, 89, 99, 129, 132

IoE Internet of Everything. 4, 6, 7, 15, 43, 44, 51, 86, 134

IoT Internet of Things. 1, 2, 4, 6, 9–14, 41–44, 58, 61, 62, 65, 66, 70, 71, 73–81, 84, 120, 122, 125, 129, 131, I, II

IPC Inter-Process Communication. 75

IRI Internationalized Resource Identifier. 35, 100

JSON JavaScript Object Notation. 47, 48

KB Knowledge Base. 33, 46, 47, 49–51, 57, 86, 88, 90

KDN Knowledge-Defined Networking. 44, 45

KG Knowledge Graph. 3, 4, 22, 33, 36, 49, 50

kNN k-Nearest-Neighbors. 18, 84, 85

KP Knowledge Plane. 45, 47

KRR Knowledge Representation and Reasoning. 3, 4, 31, 87, 89, 131, I

LIL Lifelong Incremental Learning. 84

LIME Local Interpretable Model-agnostic Explanation. 30, 107

LLM Large Language Model. 20–22, 56–58

LOD Linked Open Data. 107

LRP Layer-wise Relevance Propagation. 29

LSTM Long Short-Term Memory Network. 19, 83

M2M Machine-to-Machine. 14

MCU Micro-Controller Unit. 1, 2, 4, 7, 10, 43, 64–66, 73, 76–78, 84–86, 107, 111, 112, 114, 116, 118

ML Machine Learning. 1–4, 8, 12, 14, 15, 19, 22, 24, 28, 32, 41–43, 45, 50, 57, 62, 72, 78, 81–83, 87–89, 120, 121, 131–133, I

MPU Micro-Processor Unit. 1, 2, 4, 7, 64, 73

MQTT Message Queuing Telemetry Transport. 57, 66, 69, 74

NA Network Analytics. 45

NAS Neural Architecture Search. 19

NETCONF Network Configuration Protocol. 45

NFV Network Function Virtualization. 44

NLP Natural Language Processing. 19–21

NUI Natural User Interface. 56, 57

ONF Open Networking Foundation. 46

OTA Over-the-air. 67

OWA Open World Assumption. 39, 91

OWL Web Ontology Language. 3, 35–38, 46, 47, 49–52, 83, 86, 89, 91, 116

P2P Peer-to-Peer. 80

PDP Partial Dependence Plot. 30

pub/sub Publish/Subscribe. 65, 69

QoS Quality of Service. 47, 48, 69

R2RML RDB to RDF Mapping Language. 50

RAG Retrieval-Augmented Generation. 22, 57, 58

RAII Resource Acquisition is Initialization. 53

RAM Random Access Memory. 10, 126

RDF Resource Description Framework. 35, 36, 50, 57

REST REpresentational State Transfer. 57

RF Random Forest. 17, 85

RML RDF Mapping Language. 50

RNN Recurrent Neural Network. 19, 129

RTOS Real-Time Operating System. 71, 77, 80

S3 Simple Storage Service. 75, 80

SDK Software Development Kit. 77

SDN Software-Defined Networking. 44, 46, 47

SEAS Smart Energy Aware Systems. 57

SGAM Smart-Grid Architecture Model. 56

SHAP SHapley Additive exPlanations. 30, 85, 107

SHFL Serverless Hierarchical Federated Learning. 42

SQL Structured Query Language. 36

SRAM Static Random Access Memory. 64

SVM Support Vector Machine. 18, 29, 84

SWoE Semantic Web of Everything. 3, 4, 37, 38, 49–51, 56, 82, 83, 90, 131

SWRL Semantic Web Rule Language. 47, 48

T5 Text-to-Text Transfer Transformer. 20, 21

TAPI Transport API. 47

TBox Terminological Box. 90

TDC TinyDecisionTreeClassifier. 115

TinyML Tiny Machine Learning. 84, 85

TLS Transport Layer Security. 80

TM Training Matrix. 88, 89, 92–95, 97–102, 104, 107, 116, 117, 122, 130

TPU Tensor Processing Unit. 19, 21

UI User Interface. 47

URI Uniform Resource Identifier. 35

W3C World Wide Web Consortium. 35

Wasm Web Assembly. 52, 55, 56

WLAN Wireless Local Area Network. 54

XAI eXplainable Artificial Intelligence. 2–4, 22–24, 26, 28–30, 40, 82, 85, 86, 116, 131, 134, II, III

YANG Yet Another Next Generation. 45–47

Introduction

In the past decade, there has been substantial progress in the development of CPSs, complex systems that integrate sensing, computation, communication, and actuation to enable real-time monitoring and control of physical processes [29, 90, 145]. In this context, the IoT provides the hardware and protocols to connect distributed devices and sensors, enabling data acquisition, exchange, and coordination across pervasive and ubiquitous networks of devices, to support decision-making and process automation [63, 37]. These devices generate large streams of unstructured data, which must be managed and processed to extract value. AI, and specifically ML, provide the algorithms to analyze these data streams and run *predictive* (for classification and forecasting) and *prescriptive* (for decision-making) models [88]. CPSs have been deployed in a wide range of domains including manufacturing, healthcare, transportation, energy, and agriculture. The main limitations of state-of-the-art ML techniques lie in their dependence on very large amounts of computational resources and storage capacity, as well as their inherent opacity, which makes it difficult to interpret and explain decisions made by complex models. Since IoT devices comprise embedded Micro-Controller Units (MCUs), and occasionally lightweight Micro-Processor Units (MPUs), they have been used so far essentially for measuring and transmitting data, while cloud computing has traditionally handled intensive computation and storage tasks. However, reliance on a centralized cloud infrastructure introduces latency, bandwidth consumption, and privacy concerns, which may be intolerable in distributed and pervasive CPSs.

Albeit still limited, advances in embedded hardware manufacturing have led to significant increases in the computational power of relatively inexpensive IoT devices and single-board computers. Modern MCUs and low-power

MPUs now feature higher clock frequencies, increased memory, and dedicated accelerators for signal processing and ML inference. These improvements enable the local execution of some tasks that previously required remote cloud resources, promoting a paradigm shift toward *edge computing*. In this model, computation is partially or fully performed closer to data sources, reducing dependency on centralized infrastructures [47, 199].

Edge computing often integrates with cloud computing, addressing several limitations of the latter. By processing data in the local network, it decreases communication latency, reduces bandwidth consumption, and ensures data confidentiality by avoiding the transfer of sensitive information to remote servers [157]. Furthermore, it enhances system resilience by removing the cloud platform’s single point of failure, increasing service availability even during network disruptions. These properties are crucial, especially for real-time and mission-critical applications [2, 134].

Despite these advances, several gaps remain. In most deployments, IoT devices based on MCUs are still primarily used as data collectors rather than active computational nodes. Their role is often limited to sensing and forwarding measurements to edge or cloud servers, where data processing and model training take place. As a result, their potential for executing AI workloads, including on-device learning and reasoning, remains underexplored and unused [126]. The lack of unified frameworks and resource-efficient algorithms further curbs the adoption of decentralized on-device ML [177].

While *edge intelligence* focuses on distributing AI/ ML tasks across all network layers [203], XAI seeks to address the interpretability challenge by making models and outcomes more transparent and understandable to human operators. So far, however, explainability is rarely addressed in the edge intelligence domain. In fact, most approaches focus on performance optimization, model compression, and latency reduction [95], overlooking the need for transparent and interpretable models. This omission poses a critical limitation in domains where accountability, safety, and trustworthiness are essential.

A promising approach to address these challenges is the adoption of KRR technologies. KRR enable the representation of information in a machine-understandable form through formal semantics grounded on logics and shared ontologies. The adoption of KRR enhances ML explainability, since symbolic representations, such as those based on the Web Ontology Language (OWL) standard, can elicit how concepts relate to one another and how reasoning steps lead to specific outcomes [144]. When integrated with data-driven ML models, these representations enable hybrid approaches that combine statistical methods with semantic reasoning. Furthermore, by expressing the underlying AI algorithms w.r.t. OWL standard ontologies, the decision pipelines and all the processes involved are described within standard vocabularies and can be linked with external data sources. This approach aligns with the broader vision of the *Semantic Web of Everything (SWoE)* [155], which extends the principles of the Semantic Web to include not only web resources but also pervasive physical devices, sensors, actuators, and living beings, up to end users.

This dissertation positions itself in the XAI domain for interpretable on-device ML with decentralized coordination in pervasive CPSs. In particular, the contributions of this work are summarized as follows:

- **A unified framework for on-device and edge AI.** The thesis introduces MAFALDA 2.0, a novel architecture that enables training, inference, and reasoning directly on constrained devices. MAFALDA 2.0 uses symbolic reasoning to provide a interpretable ML model intelligence that can be executed ubiquitously across the Cloud-to-Thing continuum.
- **Semantic integration through ontology-based modeling.** MAFALDA 2.0 leverages OWL-based ontologies to represent features, labels, and knowledge w.r.t. a shared Knowledge Graph (KG).
- **Incremental and federated learning support.** MAFALDA 2.0 uses a compact intermediate data structure that allows model updates from continuous data streams.

- **Explainable on-device ML.** The proposed approach integrates non-standard reasoning services to provide interpretable, contrastive, and counterfactual explanations of model predictions, supporting trustworthy and human-understandable AI.
- **Implementation and validation.** A full implementation in the C programming language ensures portability across heterogeneous hardware, including MCUs, MPUs, and embedded platforms. Experimental validation on multiple public datasets and a Cloud-to-Thing framework prototype demonstrates the feasibility of the proposed solution under realistic constraints.
- **Integration within a pervasive CPS architecture.** The framework is embedded into a Cloud-to-Thing architecture based on serverless computing for dynamic scheduling of training and inference tasks.

The remainder of the dissertation is organized as follows. **Chapter 1** introduces the conceptual foundations of pervasive intelligent CPSs. It discusses the evolution of IoT and edge computing, the emergence of the Internet of Everything (IoE) and the SWoE, and reviews distributed frameworks enabling edge intelligence within pervasive CPSs.

It also provides the theoretical background and state of the art on AI, ML, XAI, and KRR. It reviews key methodologies for explainability, the role of KGs, and recent approaches to on-device and federated learning for pervasive CPS applications.

Chapter 2 presents a framework for distributed CPS that unifies the management of the Cloud-to-Thing continuum. It describes the design motivation, abstract architecture, and workflow for integrating serverless computing with federated ML operations and reports a prototype implementation and experimental validation. **Chapter 3** introduces the proposed MAFALDA 2.0 framework for explainable federated on-device learning within the SWoE. It details the system design, ontology-based modeling, training and inference mechanisms, experimental evaluation on a range of platforms and datasets,

and a case study on activity recognition integrating the Cloud-to-Thing framework introduced in Chapter 2.

Chapter 1

Towards Pervasive Intelligent Cyber-Physical Systems

CPSs combine sensing, computation, communication, and actuation to monitor and control physical processes. With the growth of the IoT, these systems now rely on large networks of connected devices that collect and process data in real time. Advances in embedded hardware have transformed many classes of devices from simple data collectors into nodes capable of local computation. This shift has accelerated the rise of edge computing, which moves data processing closer to where information is generated, reducing latency and improving data confidentiality. At the same time, the IoE broadens this vision by connecting not only devices, but also people, data, and processes into a unified domain. Together, these technologies form the foundation for pervasive intelligent CPSs.

The purpose of this chapter is to provide an overview of the fundamental concepts that support the subsequent sections of this work. Section 1.1 outlines the foundations and architecture of CPSs, including the 5C model and its main applications in fields such as manufacturing, healthcare, and digital twins. Section 1.2 describes how IoT and edge computing extend CPS capabilities by bringing computation closer to data sources. Section 1.6 reviews current distributed CPS architectures and identifies open challenges in scalability, coordination, and the integration of serverless and federated learning across the Cloud-to-Thing continuum. Finally, Section 1.7 explores

novel proposals for integrating pervasive computing and the IoE into CPSs, focusing on how semantic technologies support interoperability and reasoning. Specifically, it presents examples of semantic network management, ontology-based data integration, and lightweight reasoning engines that can run directly on embedded devices.

1.1 Concepts and Architecture of Cyber-Physical Systems

CPSs represent an approach to complex systems design, where software and hardware interact with the physical world through continuous sensing and feedback. In particular, CPSs can be characterized as advanced integrations of computation, networking, and physical processes.

CPSs are implemented in key reference domains such as Industry 4.0 [29] and Industry 5.0 paradigms [90]. In smart manufacturing, CPSs are employed for monitoring, automation, decentralized control, and decision support [145]. They can also improve the sustainability of production pipelines by optimizing energy consumption, enabling real-time quality control, and supporting predictive maintenance strategies that reduce environmental impact and resource usage [75]. In healthcare, CPS allow for continuous monitoring outside clinical settings using wearable devices, enabling timely detection of anomalies and proactive health recommendations [98].

A widely adopted model to summarize the architecture of CPSs is the *5C architecture*, shown in Figure 1.1 [87]. In particular, the model consists of five layers:

- **Connection:** interfaces directly with the physical process and comprises sensing and actuation capabilities, typically implemented via embedded MCUs and MPUs that manage communication with sensors and actuators.

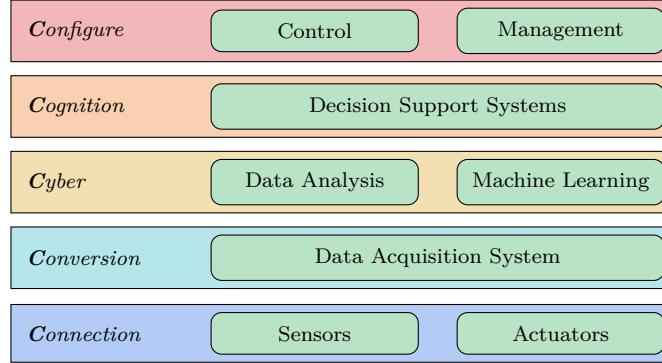


Figure 1.1: The 5C architecture of Cyber-Physical Systems.

- **Conversion:** collects data from the Connection layer and is responsible for transforming and refining them through filtering, aggregation, and basic pre-processing, allowing those data to be further analyzed in the layers above.
- **Cyber:** implements data analysis using AI, ML, and data mining algorithms. It allows implementing a wide range of tasks, spanning pattern recognition, real-time diagnostics, time-series forecasting, and decision support.
- **Cognition:** With the results of the Cyber layer analysis, CPSs can implement the ability to autonomously reason and adjust to evolving conditions or signals inferred from the data. For instance, predictive models may trigger adaptive maintenance policies based on historical trends.
- **Configuration:** enables the autonomous adjustment of operations with a set of *self*-* capabilities, *e.g.*, self-organize, self-adapt, [202] self-manage, self-repair, self-recover, self-heal [13]. Resource orchestration and the implementation of corrective measures ensure resilience and optimal performance. This layer often contains interfaces that allow human operators to monitor and act on the system itself.

CPSs are increasingly employed as both a conceptual and an engineering framework for digital twins [17]. In the literature, the term denotes a virtual

representation (the digital twin a.k.a. virtual counterpart) of a physical asset, updated in real-time to reflect its state through a real-time, bidirectional data and information flow. These virtual models replicate the state, behavior, and structure of physical assets (the physical twin). Initially used in aerospace and manufacturing, digital twins are now being developed and adopted across a wide range of domains, including energy systems [69], smart cities [6], healthcare [109], and transportation [121]. Within the context of CPSs, digital twins serve as the digital layer that enables simulation, monitoring, and control of the physical component. Conversely, CPSs provide the sensing, actuation, and communication infrastructure that makes the creation and synchronization of digital twins possible, and enable interfacing them with external actors and systems.

1.2 Edge Computing and the Internet of Things

As CPSs are systems that tightly integrate sensing, computation, and actuation to monitor and control physical processes, IoT and edge computing have emerged as two enabling technologies for their design and implementation. The IoT provides the hardware and protocols that can be used to disseminate sensors and actuators in the environment, while edge computing is an architectural paradigm that promotes the execution of data processing and control logic near the data acquisition site.

1.2.1 The Internet of Things in Cyber-Physical Systems

The IoT is a field that has been studied and popularized since its 1999 introduction by the Massachusetts Institute of Technology Auto-ID Center [85] with steady growth and market penetration [136]. The main idea behind this field is to interconnect everyday physical objects and locations (“things”) to data networks and the Internet through sensors, actuators, devices, appliances, or other embedded systems, thereby enabling them to sense, communicate,

share data, and act in real time or near-real time, often with minimal human intervention [63].

A typical IoT device is equipped with a microcontroller or microprocessor that executes a lightweight runtime to manage sensing, communication, and local data processing. In the context of CPSs, the IoT is the interface between the physical and digital worlds [37].

Early IoT devices were extremely constrained: 8bit MCU running at a few MHz, very limited Random Access Memory (RAM) (in the order of few kilobytes), minimal to no local storage, and a simple firmware stack that merely sampled sensors and forwarded data. Although device capabilities have grown considerably, the most restricting factor was and still is energy, since these devices are designed to run for prolonged periods of time on small batteries or using energy-harvesting techniques [160].

Large-scale complex applications relying on IoT device networks for data gathering usually rely on cloud computing infrastructure for processing. In cloud computing, resources granted by the provider are virtualized. This enables the cloud provider to efficiently manage and share resources across tenants, as well as scale them in accordance with user requests [71].

Since many IoT devices operate under strict constraints in terms of computation, memory, and energy, the cloud is commonly employed as the backbone for data storage and processing. In addition to offloading resource-intensive tasks, it enables centralized control of the underlying network of devices, and supports middleware for data routing and processing as well as dashboards, monitoring platforms, and decision-support engines for end-user applications. The overall model is illustrated in Figure 1.2.

Nevertheless, the integration of the IoT with the cloud presents various challenges, including:

1. **network latency**, due to the physical distance between IoT devices and cloud data centers;

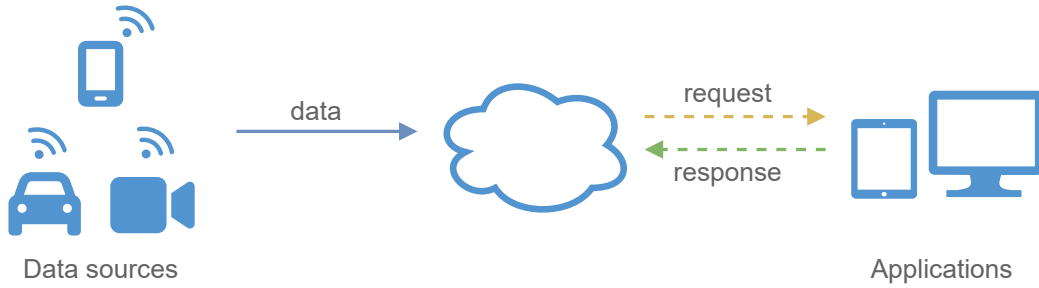


Figure 1.2: Cloud computing paradigm.

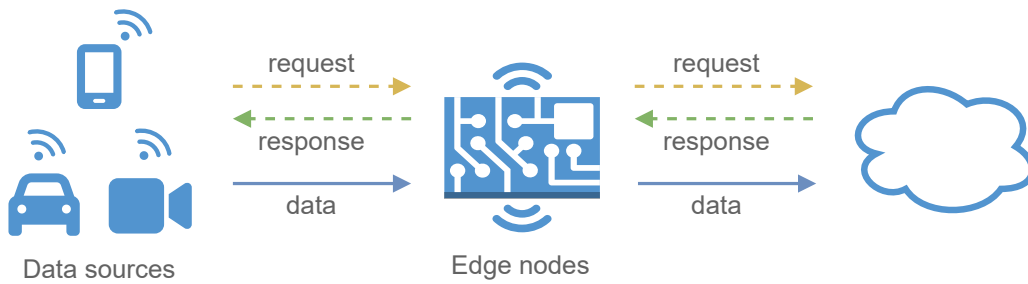


Figure 1.3: Edge computing paradigm.

2. **network bandwidth**, which can become the primary bottleneck as large volumes of data must be transmitted continuously from many interconnected nodes;
3. and **reliability**, given that connectivity disruptions can hinder access to cloud-hosted services.

Furthermore, data ownership, governance, and confidentiality can become critical concerns when sensitive information is transmitted to and stored in third-party infrastructures, often outside national borders and subject to foreign legislation [23].

1.2.2 The Edge Computing Paradigm

The improvement in manufacturing processes and miniaturization of electronic components has led to an increase in terms of computational capabilities of IoT

nodes. This has resulted in the emergence of the edge computing paradigm as a way to address the inherent limitations of cloud computing. Edge computing involves executing computational tasks directly at the periphery of the network, either on the IoT nodes themselves or on *edge nodes* at the border of the local network, minimizing the need for continuous communication with centralized infrastructures [157]. This approach has several advantages. Processing data in proximity to its acquisition site significantly lowers response latency and network bandwidth consumption. Additionally, this approach offers improved support for privacy-sensitive operations by avoiding unnecessary data transfers.

Related approaches include *fog computing* [1], *micro-datacenters* [14], and *cloudlets* [94]. All these paradigms are designed to partially offload computation from centralized cloud infrastructures while avoiding the strict resource limitations of edge devices. Fog computing typically leverages network gateways and routers to perform distributed processing and filtering closer to the data source. Cloudlets are small-scale data centers, often co-located with access points or base stations, offering localized computing resources to nearby devices with reduced latency. Similarly, micro-datacenters provide modular, geographically distributed infrastructure that can host lightweight services and support high availability with lower communication overhead.

In edge computing architectures, the cloud platform is not eliminated. Rather, it is leveraged for the most intensive computational tasks, when the resources at the edge are insufficient and the increased latency is tolerable. This collaborative approach between edge and cloud resources has led to research efforts in *cloud-edge* frameworks, which dynamically and opportunistically allocate workloads to edge or cloud nodes [150]. These architectures aim to reduce end-to-end latency, energy consumption, and network overhead, while preserving the responsiveness needed in real-time, critical domains [2, 134].

When such infrastructures are employed to support AI and ML workflows, they are often referred to in the literature as Cloud-Edge Intelligence (CEI). This paradigm distributes AI tasks across the cloud-edge continuum, *i.e.*, the whole network composed of both the cloud infrastructure and the edge nodes

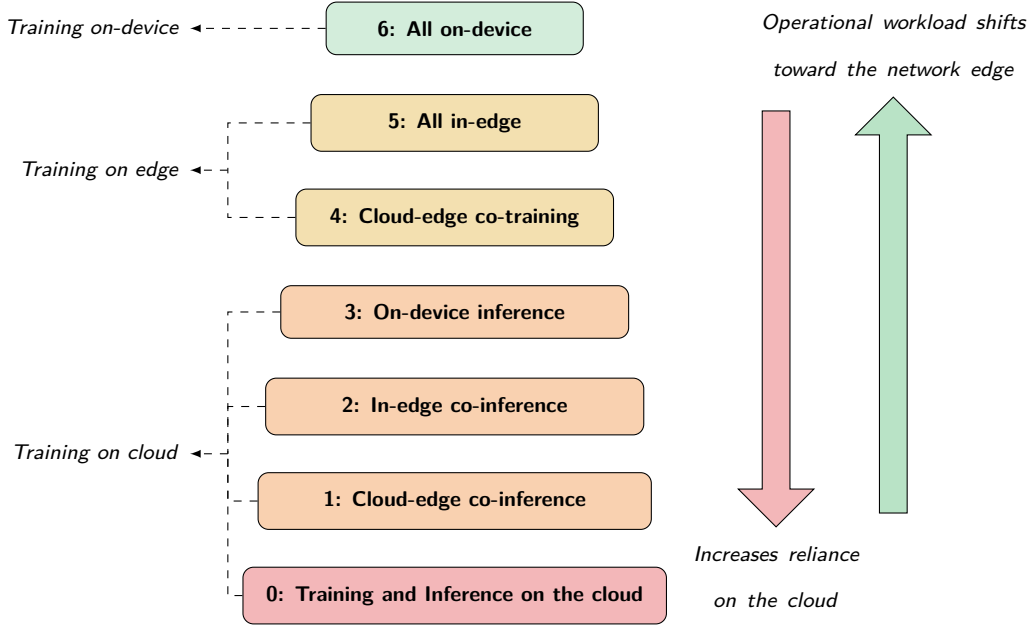


Figure 1.4: Classification of Cloud-Edge Intelligence approaches [203].

viewed as a uniform and unified entity. Most implementations commonly assign model training to cloud platforms and inference to edge or field IoT devices, in order to ensure both large computational capacity and low-latency responses [177]. However, with the increase in computational power at the edge of the network, alternative cooperation schemes have emerged, where training, decision-making, and data analysis may also occur at the network periphery under certain conditions [47, 199]. Figure 1.4 reports a taxonomy of seven levels of CEI collaboration, ranging from fully centralized intelligence to completely on-device processing [203]:

Level 0: Both training and inference are performed entirely in the cloud, with no involvement of edge or end devices. This configuration, often referred to as *Cloud Intelligence*, centralizes all processing but may suffer from all the issues discussed in Section 1.2.1.

Level 1: Training remains in the cloud, while inference is shared between the edge and cloud (cloud-edge coinference). This cooperation involves partial data offloading to the cloud during inference.

- Level 2:** This level further reduces cloud dependency by executing inference entirely within the edge network (in-edge coinference), while training still occurs in the cloud. Here, data may be processed by edge servers or nearby devices using Machine-to-Machine (M2M) communication.
- Level 3:** Inference is fully performed on the end device itself (on-device inference), eliminating the need for data transmission during inference. However, model training still takes place in the cloud, requiring the training data to be transmitted over the network.
- Level 4:** Introduces cloud-edge co-training, where both training and inference are performed collaboratively across cloud and edge nodes. This approach enables more adaptive and decentralized learning strategies.
- Level 5:** Both training and inference are executed entirely within the edge segment (all in-edge).
- Level 6:** pushes the intelligence completely to the device level (all on-device), achieving maximal data locality and autonomy at the cost of higher resource demands on end devices.

1.3 Elements of Artificial Intelligence and Machine Learning

The widespread adoption of IoT technologies in CPSs has led to an increase in the collection of high-frequency, heterogeneous data streams. Given this, AI and ML are increasingly employed in CPSs as an effective solution for extracting value from the generated data and supporting decision-making [88]. These techniques are used to implement functionalities such as fault detection, predictive maintenance, and process control. In addition, as discussed in Section 1.2.2, modern IoT devices have become more capable in terms of computing power, allowing for on-device learning and local inference in real time.

When designing a ML system, several factors must be considered. A primary requirement is the availability of large, high-quality datasets related to the target problem. While data are often available in IoE environments, the presence of incomplete, noisy, or irrelevant samples can negatively impact model accuracy. Proper data preprocessing, through filtering, normalization, and feature selection, is therefore essential.

At the other end of the spectrum, excessively large and high-dimensional datasets may not be beneficial: as the number of features grows, the volume of the feature space increases exponentially, causing data to become sparse and distances between samples less meaningful. This sparsity reduces the effectiveness of methods based on similarity and distance metrics, and increases the risk of overfitting. These issues are typically mitigated using dimensionality reduction techniques.

A key consideration in ML system design is the selection of learning algorithms appropriate for the specific objectives and constraints of the problem. ML algorithms are typically categorized into three broad families: **supervised**, **unsupervised** and **reinforcement** learning. Supervised learning methods are further divided into two main types: **regression** and **classification**. In regression, the model outputs continuous numerical values, whereas in classification, it assigns inputs to discrete categories or classes. In both cases, models are trained using input-output pairs, where the objective is to learn a mapping from inputs to known target outputs. Training is often modeled as an optimization task, where an algorithm adjusts the model's internal parameters to minimize an error function, often referred to as the *loss* or *cost function*, which quantifies the discrepancy between the predicted outputs and the actual targets. Once training is complete, the model enters the inference phase, during which it is used to generate predictions on previously unseen data. Inference involves applying the learned model to new input instances to estimate the output.

In addition to trainable model parameters, most ML algorithms require *hyperparameters*, external settings that govern the learning process and must be defined before training. Hyperparameters, such as the learning rate,

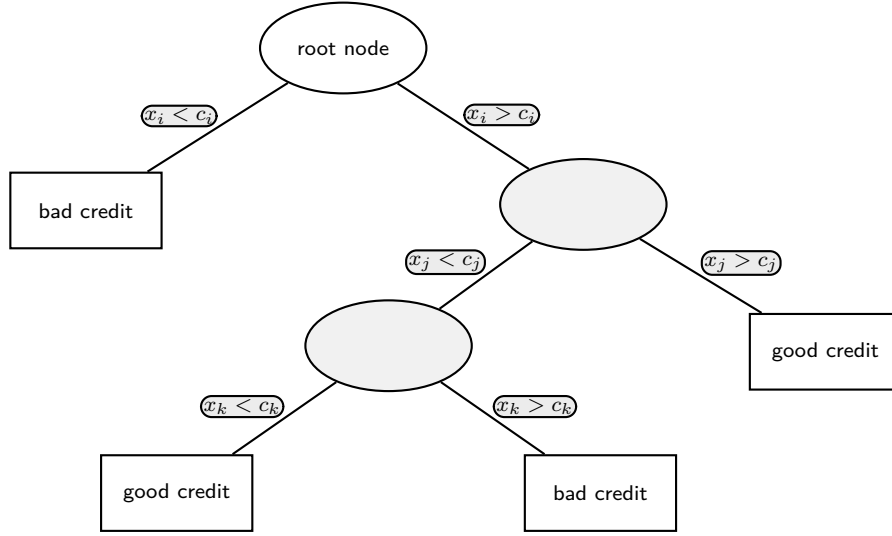


Figure 1.5: Decision tree example [18].

tree depth, number of clusters, or regularization strength, strongly influence convergence and generalization performance [116]. Selecting appropriate hyperparameters is critical, as suboptimal choices can lead to underfitting or overfitting. The process of determining the best configuration is known as *hyperparameter tuning* and is typically performed using techniques such as grid search or random search. These methods evaluate model performance across different hyperparameter combinations using a held-out validation dataset. This subset is excluded from training to prevent bias in the selection of the hyperparameters. The configuration that minimizes the cost function on the validation set, or equivalently maximizes predictive performance in terms of accuracy, is considered optimal. Finally, to obtain an unbiased estimate of model accuracy, a third partition of the dataset, the test set, is kept separate from both training and validation. After determining the optimal hyperparameters, the model is retrained on the combined training and validation data and evaluated on the test set to obtain the final accuracy.

A wide range of supervised and unsupervised learning algorithms have been developed over the years. *Linear regression* estimates a continuous outcome by fitting a straight line (or hyperplane) to the data. It models the target as a weighted sum of input features plus an intercept, minimizing the difference

between predicted and expected values.

Logistic regression is used for classification: it applies the same linear combination of features but passes it through a logistic (sigmoid) function to map the output to a probability between 0 and 1, with the final class determined by thresholding this probability, usually at 0.5.

Decision Trees (DTs) provide an intuitive model for classification and regression tasks (depicted in Figure 1.5): they recursively partition the input space based on feature values, forming a tree structure where each path from root to leaf represents a decision rule [56]. DTs are built by selecting, at each step, the feature and threshold that best separate the target values, continuing until stopping criteria are met. During inference, a new sample is routed through the tree according to its feature values, and the prediction corresponds to the majority class or average value of the samples in the reached leaf.

Random Forest (RF) extends DTs by constructing an ensemble of multiple trees [26]. Each tree in the forest is trained on a random subset of the training data (bootstrap sampling) and uses a random subset of features at each split. During inference, each tree produces an independent prediction, and the final output is obtained by aggregating their results, typically through majority voting for classification or averaging for regression. Although RFs increase computational and memory demands compared to single trees, they offer improved accuracy and stability. This ensemble strategy reduces variance and mitigates *overfitting*, a common limitation of individual DTs.

Artificial Neural Networks (ANNs) simulate interconnected neurons through multi-layered architectures with trainable weights and activation functions. During training, the network iteratively adjusts its parameters to reduce the difference between predicted and expected values. Inference is obtained by propagating the input through successive layers, where each neuron computes a weighted sum of its inputs, applies a nonlinear activation, and passes the result forward. The final layers activations yield the networks output, representing a class score, probability, or continuous value depending on the

task [200, 58].

Bayesian networks provide a probabilistic framework based on directed acyclic graphs, where nodes represent random variables and edges capture conditional dependencies. A common simplification is the Naive Bayes classifier, which assumes conditional independence among features given the class [57].

Support Vector Machines (SVMs) are a family of models where the objective is to compute the optimal hyperplane that maximizes the margin between classes in the feature space [39]. During inference, an SVM evaluates the distance of the input from the separating hyperplane; the sign determines the class, and the magnitude reflects confidence.

k-Nearest-Neighbors (kNN) assumes that similar instances are close in the input space and classifies a sample based on the majority label of its nearest neighbors, using a distance metric such as Euclidean or cosine similarity [190].

In contrast, unsupervised learning is applied when labeled data is unavailable and focuses on discovering patterns or structures in the input data alone. One of the primary tasks in this domain is clustering, which involves partitioning the dataset into distinct groups by maximizing intra-cluster similarity and minimizing inter-cluster similarity, often using metrics such as Euclidean distance [53]. Another technique is anomaly detection, which aims at identifying significant deviations from expected behavior or known patterns; these deviations can signal faults, outliers, or emerging conditions [107]. Additionally, unsupervised methods such as association rule mining can be employed to extract statistically relevant relationships among features, forming the basis for hypothesis generation and causal inference.

Finally, reinforcement learning focuses on teaching a system how to make decisions that maximize a numerical reward representing a long-term goal [171]. Unlike supervised learning, where the learner receives full feedback on its predictions, reinforcement learning provides only partial feedback about the outcomes of its actions.

Building on traditional methods, **Deep Learning** has become a key subset of

ML, particularly within ANNs. Deep learning relies on Deep Neural Networks (DNNs), which extend conventional ANN in depth and scale. The focus of this field is on optimizing and training large, multilayer models on massive datasets, often requiring distributed or parallel computation. DNNs consist of multiple hidden layers that apply nonlinear transformations, progressively extracting higher-level abstractions from the input data. Among the most common architectures are *Convolutional Neural Networks (CNNs)*, designed to process data structured as grids, such as images or time or frequency signals. CNNs are widely used for visual inspection, defect detection, surface analysis, and video analytics, owing to their capacity to capture spatial feature hierarchies [79, 73].

Another important family of deep learning models is *Recurrent Neural Networks (RNNs)*, designed to process sequential or temporal data. Unlike CNNs, which capture spatial correlations, RNNs model temporal dependencies within ordered data streams. Variants such as *Long Short-Term Memory Networks (LSTMs)* and *Gated Recurrent Units (GRUs)* are widely adopted in predictive maintenance and anomaly detection, as they can learn long-term temporal relationships and evolving patterns from time-series sensor data [108, 161].

More recently, *transformer* architectures have innovated deep learning by introducing attention mechanisms in place of recurrent or convolutional layers. Initially proposed for Natural Language Processing (NLP), transformers have since achieved significant results in several domains, including multivariate time-series forecasting. Attention relies primarily on matrix multiplications, which makes transformers highly compatible with parallel computation on modern hardware accelerators such as Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs), allowing those models to scale effectively on very large datasets [165].

Despite their effectiveness, deep learning models are computationally demanding and require large volumes of data, which are often impractical to handle in resource-constrained environments. To mitigate these limitations, techniques such as model quantization, pruning, knowledge distillation, and Neural Architecture Search (NAS) have been introduced to produce lightweight ar-

architectures suitable for edge deployment and on-device inference [95]. As a result, most CPS research applies deep learning primarily for inference, using pre-trained models that are adapted or fine-tuned offline before deployment. According to the taxonomy introduced in Section 1.2 and illustrated in Figure 1.4, such systems typically operate at collaboration Level 3 or lower.

1.3.1 Generative AI and Large Language Models

With Large Language Models (LLMs), recent literature refers to deep learning models trained on massive text corpora that learn to generate and manipulate natural language in a generalizable way. They are typically based on transformer architectures and comprise billions of parameters. LLMs constitute a core technology in modern NLP. These models perform a wide range of language tasks, such as translation, summarization, question answering, sentiment analysis, and dialogue generation, often without task-specific retraining [137]. They are typically composed of stacked encoder or decoder layers, depending on the model family, each integrating multi-head *attention* and feed-forward neural networks, along with residual connections and normalization layers. Input is tokenized and tokens are first embedded and combined with positional encodings, allowing the model to maintain information about sequence order. The multi-head attention mechanism enables the model to focus on different parts of the sequence simultaneously, improving their ability to understand contextual information. In encoder-decoder models, the decoder incorporates an additional cross-attention mechanism to focus on encoder outputs, enabling complex sequence-to-sequence generation tasks [183].

Notable examples of LLMs include *Generative Pre-trained Transformer (GPT)* [27], *Bidirectional Encoder Representations from Transformers (BERT)* [45], and *Text-to-Text Transfer Transformer (T5)* [135], each reflecting different modeling philosophies. For instance, *GPT-3* is a generative model trained with 175 billion parameters and used for a wide range of tasks through prompt-based learning. Its pretraining followed by optional fine-tuning allows

adaptation to specific domains. In contrast, BERT is a bidirectional model trained using masked language modeling and next sentence prediction, and it is especially effective for tasks involving understanding and classification. T5 unifies NLP tasks into a single text-to-text format, allowing the same model architecture to be reused across translation, summarization, and question generation.

Recent works have extended LLM by integrating *language*, *vision*, and in some cases *action planning*, enabling systems to reason across different modalities. These *multimodal* models process combinations of textual and visual inputs to perform complex tasks such as image classification, scene understanding, or robotic control based on verbal commands [189].

Training of LLMs generally involves two stages: large-scale **self-supervised**¹ pre-training on unlabeled text using objectives such as next-word prediction or masked token infilling, followed by supervised fine-tuning or instruction tuning for specific tasks [195]. This process enables the model to encode linguistic patterns and its semantics without manual annotation. Once trained, these models achieve state-of-the-art performance across various NLP tasks. However, this process requires massive computational resources, typically involving multiple high-performance GPUs or TPUs. This introduces limitations in terms of energy consumption, carbon footprint, and accessibility, often restricting training to large organizations. Additionally, the reliance on large-scale, web-sourced datasets raises concerns around bias, fairness, and data quality, as models may inherit and amplify patterns from uncured data sources.

A critical limitation of LLMs lies in their tendency to generate *hallucinations*, outputs that are syntactically plausible but factually incorrect. This issue arises from the probabilistic nature of next-token prediction and the absence of explicit grounding in verified knowledge sources. Furthermore, while LLMs can encode some degree of factual knowledge in their weights, they

¹Self-supervised learning is a training paradigm in which models learn from unlabeled data by generating supervision signals from the data itself, such as predicting masked or missing parts of the input.

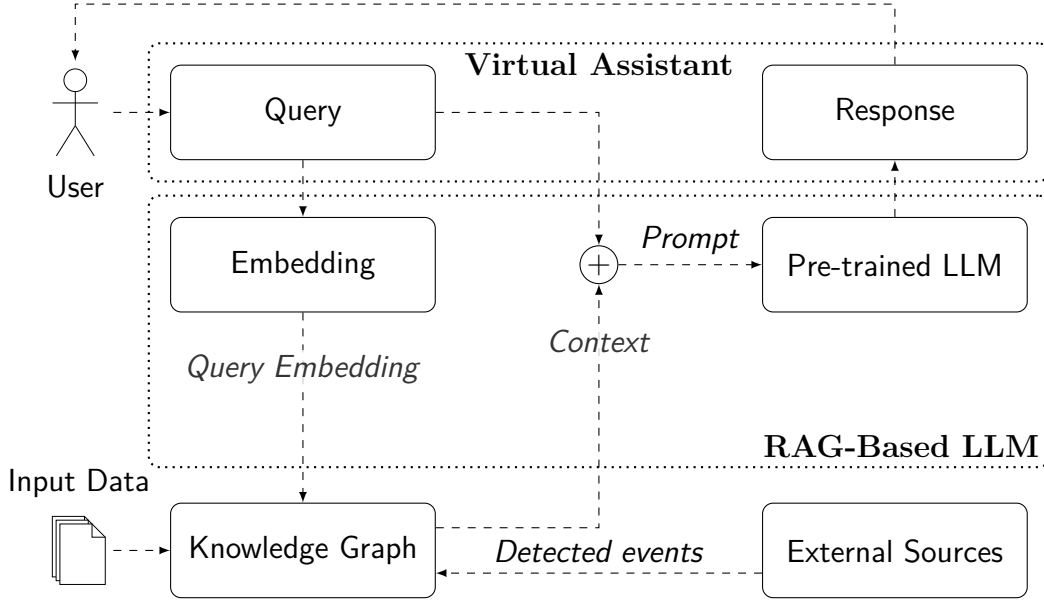


Figure 1.6: Retrieval augmented generation architecture [69].

lack a proper and updatable memory, which limits their ability to access or incorporate new knowledge after training.

A practical approach to mitigate these issues is Retrieval-Augmented Generation (RAG), illustrated in Figure 1.6. This approach is a prompt-engineering technique that complements the LLM with an external retrieval component that searches a KG or a document collection at inference time. The retrieved information is then provided as additional context input in the prompt to the LLM. This architecture enables the generation of responses with improved precision and traceability, thus reducing hallucinations [91].

1.4 Methods for eXplainable Artificial Intelligence

The importance of XAI has grown significantly with the increasing adoption of AI models in critical sectors. In these domains, the ability to interpret and understand the decisions produced by ML models is essential not only

to ensure transparency and reliability, but also to promote the trust in AI technologies by end users and stakeholders. In this context, explainability emerges as a key property, aiming to make AI systems not only effective but also understandable and accountable [147].

The main objectives of XAI can be summarized as follows [10]:

- **Transparency:** make the internal decision processes of AI models understandable, including the influence of input features and the transformations leading to an output. This involves both structural and procedural aspects of model behavior.
- **Verifiability:** support the evaluation of model outputs by making the reasoning behind decisions accessible and subject to external scrutiny.
- **User Acceptance:** favor trust and integration into human workflows by providing explanations that are aligned with users cognitive models and domain knowledge. Explanations serve as a bridge between algorithmic decisions and human interpretability.
- **Regulatory Compliance:** ensure that AI systems meet legal and ethical standards, including requirements for transparency, fairness, and accountability.
- **Model Diagnosis and Improvement:** assist developers in identifying errors, biases, or unintended feature dependencies.

These objectives position explainability as a foundational requirement for the deployment of AI in domains where traceability, accountability, and human oversight are essential. In healthcare, XAI supports clinical decision-making by clarifying how features like biomarkers or test results influence diagnoses [99]. In finance, it enables transparency in credit scoring by justifying decisions based on variables such as credit history or income [35]. In legal contexts, it helps interpret predictions by linking outcomes to legal precedents and evidence [188]. In cybersecurity, it allows analysts to understand threat detection

by exposing the factors behind flagged anomalies [31]. In autonomous driving, XAI explains system behavior in critical scenarios, such as obstacle avoidance, based on sensor data, supporting both safety and post-event analysis [105].

A central concept in this discussion is the degree of *model transparency*. ML models can be categorized into three broad classes, *white-box*, *gray-box*, and *black-box*, depending on how easily a human can interpret their internal logic.

- **Black-box models** are highly complex and opaque, with decision processes that are not readily accessible to human interpretation. Despite offering high predictive performance, they often lack transparency. Typical examples include DNNs, which can model intricate, non-linear relationships but provide little insight into how individual prediction outcomes are obtained.
- **Gray-box models** offer a partial degree of interpretability. Some components of the model are transparent and understandable, while others remain opaque. For instance, ensemble methods like random forests or hybrid architectures combining interpretable and non-interpretable components may fall into this category.
- **White-box models** are fully transparent: their structure and decision logic are directly accessible and understandable by humans. Examples include simple decision trees and linear models, where the contribution of each feature to the outcome can be explicitly traced.

It is important to note that a trade-off often exists between model interpretability and predictive performance. Highly accurate models such as DNNs are often black-box by nature, due to their depth and complexity. Although they are capable of capturing nuanced and non-linear interactions among features, their internal representations generally are not human-readable. This lack of transparency can limit their adoption in domains where explanations are required and accountability is critical [4, 192].

XAI research aims to bridge this gap by developing techniques that either make inherently interpretable more expressive models or provide *post hoc*

explanations for black-box models. These approaches are further discussed in the next sections.

1.4.1 A Taxonomy of Explainability Methods in Artificial Intelligence

Explainability in AI can be characterized through several complementary dimensions, each offering a distinct perspective on how to make the behavior of an AI model more understandable to humans [10].

- **Model-Agnostic vs Model-Specific Methods:**
 - *Model-agnostic* techniques are independent of the internal structure of the model. They treat the model as a black box and focus on learning the relationship between inputs and outputs to generate explanations. These methods are widely applicable and versatile.
 - *Model-specific* techniques, by contrast, leverage knowledge of the model's internal components. They are specialized to particular model architectures and can often produce more precise explanations. Examples include saliency maps for DNNs or feature importances derived from decision trees.
- **Local vs Global Explainability:**
 - *Local explainability* refers to techniques that explain individual predictions.
 - *Global explainability* aims to describe the model's overall behavior across the entire dataset.
- **Post hoc vs Intrinsically Interpretable Models:**
 - *Post hoc* explanations are applied after the model has been trained. These techniques do not alter the model itself but provide auxiliary tools to interpret its decisions retrospectively.

- *Intrinsically interpretable models* are designed to be transparent by construction. Their internal logic is accessible and understandable without additional tools. Examples include linear models, rule-based systems, and shallow decision trees.

- **Target User Perspective:**

- *Domain experts* require technical and detailed explanations of the underlying models, often involving their mathematical formulations. These methods are often used for model validation and debugging.
- *Non-expert users* need simplified, intuitive explanations. These may include natural language summaries, simple charts, or analogies. For example, in a credit-scoring application, a graphical display of how income and credit history influenced the decision would enhance user understanding.

- **Temporal Dimension:**

- *Static explanations* are suitable for models that do not evolve over time. Once generated, the explanation remains valid unless the model is retrained. Examples include fixed feature importance scores or decision rules.
- *Dynamic explanations* apply to models that change over time, such as in online or continual learning. In these cases, explanations must be updated accordingly. For instance, local approximations or surrogate models may need to be recomputed after each update.

This taxonomy serves as a conceptual framework to classify and compare XAI methods based on their technical characteristics and intended use case. It is particularly relevant for pervasive systems, where different layers (from edge to cloud) and actors (from developers to end users) may require different levels and types of explanation.

In addition to the dimensions presented earlier, a complementary taxonomy

classifies explanation types not only by the methodology but also according to their *informational goals*, *reasoning strategies*, and *user context* [34].

In particular, explanation types include:

- **Contrastive**: explain why one outcome occurred instead of another by comparing alternative system recommendations and the distinguishing input features.
- **Counterfactual**: explore how a different outcome could have occurred under a hypothetical change in input conditions.
- **Trace-Based**: describe the internal steps or reasoning process that led to the outcome.
- **Statistical**: justify decisions using aggregate data, distributions, or likelihoods.
- **Case-Based**: refer to past, similar cases as analogical support for the current decision.
- **Contextual**: include information external to the input data, such as user situation or environmental conditions.
- **Scientific**: reference empirical studies, clinical guidelines, or domain-specific models as the basis of the recommendation.
- **Simulation-Based**: project the effects of actions or decisions through forward simulations.
- **Everyday**: Provide simple, common-sense justifications aligned with the users intuitive understanding.

Each explanation type corresponds to a category of user questions (e.g., “Why this and not that?”, “What if...?”, “What usually happens?”) and is associated with specific reasoning strategies (*e.g.*, inductive, deductive, abductive) and AI tasks. The framework also distinguishes between knowledge sources (*e.g.*,

contextual, scientific, experiential) and user preferences (e.g., modality, level of detail). By explicitly modeling these relationships, the work provides a common taxonomy to describe and characterize XAI systems that must serve diverse users with varying levels of expertise and goals.

1.4.2 Approaches to Explainable Artificial Intelligence

The literature on XAI involves a broad range of techniques aimed at making ML models understandable to human stakeholders.

Model-specific approaches leverage the knowledge of a model's internal structure, such as its parameters, topology, or optimization criteria. These techniques are typically applicable to inherently interpretable models, or to complex models whose internal logic can be partially analyzed.

A canonical example of a model with built-in interpretability is the **decision tree**, which expresses its decision process as a hierarchical structure of conditional rules, as shown in Figure 1.5. Each internal node corresponds to a decision based on a single input feature, and the leaves represent predicted outputs. The resulting if-then rules form a transparent path that can be easily followed to understand how a specific prediction was obtained.

Similarly, **linear models**, such as linear regression and logistic regression, offer a high degree of interpretability due to their mathematical formulation. Predictions are generated as a weighted sum of input features, where each coefficient reflects the influence of a corresponding feature on the output. The sign and magnitude of each coefficient directly indicate the direction and strength of influence, although care must be taken to consider feature scaling, as coefficients are sensitive to the numerical range of the inputs.

Another family of interpretable models comprises rule-based systems, which encode knowledge in the form of logical if-then rules. These models are particularly relevant in domains that demand traceable, symbolic reasoning and are often favored in regulatory or compliance-heavy contexts.

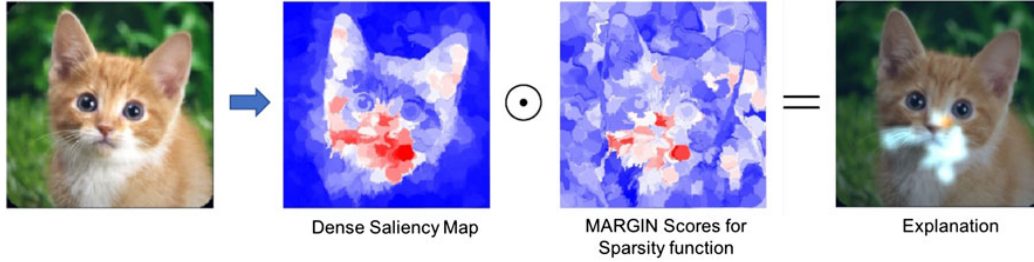


Figure 1.7: The MARGIN XAI method [8]

In contrast, SVMs are less transparent by design, yet their structure still permits certain forms of interpretation. For linear SVMs, the coefficients of the separating hyperplane can be analyzed to determine the most influential features. Moreover, the concept of support vectors, the training instances that lie closest to the decision boundary, provides insight into how the model distinguishes between classes. For non-linear SVMs, interpretability can be partially restored through approximation methods such as local linear surrogates, which fall under the *model-agnostic* family of methods.

The case of neural networks presents greater challenges due to their complexity. Nonetheless, several techniques have been developed to enhance their interpretability. Activation-based methods examine how neurons respond to specific inputs, revealing internal patterns in the network. In the context of image classification, saliency maps are used to highlight regions of an input image that contribute most significantly to the networks decision. Another method, Layer-wise Relevance Propagation (LRP) [112], propagates a relevance score backward from the output to the input layer, assigning to each feature a quantitative measure of its contribution to the final prediction. A further example is MARGIN [8], which combines model outputs with graph signal processing to produce sparse and interpretable saliency maps. It segments the image into “superpixels”, evaluates their impact on the prediction, and refines the results using a graph-based influence score. This produces a sparse saliency map that highlights the most relevant image regions contributing to the models prediction, such as a cats mouth and whiskers in Figure 1.7.

Random forests, which aggregate the output of multiple decision trees, can be interpreted by analyzing the contribution of individual features in the ensemble. A common metric is *feature importance*, computed based on the average decrease in impurity (e.g., Gini index) across all trees in which a feature is used. Additionally, local explanations can be obtained by tracing the decision paths followed by individual trees for a given input and examining their consensus.

Model-agnostic methods represent a flexible and general approach to XAI, as they enable the interpretation of AI models without relying on knowledge of their internal structure. These techniques treat the model as a black box and analyze the relationship between inputs and outputs to derive explanations. Among the most widely adopted model-agnostic methods is *Local Interpretable Model-agnostic Explanation (LIME)* [138], which approximates the behavior of the model in a local neighborhood of a prediction by fitting a simple interpretable model, such as a linear regression, to a synthetic dataset generated by perturbing the input of interest. The resulting explanation is derived from the coefficients of the surrogate model, which indicate the importance of each feature in the decision. Another prominent method is *SHapley Additive exPlanations (SHAP)* [104], which is grounded in cooperative game theory. SHAP assigns each feature a contribution value based on its marginal effect on the prediction across all possible subsets of features. SHAP values can be used to visualize both the magnitude and direction of feature contributions.

Partial Dependence Plots (PDPs) [61] offer a global view by showing how the average model prediction varies w.r.t. a single feature, keeping all others fixed. This helps identify general trends, such as whether an increase in a feature leads to higher or lower predictions. However, PDPs assume feature independence and may be misleading in the presence of strong interactions between features, such as in the case of time series forecasting. To address this limitation, Individual Conditional Expectation (ICE) plots [61] extend PDPs by displaying the effect of a feature on the prediction for individual instances. Each line in an ICE plot represents a single observation, revealing heterogeneous effects and outliers otherwise hidden in a PDP.

Another widely used technique is *Permutation Feature Importance* [113], which measures the decrease in model performance when the values of a specific feature are randomly shuffled. A significant drop in accuracy suggests that the feature is important for the model’s predictions. This method is straightforward to implement and applicable to any model, but care must be taken when interpreting results in the presence of correlated features. Lastly, *Surrogate Models* [32] offer a global approximation of a black-box model by training an interpretable model, such as a decision tree or linear model, on the outputs of the original complex model. The surrogate is expected to mimic the predictions of the black-box model while maintaining interpretability. The quality of the explanation depends on the fidelity between the surrogate and the original model. A well-fitted surrogate can reveal general decision logic while remaining interpretable by human users.

1.5 Knowledge Representation and Reasoning for explainability

KRR is the branch of AI concerned with designing formalisms for representing knowledge within a given domain. Through ontologies and logical reasoning, KRR improves interoperability across heterogeneous entities, supports reasoning tasks to derive implicit knowledge, and enables traceable, interpretable data analysis.

As Section 1.7 will demonstrate, these technologies can be integrated across the entire 5C architecture stack of CPSs (see Section 1.1), enhancing network and communication management, data and metadata representation, user interfaces, and decision support functionalities.

Among the various KRR approaches, the following subsections provide an overview of Description Logics (DL)-based languages and reasoning services developed within the Semantic Web community, which underpin the languages, domain modeling, and inference mechanisms used in this thesis to implement

an explainable ML model for resource-constrained environments.

1.5.1 Description Logics

One of the most important logics in the context of automated reasoning is First Order Logic (FOL), which provides an expressive formalism for representing knowledge and statements about a given domain. However, FOL is undecidable in its general form. To overcome this limitation, DLs are designed to be decidable subsets of FOL. DLs represent domain knowledge through *concepts* (also called *classes*, representing collections of objects), *individuals* (instances of classes), and *roles* (also called *properties*, which define relationships between pairs of individuals)[15].

In DL, the formal semantics of expressions is defined through an interpretation function, typically denoted as $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ which maps each symbol to elements or subsets of the domain Δ .

Under this interpretation:

- each **concept** C is interpreted as a subset of the domain: $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,
- each **role** R is interpreted as a binary relation over the domain: $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$,
- each **individual** a is interpreted as a single element in the domain: $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

These components combine through a set of constructors to form complex DL expressions. For instance, the conjunction of two concepts is interpreted as set intersection: $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$; the logical or is interpreted as set union: $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$; and negation as set complement: $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$. The interpretation of universal restriction $\forall R.C$ includes all individuals $x \in \Delta^{\mathcal{I}}$ such that for every $y \in \Delta^{\mathcal{I}}$, if $(x, y) \in R^{\mathcal{I}}$, then $y \in C^{\mathcal{I}}$; formally:

$$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}, (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\} \quad (1.1)$$

Conversely, existential quantification $\exists R.C$ includes all individuals $x \in \Delta^{\mathcal{I}}$ for which there exists at least one related individual $y \in \Delta^{\mathcal{I}}$ such that $(x, y) \in R^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}$; i.e.:

$$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}, (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \quad (1.2)$$

DLs represent knowledge using two components: the *TBox* ($\mathcal{T}$), a.k.a. *ontology*, which defines the conceptual schema or vocabulary through axioms describing class hierarchies and property characteristics (*e.g.*, subclass and equivalence relations), and the *ABox* ($\mathcal{A}$), which contains assertions about individual instances. Together, they form a Knowledge Base (KB), denoted as $\langle \mathcal{T}, \mathcal{A} \rangle$ and often materialized on the Web as a KG.

Each DL is characterized by its set of constructors and operators, which determine its expressiveness. Naming conventions indicate available features; for example, $\mathcal{AL}$ is the base language, while extensions such as $\mathcal{C}$, $\mathcal{N}$, $\mathcal{H}$, $\mathcal{I}$, and $\mathcal{O}$ add full negation, number restrictions, role hierarchies, inverse roles, and nominals, respectively. More expressive logics can represent more complex knowledge but increase reasoning complexity, whereas lightweight DLs trade expressiveness for scalability. The appropriate choice depends on the required balance between modeling precision and reasoning efficiency. The complete naming convention is presented in Table 1.1. This thesis adopts the $\mathcal{ALN}$ logic, an extension of the base $\mathcal{AL}$ language with unqualified number restrictions. The main constructs supported by $\mathcal{ALN}$ are summarized in Table 1.2 and listed below:

- $\top$, *universal concept*. The set of all individuals in the domain.
- $\perp$, *bottom concept*. The empty set.
- A , *atomic concept*. The interpretation of a named class $A^{\mathcal{I}}$.
- $\neg A$, *atomic negation*. The complement of $A^{\mathcal{I}}$.
- $C \sqcap D$, *conjunction*. The intersection of $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$.

Table 1.1: Naming conventions for Description Logics

Symbol	Meaning
Base Logics	
$\mathcal{AL}$	<i>Attributive Language</i> : supports atomic negation (only on the right-hand side of axioms), concept intersection, universal restrictions, and limited existential quantification.
$\mathcal{FL}$	<i>Frame-based Language</i> : includes concept intersection, universal restrictions, limited existential quantification, and role restriction.
$\mathcal{EL}$	<i>Existential Language</i> : includes concept intersection and existential restrictions (full existential quantification).
Extensions	
$\mathcal{F}$	Functional properties (uniqueness constraints).
$\mathcal{E}$	Full existential qualification (existentials with fillers other than $\top$).
$\mathcal{U}$	Concept union.
$\mathcal{C}$	Complex concept negation.
$\mathcal{H}$	Role hierarchies (e.g., <code>subPropertyOf</code>).
$\mathcal{R}$	Limited role inclusion axioms; includes reflexivity, irreflexivity, and role disjointness.
$\mathcal{O}$	Nominals (enumerated classes and value restrictions).
$\mathcal{I}$	Inverse properties.
$\mathcal{N}$	Unqualified cardinality restrictions (e.g., min/max cardinality).
$\mathcal{Q}$	Qualified cardinality restrictions (with fillers other than $\top$).
$(\mathcal{D})$	Use of datatypes, data values, or datatype properties.
Exceptions and Common Combinations	
$\mathcal{S}$	Abbreviation for $\mathcal{ALC}$ with transitive roles.
$\mathcal{FL}^-$	Sub-language of $\mathcal{FL}$ without role restriction; equivalent to $\mathcal{AL}$ without atomic negation.
$\mathcal{FL}_i$	Further restriction of $\mathcal{FL}^-$, disallowing existential quantification.
$\mathcal{EL}^{++}$	Alias for $\mathcal{ELRO}$.

- $\forall R.C$, *universal restriction*. The set of individuals related through R only to elements of $C^{\mathcal{I}}$.
- $(\geq nR)$, $(\leq nR)$, *unqualified number restrictions*. Sets of individuals having at least or at most n R -successors.
- $C \sqsubseteq D$, *concept inclusion*. Assert that every instance of C is also an instance of D ($C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$).
- $C \equiv D$, *concept equivalence*. Assert that C and D denote the same set of individuals ($C^{\mathcal{I}} = D^{\mathcal{I}}$).

Table 1.2: Syntax and semantics of $\mathcal{ALN}$ constructors and axiom types.

Name	Syntax	Semantics
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\perp$	$\emptyset$
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Atomic negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
Universal quantification	$\forall R.C$	$\{d_1 \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow d_2 \in C^{\mathcal{I}}\}$
Number restrictions	$\geq nR$	$\{d_1 \mid \#\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \geq n\}$
	$\leq nR$	$\{d_1 \mid \#\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \leq n\}$
Inclusion	$A \sqsubseteq D$	$A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Equivalence	$A \equiv D$	$A^{\mathcal{I}} = D^{\mathcal{I}}$

1.5.2 Web Ontology Language

The primary World Wide Web Consortium (W3C) recommendation for knowledge representation in the Semantic Web is OWL 2 [129], which implements DLs on top of the Resource Description Framework (RDF) [64] data model. Entities, such as classes, individuals, and properties, are identified through Internationalized Resource Identifiers (IRIs)², which provide globally unique identifiers on the Web.

The initial version of OWL is subdivided into three progressively expressive sub-languages: *OWL Lite*, supporting simple taxonomies and basic constraints; *OWL Full*, providing maximum expressiveness at the cost of computational undecidability; and *OWL DL*, offering a balance between the two. OWL Lite is a subset of OWL DL, which in turn is contained within OWL Full.

²IRIs extend the traditional concept of Uniform Resource Identifiers (URIs) by supporting a wider range of character sets.

Table 1.3: Mapping between OWL RDF/XML and DL syntax.

OWL RDF/XML syntax	DL syntax
<code><owl:Thing></code>	$\top$
<code><owl:Nothing></code>	$\perp$
<code><owl:Class rdf:ID="C"></code>	C
<code><owl:ObjectProperty rdf:ID="R"></code>	R
<code><rdfs:subClassOf></code>	$\sqsubseteq$
<code><owl:equivalentClass></code>	$\equiv$
<code><owl:disjointWith></code>	$\sqcap$
<code><owl:intersectionOf></code>	$\sqcap$
<code><owl:allValuesFrom></code>	$\forall$
<code><owl:maxCardinality></code>	$\leq$
<code><owl:minCardinality></code>	$\geq$

The subsequent version, OWL 2 [129], extends the original language by increasing expressiveness, broadening datatype and annotation support, and introducing multiple syntaxes for ontology representation. Ontologies can be serialized using any RDF syntax, such as *RDF/XML* [152] or *Turtle* [133]. Additional OWL-specific syntaxes, including the *Functional Syntax* [128] and *OWL/XML* [115], encode OWL axioms directly, bypassing RDF triples. The *Manchester Syntax* [66] provides a more human-readable alternative. The full language, OWL DL, implements the $\mathcal{SHOIN}(\mathcal{D})$ description logic, which combines transitive roles, role hierarchies, nominals, inverse properties, number restrictions, and datatypes.

As with the first version, OWL 2 defines multiple *profiles*, restrictions over the full language, designed to address specific application requirements. *OWL EL* (Existential Language) enables tractable reasoning with conjunction and existential restriction, omitting disjunction, negation, universal quantification, and cardinality constraints to ensure polynomial-time classification. It is frequently used for large KGs with simple relationships, as found *e.g.*, in life sciences domains. *OWL RL* (Rule Language) translates OWL axioms into Horn rules for forward-chaining rule-based inference with polynomial data complexity but reduced expressiveness, excluding non-Horn constructs. *OWL QL* (Query Language) supports Structured Query Language (SQL)-like, first-order reasoning while restricting role composition and complex property

chains. Unlike the original OWL sublanguages, the OWL 2 profiles are independent, allowing systems to selectively implement subsets of constructs to achieve the desired expressiveness for the specific use case. The elements of *OWL DL* that correspond to the $\mathcal{ALN}$ DL are presented in Table 1.3 using the *RDF/XML* syntax.

1.5.3 Reasoning Services

Several automated reasoning algorithms are discussed in the literature. *Structural* algorithms simplify logical expressions through normalization and decomposition, achieving high efficiency and typically polynomial-time performance; however, they are incomplete for expressive description logics and may fail to derive all valid inferences. *Tableaux-based* algorithms construct completion trees to test concept satisfiability and knowledge base consistency, offering high expressivity but exhibiting exponential or double-exponential worst-case complexity. *Rule-based algorithms* apply inference rules, such as forward or backward chaining, to derive implicit facts; they are computationally efficient and suitable for large data sets but limited in handling non-monotonic reasoning, where previously derived conclusions may need to be retracted.

Traditional Semantic Web systems assume reasoning is performed on high-performance nodes. In contrast, as will be discussed in Section 1.6, the SWoE and CPSs operate across distributed, resource-constrained devices with limited memory, energy, and communication bandwidth. In these environments, lightweight autonomous agents must support on-board real-time reasoning, query processing, and stream interpretation [52, 93].

Early examples of DL reasoners for mobile and ubiquitous contexts include *3APL-M* [81], *COROR* [172], *MIRE4OWL* [80], and the *Delta-Reasoner* [114]. *Pocket KRHyper* [162] introduced a tableaux-based reasoner founded on the hyper tableau calculus. Subsequent optimizations, such as *mTableaux* [164], improved memory efficiency by refining tableau construction and adapting the *Pellet* reasoner [163].

Efforts to adapt Semantic Web reasoners for mobile and embedded environments include *CB* [77], *Hermit* [60], and *JFact* [201] (a Java implementation of *Fact++* [178]), along with optimized variants of *ELK* [78] and the RETEpool algorithm [181], which provides an efficient rule engine for OWL RL.

The proposed approaches in this dissertation leverage *Tiny-ME* [141], a lightweight, multiplatform matchmaking and reasoning engine designed with SWoE constraints in mind. It supports an extended range of standard and non-standard inference services in the $\mathcal{ALN}(\mathcal{D})$ DL. The standard reasoning services implemented in Tiny-ME are:

- **Concept Subsumption**, which verifies whether a concept A is more specific than another concept B (i.e., $\mathcal{T} \models A \sqsubseteq B$), meaning that all instances of A are also instances of B .
- **Concept Satisfiability**, which determines whether a concept A is logically consistent (i.e., $\mathcal{T} \models A \not\sqsubseteq \perp$) and therefore can have instances.
- **Ontology Consistency**, which checks whether the ontology $\mathcal{T}$ is free from logical contradictions.

These services are implemented through *structural reasoning algorithms*, which operate in polynomial time for ontologies expressed in $\mathcal{ALN}(\mathcal{D})$. Tiny-ME also supports several non-standard reasoning tasks, summarized below:

- **Concept Contraction** is applied when the conjunction $R \sqcap S$ of two concepts is unsatisfiable. This task computes a pair $\langle G, K \rangle$ such that $\mathcal{T} \models R \equiv G \sqcap K$, where G represents the conflicting component of R to be discarded, and K is the consistent remainder.
- **Concept Abduction** is applied when $R \sqcap S$ is satisfiable but $S \not\sqsubseteq R$. It computes a concept H such that $\mathcal{T} \models S \sqcap H \sqsubseteq R$. The hypothesis H represents the additional information that, if assumed, would make S a valid match for R .

- **Concept Bonus** identifies properties in the resource S that are not required by the request R but may still be relevant. It is computed as a reversed abduction problem, where the roles of R and S are swapped.
- **Concept Difference** defines a way to subtract information in a concept description from another one: in the original definition by Teege [174], if $\mathcal{T} \models R \sqsubseteq S$, then the output of difference $D = CD(R, S)$ is a concept such that $R \equiv S \sqcap D$. Tiny-ME extends CD from the subsumption case to include approximate matches as well, by exploiting CC and CB.
- **Concept Covering** (CCov) non-standard inference for many-to-one composition of a set of elementary instances to answer complex requests. It takes a set of resources and a request and aims to (i) cover (*i.e.*, satisfy) features expressed in the request as much as possible through the conjunction of resources, and (ii) provide an explanation of the possibly uncovered part, by leveraging abduction. In formulae, given a request R and a set of resources $\mathcal{S} = \{S_1, S_2, \dots, S_n\}$, the output of CCov is a pair $\langle \mathcal{S}_c, H \rangle$ where $\mathcal{S}_c \subseteq \mathcal{S}$ contains resources in S forming a (possibly incomplete) covering of R w.r.t. $\mathcal{T}$ and concept H is the (possible) part of R not covered by elements in $\mathcal{S}_c$.

In the general case, these reasoning tasks are non-commutative. All operate under the Open World Assumption (OWA), where missing information is treated as unknown rather than false. These algorithms form the foundation for the work presented in Chapter 3.

Non-standard reasoning tasks address the *Semantic Matchmaking* problem, which seeks to identify the most suitable supply S for a given request R . For each pair $\langle R, S \rangle$, the three reasoning tasks can be composed, as illustrated in Figure 1.8, to compute a penalty or score that enables the ranking of candidate resources and the selection of the one with the lowest penalty (or highest score). This value is derived from the Conjunctive Normal Form (CNF) norm of the concepts G and H [140]. At the same time, G and H returned by Concept Contraction and Concept Abduction act as formal logic-based justifications of unsatisfiability and missed subsumption, respectively: this

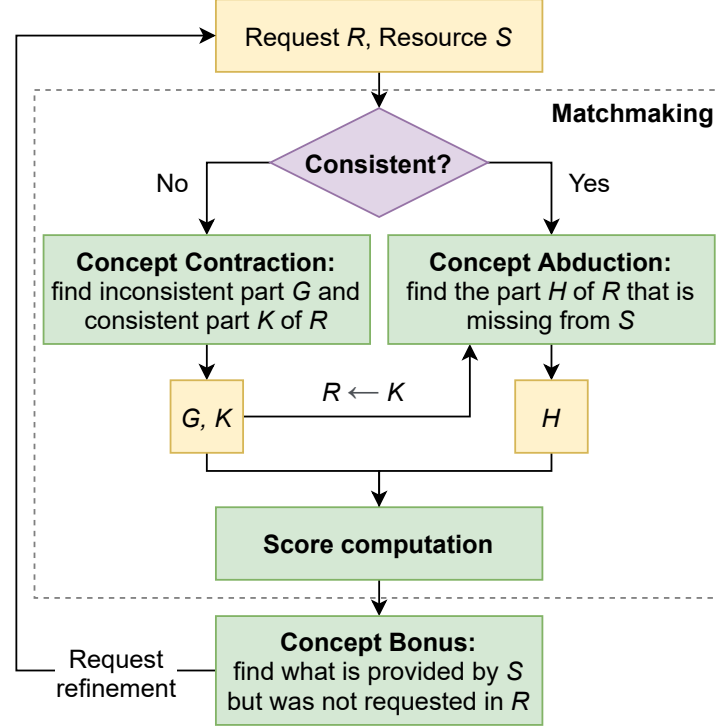


Figure 1.8: Semantic matchmaking framework

lays the groundwork for explainability in XAI frameworks leveraging these non-monotonic inference services in semantic matchmaking.

1.6 State of the Art in Distributed Architectures for Cyber-Physical Systems

Given that CEI allows to solve some issues in decentralized architectures, as discussed in Section 1.2.2, recent research has explored its integration across multiple domains, including CPSs [30, 102], healthcare [169], and intelligent transportation systems [12]. Still, there exists several challenges that must be addressed:

- **Latency.** Minimizing latency is a core objective, yet achieving ultra-low communication delay between edge and cloud remains difficult in prac-

tice due to fluctuating network conditions and constrained processing capabilities of some edge devices.

- **Scalability.** As the number of connected devices and applications grows, edge infrastructures must dynamically scale to meet increasing computational demands while preserving performance and reliability.
- **Data management.** Edge and IoT devices produce large data streams: appropriate strategies must be established to decide where information should be processed and stored.
- **Complexity.** Effectively managing multiple heterogeneous devices, disseminated throughout the environment and potentially distributed geographically, introduces major challenges in orchestrating and coordinating use-case tasks compared to traditional centralized approaches.

Serverless computing represents a promising paradigm. Compared with traditional on-premise approaches, serverless computing [156] aims to provide function-execution environments in which both infrastructure and underlying software-platform details are abstracted from customers. In this way, users can invoke the desired application functionality while paying only for the resources they actually consume. The invocation of the functions is delegated to one of the available computation nodes (*e.g.*, cloud containers, decentralized edge environments, or specific IoT devices) and the obtained results are sent back to the user. The fusion of serverless architecture with the Cloud-to-Edge continuum holds great potential in the field of Federated Learning (FL) [92]. By deploying serverless functions on the most suitable devices, initial model training can be performed at the network edge near data sources, while resource-intensive tasks such as model aggregation and global updates are executed on more capable computing instances.

A basic architecture for the orchestration of containerized microservices and CEI was proposed in [102]. It is based on Osmotic Computing principles [186] allowing data mining with predictive ML models trained and executed on the edge and in the cloud, exploiting computational devices opportunistically to

reach the best prediction accuracy at the lowest operational cost in terms of resource consumption. However, the proposed work did not fully utilize IoT devices for either training or inference. Additionally, the architecture did not employ FL approaches, as the data needed to be sent to the cloud for processing.

In [176], a novel cloud-edge AI framework and architecture was presented to enhance ML efficiency, focusing on reducing transmission latency and bandwidth consumption. The framework efficiently manages task allocation and data processing through container orchestration. It employs a specialized BranchyNet DNN model with early-exit branches for rapid inference on edge devices, optimizing response times and system load. Nevertheless, IoT devices within this framework are used only as data stream sources, being able to run neither training nor inference tasks.

FLoX [83] is a FL framework aiming to train and deploy neural network models over heterogeneous and distributed computing resources. It is built on the *funcX* [33] federated serverless computing platform to decouple FL model training/inference from infrastructure management. In this way, users can easily deploy models on different network devices. The approach, however, does not include training or inference on IoT devices.

The work in [130] also exploits *funcX* for defining *Rural AI*, a prototype system based on a federated Function as a Service (FaaS) architecture aiming to demonstrate the capabilities serverless computing offers to traditional FL in rural precision agriculture scenarios, often characterized by limited and unreliable networking infrastructures.

Similar approaches are also in *FedLess* [62], a serverless framework for training and using DNN models on heterogeneous FaaS platforms, and in the Serverless Hierarchical Federated Learning (SHFL) framework [123], which adopts a two-layer FL architecture where nodes are grouped into clusters under cluster heads exchanging the local model parameters among a neighborhood of workers.

Despite the recent advancements, the literature continues to fall short in addressing two critical and interrelated challenges: (i) the extension of serverless FL across the entire CloudEdge IoT continuum, and (ii) the realization of full on-device intelligence, corresponding to level 6 of the taxonomy in Figure 1.4. While many existing serverless FL frameworks integrate cloud and edge components, they do not actively use IoT devices, which are typically limited to passive data acquisition roles. In particular, the execution of training and inference tasks on MCU remains an open issue, primarily due to their limited computational resources and the scarcity of ML algorithms specifically optimized for these devices. Furthermore, the integration of IoT devices into existing distributed architectures is hindered by substantial heterogeneity in execution environments, platform-specific dependencies, and the lack of standardized packaging and deployment formats. This fragmentation significantly complicates the development and deployment of cross-layer applications. Finally, although real-world FL deployments increasingly demand hierarchical aggregation schemes, most existing frameworks adopt flat, centralized aggregation models. The use of intermediate representations instead of raw data or full model parameters, which could reduce communication overhead and enhance privacy, is still rarely supported.

1.7 Semantic-based Enhancements of Pervasive Cyber-Physical Systems

The increase in computational power and the reduction in the cost of electronic micro-components have enabled the embedding of computational capabilities into everyday objects. This technological progress has contributed to the realization of Mark Weiser’s seminal vision of Pervasive Computing, which aims for transparent and unobtrusive integration of computing into the environment [191]. In parallel, these advancements have driven the development of distributed AI and Edge computing architectures, key enablers of the broader paradigm known as the Internet of Everything [54].

The IoE extends traditional IoT systems by incorporating not only devices and data sources, but also living beings, human actors and organizational processes into unified, distributed information systems. This evolution introduces new challenges and opportunities for the design and deployment of CPSs, which increasingly operate across heterogeneous platforms and dynamic network environments.

In this context, semantic technologies offer a formal foundation for modeling, integrating, and reasoning over the heterogeneous data formats, real-time streams, and contextual metadata that define IoE-enabled CPSs. This section describes the main strategies for implementing these technologies across the CPS stack.

1.7.1 Connection Layer: Semantic-Oriented Infrastructure and Connectivity Management

Distributed network infrastructures in hybrid cloud and IoE environments place significant demands on configurability, responsiveness, and resource scalability. Traditional configuration models often fall short in supporting the adaptability required by CPS applications operating across heterogeneous and dynamic networks.

The Software-Defined Networking (SDN) paradigm separates the control and data planes, allowing a centralized software controller to configure forwarding behavior across distributed devices via programmable interfaces. This allows centralized definition and distributed enforcement of traffic management and security policies. Combined with Network Function Virtualization (NFV), which enables the virtualization of network functions (*e.g.*, routing, firewalls, load balancing), SDN separates the control logic from these functions, allowing them to be deployed as software components on general-purpose hardware [49].

Building upon this foundation, the Knowledge-Defined Networking (KDN) paradigm introduces a knowledge-centric control model that combines SDN,

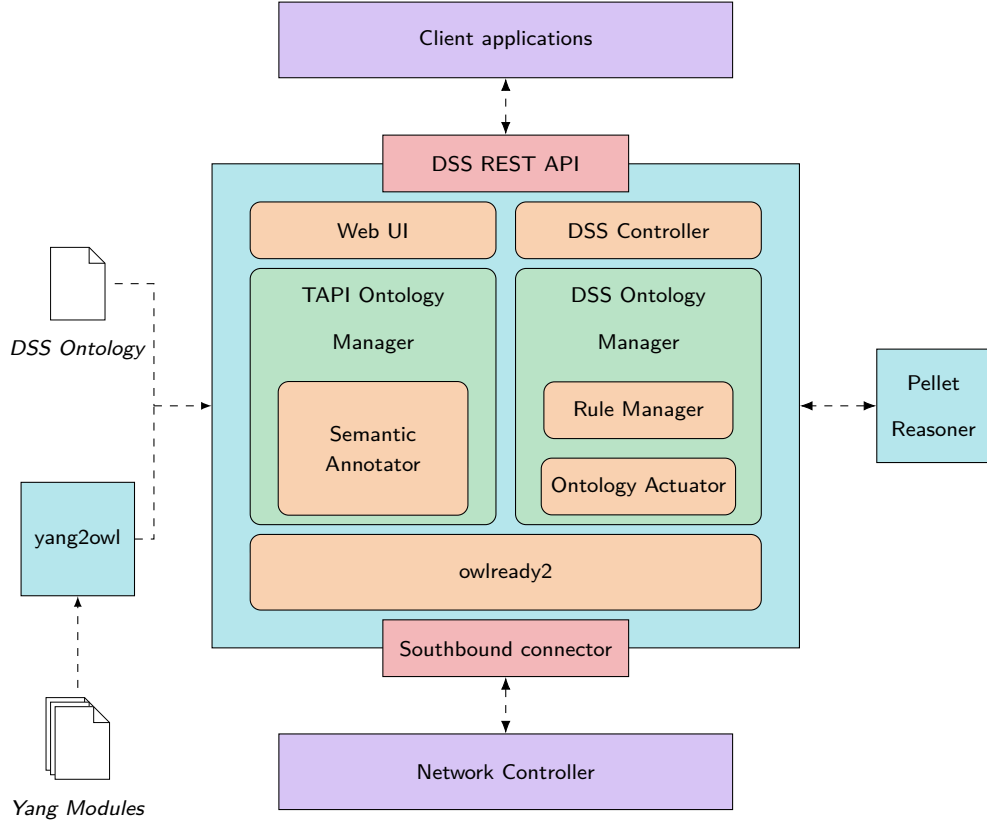


Figure 1.9: Semantic-enabled network management architecture.

Network Analytics (NA), and AI techniques to enhance decision-making in large-scale, distributed networks [110]. The central component of this model is the Knowledge Plane (KP), which integrates ML-based inference with network telemetry to maintain an up-to-date view of the network state. This knowledge-driven feedback loop enables continuous refinement of control strategies, supporting adaptive and predictive behaviors aligned with application-level requirements.

A concrete realization of the KDN paradigm has been proposed in the form of a Decision Support System (DSS) architecture [70]. In this architecture, shown in Figure 1.9, the KP is based on a two-layer ontology. The “low-level” ontology fragment is populated at runtime with controller data and schemas defined in Yet Another Next Generation (YANG) format, a data modeling language used in network management protocols such as Network

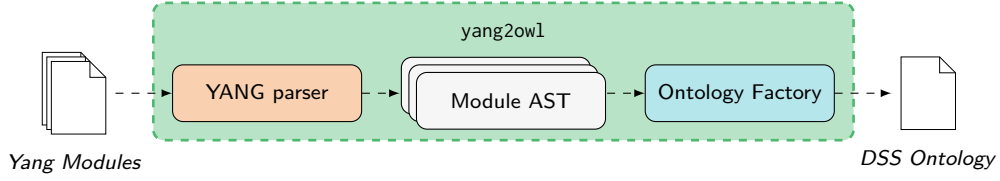


Figure 1.10: yang2OWL architecture.

```

module: tapi-common
+--rw context!
+--rw uuid?  uuid
+--ro tapi-topology:topology* [uuid]
|  +--ro tapi-topology:node* [uuid]
|  |  |  +--ro tapi-topology:uuid  uuid
|  +--ro tapi-topology:link* [uuid]
|  |  |  +--ro tapi-topology:uuid  uuid
|  +--ro tapi-topology:uuid  uuid
...

```

(a) YANG module snippet.

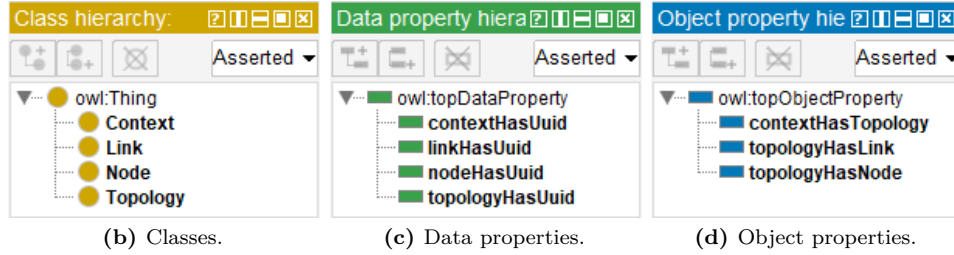


Figure 1.11: Example of the OWL ontology generated from YANG module.

Configuration Protocol (NETCONF) [24, 50], and in standards like European Telecommunications Standards Institute (ETSI) NFV-SOL [82] and Open Networking Foundation (ONF) TAPI [100].

The ontology is populated by translating YANG schemas into equivalent OWL ontologies [129] using the *yang2OWL* tool, illustrated in Figure 1.10.

The system operates iteratively, continuously performing three operations:

- **Update:** populates the KB from SDN controller data;

- **Sync:** applies reasoning via the *Pellet* OWL reasoning engine [163];
- **Apply:** executes inferred actions.

Other elements of the architecture include the compatibility with external plugins to support extensibility and integration with external clients via a RESTful Application Programming Interface (API) and a minimal web User Interface (UI).

A translation example is depicted in Figure 1.11.

During the update phase, configuration data retrieved from a SDN controller (typically in JavaScript Object Notation (JSON) format, and compliant with the Transport API (TAPI) specification) is mapped to OWL individuals through the *SemanticAnnotator*, producing a populated KB in the KP that reflects the current network state. An example of this procedure is depicted in Figure 1.12.

Additional facts are expressed in the form of Semantic Web Rule Language (SWRL) rules [67], which extends OWL by allowing the specification of Horn-like rules through antecedent-consequent (ifthen) expressions.

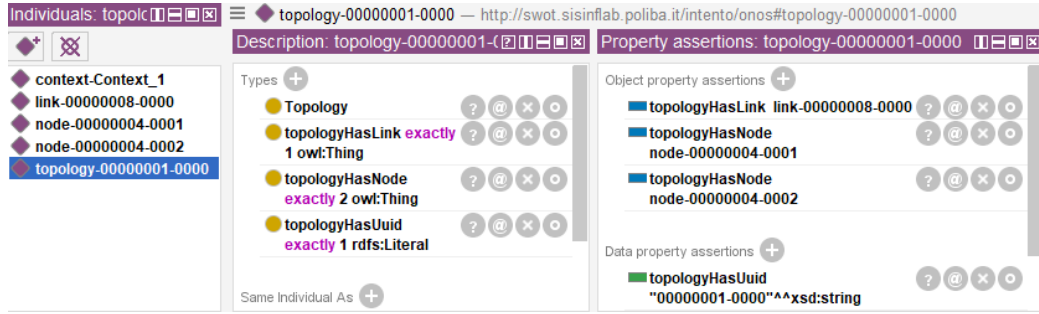
To capture higher-level semantics and constraints, these individuals are mapped to a second ontology fragment, the DSS ontology, which introduces abstract entities (e.g., nodes, links, services) and service-level metrics (e.g., bandwidth, latency, energy). This two-level ontology approach follows a known modeling pattern that separates low-level data schemas from contextual abstractions [146]. Quality of Service (QoS) requirements are modeled as constraint individuals, and violations are flagged via classification logic. During the sync phase, the reasoner is invoked to solve the SWRL rules and (i) derive implicit facts, (ii) detect constraint violations, and (iii) suggest a list of corrective actions. Execution of actions is delegated to the *OntologyActuator* during the apply phase, which links rule outcomes to concrete functions (e.g., path recomputation). The described DSS enables semantic-based network management and orchestration by translating YANG models into OWL ontologies and enriching them through a two-level semantic

```

{
  "uuid": "Context_1",
  "topology": [{
    "uuid": "00000001-0000",
    "node": [
      {"uuid": "00000004-0001"},
      {"uuid": "00000004-0002"}
    ],
    "link": [{"uuid": "00000008-0000"}]
  }]
}

```

(a) JSON configuration.



(b) Individuals generated from JSON document.

Figure 1.12: Example of ABox generation from JSON document.

architecture. The lower ontology captures device configuration data, while the higher level ontology introduces abstract entities and QoS constraints, linked through SWRL rules that enable inference and corrective actions via automated reasoning.

1.7.2 Conversion Layer: Semantic Reasoning and Data-to-Knowledge Mapping for CPS

As discussed in Section 1.1, the Cognitive Layer of the CPS architecture is responsible for inferring knowledge and making decisions based on the data acquired by the Connection Layer and the analysis conducted in the Cyber

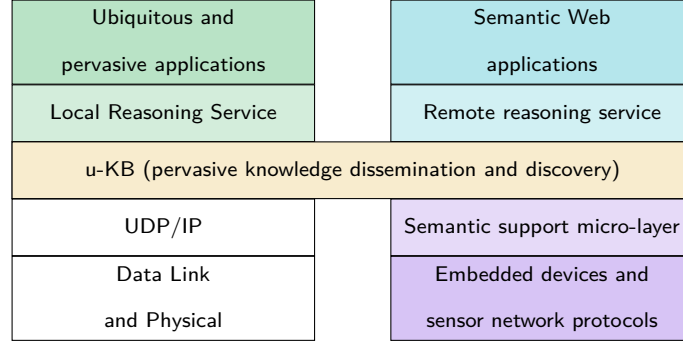


Figure 1.13: Semantic Web of Everything Architecture.

and Conversion layers. Similarly, in the SWoE, semantic technologies are integrated throughout the system stack, from the physical layer to the user application. The key component of the architecture is the u-KB layer, which represents a unified, distributed KB that abstracts and links information generated across devices, network nodes, and cloud services [155]. It provides a common semantic framework for representing contextual data and supports tasks such as service discovery and orchestration [143], as well as knowledge dissemination [142].

Unlike classical Semantic Web approaches, which assume the availability of computational, storage, and networking resources typical of cloud data centers, the SWoE model must address high heterogeneity, intermittence, and volatility of embedded platforms. This has led to the development of tools like *Cowl* [22], a cross-platform OWL 2 library to read and manipulate OWL ontologies, and *Tiny-ME* [141], a multiplatform and lightweight reasoning engine, both enabling reasoning over OWL ontologies on constrained embedded platforms. Semantic engines for the SWoE paradigm implement both *standard inference services* (e.g., ontology classification, concept subsumption, satisfiability, and coherence checking) and *non-standard reasoning tasks*, including concept abduction, contraction, difference, and covering, analyzed in Section 1.5. These tasks are particularly relevant in scenarios involving approximate service discovery, semantic matchmaking, or failure explanation, where incomplete or conflicting information must be handled appropriately.

Transforming raw or semi-structured datasets into a usable KG poses sev-

eral challenges. Notably, it requires bridging the *conceptual gap* between real-world data sources and the formalism of RDF [152] and OWL [129] representations [111]. Pervasive CPSs typically emit large volumes of observations, making data governance and curation difficult [173]. Conventional solutions often rely on centralized cloud-based *data lakes*, which aggregate and preprocess data before mapping it to a KB. While flexible, this strategy can be misaligned with the goals of SWoE, as it requires a dedicated data collection and storage infrastructure and limits the autonomy of the edge devices. Some works in academic research have addressed the problem of mapping structured data, especially tabular formats, into KGs. These approaches support mappings beyond the row-entity assumption and are capable of handling complex schemas [96]. However, many such systems require deep expertise in Semantic Web technologies and often involve complex configuration processes. Declarative mapping languages, such as RDB to RDF Mapping Language (R2RML) [40] and its extension RDF Mapping Language (RML) [46], are widely used for translating tabular data into RDF. RML supports heterogeneous input sources and data formats beyond relational databases. Further solutions, such as *RML-star* [44], incorporate features from the *RDF-star* [153] model to allow annotations on statements.

To assist users in the semantic annotation process, some systems combine mapping with external linked data resources for entity resolution. For instance, *JenTab* [3] defines modular pipelines that leverage *Wikidata* [180] to annotate cells in tabular data. Other systems, such as *DAGOBAAH* [97], aggregate information from multiple open knowledge sources, including *DBpedia* [89] and Wikipedia to improve disambiguation.

Several approaches adopt ML or probabilistic reasoning to infer the semantic structure of data tables automatically [187, 42]. These models can learn from annotated corpora and leverage external ontologies, but they often involve opaque decision logic and task-specific engineered datasets.

Whereas the majority of the aforementioned tools target RDF, a few systems focus on generating OWL KB. Examples include *Mapping Master* [125], which provides a mapping language for tabular data, and *BootOX* [74], which

performs ontology generation and mapping from relational schemas to OWL axioms. The bootstrapping workflow in such systems typically includes vocabulary extraction and ontology creation to create a base schema, and rule specification to extract individuals from data.

While various frameworks exist for data transformation and integration into u-KBs, their complexity and infrastructural demands pose challenges for pervasive computing applications. The *KBgen* library [132] supports automatic OWL 2 KB generation from observation data sets in tabular format. It aims at simplicity, in order to enable further reasoning with SWoE inference engines in pervasive and embedded devices. Lightweight and interpretable mapping strategies remain essential to support automated knowledge base construction in scenarios requiring runtime reasoning, limited connectivity, and constrained resources, such as those found in the information plane of SWoE and pervasive CPSs.

1.7.3 Cyber Layer: Local Semantic Reasoning at the Edge for Autonomous Decision Support

In pervasive and social CPSs, decision support increasingly depends on the ability to interpret and act upon knowledge generated locally, rather than relying on centralized infrastructure. At the same time, the IoE paradigm requires human actors to participate within the system. Web technologies provide a standardized execution and interaction environment for this purpose. Most reasoners are platform-specific and depend on native runtimes, making them unsuitable for Web deployment. Most of the web applications for conventional desktop clients rely on remote services for reasoning.

JavaScript is the main language for implementing client-side code on the Web, but its limited performance has traditionally constrained the deployment of reasoning engines within Web applications, leading most systems to depend on remote, server-side inference services unsuited to mobile and pervasive environments. Only a few mobile Web apps, such as *DBpedia Mobile* [19],

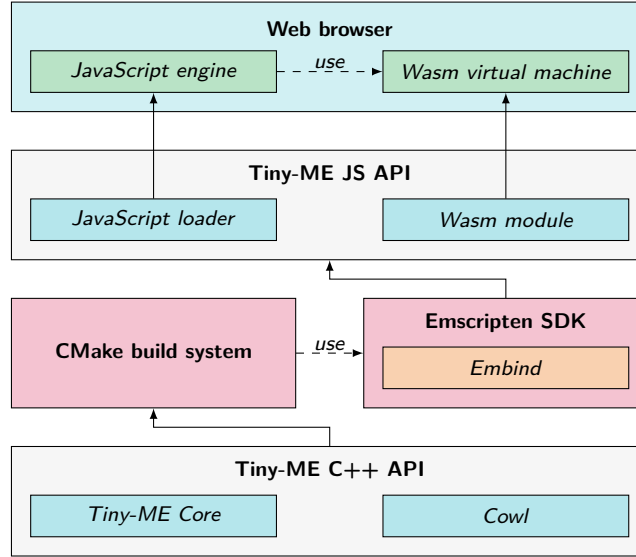


Figure 1.14: Porting workflow and high-level architecture of Tiny-ME Wasm.

LOD4AR [185], and *OntoWiki Mobile* [51], provide partial local reasoning capabilities, while cross-platform solutions remain limited. Attempts to integrate reasoning into Web technologies include *EYE* [184], and *HyLAR+* [175] for reasoning, and *MobiBench* [182] for benchmarking. *Web Assembly (Wasm)* introduces a practical alternative, enabling near-native reasoning performance within browsers and *Node.js*.

The *Tiny-ME* OWL reasoner [141] is designed for portability across heterogeneous platforms, which allows compiling its codebase to target Wasm. The porting workflow is depicted in Figure 1.14. Its core implements inference logic and exposes high-level APIs. For Web deployment, the system is cross-compiled to Wasm using *Emscripten*³.

Figure 1.15 represents the low-level workflow of the port.

A C++ wrapper and *Embind*⁴ bindings provide an object-oriented JavaScript API that handles memory management through the C++ runtime. The build process produces a Wasm module and JavaScript loader that integrate

³Emscripten home: <https://emscripten.org>

⁴Embind documentation: https://emscripten.org/docs/porting/connecting_cpp_and_JS/embind.html

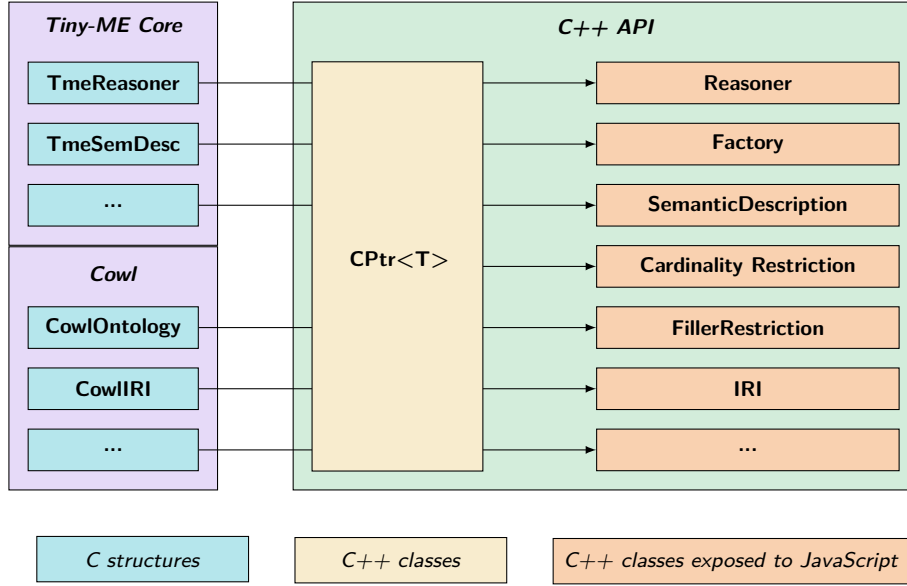


Figure 1.15: Low-level architecture of Tiny-ME Wasm.

directly into Web applications.

Memory management is unified via a `CPtr<T>` C++ Resource Acquisition is Initialization (RAII) template to manage under a single interface both Tiny-MEs `malloc/free` logic and Cows reference counting. The port exposes a `Reasoner` class with methods for executing both standard and non-standard inference tasks.

This port underwent a performance evaluation to assess the execution time of inference tasks. The analysis focused on three metrics: (i) fetching, the time to download ontologies; (ii) parsing, the time to deserialize them; and (iii) reasoning, the time to execute inference. Tests were performed on a MacBook

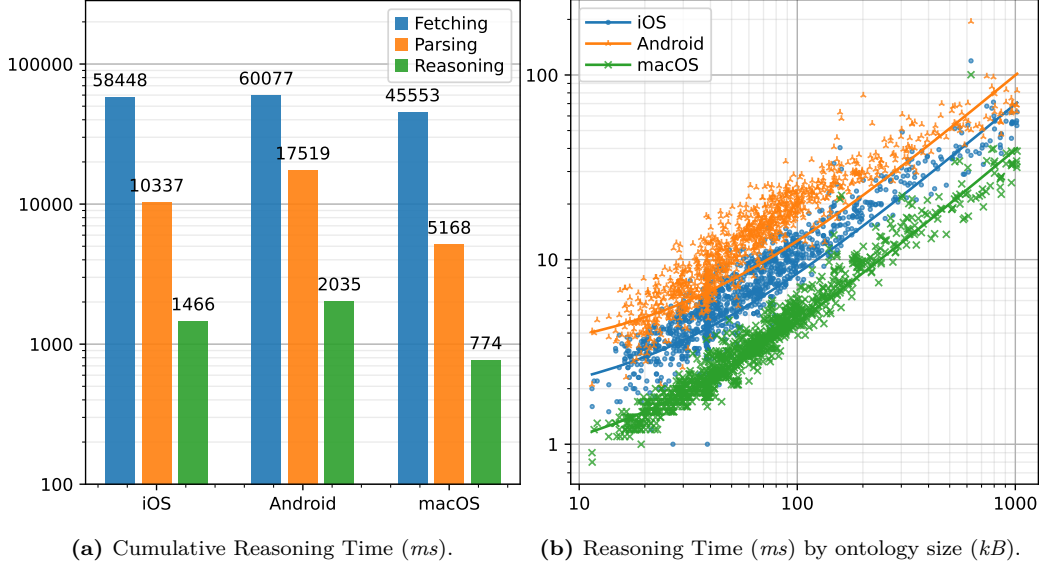


Figure 1.16: Tiny-ME Wasm ontology classification results.

Pro (M1 Max)⁵, iPhone 12 Pro⁶, and OnePlus 7T⁷ to cover different hardware and browser configurations.

Datasets included ontologies from Tiny-ME [141] knowledge bases for non-standard reasoning and a subset of the *ORE2014* [16] ontologies (smaller than 1 MB, $\mathcal{ALN}$ expressiveness) for standard reasoning. Each test was executed via a web page running client-side JavaScript, which downloads the ontology, invokes the reasoner, and records results to Comma Separated Value (CSV) files. Visualizations were generated using *evOWLuator* [154]. Ontologies and scripts were served by a *Nginx* instance on a local Wireless Local Area Network (WLAN) without caching.

⁵Apple M1 Max SoC, 8 performance cores @3.2GHz and 2 efficiency cores @2.0GHz, 64 GB UM RAM, 1 TB SSD, macOS Ventura 13.3.1, Safari 16.4 browser.

⁶Apple A14 Bionic, 2 high-power cores @3.1 GHz and 4 low-power cores @1.8 GHz, 6 GB LPDDR4X RAM at 4266 MT/s, 128 GB NVMe SSD, iOS 15.1, Safari Mobile 15.1 browser.

⁷Qualcomm Snapdragon 855+, 1 core @2.96 GHz, 3 cores @2.42 GHz and 4 cores @1.8 GHz, 8 GB LPDDR4X RAM, 128 GB UFS 3.0 storage, Android 12, Chrome 112.0.5615.47 browser.

Timing data were collected using the `performance.now()`⁸ API, with reasoning times averaged across multiple runs (10 on mobile, 50 on laptop) to offset browser timing resolution limits.

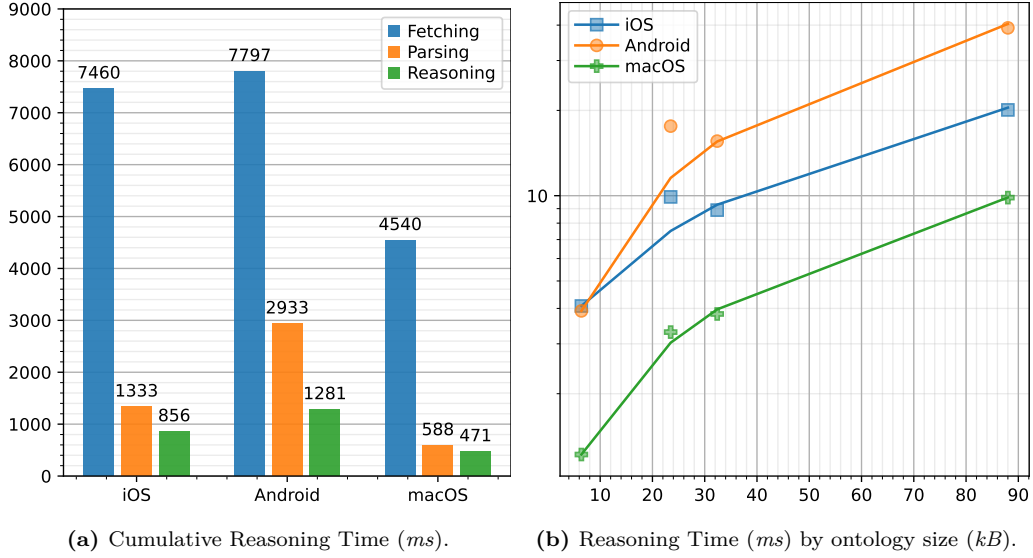


Figure 1.17: Tiny-ME Wasm semantic matchmaking results.

Figure 1.16 presents the results for the *ontology classification* standard inference task. The bar chart in Figure 1.16a reports the cumulative time for fetching, parsing, and reasoning across all tested ontologies, while the scatter plot in Figure 1.16b shows parsing and reasoning times as a function of ontology size. The maximum total time observed on the least capable device was approximately 200 ms. These results indicate that the Wasm reasoner achieves adequate performance for ontology management tasks, supporting the requirements of interactive semantic Web applications on mobile platforms. Fetching dominates total time, but this impact is negligible in practice, as ontologies are typically retrieved once and cached for subsequent operations.

The *matchmaking* non-standard inference task iterates over all individuals in each of the four reference ontologies to identify the best match for a *request* individual. The process begins with a *compatibility check* between a $\langle request, resource \rangle$ pair: if the conjunction is satisfiable, *concept abduction* is

⁸<https://developer.mozilla.org/en-US/docs/Web/API/Performance/now>

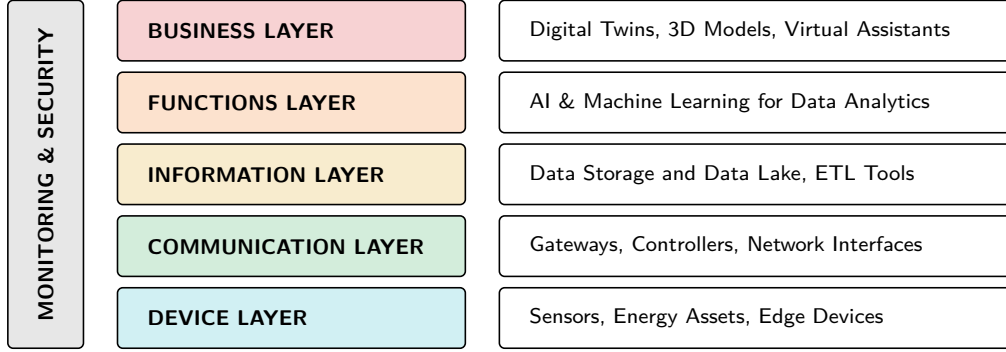


Figure 1.18: Smart Grid Architecture Model [69].

performed; otherwise, *concept contraction* is executed, followed by abduction on the contracted request.

Results in Figure 1.17 show that the Wasm reasoner executes this inference within a few milliseconds, with a maximum of about 40 ms for the largest ontology. These findings are consistent with the results obtained for standard inference tasks, yet more significant given that non-standard reasoning is central to many SWoE use cases targeted by Tiny-ME Wasm. Overall, the evaluation confirms that the reasoner can be deployed in real-time applications to provide both standard and non-standard inference services efficiently, without compromising responsiveness or user experience.

1.7.4 Cognition Layer: Retrieval-Augmented Generation for Digital Twins

LLMs have demonstrated versatility and performance, making them central to modern AI research and applications, including their integration into CPS for enhancing humanmachine interaction through Natural User Interface (NUI) interfaces [69].

Figure 1.18 illustrates the Smart-Grid Architecture Model (SGAM), which provides a reference framework for smart grid infrastructures [179]. This architecture consists of the following functional layers:

- The **Device layer** includes sensors and actuators that monitor and control physical assets in the grid.
- The **Communication layer** manages data transmission across the infrastructure using standardized protocols such as Message Queuing Telemetry Transport (MQTT) and REpresentational State Transfer (REST) API.
- The **Information layer** structures and integrates heterogeneous data from field devices and network components into a unified data model stored within a centralized data lake.
- The **Analytics layer** applies ML for tasks such as predictive maintenance, fault detection, and load forecasting, combining real-time and batch processing.
- The **Visualization layer** implements three-dimensional views of assets, dashboards, and NUI that allow operators to interact with the system.
- Finally, the **Security layer**, which interacts with all the layers in the stack, provides access control, continuous monitoring, and compliance management to ensure the resilience and integrity of the infrastructure.

Both LLMs and RDF-based vocabularies for RAG can be integrated into this architecture [69]. At the **Information layer**, data acquired from field devices are represented using reference vocabularies such as Digital Twins Definition Language (DTDLE) and Smart Energy Aware Systems (SEAS) within a KB, as illustrated in Figure 1.19.

At the **Analytics layer**, the KB and the LLM are connected through a RAG-based mechanism, enabling the model to retrieve domain-specific facts from the KB and generate grounded responses. At the **Visualization layer**, the LLM provides a natural-language interface to the operators, enabling conversational queries and delivering recommendations or corrective actions based on real-time knowledge.

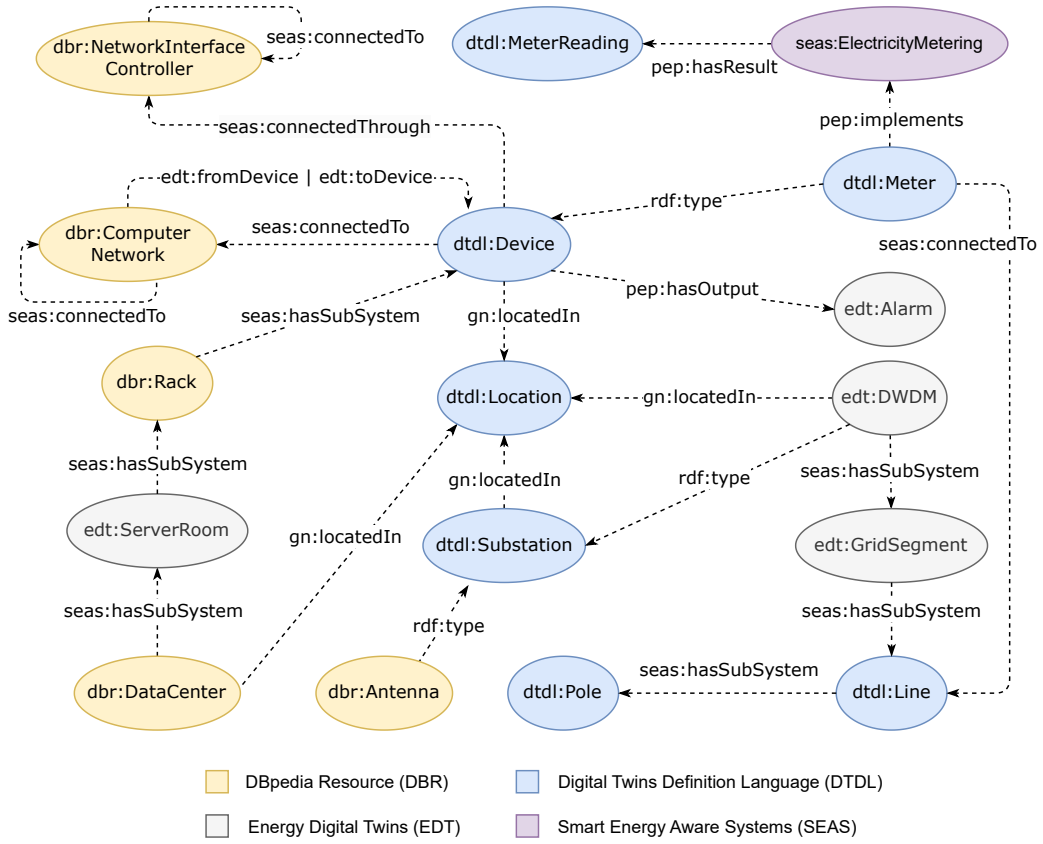
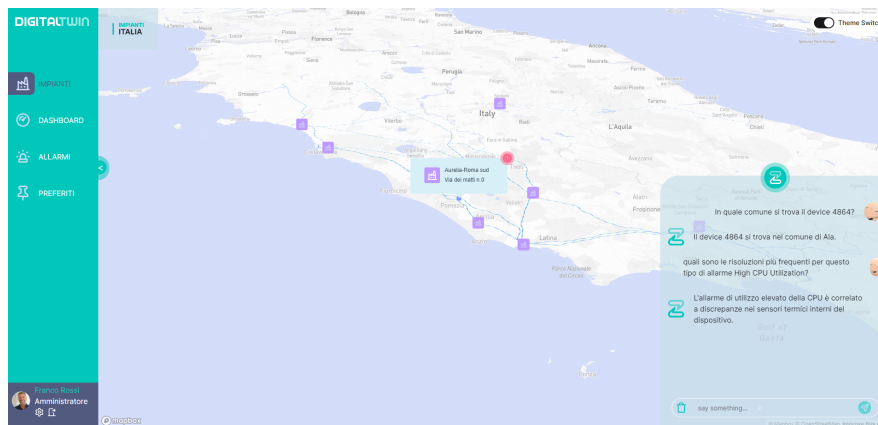
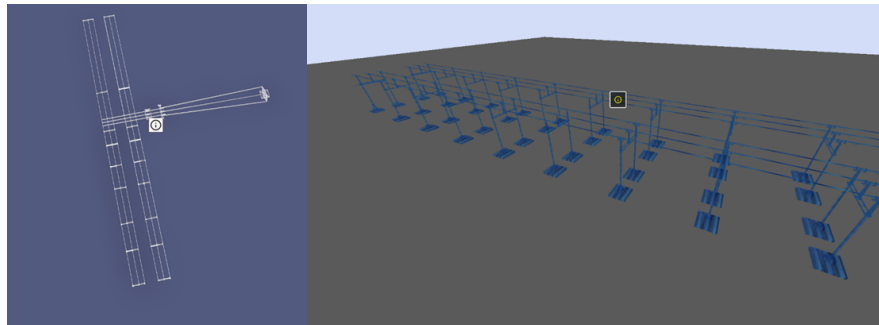


Figure 1.19: Reference graph-based data model [69].

This has resulted in a Digital Twin platform for large-scale energy infrastructures that integrates edge and cloud computing, data lakes, IoT sensor networks, and cybersecurity controls. Figure 1.20 illustrates examples of the system interface, including the web dashboard for grid visualization and 2D/3D Digital Twin representations, while Figure 1.21 shows the mobile application used by maintenance operators. It incorporates a RAG-based reasoning module combining LLMs with knowledge graphs for contextual querying and decision support. Deployed on an operational power grid covering 75,000 km of lines and 900 substations, the system supports predictive maintenance, real-time monitoring, and operator interaction through natural-language and multimodal interfaces, ensuring interoperability and scalable data management across distributed assets [69].

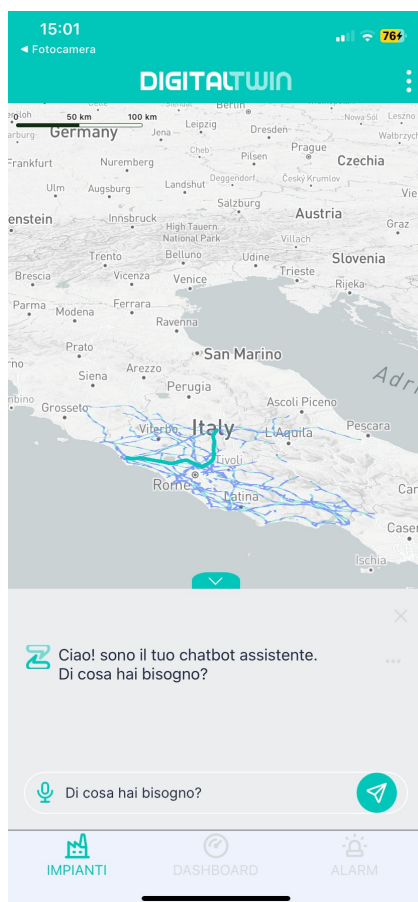


(a) Digital representation of the energy grid.



(b) 2D model of the energy plant.

Figure 1.20: Examples of digital-twin visualizations: (a) energy grid representation; (b) 2D model of plant assets [69].



(a) Map of the energy grid



(b) Details about an alert message

Figure 1.21: Screenshots of the mobile application. [69]

Chapter 2

Expanding pervasive Cyber-Physical Systems to the Serverless Cloud-to-Thing continuum

Recent advancements in the capabilities of IoT devices have enabled their use beyond simple data acquisition, positioning them as lightweight computing nodes capable of supporting both inference and, under certain conditions, model training. However, the design of unified, end-to-end architectures capable of fully leveraging the whole Cloud-to-Thing continuum remains an open challenge. Therefore, this chapter presents a serverless computing framework that extends pervasive intelligence down to the IoT layer. The architecture supports function deployment across cloud, edge, and device nodes through a uniform interface and with minimal adaptation, where IoT devices participate in both training and inference.

2.1 Design Motivation and System Requirements

As discussed in Chapter 1, and in particular in Section 1.6, the development of pervasive CPSs introduces a series of challenges. These include the need to minimize *latency*, manage the growing *complexity*, and implement efficient *data*

management strategies. Traditional cloud-centric approaches, while simple and powerful, are often inadequate to meet such constraints, especially when dealing with real-time, resource-constrained, or privacy-sensitive applications. Furthermore, although recent works have focused on extending the Cloud-to-Edge continuum toward IoT, their management within a unified architecture remains an open challenge.

Serverless computing offers a promising abstraction model. It enables event-driven execution of small, self-contained, stateless functions without the need to manage underlying infrastructure. This chapter introduces a serverless computing framework designed to support pervasive CPSs by deploying ML and AI workflows across cloud, edge, and IoT layers, that is, the Cloud-to-Thing continuum. The architectural design is guided by the following key requirements:

- **Uniform execution model.** Functions must be deployable across the continuum with minimal code modification, enabling flexible task placement and resource utilization.
- **Transparency.** The system should ensure functional transparency for both devices and users, such that service interaction does not depend on the physical location of computation.
- **Federated learning support.** The architecture should allow for local data processing and decentralized training workflows, minimizing data movement and preserving privacy.
- **Lightweight deployment.** Runtime environments must comply with the resource constraints of IoT nodes, supporting short execution times, low memory usage, and small footprint libraries.

Table 2.1: Reference development boards for Microcontroller Units.

Board	MCU	Processor	Memory	Storage
Arduino Uno	ATmega328P (8-bit AVR RISC)	20 MHz Single core	2 KB	32 KB
NUCLEO-C031C6	STM32C031C6T6 (ARM Cortex-M0+)	48 MHz Single core	12 KB	32 KB
Blue Pill	STM32F103C8T6 (ARM Cortex-M3)	72 MHz Single core	20 KB	64 KB
Arduino Due	AT91SAM3X8E (ARM Cortex-M3)	84 MHz Single core	96 KB	512 KB
Raspberry Pi Pico	RP2040 (ARM Cortex-M0+)	133 MHz Dual core	264 KB	16 MB
ESP32-Gateway	Xtensa LX6 (Tensilica RISC)	240 MHz Dual core	520 KB	4 MB
B-L4S5I-IOT01A	STM32L4S5VIT6 (ARM Cortex-M4)	120 MHz Dual core	640 KB	2 MB
ESP32-DevKitC	ESP32-WROOM-32 (Xtensa LX6)	240 MHz Dual core	520 KB	4 MB
nRF52840 Feather	nRF52840 (ARM Cortex-M4)	64 MHz Single core	256 KB	1 MB
nRF52840-DK	nRF52840 (ARM Cortex-M4)	64 MHz Single core	256 KB	1 MB
Adafruit HUZZAH32	ESP32 (Xtensa LX6)	240 MHz Dual core	520 KB	4 MB
ATSAMD21 Metro	ATSAMD21G18 (ARM Cortex-M0+)	48 MHz Single core	32 KB	256 KB

Table 2.2: Reference development boards for Microprocessor Units.

Board	MPU	Processor	Memory	Storage
Raspberry Pi B	Broadcom BCM2835 (ARM1176JZF-S)	0.7 GHz Single core	512 MB	MicroSD
BeagleBone Black	TI AM335x (ARM Cortex-A8)	1.0 GHz Single core	512 MB	4 GB
Raspberry Pi 3B	Broadcom BCM2837 (ARM Cortex-A53)	1.2 GHz Quad core	1 GB	MicroSD
Raspberry Pi 3B+	Broadcom BCM2837B0 (ARM Cortex-A53)	1.4 GHz Quad core	1 GB	MicroSD
Asus Tinker Board	Rockchip RK3288 (ARM Cortex-A17)	1.8 GHz Quad core	2 GB	MicroSD
Jetson Nano	NVIDIA Tegra X1 (ARM Cortex-A57/A53)	1.9 GHz 4C + 1.3 GHz 4C	4 GB	MicroSD
ROCKPro64	Rockchip RK3399 (ARM Cortex-A72/A53)	1.8 GHz 2C + 1.4 GHz 4C	4 GB	128 GB
Raspberry Pi 4B	Broadcom BCM2711 (ARM Cortex-A72)	1.5 GHz Quad core	8 GB	MicroSD
Variscite DART-MX8M	NXP i.MX 8M Plus (ARM Cortex-A53)	1.5 GHz Quad core	8 GB	64 GB

The framework is designed to operate on heterogeneous hardware and run effectively on both MCUs and MPUs. MCUs, typically integrating on-chip flash and Static Random Access Memory (SRAM), provide fast boot times, deterministic execution, and are energy efficient, making them well-suited for real-time tasks such as data acquisition at the sensing layer, and long-term battery-powered operation. In contrast, MPUs interface with external high-capacity Dynamic Random Access Memorys (DRAMs) and non-volatile flash storage, allowing them to execute general-purpose operating systems, run containerized workloads, data aggregation frameworks, and serverless compute environments on edge infrastructure, which, in turn, makes their execution non-deterministic. MPUs commonly offer hundreds of megabytes to several gigabytes of memory, while even the most advanced MCUs are constrained to around 2 MB of flash and a few hundred kilobytes of SRAM. Both classes of devices are available as standalone chips or commercial development boards. Examples of supported MCUs and MPUs platforms are listed in Table 2.1 and Table 2.2, respectively.

Functions can be embedded as built-in, fixed routines or tasks (typically on MCUs) or deployed dynamically in a serverless execution environment. This approach supports a range of use cases. Field IoT devices may perform on-device inference or transmit data to a local edge node for more complex processing. In FL scenarios, model updates can then be shared with a centralized cloud service for aggregation and global optimization. Edge nodes perform local analytics while preserving data privacy. In industrial workplace safety, wearable IoT sensors can monitor workers movements and environmental conditions. The architecture is also applicable in rural or infrastructure scenarios, such as remote agriculture and smart grids, where connectivity is limited. Here, local IoT devices monitor environmental or infrastructure metrics and perform real-time processing. When connectivity is available, data are transmitted to an edge or cloud node for further analysis.

2.2 Abstract Architecture for Cloud-to-Thing Orchestration

One of the main challenges in trying to include heterogeneous devices having different resource availability into a unified framework is that not all nodes may have the capabilities to implement all the required functions. The architecture which is being proposed revolves around a key abstraction, which is the **Computing Node**, representing any computation-capable component within the system, ranging from resource-constrained embedded devices to powerful cloud servers.

The proposed infrastructure is represented as a network of interconnected **Computing Nodes**. Communication is achieved via a **Message Bus**, which follows a Publish/Subscribe (pub/sub) model. In this pattern, a **Message Bus Server** manages a set of topics, and any number of **Message Bus Clients** may publish or subscribe to these topics. Clients may publish messages without being subscribers themselves. When a message is published on a topic, the Message Bus Server broadcasts it to all subscribed clients. This

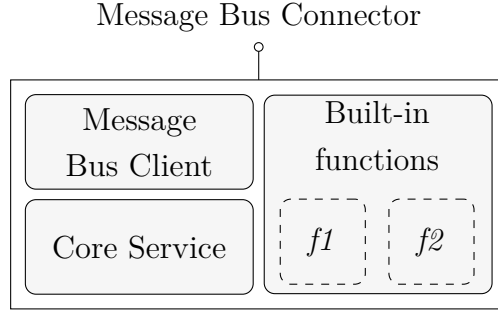


Figure 2.1: Minimal Computing Node.

pattern aligns with standard IoT messaging protocols such as MQTT [124] and Constrained Application Protocol (CoAP) with the observer pattern [25].

Given the heterogeneity of CPS environments, a uniform software stack across all nodes is not feasible. For example, MCU devices are often constrained in memory, processing power, and storage, making it impractical to deploy full runtimes or persistent storage. Furthermore, due to their low-energy profiles and limited availability, they are generally unsuitable as Message Bus Server hosts.

To address this, the framework defines two main classes of nodes:

1. Minimal Computing Node, which implements only the mandatory components required for participation in the system;
2. Complete Computing Node, which can host optional modules supporting more advanced functionality.

These configurations represent opposite ends of a capability spectrum. Actual deployments may adopt hybrid configurations that include all required modules from the Minimal Computing Node and a subset of optional components from the Complete Computing Node, depending on platform resources and application needs.

In particular, each Computing Node must implement the following **mandatory components** shown in Figure 2.1:

1. **Message Bus Client:** Manages the nodes connection to the messaging infrastructure, handling publication and subscription to topics and communication with the nearest Message Bus Server.
2. **Message Bus Connector:** The networking interface that enables the Message Bus Client to establish and maintain communication. It encapsulates the transport protocol stack and connection logic.
3. **Core Service:** The control unit of the node, coordinating internal components, parsing incoming messages, and orchestrating task execution. It evaluates whether to locally handle a request, forward it to another node, or return an error if the request cannot be satisfied.
4. **Built-in functions:** A set of embedded functions hard-coded into the node. These are typically used on constrained devices that do not support Over-the-air (OTA) updates or dynamic function deployment. A device can have zero or more built-in functions.

More capable nodes can include any combination of the following **optional modules**:

1. **Local Storage:** Supports persistent or semi-persistent data storage for sensor measurements, intermediate results, logs, and model checkpoints.
2. **Message Bus Server:** Enables the node to act as a local message broker, routing messages for a cluster of nearby nodes.
3. **Function Runtime:** A serverless execution environment allowing dynamic deployment, migration, and invocation of distributed functions. Unlike Built-in functions blocks, these functions can be updated or reconfigured at runtime.
4. **Sensor Connector and Actuator Connector:** Provide interfaces to physical sensors and actuators, respectively, to bridge the cyber and physical layers of CPSs.

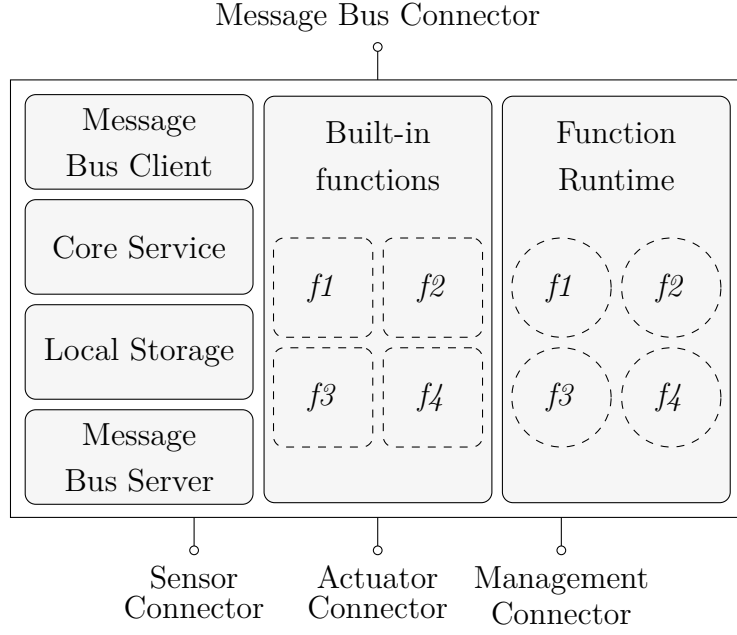


Figure 2.2: Complete Computing Node.

5. **Management Connector:** Exposes administrative interfaces for remote management, monitoring, and configuration. It enables inspection of status, updates of functions, or control of lifecycle transitions from external systems.

A node implementing all the modules is shown in Figure 2.2.

By employing this abstraction, CPS components can be integrated into the architecture considering application constraints and hardware capabilities.

All operations are executed and coordinated using a decentralized approach. A node joins the network by establishing a connection with a nearby Message Bus Server. Each node is assigned a unique identifier and can manage connections with multiple neighboring servers. Upon successful registration, it publishes its local capabilities, such as available functions, hardware interfaces, or processing resources, which are stored and managed by the Core Service component of the neighboring node, allowing the system to support service discovery and request routing based on the node capabilities.

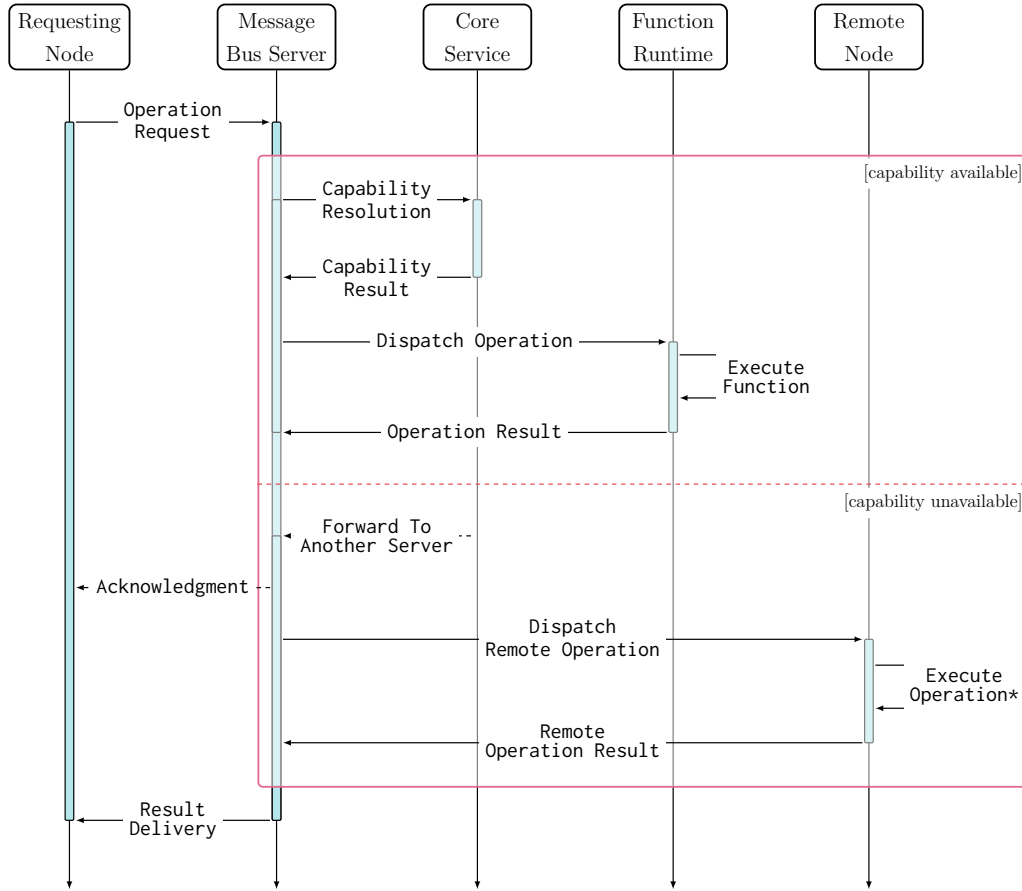


Figure 2.3: Sequence diagram of a generic operation dispatch.

As cited earlier, the communication model follows the pub/sub pattern. Requests are sent on shared topics managed by the Message Bus Server, while responses are directed to node-specific topics associated with each client-server pair.

Given the asynchronous and distributed nature of the framework, node failures are managed through a combination of application-level timeout mechanisms and the reliability features of the underlying transport layer (e.g., MQTT QoS, CoAP confirmations). At the application layer, the Core Service maintains a neighborhood registry. When a node first announces its presence, a three-way handshake procedure is performed to estimate the round-trip latency. This value is used to initialize the timeout threshold for subsequent interactions with that specific peer (e.g., the timeout value can be set to be a multiple of

this value).

As shown in Figure 2.3, if the local node cannot fulfill a request, it attempts to forward the message to another node with the required capability. In the absence of suitable candidates, an appropriate failure message is returned to the requester. To support multi-hop request resolution, where the total execution time may exceed this value, the Core Service performing request forwarding sends intermediate acknowledgment messages to the requester. These acknowledgments serve to extend the timeout window of the requester. However, in real-world deployments, additional constraints such as a maximum hop count (Time-To-Live) and a hard upper bound on the total discovery time must be enforced to limit the search space. If a target node fails to respond within this window, the Core Service assumes a failure. In the current version, the requester is responsible for handling this exception. In the sequence diagram, the messages that concern capability discovery, communication with remote core service, and remote function invocation are omitted for brevity, as they work like the local operation execution case.

2.3 Functional Components and System Workflow

Building on the abstraction presented in Section 2.2, this section details the main functional components and the overall system workflow.

What follows is designed to address the key challenges identified in Section 1.6, namely the lack of unified support for serverless AI workflows across the full Cloud-to-Thing continuum and the limited realization of on-device intelligence in resource-constrained environments (that is, the Level 6 of Figure 1.4). As detailed earlier in Section 2.1, the system design follows four key requirements: a uniform execution model for cross-layer deployment, transparency for location-independent operation, FL support for decentralized intelligence, and lightweight deployment suited to constrained IoT nodes.

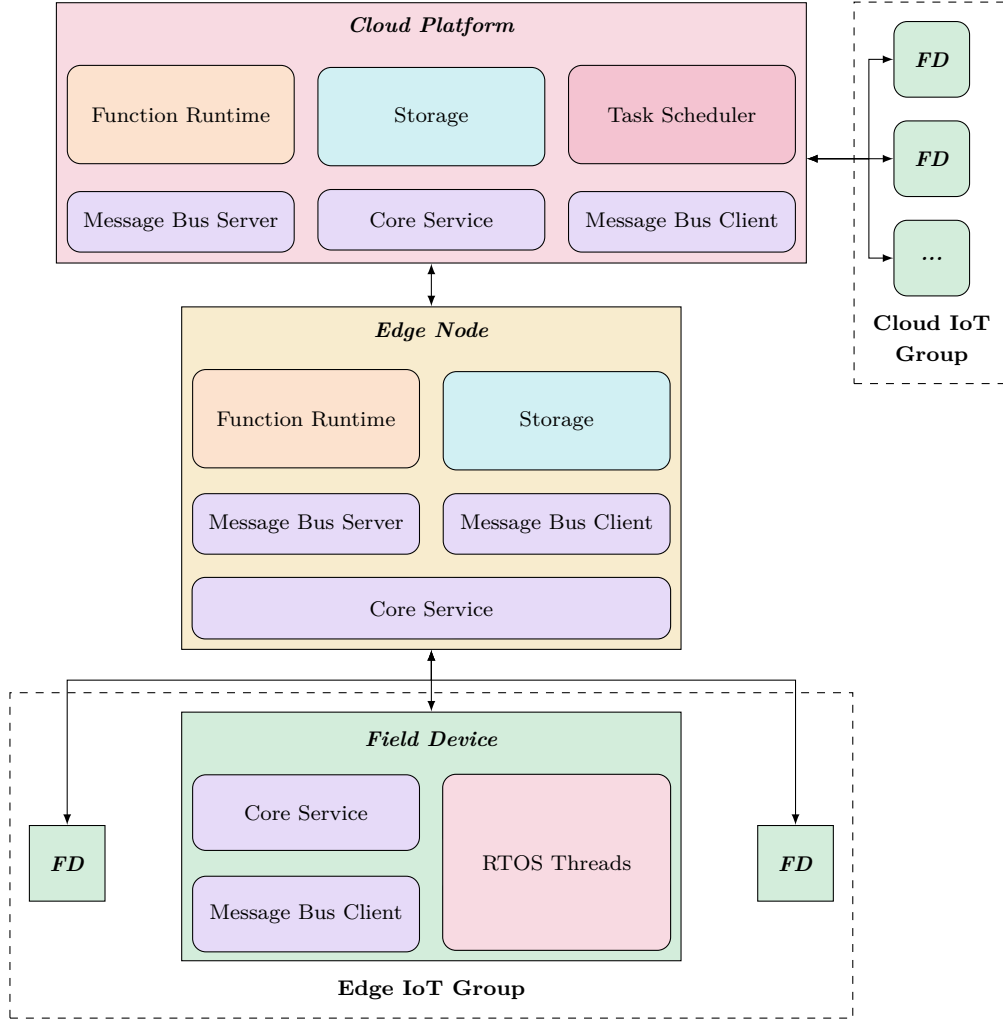


Figure 2.4: Reference framework architecture.

The framework is intentionally general-purpose. It supports the distributed execution of serverless functions for a broad class of applications involving data processing and model inference across cloud, edge, and IoT resources.

The network is conceptually represented in a high-level diagram in Figure 2.4, which includes the cloud infrastructure, edge devices, and IoT field nodes. **Cloud Provider**, **Edge Node**, and **Field Device** represent the three tiers of computational nodes. The cloud handles centralized processing and coordination, the edge offers localized compute and storage, and field devices are constrained IoT nodes running lightweight Real-Time Operating System

(RTOS).

Table 2.3: Framework operations grouped by functional category.

Category	Operation	Description
Data Gathering	<code>request_data(desc)</code>	Request data matching the description from neighbors.
	<code>push_data(payload, desc)</code>	Send raw or preprocessed data to a target node.
Model Training	<code>update(model, data, config)</code>	Train or fine-tune a model.
	<code>merge(models)</code>	Create a new model by merging a list of existing ones
	<code>store(model, desc)</code>	Save a model (trained or empty) to external storage.
	<code>load(desc)</code>	Retrieve a model matching a descriptor.
Inference	<code>infer(input, model)</code>	Run inference on input data using a specified model.
	<code>batch_infer(data, model)</code>	Apply inference across a dataset.
Evaluation	<code>evaluate(testset, model)</code>	Evaluate a model's performance using validation data.
	<code>log_metrics(query)</code>	Report evaluation results for monitoring or analysis.
Management	<code>announce(node_desc)</code>	Announce the nodes available resources and functions.
	<code>list_datasets(desc)</code>	List datasets accessible from the current node.
	<code>list_models(desc)</code>	List available models or descriptors on the current node.

The functions listed in Table 2.3 have been implemented to support distributed and serverless execution of ML pipelines. Each category groups operations according to their role in the system lifecycle:

- **Data Gathering:** Functions for data exchange between nodes.
- **Model Training:** Functions managing model creation, training, and aggregation.
- **Inference:** Functions that apply trained models to label new, unlabeled data.
- **Evaluation:** Functions for assessing model performance.

- **Management:** Functions enabling resource discovery within the network.

Two additional functions are implicitly integrated into the core service of each node: one is responsible for announcing the nodes capabilities and available functions to the network, while the other maintains a lightweight, continuously updated view of the surrounding neighborhood.

The architecture adopts an event-driven paradigm, where functions are invoked by incoming messages. Triggers include periodic events from the task scheduler, sensor readings, actuator state changes, node connection or disconnection, and messages exchanged with neighboring nodes.

2.4 Prototype Deployment of Serverless Machine Learning in a Cloud-to-Thing Architecture

Building on the architectural design and functional specifications presented in the previous sections, this section describes a prototype implementation of the proposed framework deployed on Amazon Web Services (AWS), integrating selected MCU and MPU devices serving as IoT and edge nodes, respectively. The objective is to validate the feasibility of the proposed approach, demonstrating how the defined components and functions operate in a real deployment scenario. Wherever possible, Commercial Off-the-Shelf (COTS) tools were employed to implement the different modules of the architecture. The AWS prototype, illustrated in Figure 2.5, comprises the following components:

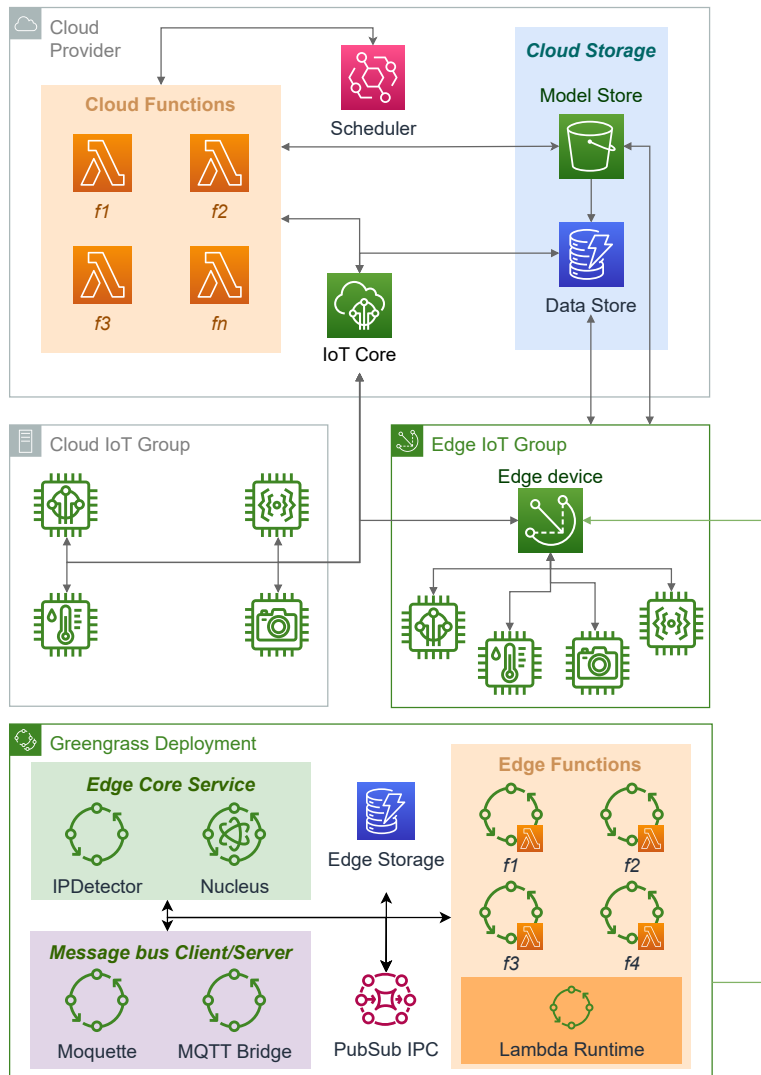


Figure 2.5: AWS-based prototype architecture.

- **AWS IoT Core:** Implements the Message Bus Server, Management Connector, and Core Service components of the architecture in the cloud. It provides the MQTT-based messaging backbone for inter-node communication, along with device provisioning, management, and authentication services. It also enables event-driven invocation of other AWS components in response to messages published on the MQTT server.

- **Storage:** Implemented through *Amazon Simple Storage Service (S3)* and *DynamoDB*, corresponding to the **Local Storage** component of the architecture. *Amazon S3* stores global model snapshots, while *DynamoDB* stores batched sensor data with configurable aging policies.
- **Lambda Functions:** Represent the **Function Runtime**, containing an implementation of the core tasks described in Section 2.2.
- **Task Scheduler:** Is treated as a built-in function of the cloud, and it triggers Lambda executions through time-based or event-based policies and is implemented with *AWS EventBridge*.
- **Edge Device:** Functions as an edge **Complete Computing Node**. Its **Core Service** is implemented through *AWS IoT Greengrass CoreDevice*, which also includes a **Function Runtime** component enabling the execution of serverless functions locally on the edge node.
- **Greengrass Deployment:** Represents the configured set of components deployed to an edge node through AWS IoT Greengrass. In particular, the deployment is composed of:
 - **Nucleus:** Manages node lifecycle and coordinates communication with the cloud, extending the functionality of the **Core Service**.
 - **PubSub IPC, MQTT Bridge, and Moquette:** Implement the local, intra-node Inter-Process Communication (IPC) system and the **Message Bus Client/Message Bus Server** components for inter-node messaging.
 - **Functions:** Local Lambda components representing deployable serverless functions inherited from the cloud **Function Runtime**.
 - **Local Data Store:** Local *DynamoDB* instance implementing the nodes **Local Storage** module for caching data batches and model updates.
 - **IPDetector:** Handles capability discovery and cloud registration processes within the **Core Service**.

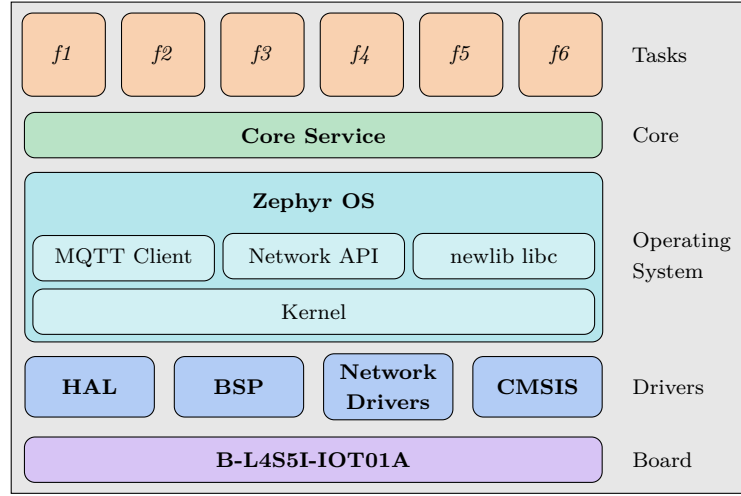


Figure 2.6: IoT Field Device firmware.

In the prototype deployment, AWS serves as the **cloud node**, a **Raspberry Pi 4 Model B+** acts as the **edge node**, and **B-L4S5I-IOT01A** boards are used as **IoT nodes**. For the IoT devices, the firmware has been developed from scratch, following the architecture shown in Figure 2.6. It adopts a layered design and runs on the STM32L4S5VIT6 MCU installed on the B-L4S5I-IOT01A development board, which provides wireless connectivity and various onboard sensors. The firmware is written in C and builds upon the **Zephyr RTOS**, which simplifies hardware portability and offers native support for key features such as concurrency, networking, and device management.

The software stack includes the Hardware Abstraction Layer (HAL), Board Support Package (BSP), and Wi-Fi drivers, as well as Common Microcontroller Software Interface Standard (CMSIS) for hardware abstraction and portability. Application logic is built on ZephyrOS’ primitives and libraries to ensure secure communication with AWS IoT Core. A subset of the frameworks functions, specifically those not requiring server-like capabilities or persistent storage, has been implemented on the devices to enable local processing and inference triggered by sensor events. Device provisioning involves firmware deployment, registration on AWS IoT Core, certificate and policy association, and discovery via Greengrass. Once configured, devices communicate securely through authenticated and encrypted channels.

2.4.1 Portability

The prototype serves as a reference implementation rather than a fixed configuration. This section highlights different services and tools that can be used to implement the framework, demonstrating that the proposed architecture is not bound to a specific hardware or software platform. Regarding the IoT nodes, Zephyr RTOS and its Software Development Kit (SDK) provide a unified development environment supporting a wide range of heterogeneous devices. The B-L4S5I-IOT01A board is supported natively, but the same setup can be adapted to other MCUs as long as a Zephyr BSP is available or can be created. This approach makes it easy to reuse the same code across different platforms. On the edge node, AWS IoT Greengrass handles the orchestration and execution of serverless functions and can be installed on any Linux device with a compatible Java Runtime Environment.

AWS has been selected as the technology provider for the prototypical implementation of the proposed framework due to comprising available building blocks that can be mapped directly to all components of the architecture. However, it is essential to note this is not a mandatory choice. The same framework is portable to other providers by substituting the adopted managed services with equivalent offerings.

Microsoft Azure includes *Azure Functions* supporting multiple languages, *Azure Service Bus* for message management, *Blob Storage* and *Table Storage* for data storage, and *Azure IoT Edge* for deploying workloads on edge devices with enhanced device management capabilities. Unlike AWS Greengrass, Azure IoT Edge requires user code to be run in containers. This can limit the choice and increase the cost of devices to be used. Conversely, Azure IoT Edge automatically manages multiple-level hierarchies of devices, whereas AWS Greengrass requires the user to configure the device group hierarchy. This can be an important feature for large-scale FL applications.

IBM Cloud services include *IBM Cloud Functions* based on *OpenWhisk*¹,

¹Apache OpenWhisk: <https://openwhisk.apache.org/>

IBM Event Streams using *Kafka*², *IBM Cloud Object Storage* and *Cloudbant*, based on *CouchDB*³, for data storage, and *IBM Edge Application Manager* for managing edge device services, similar to Azure IoT Edge, with additional support for serverless computing through *Edge functions*. Edge computing solutions rely on the *IBM Cloud Satellite* service for managing hybrid cloud deployments, which is more powerful but more complex than AWS Greengrass or Azure IoT Edge.

Other Platform-as-a-Service offerings, focused on the Cloud-to-Edge and the Cloud-to-Thing continuum, have been introduced recently. For instance, **Particle**⁴ provides a cloud-based management platform for devices running their *Device OS*, which simplifies user application integration into firmware and lifecycle management. Similarly, the **Arancino**⁵ [59] platform comprises (i) a family of open-hardware dual-board devices, which can perform as edge computing devices as well as IoT field devices for sensing and actuation, and (ii) a cloud-based IoT management platform grounded on OpenStack. **EdgeImpulse**⁶ supports Cloud-to-Edge ML training and can integrate *Arduino*⁷ MCU for inference tasks.

Ideally, the ML deployment framework should include several key characteristics: support for Incremental Learning (IL), FL, and capabilities for both training and inference from cloud to IoT environments. Additionally, an important requirement is that the framework’s installation size must be sufficiently compact to fit within the constraints of a serverless function runtime. Specifically, for AWS Lambda, this means the compressed package size must be kept under 50 MB.

A significant gap has been identified in this regard, as there is a lack of a comprehensive framework that encompasses all these essential features, which becomes evident when considering the capability for both training and

²Apache Kafka: <https://kafka.apache.org/>

³Apache CouchDB: <https://couchdb.apache.org/>

⁴Particle: <https://www.particle.io/>

⁵Arancino: <https://arancino.cc/>

⁶EdgeImpulse: <https://edgeimpulse.com/>

⁷Arduino: <https://www.arduino.cc/>

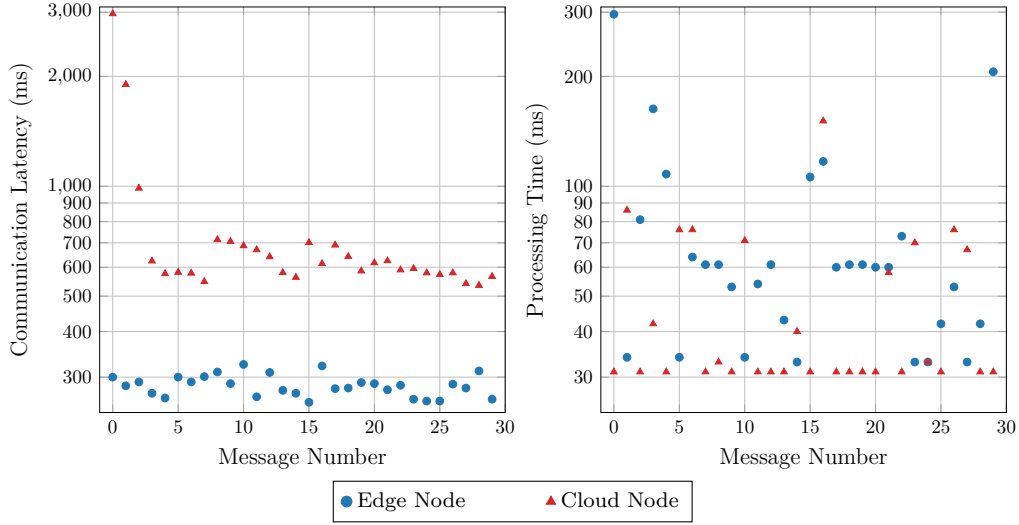


Figure 2.7: Latency and processing time for edge and cloud nodes over message sequence.

inference on IoT environments. For the deployment on Cloud-to-Edge, some libraries, such as **scikit-learn**⁸ can be easily integrated into a AWS Lambda function. However, larger frameworks such as **TensorFlow**⁹ and **PyTorch**¹⁰ exceed size limits and are not as easily integrated. **TensorFlow Lite** can be used for inference and fine-tuning on both cloud and edge devices, but its models can be run on IoT field devices only for inference.

2.4.2 Experimental Evaluation and Results

This section reports on a set of integration and communication tests conducted with the prototype described in Section 2.4. A simplified data collection task has been implemented to evaluate the messaging pipeline performance in terms of communication latency and node-level processing time.

The experimental setup begins with the configuration of both AWS IoT Core and AWS IoT Greengrass. This involves the registration of each

⁸scikit-learn: <https://scikit-learn.org>

⁹TensorFlow: <https://www.tensorflow.org/>

¹⁰PyTorch: <https://pytorch.org/>

device as a *Thing* in IoT Core, the provisioning of X.509 certificates for mutual Transport Layer Security (TLS) authentication, and the attachment of proper IoT policies to establish permissions. To authorize Peer-to-Peer (P2P) communication and access to other AWS services, such as S3, each device is also assigned appropriate Identity and Access Management (IAM) roles.

The second step involves configuring the edge device by installing the *Greengrass Core Device* software. Once the device appears on the AWS Console, a Greengrass deployment is initiated, including all essential components: Message Bus Server, Message Bus Client, Function Runtime, Local Storage, and the relevant Lambda functions.

The third setup step concerns the IoT devices, which run a custom firmware based on the Zephyr RTOS. Identical firmware is deployed across all twelve devices, differing only in authentication credentials and communication targets: six devices send messages directly to the cloud node (C_i , $i = 1, \dots, 6$) and six to the edge node (E_i , $i = 1, \dots, 6$).

Table 2.4: Average and standard deviation (in milliseconds) of communication latency and processing time per node.

	Metric	E1	E2	E3	E4	E5	E6	Edge
Communication	<i>Avg.</i>	293.80	287.80	293.00	279.60	274.00	279.60	284.63
Latency	<i>Std.Dev.</i>	25.06	21.39	12.08	18.42	14.47	21.89	19.19
Processing	<i>Avg.</i>	106.00	65.80	67.20	72.20	57.60	75.20	74.00
Time	<i>Std.Dev.</i>	110.21	30.86	9.39	52.17	30.75	73.46	57.55
	Metric	C1	C2	C3	C4	C5	C6	Cloud
Communication	<i>Avg.</i>	1111.00	877.60	691.40	631.00	602.00	558.80	745.30
Latency	<i>Std.Dev.</i>	1039.83	573.04	174.18	52.34	58.73	19.65	487.74
Processing	<i>Avg.</i>	48.00	80.40	33.60	38.80	33.20	47.20	46.87
Time	<i>Std.Dev.</i>	23.35	44.66	4.77	17.44	3.90	22.41	27.10

The edge and cloud nodes each run a Lambda function that stores incoming messages in local storage during the experiment. Every IoT device transmits one message per second for five seconds, reading onboard sensor data and publishing it through the message client. When a message is received, the

edge or cloud node immediately returns an acknowledgment on a dedicated topic via its Message Bus Client. The IoT device measures communication latency as the time between sending the message and receiving this first acknowledgment. A second acknowledgment is sent once the data has been successfully stored, enabling the device to compute processing time as the interval between the two acknowledgments.

Figure 2.7 and Table 2.4 present the experimental results. In the scatter plot, messages are ordered by the timestamp of the data message received from the IoT devices, as recorded by the respective Message Bus Server. The cloud node has experienced a substantially higher latency during the initial few messages it receives, which is likely due to the cold start behavior of Lambda functions. As expected, the edge node demonstrates significantly lower communication latency compared to the cloud. In contrast, the availability of computational resources on the cloud is reflected in its shorter processing times. In some cases, the edge node has achieved processing times comparable to the cloud, but with greater variance. Considering communication latency, the edge node tends to have a shorter overall turnaround time. These findings suggest the overall operational efficiency can be optimized through intelligent and context-aware task allocation. However, without careful management, task distribution may lead to suboptimal outcomes and even degrade performance compared to a purely cloud-based solution.

The experiments demonstrate the feasibility and benefits of executing distributed functions across the Cloud-to-Thing continuum, particularly in reducing communication latency and enabling localized processing. However, the current prototype focuses mainly on communication and orchestration, without incorporating ML workflows. These aspects, along with the current state of the art in on-device, federated, and incremental learning, are discussed in the next chapter.

Chapter 3

Explainable Federated On-Device Learning for the Semantic Web of Everything

AI, and particularly ML, has become integral to applications ranging from industrial automation to scientific research. Progress in ML has been driven by advances in model architectures, data availability, and computing infrastructure. However, most ML frameworks still assume centralized or homogeneous computing environments, limiting their applicability in *distributed* and *resource-constrained* contexts such as pervasive CPSs.

Several critical fields require that AI systems provide not only accurate predictions but also transparent and interpretable reasoning. As the size and complexity of ML models increase, their outcomes become less interpretable and harder to justify to human operators. In such contexts, ensuring *interpretability*, *transparency*, and *trustworthiness* becomes essential.

Despite the growing number of proposals for on-device learning and XAI, current approaches remain fragmented and incomplete. Systems typically support either IL or FL but rarely both. Likewise, explainability is often limited to post-hoc feature attribution techniques, whose computation is unfeasible on SWoE devices. Most critically, the integration of symbolic reasoning for interpretable explanations remains largely absent from state-of-the-art solutions.

This chapter introduces MAFALDA 2.0, an on-device, inherently explainable classifier for the SWoE that combines statistical training with symbolic representation, allowing classification problems to be reframed as semantic matchmaking tasks. Predictions are both computed and explained, using non-standard reasoning services over OWL ontologies.

3.1 On-Device Federated and Incremental Learning for Pervasive Cyber-Physical Systems

Chapter 2 has presented a Cloud-to-Thing framework enabling on-device CEI; however, a gap remains in algorithmic support for distributed reasoning and learning.

As described in Section 1.3, ML algorithms are typically designed for the computational, memory, and energy resources available in cloud environments, making their direct deployment on pervasive CPSs inefficient or infeasible [126]. Although models trained in the cloud can be optimized for execution on resource-limited devices [151], such approaches remain unsuitable for latency-critical or privacy-sensitive applications due to the limitations identified in Chapter 2, namely the inability of the network to handle large-scale data streams, the latency involved in the inter-node communication, and the risks associated with uploading sensitive information.

Edge computing has been introduced in Section 1.2.2 to mitigate these limitations by enabling localized data processing. Empirical studies illustrate this trade-off: heuristic-based algorithms can execute on edge devices with acceptable response times, while hybrid cloudedge ML architectures employing CNN and LSTM units improve accuracy but remain constrained by device computational capacity [127]. Lightweight deep learning models deployed on microcontrollers further demonstrate the feasibility of on-device inference [84].

Multiple frameworks have been proposed to enable on-device training and inference without centralized coordination. *Edge2Train* [166] supports the

execution of SVMs on microcontrollers, while its successors, *Train++* [168] and *ML-MCU* [167], extend this capability through online self-learning for real-time data streams. The lightweight *AIfeS* framework [197] allows the deployment of fully connected and convolutional neural networks on embedded platforms such as the *Nordic nRF52840*, whereas *X-Cube-AI* [43] enables incremental training on *STM32* devices for anomaly detection.

Applications in industrial and activity recognition domains further demonstrate the feasibility of embedded learning. Edge anomaly detection frameworks integrate IoT and Tiny Machine Learning (TinyML) components, sometimes combined with blockchain for enhanced traceability [9], while sensor-integrated platforms support on-device training directly on inertial processing units [36].

Given this, on-device training represents a critical step towards reducing dependence on centralized cloud infrastructure while preserving data confidentiality [126]. In this domain, IL and FL have emerged as complementary paradigms that distribute the computational burden of training across spatial and temporal dimensions, respectively.

IL (a.k.a. online learning) enables model adaptation through sequential updates as new data become available, eliminating the need for complete retraining. This approach is particularly suitable for pervasive CPSs, where data distributions evolve under dynamic environmental or operational conditions, and reduces the need for large-scale local data storage and supports continuous model refinement.

IL has been applied to domain-specific tasks such as heart rhythm classification [158] and face recognition on MCUs [41]. Recent advances in Lifelong Incremental Learning (LIL) [5] seek to maintain continuous model adaptation while mitigating catastrophic forgetting. Hybrid approaches that combine transfer learning with kNN classifiers have been demonstrated on *Raspberry Pi* and *STM32* platforms for image and audio recognition [48].

Conversely, FL enables multiple edge or embedded devices to collaboratively

train a shared global model without exchanging raw data [198]. Each client performs local optimization and transmits model updates, typically parameter gradients or weight deltas, to a coordinating entity, which aggregates them using algorithms such as Federated Averaging [170].

A step towards fully decentralized learning is represented by *TinyFL*, which implements a leaderfollower coordination protocol among MCUs to enable federated training without cloud infrastructure [198].

While all these techniques have advanced significantly, integrating XAI into federated and incremental paradigms remains an open research challenge. Most on-device FL frameworks therefore prioritize computational efficiency over explainability, and global model aggregation further limits interpretability. Recent advances in federated XAI intrusion detection systems combine distributed training with local feature-relevance explanations, showing that interpretable and privacy-preserving learning is feasible, though still challenging for highly resource-constrained devices [55].

Explainability is increasingly recognized as essential in domains such as healthcare, where transparent models enable clinicians to extract meaningful clinical insights from sensor and patient data [149]. A TinyML healthcare architecture [11] implements training and inference using TensorFlow Lite on a *Raspberry Pi*, employing post hoc interpretability methods such as SHAP values and the Wasserstein distance. Complementary approaches explore symbolicstatistical integration, combining formal reasoning with data-driven models to enhance interpretability in hybrid, gray-box models [28].

When analyzing specifically explainable on-device AI, these solutions predominantly employ DT models due to their interpretability and low computational cost. Examples include a self-supervised learning system implemented on a *NUCLEO-H743Z12* MCU, where an adaptive engine periodically retrains k -means, kNN, or DT classifiers [148]. Likewise, decentralized healthcare monitoring frameworks integrate local DT inference with distributed RF consensus and blockchain-backed rule encoding to ensure model integrity [159]. Complementing these efforts, the *TinyDecisionTreeClassifier* library [76] provides

a C++ implementation for embedded training and deployment, validated across multiple MCUs and datasets.

The literature shows that applying IL, FL, and XAI in resource-constrained settings is feasible. However, combining these techniques into a unified framework remains a major challenge, one that the approach proposed here seeks to address.

3.2 Design of an Explainable On-Device Learning Framework

MAFALDA 2.0 classifier addresses all key requirements for embedded on-device XAI, thereby supporting semantic XAI and integration within pervasive CPSs and the IoE as defined in Section 1.6:

- it is implemented in C, with particular attention to time and memory efficiency, allowing both training and inference to run entirely on constrained devices;
- IL is supported through a compact internal data structure that enables iterative model updates from data streams without storing the entire dataset;
- FL is supported by design, as intermediate data structures can be exchanged and merged across devices and made interpretable through a shared OWL ontology;
- the model combines semantic inference with traditional AI methods, ensuring explainability and providing interpretable representations of the models and their outcomes;
- the adoption of OWL for feature and label encoding enables these explanations to align with standard vocabularies and shared ubiquitous KBs.

MAFALDA 2.0 borrows general ideas from [144], such as the use of KRR techniques and automated reasoning for constructing a *glassbox* ML classifier. Compared with the previous framework, MAFALDA 2.0 is redesigned from scratch to achieve higher classification accuracy and computational efficiency.

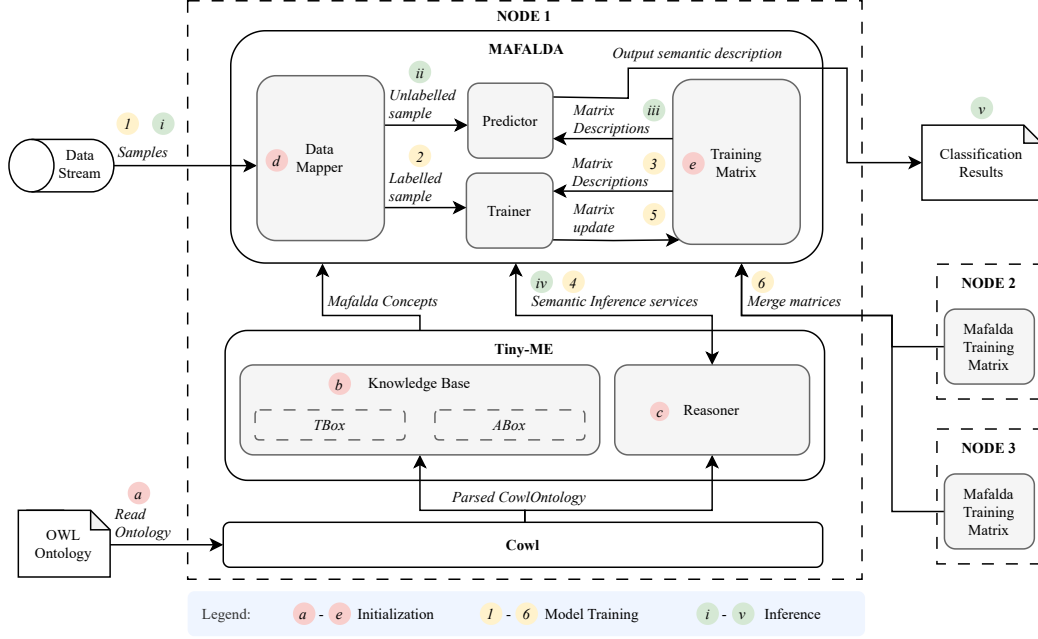


Figure 3.1: MAFALDA 2.0 Architecture

The architecture of MAFALDA 2.0 is shown in Figure 3.1. Ontology parsing, serialization, and management are handled by Cowl [20], while reasoning services are provided by Tiny-ME [141], and *NanoPB*¹ is used for serialization and deserialization. Both tools are introduced in Section 1.6 and Section 1.5. From a technical standpoint, MAFALDA 2.0 is implemented as a modular ML C library with minimal dependencies, facilitating integration across a broad range of platforms and systems. It also includes a stand-alone Command Line Interface (CLI) for convenience, allowing easy evaluation on desktop systems.

The core idea underlying MAFALDA 2.0 is to transform numerical input samples into symbolic representations and frame the classification process as

¹<https://jpa.kapsi.fi/nanopb/>

a semantic matchmaking task, where non-standard reasoning tasks determine outcomes by identifying the closest semantic match.

The system architecture comprises the following components:

- **Training Matrix (TM):** an n -dimensional data structure that maintains a compact, aggregated representation of the training dataset, capturing the most representative semantic descriptions;
- **Data Mapper:** converts numerical input values into symbolic descriptions (concepts or semantic statements) defined with respect to the reference ontology;
- **Trainer:** during training, it receives semantic descriptions from the Data Mapper and incrementally updates the TM;
- **Predictor:** during inference, it classifies unseen samples by comparing their semantic descriptions with those stored in the TM and selecting the most similar output class.

MAFALDA 2.0 follows a typical ML pipeline, as outlined in Section 1.3. It consists of a training phase, where the model learns from data, and an inference phase, where the trained model produces predictions. However, MAFALDA 2.0 extends this structure by training incrementally on a sample-by-sample basis. In particular, Figure 3.1 shows how the workflow is organized into three main phases: ontology parsing and model initialization (red markers ae), training (yellow markers 16), and inference (green markers iv). Initialization occurs once when MAFALDA 2.0 is loaded, whereas training and inference are executed incrementally for each incoming sample.

The process begins with ontology parsing (a), where the domain ontology is loaded using Cowl, producing a `CowlOntology` object. This object is then used to instantiate (b) the Tiny-ME KB and (c) its associated reasoner. These components initialize (d) the Data Mapper and (e) the TM, whose dimensions are determined by the underlying knowledge base. Ontology parsing is executed once, when MAFALDA 2.0 is first instantiated.

During training, each labeled sample received from a data stream (1) is transformed into a semantic description by the Data Mapper (2). The Trainer integrates this description into the current TM (3), using ontology-based reasoning (4) to compute the appropriate update (5). FL is supported by design (6), enabling the aggregation of multiple training matrices from distributed sources.

At inference time, an unlabeled sample (i) is similarly mapped to a semantic description (ii), which is then processed by the Predictor (iii). Classification is performed by comparing the input against the semantic profiles in the TM, applying reasoning (iv) to identify the best-matching output class, and to generate an explanation (v).

Both training and inference operate on a sample-by-sample basis, inherently supporting IL. MAFALDA 2.0 does not impose any constraint on the order of samples. Any number of training samples can be followed by any number of unlabeled samples, and vice versa. These three phases are described and analyzed in detail in the following subsections.

3.2.1 Ontology schema and data annotation

Unlike conventional ML approaches that operate solely on numerical feature vectors, MAFALDA 2.0 uses KRR languages to map raw input data into OWL ontology concepts. The ontology defines the conceptual hierarchies and relations within the target domain. Each feature extracted from the training dataset is associated with a corresponding concept in the $\mathcal{T}$, thus enabling symbolic encoding of inputs and outputs aligned with standard Semantic Web formalisms.

To ensure compatibility with the framework, MAFALDA 2.0 defines constraints on the axioms that must be declared within the ontology. Reserved OWL entities are identified using the dedicated namespace:

`mafalda:` <http://swot.sisinflab.poliba.it/mafalda#>

The constraints on the Terminological Box (TBox) are:

- Dataset features and output labels are represented as `owl:Classes`:
 - Each independent variable (feature), such as *Temperature* or *Week-Day*, is denoted as a class $\mathcal{F}$ with `owl:subClassOf mafalda:Input`;
 - Each dependent variable (label), such as *Season*, is denoted as a class $\mathcal{L}$ with `owl:subClassOf mafalda:Output`.
- Each possible label outcome (e.g., *Summer*, *Winter*) is modeled as a class C^y , declared as a subclass of the corresponding label superclass $\mathcal{L}$. The class name must match its representation in the dataset.
- For categorical features (e.g., *WeekDay*), each category (e.g., *Monday*, *Tuesday*) is defined as a class D^x with `owl:subClassOf  $\mathcal{F}$` . Class names must correspond to the values used in the dataset.
- Numerical features (e.g., *Temperature*) are represented through subclass partitions of $\mathcal{F}$. Each subclass (e.g., *LowTemperature*, *HighTemperature*) corresponds to a predefined value range, specified using the annotation properties `mafalda:minValue` and `mafalda:maxValue`. These range-based classes are declared as `owl:subClassOf` the parent feature class $\mathcal{F}$.

For generality, the notation C^x denotes any named `owl:Class` representing a feature value, which may correspond to either a categorical class D^x or a numerical class N^x . After data mapping, both categories are handled equally within MAFALDA 2.0.

The $\mathcal{T}$ can be constructed from expert-defined knowledge, standard domain vocabularies, technical specifications, external KBs, Linked Data sources [196], or derived through data-driven approaches such as clustering and feature selection [132]. Within the SWoE domain, this KB can be derived from the disseminated knowledge of the u- KB layer.

Starting from this ontology, MAFALDA 2.0 allocates the required internal data structures and transforms streams of data into $\mathcal{ALN}$ concept descriptions.

The data mapping procedure is conceptually similar to discretization by binning but extends this with formal semantic annotations defined using standard OWL constructs.

A labeled sample (or training instance) is defined as a pair $\langle \mathbf{x}, \mathbf{y} \rangle$, where $\mathbf{x} = [x_1, \dots, x_n]$ is the feature vector and $\mathbf{y} = [y_1, \dots, y_m]$ is the output label vector. An unlabeled sample (or prediction instance) is denoted as $\langle \mathbf{x} \rangle$, containing only the feature vector.

Each element x_i in the feature vector $\mathbf{x}$ represents either a numerical or a categorical feature. Numerical features are defined by a feature name corresponding to a parent class in the ontology and an associated scalar value. This value is mapped to the most specific subclass whose range is specified via the `mafalda:minValue` and `mafalda:maxValue` annotation properties. If either bound is undefined, the feature is treated as unbounded in that direction.

Categorical features consist of a feature name and a category label. Mapping is performed by associating the label with the corresponding ontology class D^x , defined as a subclass of the relevant feature class. Each output label y_i in $\mathbf{y}$ is similarly mapped to a class C^y declared within the `mafalda:Output` hierarchy, since MAFALDA 2.0 operates exclusively in a classification setting.

According to the OWA (see Section 1.5), if a feature value is missing, the corresponding x_i is not mapped to any ontology concept.

Each sample is annotated as a semantic description in $\mathcal{ALN}$, represented as the conjunction of concepts corresponding to its feature values. Formally, the semantic representation of a sample is defined as:

$$S \equiv S^x \sqcap S^y \equiv C_1^x \sqcap \dots \sqcap C_f^x \sqcap C_1^y \sqcap \dots \sqcap C_l^y$$

where S^x denotes the conjunction of concept expressions derived from the feature vector $\mathbf{x}$, and S^y denotes those derived from the label vector $\mathbf{y}$.

When the sample is used for training, both S^x and S^y are present. During inference, only the feature vector is available, and the label component is empty, i.e., $S^y \equiv \top$.

Table 3.1: Example dataset with two features and one label.

Temperature (Celsius)	DayType	Season
28	Weekend	Summer
4	Weekend	Winter
22	Weekday	Summer
2	Weekday	Winter
10	Weekday	Summer
27	Weekday	Summer

Table 3.1 reports a synthetic dataset with two features (*Temperature* and *DayType*) and one label (*Season*), which serves as a running example in the subsequent discussion. The *Temperature* feature is discretized into two intervals: **LowTemp** for values in $[0, 15]$ and **HighTemp** for values in $[15, 30]$.

As illustrative examples, the sample (28, Weekend, Summer) is mapped to the semantic description:

HighTemp $\sqcap$ **Weekend** $\sqcap$ **Summer**

whereas the sample (4, Weekend, Winter) is mapped to:

LowTemp $\sqcap$ **Weekend** $\sqcap$ **Winter**

The mapped samples are used for either training or inference, depending on whether they originate from a labeled or unlabeled case. Training samples are passed to the trainer module, which updates the TM appropriately. At inference time, the predictor module uses the knowledge encoded in the TM to generate concept descriptions for each possible outcome and then applies non-standard inference services to find the closest match to the input sample.

3.2.2 Incremental and federated learning

In MAFALDA 2.0, training consists of incrementally constructing and storing semantic descriptions, referred to as *profiles*, *concepts*, or *concept descriptions*, for each output class, based on the samples seen from the training set. When an unlabeled sample arrives, its semantic description is compared with the stored profiles to determine the closest, which then defines the classification outcome.

The issue with this approach is that storing a full semantic description for every training sample is infeasible. This holds even for small datasets in on-device learning scenarios due to memory and computational constraints. To address this, MAFALDA 2.0 introduces an intermediate data structure, the Training Matrix (TM), which maintains a compact, aggregated representation of the training set.

The behavior of the TM is governed by the following hyperparameters:

- d : the number of representative semantic descriptions maintained per output class C^y defined in the ontology $\mathcal{T}$;
- w : the size of the observation window used to retain recent samples;
- f : the falloff factor controlling the temporal decay of older observations.

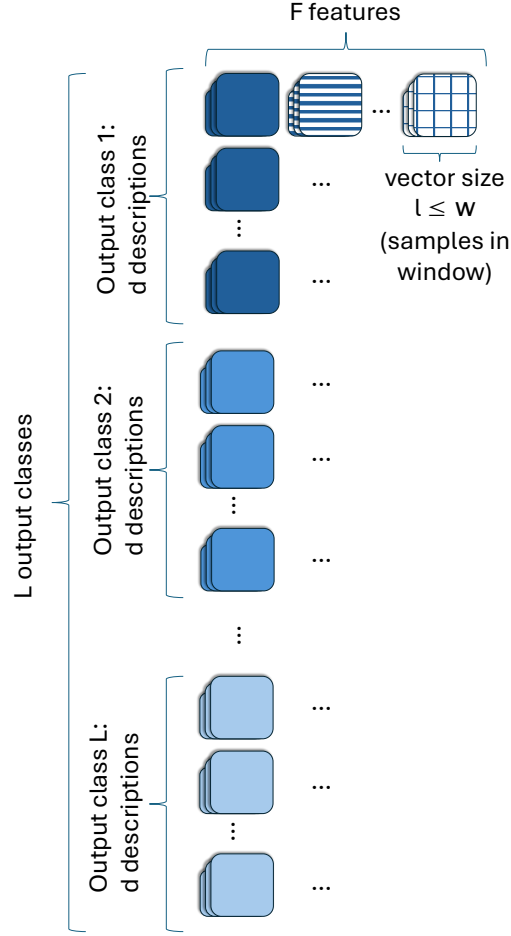


Figure 3.2: Training Matrix structure.

As illustrated in Figure 3.2, the TM is a three-dimensional matrix $\mathbf{M}$ of size $(L \cdot d) \times F \times w$, where:

- F is the total number of feature classes C^x defined in the ontology, corresponding to all possible symbolic representations of input features;
- d is the number of semantic descriptions maintained per output class;
- L is the number of distinct output label classes C^y defined in the ontology.

The total number of profiles stored in the TM is equal to the number of rows, that is $(L \cdot d)$.

Each matrix element $\mathbf{M}_{k,j}$ is a vector of length l , defined as:

$$\mathbf{M}_{k,j} = [M_{k,j,1}, M_{k,j,2}, \dots, M_{k,j,h}] \quad \text{with} \quad M_{k,j,h} \in \mathbb{N}$$

These vectors encode the temporal evolution of feature-class occurrences over a fixed observation window of size w .

When a new labeled training sample $S \equiv S^x \sqcap S^y$ is received, MAFALDA 2.0 invokes Algorithm 1 to incorporate the corresponding semantic description into the matrix $\mathbf{M}$.

The algorithm iterates over each label concept C^y in S^y , retrieves its associated index i , and updates a count vector that tracks how many times each label has been observed. Concept indices are precomputed during ontology parsing and stored in a compact hash map, which assigns a unique integer identifier to each relevant class and thereby allows identification of matrix positions.

Given index i , the algorithm selects an appropriate semantic profile k within the range $[i \cdot d, (i + 1) \cdot d)$. If an unused profile slot is available, the sample is assigned to the first empty row. Otherwise, it identifies the row within the label class group whose existing profile has the highest semantic similarity to the current sample. The similarity metric used for this selection is the same as that employed during inference, that is the Equation 3.6 described in Section 3.2.3.

Once the appropriate description row has been selected, its index is denoted k . The algorithm then iterates over all feature concepts C^x in S^x , retrieves their corresponding ontology index j , and updates the TM by incrementing $M_{k,j,1}$. Each value $M_{k,j,1}$ thus records the number of times the j -th concept has appeared in the most recent observation assigned to the k -th profile.

To model temporal dynamics, the algorithm verifies whether the count of observations for the i -th label exceeds the window size w . If so, it performs a right-shift operation along the temporal axis: each element $M_{k,j,h}$ is reassigned

Algorithm 1: Training Matrix Update

Input: S^x (features) and S^y (labels)
Data: M (Training Matrix), vector $Count$ of size l
foreach concept C^y in S^y **do**
 $i \leftarrow \text{indexOfConcept}(C^y)$
 $Count_i \leftarrow Count_i + 1$
 // Find the description to update
 if $\text{hasEmptyDescription}(M, i, d)$ **then**
 $k \leftarrow \text{firstEmptyDescription}(M, i, d)$
 else
 $k \leftarrow \text{getNearestDescription}(S^x, M, i, d)$
 // Add all concepts to the corresponding TM vector
 foreach concept C^x in S^x **do**
 $j \leftarrow \text{indexOfConcept}(C^x)$
 $M[k, j, 1] \leftarrow M[k, j, 1] + 1$
 // Shift Training Matrix if count exceeds window w
 if $Count_i \geq w$ **then**
 for $k \leftarrow i \cdot d$ **to** $(i \cdot d) + d - 1$ **do**
 for $j \leftarrow 1$ **to** F **do**
 for $h \leftarrow l - 1$ **to** 2 **do**
 $M[k, j, h] \leftarrow M[k, j, h - 1]$
 $M[k, j, 1] \leftarrow 0$
 $Count_i \leftarrow 0$

to $M_{k,j,h+1}$ for $1 \leq h < l$. The last entry is discarded, and a zero is inserted at the head of the vector.

This mechanism maintains a bounded history of recent observations, where the first element of each vector $\mathbf{M}_{k,j}$ corresponds to the most recent samples, and the last element encodes the oldest. This algorithm’s complexity is determined by the search for the description to update and the matrix-shifting mechanism. Specifically, for each input concept in S^y , the retrieval phase must generate and compare d candidate descriptions. Previously generated descriptions are cached and reused to improve performance. In steady-state operation, only the single modified description requires per-sample regeneration; however, the complexity analysis assumes the worst-case scenario (*e.g.*, system initialization) where all d descriptions must be generated. The terms $\mathcal{X}_{gen}$ and $\mathcal{X}_{cmp}$ represent the total computational costs of all the Tiny-ME reasoning services invoked for, respectively, generating a description and retrieving the semantic distance between two descriptions. Since Tiny-ME exploits the $\mathcal{ALN}(D)$ logic, these operations exhibit polynomial complexity w.r.t. the length of concept descriptions and the depth of the concept hierarchy in $\mathcal{T}$, ensuring the overall problem remains computationally tractable [141]. Consequently, the retrieval step scales as $\mathcal{O}(d \cdot (F \cdot \mathcal{X}_{gen} + \mathcal{X}_{cmp}))$. Subsequently, the shift operation traverses the entire matrix block with a complexity of $\mathcal{O}(d \cdot F \cdot l)$. The total time complexity is thus $\mathcal{O}(|S^y| \cdot d \cdot (F \cdot (\mathcal{X}_{gen} + l) + \mathcal{X}_{cmp}))$. Crucially, because the dimensions d, F, l are fixed at initialization, this cost is independent of the cumulative number of processed samples.

Table 3.2: Training matrix at the fourth sample of the example dataset

Season	Temperature		DayType	
	LowTemp	HighTemp	Weekend	Weekday
Winter1	[1,0]	[0,0]	[0,0]	[1,0]
Winter2	[1,0]	[0,0]	[1,0]	[0,0]
Summer1	[0,0]	[1,0]	[0,0]	[1,0]
Summer2	[0,0]	[1,0]	[1,0]	[0,0]

As shown in Table 3.2, the TM after processing the fourth sample from Table 3.1 contains two semantic profiles for each output class (*Winter* and

Summer). The matrix is configured with depth $d = 2$, window size $w = 3$, and stores two representative descriptions per class.

Table 3.3: Training matrix after

Season	Temperature		DayType	
	LowTemp	HighTemp	Weekend	Weekday
Winter1	[1,0]	[0,0]	[0,0]	[1,0]
Winter2	[1,0]	[0,0]	[1,0]	[0,0]
Summer1	[0,1]	[0,1]	[0,0]	[0,2]
Summer2	[0,0]	[0,1]	[0,1]	[0,0]

Since no row is empty, the fifth training sample, labeled as *Summer*, must be assigned to the most similar existing *Summer* profile. To determine the appropriate row, the semantic description R^k is generated for each candidate using Algorithm 2, and the semantic similarity score (defined in Section 3.2.3 in Equation 3.6) is computed. The row with the lowest score is selected for the update, which is determined to be *Summer1*. The resulting TM is shown in Table 3.3.

Table 3.4: Training matrix at the fourth sample of the example dataset

Season	Temperature		DayType	
	LowTemp	HighTemp	Weekend	Weekday
Winter1	[1,0]	[0,0]	[0,0]	[1,0]
Winter2	[1,0]	[0,0]	[1,0]	[0,0]
Summer1	[0,1]	[0,1]	[0,0]	[0,2]
Summer2	[0,0]	[0,1]	[0,1]	[0,0]

The new sample is inserted at the head of the selected vector, causing the *Summer* profile to exceed its observation window and triggering a shift operation. The updated matrix state after incorporating also the sixth sample in *Summer1* is shown in Table 3.4.

Each vector $\mathbf{M}_{k,j}$ can be expressed as a scalar $M_{k,j}$ defined as follows:

$$M_{k,j} = M_{k,j,1} + \sum_{n=2}^l \frac{1}{f \cdot (n-1)} \cdot M_{k,j,n} \quad (3.1)$$

where f is the *falloff* parameter that controls the exponential decay applied to older observations. A higher value of f increases the rate at which past contributions diminish.

For example, consider the value of the *Summer1* row corresponding to the *HighTemp* feature, assuming a falloff factor $f = 2$:

$$M_{3,2} = 1 + \frac{1}{2} \cdot 1 = 1.5 \quad (3.2)$$

Starting from the scalar value $M_{k,j}$, a normalized value $\hat{M}_{k,j}$ is defined. This value represents the relative frequency of the j -th concept within the k -th profile relative to its feature superclass. The normalized value is computed as:

$$\hat{M}_{k,j} = \frac{M_{k,j}}{\sum_{j' \in C_j^{\mathcal{F}}} M_{k,j'}} \quad (3.3)$$

where $C_j^{\mathcal{F}}$ denotes the set of feature classes sharing the same parent class $\mathcal{F}$ as the j -th concept.

This normalization ensures that contributions are interpreted relative to their feature group and avoids bias from features that are observed with different frequencies.

As a continuation of the previous example, the normalized value for the *Summer1* row with respect to the *HighTemp* feature is computed as:

$$\hat{M}_{3,2} = \frac{M_{3,2}}{M_{3,1} + M_{3,2}} = \frac{1.5}{1.5 + 0.5} = 0.75 \quad (3.4)$$

The training algorithm operates online, in a sample-by-sample fashion, enabling IL without requiring storage of the full dataset. The resulting TM serves as the internal representation of the model and is later used for inference, as detailed in Section 3.2.3.

Compared to the approach described in [144], the TM has evolved from a two-dimensional to a three-dimensional structure, explicitly supporting sliding-window updates over data streams. This extension facilitates incremental lifelong learning and introduces temporal dynamics into the model.

Moreover, while the original algorithm maintained a single semantic profile per output class, MAFALDA 2.0 generalizes this behavior through the d parameter, allowing multiple concurrent descriptions to capture intra-class variability.

MAFALDA 2.0 also supports FL scenarios by enabling distributed instances to exchange and merge their internal TMs.

Consider two instances with TMs $\mathbf{M}_1$ and $\mathbf{M}_2$. The first matrix, $\mathbf{M}_1$, is configured with d_1 descriptions, depth l_1 , and observation window w_1 ; the second, $\mathbf{M}_2$, with d_2 descriptions, depth l_2 , and window w_2 . Merging these produces a new matrix $\mathbf{M}_3$ with $d_1 + d_2$ rows, depth l_2 , and window size w_2 . Rows associated with the same output class are kept contiguous, preserving compatibility with the training and inference procedures.

If $l_1 < l_2$, the vectors in $\mathbf{M}_1$ are extended by appending zeros to match the required depth. Conversely, if $l_1 > l_2$, the system may either discard older observations or increase the receiving instance’s depth to l_1 , depending on configuration parameters. Regardless of depth adjustments, the matrix update logic and count vectors used to manage temporal shifts and recurrence tracking, as defined in Algorithm 1, remain unchanged after merging.

Since numeric indexing is arbitrary and depends on the order in which the axioms are parsed from the ontology, rows and columns are indexed by IRIs in the serialized form, which are fixed identifiers. This allows the receiving end to reconcile the indices with its internal representation. It also supports small differences in the ontologies: if a concept is absent on the receiving side, the training matrix can still be merged, and the missing information is simply discarded.

3.2.3 Inference and explainability

As introduced earlier, MAFALDA 2.0 performs classification based on the TM constructed during training. For each unlabeled input, a semantic description is generated through the feature mapping process described in Section 3.2.1.

This description is then compared against the reference profiles stored in the TM to identify the most similar representation.

Explainability is an intrinsic property of the classification process. It arises from (i) the symbolic structure of the model itself, namely, the profiles associated with each output class, and (ii) the application of non-standard reasoning services for semantic matchmaking, as detailed in Section 1.5. These services return a set of profiles, namely, G , K , and H , which represent the incompatible, compatible, and missing concepts between two profiles. These elements enable the system to justify classification outcomes in an interpretable form.

In detail, MAFALDA 2.0 derives symbolic representations from the TM by generating a semantic description for each stored profile. The description associated with the k -th row of the matrix $\mathbf{M}$ is denoted by R_k and is computed using Algorithm 2.

This process is controlled by a model hyperparameter $\theta \in (0, 1]$, which defines the minimum normalized contribution $\hat{M}_{k,j}$ required for a feature concept to be included in R_k . Higher values of θ impose stricter inclusion criteria, producing more compact descriptions that retain only the most frequent concepts. Lower values yield longer descriptions that include less frequent (and potentially noisy) features that may not be representative of the reference profile. The choice of θ is task-dependent and typically requires empirical tuning.

Algorithm 2 begins by initializing R_k as an empty semantic description. It then iterates over all feature indices j , computing the corresponding normalized matrix value $\hat{M}_{k,j}$. If $\hat{M}_{k,j} > \theta$, the associated ontology concept C^x is retrieved and added to a temporary candidate description R_{tmp} .

Adding each C^x concept to the profile without checks may lead to inconsistent profiles. This requires a procedure to gradually incorporate concepts while avoiding conflicts during generation. First, a concept *weight* is defined. Specifically, the weight of a concept R_k with respect to the k -th profile is

Algorithm 2: Generate a semantic description for the k -th TM row

Input: k (index of the row to map to a semantic description)

Data: M (Training Matrix k -th row), θ (threshold)

Result: R_k (semantic description of the k -th row)

$R_k \leftarrow \top$

for $j = 1$ *to* F **do**

$\hat{M}_{k,j} \leftarrow \frac{M_{k,j}}{\sum_{j' \in C_j^F} M_{k,j'}}$

if $\hat{M}_{k,j} > \theta$ **then**

$C^x \leftarrow \text{getConcept}(j)$

// Store the concept in a temporary semantic
description

$R_{tmp} \leftarrow C^x$

$\langle G, K \rangle \leftarrow \text{conceptContraction}(R_{tmp}, R_k)$

if $G \equiv \top$ **then**

$R_k \leftarrow R_k \sqcap R_{tmp}$

else if $\text{weight}(k, G) \leq \text{weight}(k, R_{tmp})$ **then**

$R_k \leftarrow K \sqcap R_{tmp}$

return R_k

defined as the maximum normalized matrix value among the concepts included in the description:

$$\text{weight}(k, R_k) = \max_{C^x \in R_k} \hat{M}k, j \quad (3.5)$$

where:

- C^x is a concept included in the semantic description R_k ;
- j is the index of C^x in the feature dimension of row k in the matrix M ;
- $\hat{M}k, j$ is the normalized value as defined in Equation 3.3.

If a concept C^x is not present in the k -th row, its weight is assumed to be zero. The weight measures the concept importance w.r.t. the k -th semantic profile.

Before adding R_{tmp} to R_k , a concept contraction is performed between R_{tmp} (the *resource*) and the current R_k (the *request*). This operation yields two components: G (the conflicting part) and K (the consistent remainder). The algorithm then incorporates into R_k only the most significant and compatible concepts. Specifically:

- If $G \equiv \top$, the candidate description R_{tmp} is fully compatible and added to R_k without modification;
- Otherwise, the algorithm selects the most significant part to include in R_k between K and R_{tmp} :
 - If the weight of the conflicting part G is lower than that of R_{tmp} , the updated description is set to $K \sqcap R_{\text{tmp}}$, discarding the incompatible component G ;
 - Otherwise, since the weight of G is higher than or equal to that of R_{tmp} , adding R_{tmp} would remove a more significant part of the description; thus, no change is made and $R_k \equiv G \sqcap K$ is retained.

After all feature columns have been processed, the resulting R_k is returned as the final semantic description for the profile.

The complexity of Algorithm 2 is primarily determined by the dimensionality of the feature space F and the computational cost of the underlying semantic reasoning services. The algorithm traverses the F columns of the TM to compute the normalized values. For features exceeding the threshold θ , the algorithm invokes the concept contraction reasoning service and auxiliary description manipulation functions within Tiny-ME. Let $\mathcal{X}_{gen}$ denote the computational cost of these logic operations (as defined in Section 3.2.2). As previously mentioned, since Tiny-ME operates on the $\mathcal{ALN}(D)$ Description Logic subset, $\mathcal{X}_{gen}$ is polynomial with respect to the $\mathcal{T}$ structure [141], ensuring that the reasoning remains tractable. In the worst-case scenario, where all features are deemed relevant, this semantic integration is triggered at every step. Consequently, the total time complexity is bounded by $\mathcal{O}(F \cdot \mathcal{X}_{gen})$. Similar to the training phase, the feature space F is fixed at initialization, guaranteeing bounded execution times.

Table 3.5: Normalized training matrix

	Temperature		DayType	
Season	LowTemp	HighTemp	Weekend	Weekday
Winter1	1	0	0	1
Winter2	1	0	1	0
Summer1	0.25	0.75	0	1
Summer2	0	1	1	0

Continuing the example from the TM in Table 3.4, the normalized values of the training matrix are shown in Table 3.5. For a threshold $\theta = 0.5$, the semantic description generated for the *Summer1* profile is:

$$\text{HighTemp} \sqcap \text{Weekday}$$

Without any semantic check, reducing the threshold to $\theta = 0.2$ results in additional concepts being included, yielding:

$$\text{HighTemp} \sqcap \text{LowTemp} \sqcap \text{Weekday}$$

However, assuming that $\mathcal{T} \models \text{LowTemp} \sqcap \text{HighTemp} \sqsubseteq \perp$, this description is inconsistent.

In such cases, Algorithm 2 resolves conflicts by iteratively evaluating and selectively incorporating feature concepts. For the *Summer1* profile, the initial description is $R_3 \equiv \text{LowTemp}$. When the algorithm attempts to add $R_{\text{tmp}} \equiv \text{HighTemp}$, concept contraction returns $G \equiv \text{HighTemp}$ and $K \equiv \top$. The corresponding weights are 0.25 for R_{tmp} and 0.75 for G , so the current description is replaced with $R_3 \equiv \text{HighTemp}$ because the conflicting term is more significant.

In the following step, the concept **Weekday** is evaluated. Since it is fully compatible with the current description ($G \equiv \top$), it is added directly. The final semantic profile is thus:

$$R_3 \equiv \text{HighTemp} \sqcap \text{Weekday}$$

Classification in MAFALDA 2.0 relies on a semantic similarity metric computed between descriptions. The proposed metric combines the non-standard inference services described in Section 1.5 with the CNF Norm, which for a generic concept D is denoted as $\|D\|$, with $\|D\| \in \mathbb{R}$. This value correlates with the number of concepts in the unfolded description of D [140].

Given an input description S^x and a profile R_k , the similarity score is defined as:

$$s(S^x, R_k) = \left(\frac{\|G'\| + \|H'\|}{2\|S^x\|} \right) + \left(\frac{\|G''\| + \|H''\|}{2\|R_k\|} \right) \quad (3.6)$$

The score comprises two terms, each quantifying semantic similarity via concept contraction and abduction. In the first term, G' and H' are obtained by applying contraction and abduction with S^x as the request and R_k as the resource. The numerator, $\|G'\| + \|H'\|$, measures the number of missing or conflicting concepts relative to S^x , and the normalization factor $2\|S^x\|$ ensures that the term lies in $[0, 0.5]$. If S^x and R_k are fully compatible, the numerator is zero. If they are disjoint, the term evaluates to 0.5.

The second term is computed analogously, but with S^x and R_k swapped.

This accounts for the asymmetry of non-standard reasoning, since concept contraction and abduction are not commutative operations. This term is bounded by $[0, 0.5]$ as well.

The resulting similarity score $s(S^x, R_k)$ lies in the range $[0, 1]$, where 0 indicates a perfect semantic match and higher values denote increasing dissimilarities. This score is used during training to identify the closest semantic profile for update and during inference to determine the best-matching output class. The term $\mathcal{X}_{cmp}$ (introduced in Section 3.2.2) denotes the computational cost of Tiny-ME associated with this formula. Specifically, calculating $s(S^x, R_k)$ requires executing two concept contraction and two abduction tasks, followed by the computation of six semantic description norms.

Given an unlabeled sample $S \equiv S^x$, the classification process proceeds as follows:

1. **Iterate over each label superclass $\mathcal{L}$** , corresponding to subclasses of `mafalda:Output` as defined in Section 3.2.1.
2. **For each subclass $C^y \sqsubseteq \mathcal{L}$** , retrieve all k -th rows in the Training Matrix associated with that label class.
3. **For each row index k** corresponding to a description linked to C^y :
 - (a) Generate the semantic description R_k using Algorithm 2;
 - (b) Compute the semantic similarity score $s(S^x, R_k)$ as defined in Equation 3.6.
4. **Assign the predicted label** for superclass $\mathcal{L}$ as the subclass C^y whose corresponding row R_k yields the lowest score. This label is added to the output concept S^y .

After completing this procedure, the vector S^y contains the predicted labels for the input sample S^x . This workflow generalizes the approach presented in [144], extending it to support both multi-class and multi-label classification.

Beyond prediction, the classification procedure supports both global and local explainability. The semantic description R_k reflects the knowledge extracted from the training data, where each concept is ranked by its normalized contribution in the TM. During inference, the reasoning components, namely G' , K' , H' , and H'' , capture how and why an input S^x was classified in a certain way. Specifically:

- G' and G'' identify incompatible features that must be discarded to enable a match (i.e., the *give up* components in concept contraction);
- H' and H'' identify concepts missing from S^x or R_k , respectively, that would be required to complete a full match (i.e., the *hypotheses* in concept abduction).

Since this analysis is performed for all candidate rows R_k , it enables introspection into both the decision-making process and the internal model representation. This supports not only the justification of outcomes but also *contrastive* and *counterfactual* explanations [34], identifying why a sample was not assigned a specific label and what changes would be necessary to alter the classification result.

Unlike model-agnostic approaches such as SHAP and LIME, which generate post-hoc explanations by approximating the behavior of a black-box model through input perturbation, MAFALDA 2.0 offers *intrinsic* interpretability. This difference yields distinct advantages for pervasive CPS. Primarily, it ensures resource efficiency by eliminating the need for auxiliary explainer modules and multiple inference passes, which are often prohibitive for constrained MCUs. Furthermore, the explanations are faithful to the internal logic operations executed by the learning framework, a critical factor for auditability. Additionally, unlike numerical feature importance scores MAFALDA 2.0 produces symbolic descriptions that facilitate semantic interoperability with Linked Open Data (LOD) and knowledge bases.

3.3 Performance and Accuracy Evaluation

The experimental evaluation investigates the performance, classification accuracy, and interpretability of the proposed algorithm. Two publicly available benchmark datasets have been selected to support training and prediction tasks:

- The **Breast Cancer Wisconsin Dataset** [194] contains 569 instances with 30 numerical features extracted from digitized images of fine needle aspirates of breast masses. These features include mean values (suffix 1), standard errors (suffix 2), and worst-case measurements (suffix 3) for attributes such as texture, radius, perimeter, area, smoothness, and symmetry. The target is a binary classification task: identifying breast tissue as benign (B) or malignant (M).
- The **Heart Disease Dataset** [72] comprises 303 instances described by 13 clinical and diagnostic attributes, including age, blood pressure, cholesterol level, and maximum heart rate. The classification task consists of predicting the presence (positive outcome) or absence (negative outcome) of heart disease.

These datasets have been selected because they provide a direct basis for comparison with other on-device AI approaches found in the literature.

As described in Section 3.2.1, the deployment of MAFALDA 2.0 requires an input ontology that enables the transformation of numerical samples into semantic descriptions. This ontology must be provided prior to the training and inference phases, as it is used to correctly allocate and initialize the data structures.

When multiple instances operate collaboratively, they can use the same ontology to interpret and exchange knowledge; partial interoperability is also supported, allowing reasoning over shared subsets of knowledge when ontologies differ. This design supports federated and incremental workflows

and enables scalability with device capabilities. More powerful nodes can thus maintain larger ontologies, support more output classes, and store more detailed semantic profiles.

Ontologies can be manually engineered using standard vocabularies or expert knowledge, or they can be derived through data-driven strategies. Since automatic generation is not yet integrated into the current MAFALDA 2.0 implementation, this study adopts a data-driven method, executed offline, to construct the required ontologies. The adopted procedure is general-purpose and applicable to datasets that can be represented in a tabular form.

The construction process begins with an empty ontology containing the base classes `mafalda:Input` and `mafalda:Output`, along with the annotation properties `mafalda:minValue` and `mafalda:maxValue` for defining numerical intervals. The dataset is then processed column-wise to identify features and labels.

- For each feature column, a new class is created as a subclass of `mafalda:Input`, named after its corresponding column. If the feature is categorical, each distinct category is mapped to a subclass. If the feature is numerical, its values are partitioned into discrete bins, each mapped to a subclass with appropriate `mafalda:minValue` and `mafalda:maxValue` annotations.
- For each label column, a new class is introduced as a subclass of `mafalda:Output`, and each distinct label is mapped to a subclass of that label class.

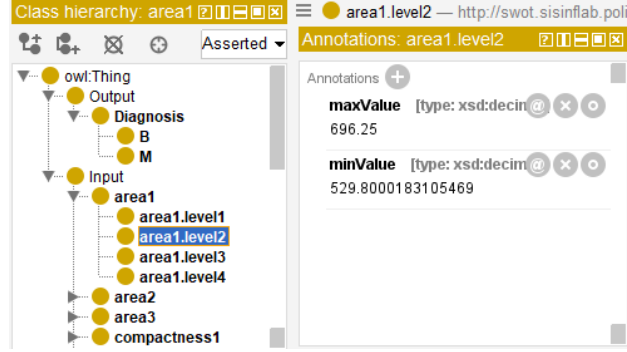


Figure 3.3: Generated Ontology

This procedure yields an ontology compliant with the modeling schema defined in Section 3.2.1. A fragment of the class hierarchy generated for the Breast Cancer dataset is shown in Figure 3.3.

For numerical features, two binning strategies are supported:

- **KBins** is an unsupervised method that applies K-Means clustering to a one-dimensional feature column. The number of clusters $k \in [k_{\min}, k_{\max}]$ is selected by maximizing the silhouette coefficient [139]. Bin boundaries are defined as midpoints between adjacent centroids.
- **DTree** is a supervised method based on a decision tree induction algorithm. A feature column and its corresponding labels are used to iteratively partition the feature space. Thresholds are selected to maximize the reduction in the Gini index. The recursion continues until a specified maximum depth is reached or until each leaf node contains fewer than two samples. Bin boundaries are defined as the thresholds at decision branches.

The choice of binning method, as well as hyperparameters such as $k_{\min}$, $k_{\max}$, and the maximum tree depth, is part of the hyperparameter tuning process. Table 3.6 summarizes the main differences between the two approaches.

In the present experiments, the ontology has been constructed offline using

Table 3.6: Comparison of ontology generation strategies for numerical features.

Feature	KBins Strategy	DTree Strategy
Learning Paradigm	Unsupervised	Supervised
Input Data	Single feature column only	Single feature column + all class labels
Optimization Goal	Maximize Silhouette Coefficient	Maximize Gini Index reduction
Bins Boundaries	Midpoints between centroids	Decision node split thresholds
Hyperparameters	Cluster range (k_{min}, k_{max})	Max tree depth, min samples per leaf

Python², with *scikit-learn* [131] for preprocessing and *Owlready2* [86] for constructing the ontology. Equivalent pipelines can be implemented in Java,³ using *WEKA* [193] and the *OWL API* [65].

Although ontology construction is currently performed off-device, these strategies are generalizable. Future work will concern the extension of MAFALDA 2.0, specifically:

- (i) autonomously generate baseline ontologies from scratch or specialize existing upper ontologies;
- (ii) support the generation of ontologies at runtime, with learnable `mafalda:minValue` and `mafalda:maxValue` parameters.

Such extensions would be feasible via lightweight libraries such as Cowl [20], that fully support on-device execution and ontology management.

The experimentation involved four devices, each representing a distinct class of computing hardware: an edge computing node E , represented by a *Raspberry Pi 4*; a high-end MCU N_{high} , namely a *B-L4S5I-IOT01A* development board; a mid-range MCU N_{mid} , the *Nordic nRF52840* development kit; and a low-end MCU N_{low} , the *Arduino Due*. These devices and their characteristics are listed in Table 2.2 and Table 2.1.

²<https://www.python.org/>

³<https://www.java.com>

Regarding the experimental workflow, each dataset is evaluated using a 30% test split. A five-fold cross-validation procedure is applied to the remaining training data to optimize the model configuration via grid search over the following hyperparameter space:

- **Threshold (θ):** 0.1 to 1.0 in increments of 0.1;
- **Number of ontology classes:** 5 to 9 in increments of 2 (controls Decision Tree depth in DTree or maximum $k_{\max}$ in KBins);
- **Number of semantic descriptions per class (d):** 1 to 20;
- **Ontology preprocessing method:** KBins or DTree.

The samples in the datasets lack temporal continuity, and therefore the shifting mechanism is disabled. This is achieved by configuring the observation window size to $+\infty$ (i.e., the largest integer representable for the bit width of the target platform), and the depth is fixed to 1. The framework is evaluated using standard performance metrics:

- **Accuracy:** the proportion of correctly classified samples over the test set;
- **Inference time:** the per-sample prediction latency on each node;
- **Training time:** the total time required to process all training samples (measured separately on edge and MCU platforms);
- **Model size:** the serialized representation of the trained model, including parameters and metadata;
- **Serialization and deserialization time:** measured to assess efficiency in model storage, transfer, and initialization;
- **Peak memory usage:** estimated on MCUs via `malloc_stats` from *newlib*⁴, combined with observed maximum stack usage; on the edge node, memory profiling is conducted using *Massif* [122].

⁴<https://www.sourceware.org/newlib/>

Figure 3.4 reports the confusion matrices for both datasets, and Table 3.7 summarizes the corresponding performance metrics.



Figure 3.4: Classification results: confusion matrices

Table 3.7: Performance Metrics for Heart Disease and Breast Cancer Datasets

Dataset	Precision	Recall	F1	Acc
Breast Cancer (Malignant)	1.000	0.857	0.923	0.947
Heart Disease (Positive)	0.886	0.848	0.867	0.868

For Breast Cancer, the best configuration employs DTree ontology generation with four classes, one semantic description per class, and $\theta = 0.1$. For Heart Disease, the best configuration uses KBins with four classes, one semantic description per class, and $\theta = 0.3$.

Table 3.8: Comparison of Breast Cancer Wisconsin dataset results with related works

Algorithm	Device	Acc (%)	Training Time (ms)	Inference Time (ms)	Flash (kB)	RAM (kB)
ML-MCU 2.0 [167]	nRF52840	63	273	0.005	111.864 ¹	77.32 ¹
	STM32F103C8	78	3837	0.1	161.9 (+33.9) ¹	74.33 (+54.33) ¹
	ESP32	62	112	0.005	287.43 ¹	241.73 ¹
	ATSAMD21G18	78	13806	0.33	288.6 (+32.6) ¹	72.58 (+40.58) ¹
Train++ [168]	nRF52840	85.0	5	1	110.41 ¹	77.32 ¹
	STM32F103C8	78.0	66	9	181.6 (+53.6) ¹	49.0 (+29) ¹
	HUZZAH32	78.0	2	<1	285.94 ¹	859.32 ¹
	ESP32	78	2	<1	285.94 ¹	859.32 ¹
	ATSAMD21G18	85.0	286	286	302.7 (+46.7) ¹	72.58 (+40.58) ¹
AIfES [197]	nRF52840 1 x 10 arch	91 ²	15.04	N/A	20.96	8.34
	nRF52840 1 + 10 arch	91 ³	40.34	N/A	21.22	21.27
	nRF52840 10 x 10 arch	91 ³	55.74	N/A	23.33	29.85
TDC [76]	nRF52840	_4	_4	_4	_4	_4
	ESP32	92	2.27	N/A	N/A	N/A
MAFALDA 2.0	BCM2711	94	14	0.524	395 ²	94.27
	STM32L4S5VIT6	94	1513	82	358.80	49.90
	nRF52840	94	1941	102	363.19	49.90
	AT91SAM3X8E	94	1722	90	321.62	49.44

¹ Compiler-generated estimates; values in parentheses represent the portion that exceeds available device memory.

² This measurement refers to the executable size of MAFALDA 2.0.

³ [197] do not include accuracy results. The experiments with results were reproduced by [76].

⁴ The model could not fit into the MCU memory.

Results for Breast Cancer appear in Table 3.8, and for Heart Disease in Table 3.9. Comparisons include target MCU, accuracy, total training time, per-sample inference time, and memory footprint.

Table 3.9: Comparison of Heart Disease Binary dataset results with related works

Algorithm	Device	Acc (%)	Training Time (ms)	Inference Time (ms)	Flash (kB)	RAM (kB)
ML-MCU 2.0 [167]	nRF52840	80.1	6	0.015	53.63 ¹	18.76 ¹
	STM32F103C8	82	77	0.03	40.832 ¹	16.24 ¹
	ESP32	83	3	0.015	228.84 ¹	27.38 ¹
	ATSAMD21G18	83	260	0.093	35.73 ¹	17.26 ¹
Train++ [168]	nRF52840	80.18	<1	<1	51.95 ¹	18.74 ¹
	STM32F103C8	82.08	13	1	36.93 ¹	16.23 ¹
	HUZZAH32	80	<1	<1	227.35 ¹	27.35 ¹
	ESP32	83	<1	<1	227.35 ¹	27.35 ¹
	ATSAMD21G18	80.18	44	44	20.47 ¹	19.17 ¹
MAFALDA 2.0	BCM2711	86	42	0.236	395 ²	46.16
	STM32L4S5VIT6	86	204	22	270.07	30.19
	nRF52840	86	261	27	231.46	30.19
	AT91SAM3X8E	86	308	29	244.78	30.19

¹ Compiler-generated estimates.² This measurement refers to the executable size of MAFALDA 2.0.

Across devices, MAFALDA 2.0 attains 94% accuracy on Breast Cancer and 86% on Heart Disease, slightly exceeding the best alternative approaches. This improvement is accompanied by a higher resource usage. Training time is of the same order of magnitude as that of ML-MCU 2.0 [167], whereas Train++ [168] and TinyDecisionTreeClassifier (TDC) [76] are faster. Inference latency is likewise higher for MAFALDA 2.0 relative to these baselines. In terms of memory, MAFALDA 2.0 requires more Flash/SRAM than AIfES [197], yet remains deployable on the evaluated boards and is comparable to ML-MCU 2.0 and Train++. The latter two, however, report compiler estimates rather than runtime measurements.

Table 3.10: Serialization and Deserialization Times

N	Breast Cancer (6.38 kB)		Heart Disease (2.43 kB)	
	Ser. (ms)	Des. (ms)	Ser. (ms)	Des. (ms)
E	0.848	17.41	0.587	4.61
N_{high}	2.61	964.83	1.50	228.79
N_{mid}	2.59	1232.63	1.48	284.65
N_{low}	3.61	1292.34	2.05	302.35

MAFALDA 2.0 implements methods for serializing and deserializing the TM using the Protocol Buffers⁵ format. This allows the exchange of TMs between devices, enabling FL workflows. These methods have been profiled, and the results are presented in Table 3.10. Encoded models are 6.38 kB (Breast Cancer) and 2.43 kB (Heart Disease). As expected, E outperforms MCU nodes by roughly one order of magnitude in serialization and nearly two orders in deserialization.

An additional experiment evaluated incremental and FL on the Breast Cancer dataset. Using a 20% test split, node E performed model selection and ontology generation, distributing the resulting configuration to all MCU nodes. Each MCU trained locally and returned its updated model to E for aggregation. As shown in Table 3.11, E achieved 0.918 accuracy after initial training, while N_{high} , N_{mid} , and N_{low} reached 0.924, 0.930, and 0.924, respectively. The aggregated model obtained 0.930 accuracy, slightly below the accuracy evaluated in the full local training case.

Table 3.11: Accuracy for Breast Cancer Dataset of all devices

Device	E	N_{high}	N_{mid}	N_{low}	Aggregated
Accuracy	0.918	0.924	0.93	0.924	0.93

MAFALDA 2.0 provides XAI by representing domain knowledge through symbolic OWL descriptions and formulating classification as semantic match-making. The Breast Cancer dataset is used for illustration due to its larger

⁵<https://protobuf.dev/>

feature space, though the procedure generalizes to other datasets. Figure 3.5 visualizes learned semantic profiles for the first model in Table 3.8, and Figure 3.6 shows a per-sample explanation.

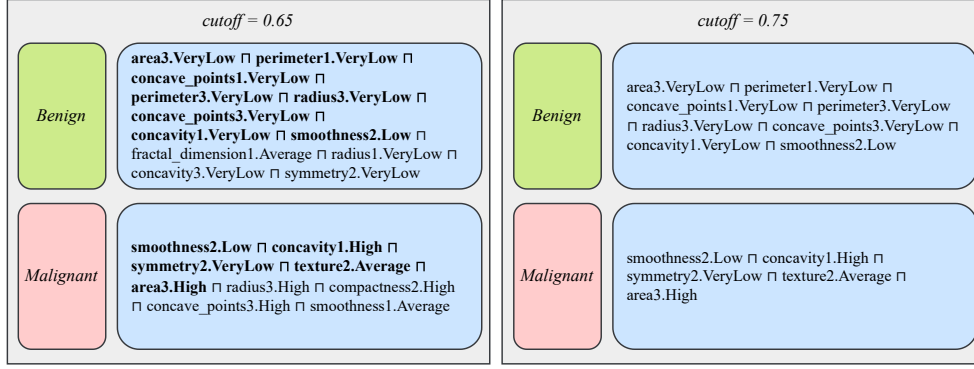


Figure 3.5: MAFALDA 2.0 global explanation: learned semantic description of output class

With 30 features, learned profiles can be lengthy. To improve readability, the textual export supports a cutoff threshold that omits concepts with TM values below a chosen level. This controls the fidelity-sparsity trade-off: higher cutoffs yield compact, readable explanations, while lower values allow the inclusion of all the concepts considered during the matchmaking process. Figure 3.5 illustrates this effect (cutoff 0.65 on the left, 0.75 on the right). Concepts are sorted by decreasing frequency. Benign instances exhibit consistently low values for shape-related features (area, perimeter, concave points, radius, concavity) and low smoothness and symmetry, indicating smaller, more regular morphology. Malignant instances show higher concavity (mean), lower symmetry (standard deviation), and larger area (maximum), corresponding to larger, more irregular structures.

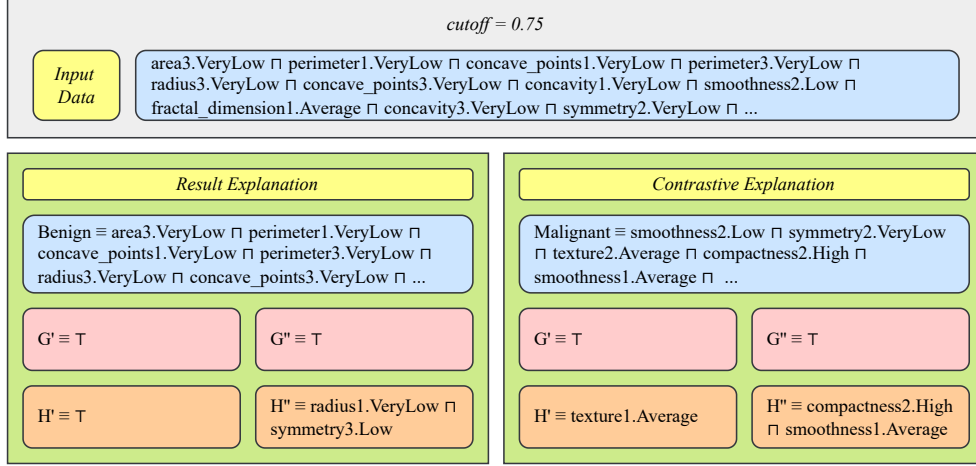


Figure 3.6: MAFALDA 2.0 local explanation: classification outcome explained by means of reasoning services.

MAFALDA 2.0 also generates local per-sample explanations. In Figure 3.6, the input is classified as benign mainly due to low values in shape-related features. Only minor concepts (H'') are missing. Contrastive analysis indicates that a malignant outcome would require higher compactness and texture variation; the missing concepts (H' in the input and H'' in the model) form counterfactuals that would alter the decision.

Finally, two complementary evaluations have also been performed to assess energy efficiency. Table 3.12 compares MAFALDA 2.0 with ML-MCU and Train++ [168] on MNIST, a widely used dataset of handwritten digit images consisting of 60,000 training and 10,000 test samples, each represented as a 28×28 grayscale image. The evaluation has been conducted using the same **nRF52840** platform and workflow adopted by ML-MCU. Table 3.13 provides an estimated comparison with TensorFlow Lite Micro [38] on UCI-HAR [7] using the same **ESP32** MCU. Measurements have been taken with the Monsoon High Voltage Power Monitor⁶, averaged over five runs, normalized per sample, and include preprocessing. Features have been extracted via 50-sample sliding windows with mean and standard deviation aggregation,

⁶Monsoon Solutions Inc. High Voltage Power Monitor. <https://www.msoon.com/high-voltage-power-monitor>

while the original model processed the raw stream directly. Since MAFALDA 2.0 requires explicit feature engineering, preprocessing time and energy have been included in the measurements. In this specific experiment, all values are reported on a per-sample basis.

Algorithm	Train		Inference		Acc. (%)
	Time (ms)	Energy (mj)	Time (ms)	Energy (mj)	
ML-MCU [167]	5.09	0.68	2.38	0.34	89.7
Train++ [168]	0.0659	0.0098	0.0138	0.002	98.0
MAFALDA 2.0	139	21	41	6	97.2

Table 3.12: Comparison on the Handwritten Digits dataset.

Alg	Train	Inference (ms)	Inference	Acc. (%)
	Time (ms)	Time	Energy (mj)	
TFMicro [38]	n/a	5–72	253–324	86–92
MAFALDA 2.0	94.875	9	3.55	90.5

Table 3.13: Comparison on the HAR dataset between TFMicro and MAFALDA 2.0

As shown in Table 3.12, MAFALDA 2.0 incurs higher training/inference time and energy than simpler approaches. Relative to Table 3.13, performance is comparable to more complex models; however, DNNs, when tuned, could achieve higher accuracy and efficiency. Conversely, MAFALDA 2.0 provides complementary capabilities that are not covered by DNNs: on-device incremental and FL with integrated symbolic explainability.

3.4 Case Study: Activity Recognition in Cloud-Edge-IoT Scenarios

This section aims to merge the contributions of Chapter 2 and Chapter 3 by presenting a pervasive CPS for an activity detection task. In particular, it integrates MAFALDA 2.0 into the Cloud-to-Thing AWS architecture described

in Section 2.4 and manages the lifecycle of the ML model and tasks through the operations identified in Section 2.3.

3.4.1 Prototype Workflow

The main operations of the system are described below.

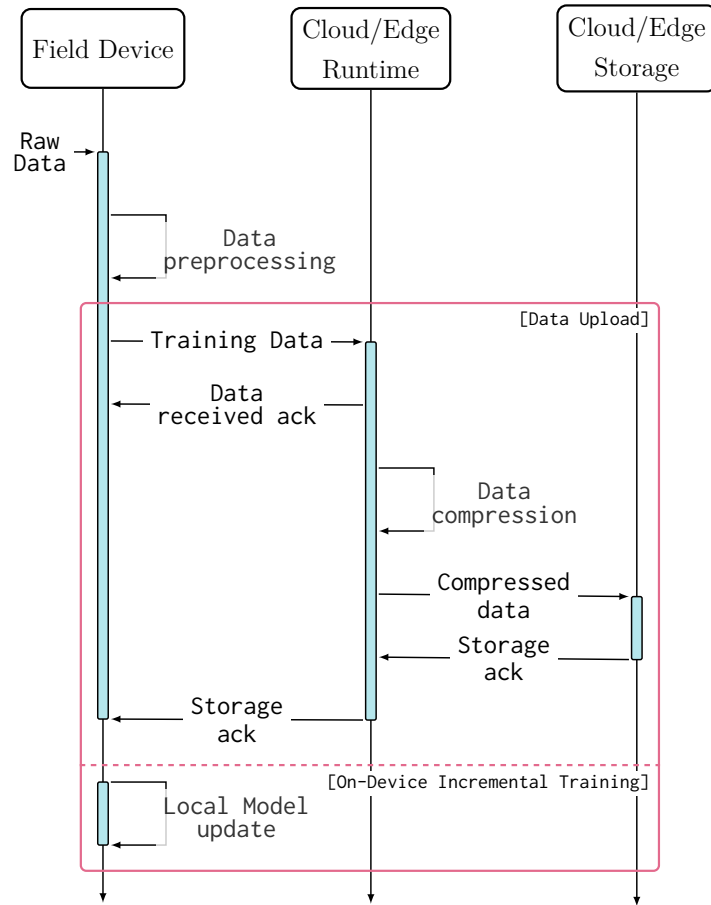


Figure 3.7: Data upload sequence diagram

The sequence begins with data collection, as shown in Figure 3.7, which represents the data collection phase. Field IoT devices collect raw data, perform lightweight preprocessing, and either use the processed data to update their local MAFALDA 2.0 instance or transmit them to a storage node. Node selection and routing are handled through an intermediary negotiation

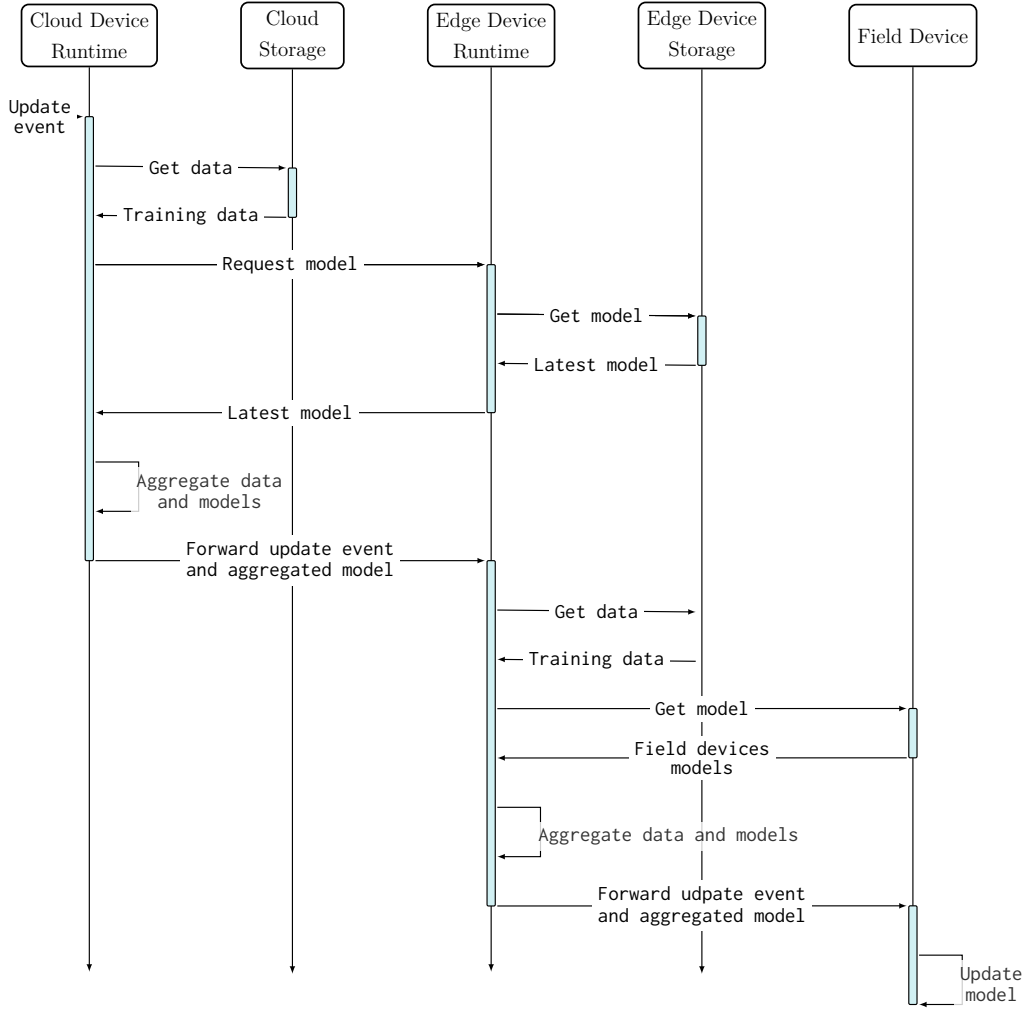


Figure 3.8: Model update sequence diagram

and discovery process, shown in Figure 2.3 but omitted from Figure 3.7 and subsequent diagrams for clarity.

The model update phase, corresponding to the training stage in a conventional ML pipeline, synchronizes and refines the ML model by aggregating data from distributed nodes. In the current prototype, this process is initiated by a centralized **update event**, generated by a task scheduler hosted on the cloud node and unique to the network. Future work will explore decentralized protocols that allow edge and field devices to autonomously decide when and how to trigger this operation. In general, each node trains its instance using

local data and aggregates it with that of neighboring nodes. Once complete, it propagates an update event containing the aggregated model to its neighbors. The receiving node replaces its model with the aggregated one and repeats the process. Figure 3.8 illustrates the sequence of interactions among the components during the update phase. The process continues recursively across lower tiers. A nonce value uniquely identifies the origin update event, preventing repeated updates caused by cyclic message propagation.

Finally, when an inference is requested, the node first verifies whether a local model is available. If present, it is loaded and used directly for prediction. If absent, the node can either retrieve the TM from the nearest storage node to instantiate a local MAFALDA 2.0 instance or forward the prediction request on the `predict` topic. In the experiments, the first configuration has been used, retrieving the TM from the nearest storage node and creating a local MAFALDA 2.0 instance. The sequence diagram in Figure 3.9 depicts this process.

A key challenge in distributed systems of this type is maintaining synchronization of data across devices that operate independently and may be intermittently offline. Concurrent updates from multiple sources can cause write conflicts when accessing the shared Model Store. To mitigate this, an optimistic concurrency control mechanism is used: updates to instances in the database are applied only if the version number remains unchanged between the initial read and the write. If a conflict is detected, the update is postponed and retried in the next execution cycle. This approach ensures eventual consistency while allowing temporary use of outdated data. For scenarios requiring stronger guarantees, more advanced coordination methods, like consensus protocols or centralized scheduling, should be used instead [68].

3.4.2 Experiments

The experimental testbed has been set up using the prototype described in Section 2.4. A Raspberry Pi 4 Model B+ served as the edge CoreDevice, and the IoT Field Devices have been implemented using B-L4S5I-IOT01A boards.

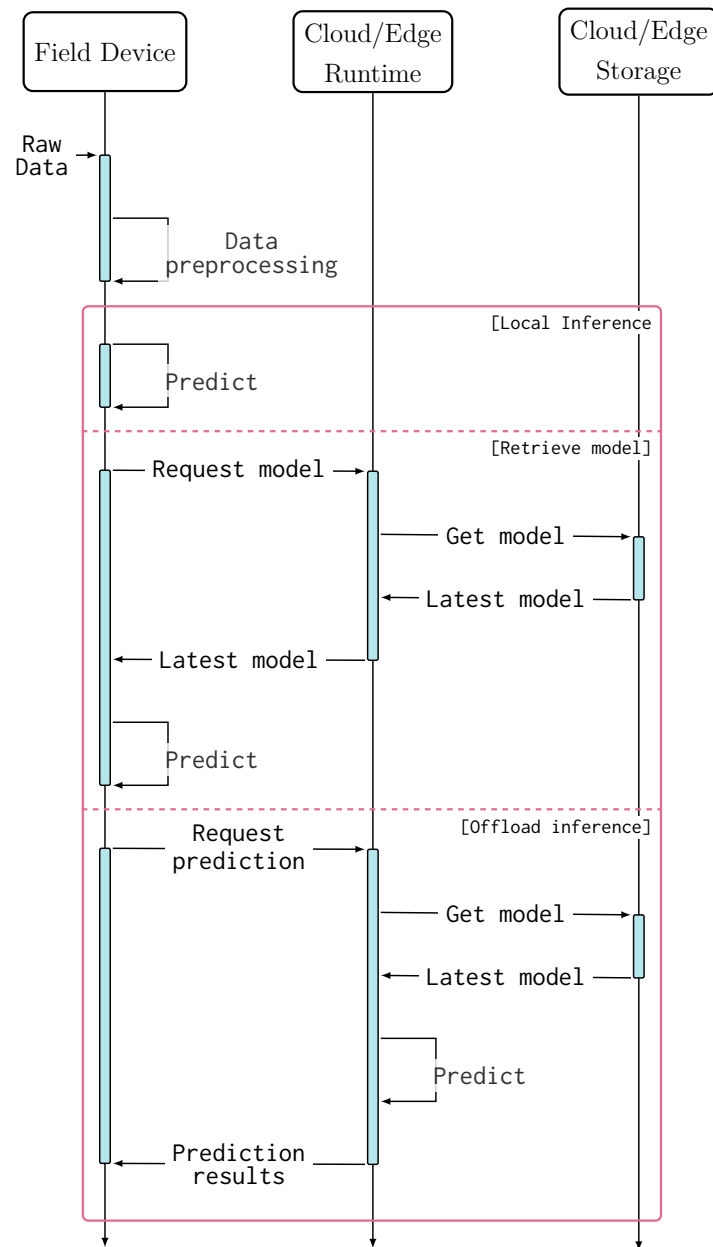


Figure 3.9: Inference workflow diagram

The first step has involved data preparation.

For the experiment, the chosen dataset is *MotionSense* [106], which provides motion sensor data collected from smartphones during various human activities. This dataset has been selected for its realistic representation of distributed sensing scenarios, where each device corresponds to a distinct user performing daily activities. It contains accelerometer and gyroscope data from 24 iPhone 6s users performing six activities: walking, jogging, sitting, standing, going upstairs, and going downstairs. Each user completed 15 trials sampled at 50 Hz, producing 12 sensor features (attitude, rotation rate, gravity, user acceleration).

For data preparation, raw measurements have been aggregated into 1-second windows (50 samples). The mean and standard deviation per feature have been computed for each 50-sample series. Starting from the 12 features, the post-processed dataset yielded 24 features per sample. This aggregation has been performed directly on Field Devices, with preprocessing taking approximately 0.2 ms per window and fitting within the available memory. This additional overhead is included in all subsequent measurements. Subjects 16 have been used to create a **bootstrap model** for initial training and hyperparameter tuning. The remaining subjects have been distributed among six Field Devices:

- Cloud IoT group: $c1$ (79), $c2$ (1012), $c3$ (1315)
- Edge IoT group: $e1$ (1618), $e2$ (1920), $e3$ (2124)

From each partition, 20% of samples have been held out to form a unique test set for model evaluation.

The second step has involved data collection and model training. First, the *bootstrap model* has been trained and loaded into the architecture. This included the ontology generation step, following the approach described in Section 3.3, and a five-fold Cross Validation (CV) to determine the best hyperparameter configuration, which is:

- **threshold:** 0.05;
- **descriptions:** 15;
- **binning algorithm:** DTree, with maximum depth set to 4;
- **depth:** 1;
- **window and factor:** N/D⁷

Table 3.14: Communication latencies for edge- and cloud-connected field devices (ms)

		Edge-connected				Cloud-connected			
		<i>e1</i>	<i>e2</i>	<i>e3</i>	<i>Overall</i>	<i>c1</i>	<i>c2</i>	<i>c3</i>	<i>Overall</i>
First Ack.	<i>Avg.</i>	284.90	285.90	284.40	285.06	992.00	609.72	600.00	741.29
	<i>Max</i>	322.00	310.00	325.00	325.00	2969.00	714.00	687.00	2969.00
	<i>Std.Dev.</i>	17.59	15.87	22.16	18.70	725.32	51.25	52.37	472.25
Second Ack.	<i>Avg.</i>	101.63	53.20	60.10	72.61	47.90	47.18	47.18	46.35
	<i>Max</i>	296.00	73.00	206.00	296.00	151.00	76.00	79.00	151.00
	<i>Std.Dev.</i>	71.89	13.81	49.47	56.18	36.18	18.47	20.06	26.36
Total	<i>Avg.</i>	386.54	308.27	313.00	357.00	1039.91	656.91	647.18	787.64
	<i>Max</i>	596.00	371.00	467.00	596.00	3000.00	747.00	758.00	3000.00
	<i>Std.Dev.</i>	76.00	25.71	46.78	58.58	726.29	48.51	61.05	437.34

Following this, six Field Devices have been connected simultaneously: three (*c1*, *c2*, *c3*) to the cloud through AWS IoT Core, and three (*e1*, *e2*, *e3*) to the edge via the Greengrass Core Device, which emulates networked devices performing data collection. Each device started by retrieving the bootstrap model from either the cloud or the edge. To test both data forwarding and local training, each minute the device either transmitted a batch of 60 samples to the nearest storage or updated its local model. The receiving end has been configured to send acknowledgment messages upon data receipt and after completing the storage operation. The Field Devices logged the time between publishing a message and receiving the first and second acknowledgments for data receipt and storage. The recorded latency metrics, reported in Table 3.14,

⁷Since the depth is 1, window and factor do not influence the algorithm behavior.

align with the observations in Chapter 2: edge devices achieved significantly lower round-trip times than those connected to the cloud, primarily due to reduced network overhead and local proximity between the Edge Node and Field Devices. However, the cloud generally performed better in terms of processing time.

Table 3.15: AWS Lambda Pricing in eu-central-1 region

Memory (MB)	Pricing (10^{-7} \$ / ms)
128	0.021
512	0.083
1024	0.167
1536	0.250
2048	0.333

When deploying the architecture, a critical aspect is selecting the appropriate tier of Lambda executor to be used. In the case of AWS, the tier of the virtual machine executing the Lambda functions can be indirectly controlled by adjusting the amount of RAM allocated. The selected region for the experiment has been **eu-central-1**. The Lambda prices for each memory tier are reported in Table 3.15.⁸

Periodically, AWS EventBridge triggers the *Update Model* event, updating the model stored in the Model Store with data from the cloud Data Store. Upon completion, notifications are sent to neighboring nodes, instructing them to perform updates using their respective datasets.

The model incorporating knowledge from all devices is hereafter referred to as the **final model**.

The following execution times have been collected:

- **Retrieve batch:** Time required by a node to query the local storage and retrieve a batch of 60 samples;

⁸The values refer to the ARM Central Processing Unit (CPU), whose costs are lower than those of the x86 architecture.

- **Train on batch:** Time required by a node to update the MAFALDA 2.0 model with a batch of 60 samples;
- **Predict batch:** Time required by a node to classify a batch of 60 samples using the local MAFALDA 2.0 model.

Table 3.16: Cloud Node: execution Time and Cost

Memory Config (MB)	Metric	Retrieve Batch	Train on Batch	Predict Batch	Total
128	<i>Time (ms)</i>	285.72	448.14	193.23	927.09
	<i>Cost (10⁻⁷\$)</i>	6.00	9.41	4.06	19.47
512	<i>Time (ms)</i>	71.16	108.62	48.31	228.09
	<i>Cost (10⁻⁷\$)</i>	5.91	9.02	4.01	18.93
1024	<i>Time (ms)</i>	60.84	49.74	31.83	142.41
	<i>Cost (10⁻⁷\$)</i>	10.16	8.31	5.32	23.78
1536	<i>Time (ms)</i>	57.00	34.41	17.89	109.30
	<i>Cost (10⁻⁷\$)</i>	14.25	8.60	4.47	27.33
2048	<i>Time (ms)</i>	51.31	31.26	11.18	93.75
	<i>Cost (10⁻⁷\$)</i>	17.09	10.41	3.72	31.22

Table 3.16 shows the metrics for each memory configuration. The optimal configurations in terms of cost are 512 MB for the Retrieve and Predict batch operations, and 1024 MB for Train on Batch.

Table 3.17: Mafalda functions training and prediction performance

		Retrieve Batch (ms)	Train on Batch (ms)	Predict Batch (ms)
Field Device	<i>Avg.</i>	N/A	10 550.00	9286.20
	<i>Max</i>	N/A	10 625.00	9527.00
	<i>Std.Dev.</i>	N/A	66.67	160.51
Edge Device	<i>Avg.</i>	166.61	68.97	26.47
	<i>Max</i>	185.14	73.22	28.07
	<i>Std.Dev.</i>	14.09	2.29	1.44
Cloud Node	<i>Avg.</i>	71.16	108.62	48.31
	<i>Max</i>	116.16	135.53	64.60
	<i>Std.Dev.</i>	23.22	16.17	21.15

Table 3.17 reports the results for the Cloud executor with 512 MB of RAM (selected for its cost-effectiveness), the Edge node, and the Field Device. Results show a clear hierarchy across the three computational layers. Field Devices exhibit the longest execution times, on the order of 910 seconds per batch, reflecting their limited processing power and resource constraints. Despite this, the ability of MAFALDA 2.0 to train and predict directly on-device confirms its suitability for distributed learning in pervasive CPS. These findings indicate that local execution is feasible when latency is less critical or connectivity is unreliable.

The Edge node outperforms the Cloud in raw compute speed for training and inference. The average batch training time on the edge (69 ms) is shorter than that of the cloud (109 ms), although batch retrieval from storage is slower on the edge. The reduced delay makes the edge an effective compromise between responsiveness and computational capability.

Cloud nodes show predictable scaling behavior with memory tier. Increasing Lambda memory allocation reduces execution time but raises cost. The 512 MB configuration provides the best costperformance ratio, while higher tiers (1024–2048 MB) are faster but offer diminishing returns.

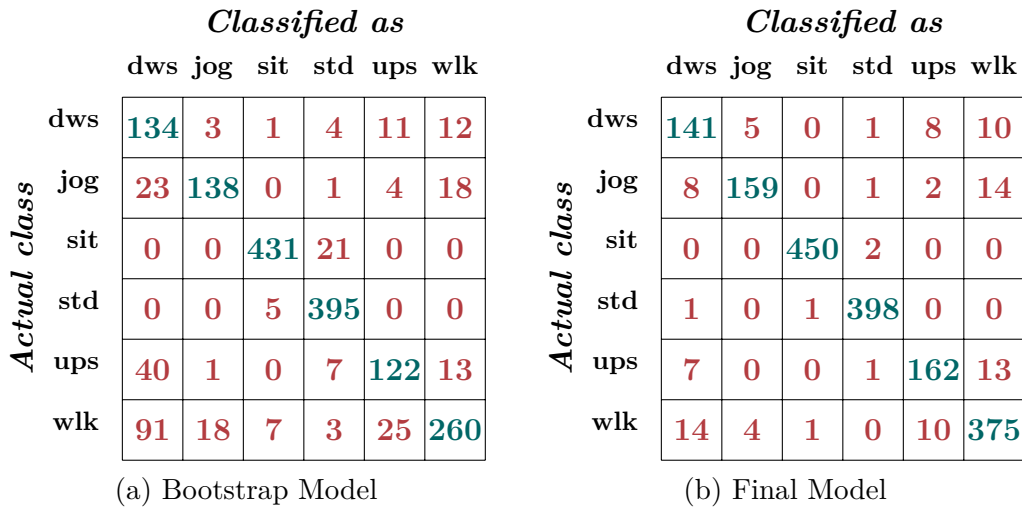


Figure 3.10: Confusion matrices: bootstrap model vs final model

Table 3.18: Comparison of evaluation summaries for three models

Class	Bootstrap Model			Final Model			Mafalda 1.0		
	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
<i>dws</i>	0.465	0.812	0.592	0.825	0.855	0.839	0.516	0.776	0.620
<i>jog</i>	0.863	0.750	0.802	0.946	0.864	0.903	0.954	0.788	0.863
<i>sit</i>	0.971	0.954	0.962	0.996	0.996	0.996	0.972	1.000	0.986
<i>std</i>	0.916	0.988	0.951	0.988	0.995	0.991	0.976	1.000	0.988
<i>ups</i>	0.753	0.667	0.707	0.890	0.885	0.888	0.737	0.721	0.729
<i>wlk</i>	0.858	0.644	0.736	0.910	0.928	0.919	0.793	0.656	0.718
Average	0.804	0.802	0.792	0.926	0.920	0.923	0.825	0.824	0.817
Accuracy	0.828			0.942			0.851		

Figure 3.10 and Table 3.18 report the accuracy evaluation results, including the confusion matrix and the evaluation summary, respectively. These have been measured on the same held-out test set for both the bootstrap model and the final model obtained from aggregating all data in the network.

Additionally, Table 3.18 includes the evaluation summary for the older MAFALDA framework [144], which has been trained once on the entire dataset and evaluated on the same held-out test set. The only tunable parameter of MAFALDA is the threshold that has been determined using five-fold CV, yielding $\theta = 0.11$. The results show an accuracy improvement in the new version. Regarding incremental and federated learning in MAFALDA 2.0, the final model demonstrates a measurable improvement over the bootstrap baseline. Overall accuracy increased from 0.828 to 0.942, confirming that the IL process functions as intended and enables continual improvement in distributed architectures.

The case study demonstrates that constrained IoT devices can actively participate in the learning process. From an algorithmic perspective, MAFALDA 2.0 has validated its federated and incremental learning capabilities, with the global aggregated model achieving a significant accuracy improvement over the bootstrap baseline. It is important to clarify how the temporal nature of this case study is addressed within the MAFALDA 2.0 framework. Unlike models like RNNs that maintain an internal hidden state to influence future

predictions, the proposed classifier operates as a stateless inference engine. The prediction for a sample at a time t depends exclusively on the input feature vector provided at that instant. Consequently, temporal dependencies have not been modeled by the classifier’s architecture but have instead been encoded directly into the feature space through a sliding window preprocessing step. The TM’s internal history mechanism is designed to manage concept drift during the *update* phase rather than to provide recurrent memory during *inference*. Future work will specifically address non-stationary scenarios to evaluate the effectiveness of the matrix shift mechanism in adapting to evolving data distributions over time.

Conclusions and Perspectives

This dissertation has addressed the challenge of enabling explainable and resource-efficient artificial intelligence in pervasive CPSs. Despite continuous progress in the development of IoT and edge computing technologies, most embedded devices remain confined to data collection and transmission, relying on cloud infrastructures for inference and learning. Furthermore, explainability is rarely considered in distributed intelligence frameworks, limiting the applicability of AI in real-time, critical contexts. The work presented here aimed to overcome these challenges by combining KRR with on-device and federated learning, ultimately contributing to the evolution of the SWoE.

Two main research contributions have been achieved to reach this goal. The first is a general-purpose framework for distributed CPSs, designed to unify the management of computational resources across the Cloud-to-Thing continuum. This framework, based on serverless computing principles, allows the orchestration of data acquisition, processing, and model training among heterogeneous nodes. It supports federated learning workflows in which models are trained locally on field devices and aggregated through edge or cloud components. The second contribution is MAFALDA 2.0, a novel architecture for explainable on-device ML that integrates symbolic knowledge representation and reasoning with lightweight statistical learning. Implemented in pure C for portability and efficiency, MAFALDA 2.0 enables incremental and federated learning directly on embedded platforms, providing semantic XAI through symbolic representations of numerical data.

The proposed solutions have been experimentally validated across multiple scenarios. The Cloud-to-Thing framework has been implemented using COTS components and AWS cloud platform. Experiments have demonstrated that distributed orchestration of ML workflows can be achieved using standard

serverless technologies supporting heterogeneous devices. MAFALDA 2.0 has been tested on several public datasets and embedded platforms, confirming its capability to perform IL, FL, and semantic reasoning in constrained environments. The results showed that the proposed approach achieves competitive accuracy compared to state-of-the-art on-device ML solutions, with the added benefit of symbolic semantic explainability. Validation has been carried out on a set of public datasets, representative of different data types and computational loads, to verify the generality and effectiveness of the framework across heterogeneous scenarios:

- **Breast Cancer** and **Heart Disease**, tabular datasets used to evaluate traditional classification models with low-dimensional numerical and categorical features;
- **MNIST**, a dataset of handwritten digits, used to test the frameworks ability to handle high-dimensional input data;
- **MotionSense**, a dataset of multivariate sensor data acquired from mobile devices, employed to analyze system behavior in human activity recognition scenarios.

The latter dataset played a central role, as it enabled testing of the entire distributed learning cycle within the AWS Cloud-to-Thing framework. This included all phases, from data acquisition at peripheral nodes to model aggregation at the edge and cloud components. Experimental results have demonstrated the feasibility of the proposed approach, showing that the architecture supports distributed learning and inference processes even on devices with limited computational resources.

Limitations and future works: Despite these advancements, several challenges remain open. The following section summarizes the primary limitations identified in this work, distinguishing between the Cloud-to-Thing framework and the on-device learning algorithm.

Cloud-to-Thing Serverless Framework

- **Configuration Complexity:** Deployment relies on a heterogeneous deployment frameworks (e.g., AWS Greengrass, ZephyrOS build system, custom build scripts), lacking a unified end-to-end provisioning solution for bare-metal devices.
- **Lack of Semantic Orchestration:** The orchestration logic is currently topic-based and does not exploit function semantics to support automatic service composition or resource-aware serverless migration.
- **Cold Start Latency:** Reliance on FaaS introduces cold start delays, which can hinder performance in hard real-time scenarios. Although the effect can be mitigated by utilizing real-time built-in functions on edge nodes, this limitation must be carefully considered when deploying such solutions in time-critical systems.
- **Concurrency Management:** The distributed Model Store utilizes Optimistic Concurrency Control, which may lead to increased write conflicts in dense networks with high-frequency updates.

MAFALDA 2

- **Accuracy Trade-off:** The model’s accuracy does not consistently match state-of-the-art opaque models on high-dimensional datasets, reflecting an inherent trade-off between performance and transparency.
- **Static Knowledge Base:** The reliance on a fixed ontology necessitates offline knowledge engineering, preventing the autonomous discovery of new feature concepts from raw data.
- **Computational Overhead:** Despite $\mathcal{ALN}$ tractability, Tiny-ME’s services incur a higher computational cost compared to optimized vector operations in bare metal implementations of the simplest traditional ML models.
- **Analysis of Explainability:** The current validation is qualitative. A formal characterization using user-centered metrics and quantitative benchmarks is an open problem.

- **Sensitivity to Discretization:** Performance is heavily dependent on the discretization strategy, which requires empirical tuning unlike end-to-end learning approaches.

Further research will seek to overcome the listed limitations. In particular, the focus will be on methods to orchestrate functions effectively on the Cloud-to-Thing framework, to optimize reasoning and MAFALDA 2.0 efficiency, and on the extension of the testbed to large-scale, heterogeneous, and realistic deployments. Moreover, conducting a formal study on privacy-preserving mechanisms, such as secure aggregation, would strengthen the frameworks applicability in sensitive domains. Finally, new experiments should include a quantitative evaluation of XAI capabilities using standard interpretability metrics and user-centered validation to objectively measure the effectiveness of the proposed explanations.

Overall, this work contributes a comprehensive approach to the design of incremental, federated, and explainable AI for pervasive CPSs. By coupling symbolic reasoning with distributed learning across the Cloud-to-Thing continuum, it lays the groundwork for novel approaches, enabling trustworthy and interoperable distributed systems for adoption in real-time, critical contexts and emerging IoE scenarios.

Bibliography

- [1] Mohammad Aazam and Eui-Nam Huh. Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 687–694. IEEE, 2015.
- [2] Alaa Awad Abdellatif, Amr Mohamed, Carla Fabiana Chiasserini, Mounira Tlili, and Aiman Erbad. Edge computing for smart health: Context-aware approaches, opportunities, and challenges. *IEEE Network*, 33(3):196–203, 2019.
- [3] Nora Abdelmageed and Sirko Schindler. JenTab: Matching Tabular Data to Knowledge Graphs. In *Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Workshop, International Semantic Web Conference*, pages 40–49, 2020.
- [4] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE access*, 6:52138–52160, 2018.
- [5] Shamil Al-Ameen, Bharath Sudharsan, Tejus Vijayakumar, Tomasz Szydlo, Tejal Shah, and Rajiv Ranjan. Poly Instance Recurrent Neural Network for Real-time Lifelong Learning at the Low-power Edge. In *2024 IEEE International Conference on Big Data (BigData)*, pages 5907–5916. IEEE, 2024.
- [6] Maira Alvi, Hrishikesh Dutta, Roberto Minerva, Noel Crespi, Syed Mohsan Raza, and Manoj Herath. Global perspectives on digital twin smart cities: Innovations, challenges, and pathways to a sustainable urban future. *Sustainable Cities and Society*, page 106356, 2025.

- [7] D. Anguita, Alessandro Ghio, L. Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition using Smartphones. In *The European Symposium on Artificial Neural Networks*, pages 437–442, 2013.
- [8] Rushil Anirudh, Jayaraman J Thiagarajan, Rahul Sridhar, and Peer-Timo Bremer. Margin: Uncovering deep neural networks using graph signal analysis. *Frontiers in Big Data*, 4:589417, 2021.
- [9] Mattia Antonini, Miguel Pincheira, Massimo Vecchio, and Fabio Antonelli. An adaptable and unsupervised TinyML anomaly detection system for extreme industrial environments. *Sensors*, 23(4):2344, 2023.
- [10] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion*, 58:82–115, 2020.
- [11] R. Arthi and V. Krishnaveni. Optimized Tiny Machine Learning and Explainable AI for Trustable and Energy-Efficient Fog-Enabled Healthcare Decision Support System. *Scientific Reports*, 2024.
- [12] Peter Arthurs, Lee Gillam, Paul Krause, Ning Wang, Kaushik Halder, and Alexandros Mouzakitis. A Taxonomy and Survey of Edge Cloud Computing for Intelligent Transportation Systems and Connected Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6206–6221, 2022.
- [13] Sisay Tadesse Arzo, Claire Naiga, Fabrizio Granelli, Riccardo Bassoli, Michael Devetsikiotis, and Frank HP Fitzek. A theoretical discussion and survey of network automation for IoT: Challenges and opportunity. *IEEE Internet of Things Journal*, 8(15):12021–12045, 2021.
- [14] Hany F. Atlam, Robert J. Walters, and Gary B. Wills. Fog computing and the Internet of Things: A review. *big data and cognitive computing*, 2(2):10, 2018.

- [15] Franz Baader. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.
- [16] Samantha Bail, Birte Glimm, Ernesto Jimenez-Ruiz, Nicolas Matentzoglou, Bijan Parsia, and Andreas Steigmiller. ORE 2014 OWL Reasoner Evaluation Live Competition. <http://dl.kr.org/ore2014>. Accessed: 2023-03-20.
- [17] Shantanu Banerjee, Naveen G Jesubalan, Amey Kulkarni, Anshul Agarwal, and Anurag S Rathore. Developing cyber-physical system and digital twin for smart manufacturing: Methodology and case study of continuous clarification. *Journal of Industrial Information Integration*, 38:100577, 2024.
- [18] João A Bastos. Predicting credit scores with boosted decision trees. *Forecasting*, 4(4):925–935, 2022.
- [19] Christian Becker and Christian Bizer. Exploring the Geospatial Semantic Web with DBpedia mobile. *Journal of Web Semantics*, 7(4):278–286, 2009.
- [20] Ivano Bilenchi, Filippo Gramegna, Giuseppe Loseto, Saverio Ieva, Floriano Scioscia, and Michele Ruta. COWL: Pushing OWL 2 over the edge. *Internet of Things*, 29:101439, 2025.
- [21] Ivano Bilenchi, Davide Loconte, Floriano Scioscia, and Michele Ruta. Evaluating Correctness, Performance and Energy Footprint of Semantic Reasoners in Mobile Edge Computing. *Journal of Systems and Software*, 233(112696), 2026 (in press; accepted 2025).
- [22] Ivano Bilenchi, Floriano Scioscia, and Michele Ruta. Cowl: A lightweight owl library for the semantic web of everything. In *International Conference on Web Engineering*, pages 100–112. Springer, 2022.
- [23] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz DaSilva, Craig Lee, and Omer Rana. The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, 3:134–155, 2018.

- [24] Martin Björklund. The YANG 1.1 Data Modeling Language. RFC 7950, August 2016.
- [25] Carsten Bormann, Angelo P Castellani, and Zach Shelby. CoAP: An application protocol for billions of tiny Internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012.
- [26] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [27] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [28] Roberta Calegari, Giovanni Ciatto, and Andrea Omicini. On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale*, 14(1):7–32, 2020.
- [29] Héctor Cañas, Josefa Mula, Manuel Díaz-Madroñero, and Francisco Campuzano-Bolarín. Implementing Industry 4.0 principles. *Computers & industrial engineering*, 158:107379, 2021.
- [30] Kun Cao, Shiyang Hu, Yang Shi, Armando Walter Colombo, Stamatis Karnouskos, and Xin Li. A Survey on Edge and Edge-Cloud Computing Assisted Cyber-Physical Systems. *IEEE Transactions on Industrial Informatics*, 17(11):7806–7819, 2021.
- [31] Nicola Capuano, Giuseppe Fenza, Vincenzo Loia, and Claudio Stanzione. Explainable artificial intelligence in cybersecurity: A survey. *Ieee Access*, 10:93575–93600, 2022.
- [32] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.
- [33] Ryan Chard, Yadu Babuji, Zhuozhao Li, Tyler Skluzacek, Anna Woodard, Ben Blaiszik, Ian Foster, and Kyle Chard. Funcx: A federated function serving fabric for science. In *Proceedings of the 29th*

International symposium on high-performance parallel and distributed computing, pages 65–76, 2020.

- [34] Shruthi Chari, Oshani Seneviratne, Mohamed Ghalwash, Sola Shirai, Daniel M Gruen, Pablo Meyer, Prithwish Chakraborty, and Deborah L McGuinness. Explanation Ontology: A general-purpose, semantic representation for supporting user-centered explanations. *Semantic Web*, 15(4):959–989, 2024.
- [35] Xun-Qi Chen, Chao-Qun Ma, Yi-Shuai Ren, Yu-Tian Lei, Ngoc Quang Anh Huynh, and Seema Narayan. Explainable artificial intelligence in finance: A bibliometric review. *Finance Research Letters*, 56:104145, 2023.
- [36] Mahesh Chowdhary and Swapnil Sayan Saha. On-Sensor Online Learning and Classification Under 8 KB Memory. In *2023 26th International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE, 2023.
- [37] Kwok Tai Chui, Brij B Gupta, Jiaqi Liu, Varsha Arya, Nadia Nedjah, Ammar Almomani, and Priyanka Chaurasia. A survey of internet of things and cyber-physical systems: Standards, algorithms, applications, security, challenges, and future directions. *Information*, 14(7):388, 2023.
- [38] Chiara Contoli and Emanuele Lattanzi. A Study on the Application of TensorFlow Compression Techniques to Human Activity Recognition. *IEEE Access*, 11:48046–48058, 2023.
- [39] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.
- [40] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language, September 2012. <https://www.w3.org/TR/r2rml/>.
- [41] Soma Dasgupta and Swarnava Dey. MCU-based TinyML Camera for Face Recognition. In *2024 IEEE International Conference on Pervasive*

Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), pages 391–393. IEEE, 2024.

- [42] Diego De Una, Nataliia Rümmele, Graeme Gange, Peter Schachte, and Peter J Stuckey. Machine Learning and Constraint Programming for Relational-To-Ontology Schema Mapping. In *International Joint Conference on Artificial Intelligence*, pages 1277–1283, 2018.
- [43] Fabrizio De Vita, Giorgio Nocera, Dario Bruneo, Valeria Tomaselli, and Mirko Falchetto. On-device training of deep learning models on edge microcontrollers. In *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCoM) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 62–69. IEEE, 2022.
- [44] Thomas Delva, Julián Arenas-Guerrero, Ana Iglesias-Molina, Oscar Corcho, David Chaves-Fraga, and Anastasia Dimou. RML-star: A declarative mapping language for RDF-star generation. In *ISWC2021, the International Semantic Web Conference*, volume 2980. CEUR, 2021.
- [45] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina 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)*, pages 4171–4186, 2019.
- [46] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Workshop on Linked Data on the Web, 23rd International World Wide Web Conference*, pages 1–5, 2014.
- [47] Chuntao Ding, Ao Zhou, Yunxin Liu, Rong N Chang, Ching-Hsien Hsu, and Shangguang Wang. A cloud-edge collaboration framework for cognitive service. *IEEE Transactions on Cloud Computing*, 10(3):1489–1499, 2020.

- [48] Simone Disabato and Manuel Roveri. Incremental On-Device Tiny Machine Learning. In *Proceedings of the 2nd International workshop on challenges in artificial intelligence and machine learning for internet of things*, pages 7–13, 2020.
- [49] Qiang Duan, Nirwan Ansari, and Mehmet Toy. Software-defined network virtualization: An architectural framework for integrating SDN and NFV for service provisioning in future networks. *IEEE Network*, 30(5):10–16, 2016.
- [50] Rob Enns, Martin Bjorklund, and Juergen Schoenwaelder. NETCONF configuration protocol, Dec 2006.
- [51] Timofey Ermilov, Norman Heino, and Sören Auer. Ontowiki mobile: knowledge management in your pocket. In *Proceedings of the 20th International Conference Companion on World Wide Web*, pages 33–34, 2011.
- [52] Timofey Ermilov, Ali Khalili, and Sören Auer. Ubiquitous semantic applications: a systematic literature review. *International Journal on Semantic Web and Information Systems*, 10(1):66–99, 2014.
- [53] Adil Fahad, Najlaa Alshatri, Zahir Tari, Abdullah Alamri, Ibrahim Khalil, Albert Y Zomaya, Sebti Foufou, and Abdelaziz Bouras. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE transactions on Emerging Topics in Computing*, 2(3):267–279, 2014.
- [54] Viviane Cunha Farias da Costa, Luiz Oliveira, and Jano de Souza. Internet of everything (IoE) taxonomies: A survey and a novel knowledge-based taxonomy. *Sensors*, 21(2):568, 2021.
- [55] Kazi Fatema, Samrat Kumar Dey, Mehrin Anannya, Risala Tasin Khan, Mohammad Mamunur Rashid, Chunhua Su, and Rashed Mazumder. Federated XAI IDS: An explainable and safeguarding privacy approach to detect intrusion combining federated learning and SHAP. *Future Internet*, 17(6):234, 2025.

- [56] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *Proceedings of the Sixteenth International Conference on Machine Learning*, volume 99, pages 124–133. Citeseer, 1999.
- [57] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2):131–163, 1997.
- [58] Stephen I Gallant et al. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990.
- [59] Maurizio Giacobbe, Francesco Alessi, Angelo Zaia, Antonio Puliafito, et al. Arancino.cc: an open hardware platform for urban regeneration. *INTERNATIONAL JOURNAL OF SIMULATION & PROCESS MODELLING*, 15(4):343–357, 2020.
- [60] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.
- [61] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.
- [62] Andreas Grafberger, Mohak Chadha, Anshul Jindal, Jianfeng Gu, and Michael Gerndt. FedLess: Secure and Scalable Federated Learning Using Serverless Computing. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 164–173, 2021.
- [63] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.
- [64] Guus Schreiber and Yves Raimond. RDF 1.1 Primer. W3C Recommendation, W3C, June 2014. <https://www.w3.org/TR/rdf11-primer/>.

- [65] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic Web*, 2(1):11–21, 2011.
- [66] Matthew Horridge and Peter Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax (Second Edition). W3C note, W3C, December 2012. <http://www.w3.org/TR/owl2-manchester-syntax>.
- [67] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosz, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, May 2014. <https://www.w3.org/Submission/SWRL/>.
- [68] Ziad Hussein, May A Salama, and Sahar A El-Rahman. Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms. *Cybersecurity*, 6(1):30, 2023.
- [69] Saverio Ieva, Davide Loconte, Giuseppe Loseto, Michele Ruta, Floriano Scioscia, Davide Marche, and Marianna Notarnicola. A Retrieval-Augmented Generation Approach for Data-Driven Energy Infrastructure Digital Twins. *Smart Cities*, 7(6):3095–3120, 2024.
- [70] Saverio Ieva, Davide Loconte, Agnese Pinto, Floriano Scioscia, and Michele Ruta. Semantic-Based Decision Support for Network Management and Orchestration. In *International Conference on Web Engineering*, pages 125–136. Springer, 2022.
- [71] Kris Jamsa. *Cloud computing*. Jones & Bartlett Learning, 2022.
- [72] Andras Janosi, William Steinbrunn, Matthias Pfisterer, and Robert Detrano. Heart Disease. UCI Machine Learning Repository, 1989. DOI: <https://doi.org/10.24432/C52P4X>.
- [73] Shashi Bhushan Jha and Radu F Babiceanu. Deep CNN-based visual defect detection: Survey of current literature. *Computers in Industry*, 148:103911, 2023.
- [74] Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G Skjæveland, Evgenij Thorstensen,

- and Jose Mora. BootOX: Practical mapping of RDBs to OWL 2. In *Proceedings of the 14th International Semantic Web Conference, Part II*, pages 113–132. Springer, 2015.
- [75] Devika Kannan, Parvaneh Gholipour, and Chunguang Bai. Smart manufacturing as a strategic tool to mitigate sustainable manufacturing challenges: a case approach. *Annals of Operations Research*, 331(1):543–579, 2023.
 - [76] Aleksei Karavaev, Jan Hejda, Patrik Kutilek, Petr Volf, Marek Sokol, and Lydie Leova. TinyDecisionTreeClassifier: Embedded C++ library for training and applying decision trees on the edge. *SoftwareX*, 27:101778, 2024.
 - [77] Yevgeny Kazakov. Consequence-driven reasoning for Horn SHIQ ontologies. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.
 - [78] Yevgeny Kazakov and Pavel Klinov. Experimenting with ELK Reasoner on Android. In *2nd International Workshop on OWL Reasoner Evaluation (ORE-2013)*, pages 68–74, 2013.
 - [79] Rahima Khanam, Muhammad Hussain, Richard Hill, and Paul Allen. A comprehensive review of convolutional neural networks for defect detection in industrial applications. *IEEE Access*, 2024.
 - [80] Taehun Kim, Insuk Park, Soon J Hyun, and Dongman Lee. MiRE4OWL: Mobile Rule Engine for OWL. In *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops*, pages 317–322. IEEE, 2010.
 - [81] F. Koch. 3APL-M platform for deliberative agents in mobile devices. In *Proc. of the fourth international joint conference on Autonomous agents and multiagent systems*, page 154. ACM, 2005.
 - [82] A Kojukhov, AM de Nicolas, B Chatras, D Druta, D Gassanov, M Brunner, M Brenner, S Li, T Nguyenphu, U Rauschenbach, et al. Network functions virtualisation (nfv) release 2; protocols and data models; vnf

package specification. GS NFV-SOL 004 V2. 3.1. *Group specification*, ETSI, 2017.

- [83] Nikita Kotsehub, Matt Baughman, Ryan Chard, Nathaniel Hudson, Panos Patros, Omer Rana, Ian Foster, and Kyle Chard. FLoX: Federated Learning with FaaS at the Edge. In *2022 IEEE 18th International Conference on e-Science (e-Science)*, pages 11–20, 2022.
- [84] Dimitri Kraft, Karthik Srinivasan, and Gerald Bieber. Deep learning based fall detection algorithms for embedded systems, smartwatches, and IoT devices using accelerometers. *Technologies*, 8(4):72, 2020.
- [85] Thorsten Kramp, Rob Van Kranenburg, and Sebastian Lange. Introduction to the Internet of Things. In *Enabling things to talk: Designing IoT solutions with the IoT architectural reference model*, pages 1–10. Springer Berlin Heidelberg Berlin, Heidelberg, 2013.
- [86] Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine*, 80:11–28, 2017.
- [87] J. Lee, Behrad Bagheri, and Hung-An Kao. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing letters*, 3:18–23, 2015.
- [88] Jay Lee, Hossein Davari, Jaskaran Singh, and Vibhor Pandhare. Industrial Artificial Intelligence for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 18:20–23, 2018.
- [89] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. Dbpedia—a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

- [90] Jiewu Leng, Weinan Sha, Baicun Wang, Pai Zheng, Cunbo Zhuang, Qiang Liu, Thorsten Wuest, Dimitris Mourtzis, and Lihui Wang. Industry 5.0: Prospect and retrospect. *Journal of Manufacturing Systems*, 65:279–295, 2022.
- [91] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.
- [92] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.
- [93] Yuan-Fang Li, Jeff Z Pan, Manfred Hauswirth, and Hai Nguyen. The Ubiquitous Semantic Web: Promises, Progress and Challenges. In *Web-Based Services: Concepts, Methodologies, Tools, and Applications*, pages 272–289. IGI Global, Hershey, PA, USA, 2016.
- [94] Yujin Li and Wenye Wang. The unheralded power of cloudlet computing in the vicinity of mobile devices. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 4994–4999. IEEE, 2013.
- [95] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.
- [96] Jixiong Liu, Yoan Chabot, Raphaël Troncy, Viet-Phi Huynh, Thomas Labbé, and Pierre Monnin. From tabular data to knowledge graphs: A survey of semantic table interpretation tasks and methods. *Journal of Web Semantics*, page 100761, 2022.
- [97] Jixiong Liu and Raphaël Troncy. DAGOBAB: an end-to-end context-free tabular data semantic annotation system. In *Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Workshop, International Semantic Web Conference*, pages 41–48, 2019.

- [98] Kalee Lodewyk, Madeleine Wiebe, Liz Dennett, Jake Larsson, Andrew Greenshaw, and Jake Hayward. Wearables research for continuous monitoring of patient outcomes: A scoping review. *PLOS Digital Health*, 4(5):e0000860, 2025.
- [99] Hui Wen Loh, Chui Ping Ooi, Silvia Seoni, Prabal Datta Barua, Filippo Molinari, and U Rajendra Acharya. Application of explainable artificial intelligence for healthcare: A systematic review of the last decade (2011–2022). *Computer methods and programs in biomedicine*, 226:107161, 2022.
- [100] Victor Lopez, Ricard Vilalta, Victor Uceda, Arturo Mayoral, Ramon Casellas, Ricardo Martinez, Raul Munoz, and Juan Pedro Fernandez Palacios. Transport API: A solution for SDN in carriers networks. In *ECOC 2016; 42nd European Conference on Optical Communication*, pages 1–3. VDE, 2016.
- [101] Giuseppe Loseto, Ivano Bilenchi Filippo Gramegna, Davide Loconte, Floriano Scioscia, and Michele Ruta. Tiny-ME Wasm: Description Logics. In *Current Trends in Web Engineering: ICWE 2023 International Workshops: BECS, SWEET, WALIS, Alicante, Spain, June 6–9, 2023, Revised Selected Papers*, page 114. Springer Nature, 2024.
- [102] Giuseppe Loseto, Floriano Scioscia, Michele Ruta, Filippo Gramegna, Saverio Ieva, Corrado Fasciano, Ivano Bilenchi, and Davide Loconte. Osmotic Cloud-Edge Intelligence for IoT-Based Cyber-Physical Systems. *Sensors*, 22(6), 2022.
- [103] Giuseppe Loseto, Floriano Scioscia, Michele Ruta, Filippo Gramegna, Saverio Ieva, Corrado Fasciano, Ivano Bilenchi, Davide Loconte, and Eugenio Di Sciascio. A Cloud-Edge Artificial Intelligence Framework for Sensor Networks. In *9th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, pages 149–154, 2023.
- [104] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017.

- [105] AV Shreyas Madhav and Amit Kumar Tyagi. Explainable Artificial Intelligence (XAI): connecting artificial decision-making and human trust in autonomous vehicles. In *Proceedings of Third International Conference on Computing, Communications, and Cyber-Security: IC4S 2021*, pages 123–136. Springer, 2022.
- [106] Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, and Hamed Haddadi. Mobile Sensor Data Anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation, IoTDI '19*, pages 49–58, New York, NY, USA, 2019. ACM.
- [107] Luis Martí, Nayat Sanchez-Pi, José Manuel Molina, and Ana Cristina Bicharra Garcia. Anomaly detection based on sensor data in petroleum industry applications. *Sensors*, 15(2):2774–2797, 2015.
- [108] Balduino César Mateus, Mateus Mendes, José Torres Farinha, Rui Assis, and António Marques Cardoso. Comparing LSTM and GRU models to predict the condition of a pulp paper press. *Energies*, 14(21):6958, 2021.
- [109] Charles Meijer, Hae-Won Uh, and Said El Bouhaddani. Digital twins in healthcare: Methodological challenges and opportunities. *Journal of personalized medicine*, 13(10):1522, 2023.
- [110] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J. Hibbett, Giovanni Estrada, Khaldun Ma’ruf, Florin Coras, Vina Ermagan, Hugo Latapie, Chris Cassar, John Evans, Fabio Maino, Jean Walrand, and Albert Cabellos. Knowledge-Defined Networking. *SIGCOMM Comput. Commun. Rev.*, 47(3):210, sep 2017.
- [111] Gianfranco E Modoni and Marco Sacco. Discovering Critical Factors Affecting RDF Stores Success. In Rajic Pandey, Marcin Paprzycki, Nidhi Srivastava, Subhash Bhalla, and Katarzyna Wasielewska-Michniewska, editors, *Semantic IoT: Theory and Applications: Interoperability, Provenance and Beyond*, pages 193–206. Springer, 2021.

- [112] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: an overview. *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 193–209, 2019.
- [113] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 607–617, 2020.
- [114] Boris Motik, Ian Horrocks, and Su Myeon Kim. Delta-Reasoner: a Semantic Web reasoner for an intelligent mobile platform. In *Proceedings of the 21st International Conference on World Wide Web*, pages 63–72, New York, NY, USA, 2012. ACM.
- [115] Boris Motik, Bijan Parsia, and Peter Patel-Schneider. OWL 2 Web Ontology Language XML Serialization (Second Edition). Recommendation, W3C, December 2012. <https://www.w3.org/TR/owl2-xml-serialization>.
- [116] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- [117] Davide Loconte, Saverio Ieva, Filippo Gramegna, Ivano Bilenchi, Corrado Fasciano, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, Michele Ruta, and Eugenio Di Sciascio. Serverless microservice architecture for cloud-edge intelligence in sensor networks. *IEEE Sensors Journal*, 25(5):78757885, 2024.
- [118] Davide Loconte, Saverio Ieva, Grazia Mascellaro, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. On-device explainable artificial intelligence for the semantic web of everything. *Future Generation Computer Systems*, page 108310, 2025. In press.
- [119] Davide Loconte, Saverio Ieva, Agnese Pinto, Filippo Gramegna, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. A Serverless Cloud-to-Thing Framework for TinyMLOps Workflows. In *1st ECAI*

Workshop on Machine Learning Operations (MLOps). CEUR-WS, oct 2025.

- [120] Davide Loconte, Saverio Ieva, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. Expanding the cloud-to-edge continuum to the IoT in serverless federated learning. *Future Generation Computer Systems*, 155:447–462, 2024.
- [121] Dipanjan Nag, Freyja Brandel-Tanis, Zakiya Aryana Pramestri, Kelly Pitera, and Yngve Karl Frøyen. Exploring digital twins for transport planning: a review. *European Transport Research Review*, 17(1):15, 2025.
- [122] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. *ACM Sigplan notices*, 42(6):89–100, 2007.
- [123] Jer Shyuan Ng, Wei Yang Bryan Lim, Zehui Xiong, Xianbin Cao, Jiangming Jin, Dusit Niyato, Cyril Leung, and Chunyan Miao. Reputation-Aware Hedonic Coalition Formation for Efficient Serverless Hierarchical Federated Learning. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):2675–2686, 2022.
- [124] OASIS Standard. MQTT Version 3.1.1. <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>, 2014. OASIS Standard, 29 October 2014.
- [125] Martin J O’Connor, Christian Halaschek-Wiener, and Mark A Musen. Mapping master: a flexible approach for mapping spreadsheets to OWL. In *International Semantic Web Conference*, pages 194–208. Springer, 2010.
- [126] Franklin Oliveira, Daniel G Costa, Flávio Assis, and Ivanovitch Silva. Internet of Intelligent Things: A convergence of embedded systems, edge computing and machine learning. *Internet of Things*, page 101153, 2024.

- [127] A Paramasivam, M Jenath, TS Sivakumaran, Sakthivel Sankaran, Pavan Sai Kiran Reddy Pittu, S Vijayalakshmi, et al. Development of artificial intelligence edge computing based wearable device for fall detection and prevention of elderly people. *Heliyon*, 10(8), 2024.
- [128] Bijan Parsia, Boris Motik, and Peter Patel-Schneider. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). Recommendation, W3C, December 2012. <http://www.w3.org/TR/owl2-syntax/>.
- [129] Bijan Parsia, Sebastian Rudolph, Markus Krötzsch, Peter Patel-Schneider, and Pascal Hitzler. OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation, W3C, December 2012. <http://www.w3.org/TR/owl2-primer>.
- [130] P. Patros, M. Ooi, V. Huang, M. Mayo, C. Anderson, S. Burroughs, M. Baughman, O. Almurshed, O. Rana, R. Chard, K. Chard, and I. Foster. Rural AI: Serverless-Powered Federated Learning for Remote Applications. *IEEE Internet Computing*, 27(02):28–34, mar 2023.
- [131] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [132] Agnese Pinto, Saverio Ieva, Arnaldo Tomasino, Giuseppe Loseto, Floriano Scioscia, Michele Ruta, and Francesco De Feudis. A Framework for Automatic Knowledge Base Generation from Observation Data Sets. In *International Conference on Web Engineering*, pages 89–100. Springer, 2023.
- [133] Eric Prud’hommeaux and Gavin Carothers. RDF 1.1 Turtle. Recommendation, W3C, February 2014. <https://www.w3.org/TR/turtle/>.
- [134] Tie Qiu, Jiancheng Chi, Xiaobo Zhou, Zhaolong Ning, Mohammed Atiquzzaman, and Dapeng Oliver Wu. Edge computing in Industrial

- Internet of Things: Architecture, advances and challenges. *IEEE Communications Surveys & Tutorials*, 22(4):2462–2488, 2020.
- [135] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.
 - [136] Bharat Rathi, S Thapaswi, Meghana Kambhampati, Vineet Jain, P Akshay, Trilok Nath Pandey, and Sunil Kumar Pradhan. Realizing the potential of Internet of Things (IoT) in Industrial applications. *Discover Internet of Things*, 5(1):1–16, 2025.
 - [137] Mubashar Raza, Zarmina Jahangir, Muhammad Bilal Riaz, Muhammad Jasim Saeed, and Muhammad Awais Sattar. Industrial applications of large language models. *Scientific Reports*, 15(1):13755, 2025.
 - [138] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why should i trust you?” Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.
 - [139] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.
 - [140] Michele Ruta, Eugenio Di Sciascio, and Floriano Scioscia. Concept Abduction and Contraction in Semantic-based P2P Environments. *Web Intelligence and Agent Systems*, 9(3):179–207, 2011.
 - [141] Michele Ruta, Floriano Scioscia, Ivano Bilenchi, Filippo Gramegna, Giuseppe Loseto, Saverio Ieva, and Agnese Pinto. A multiplatform reasoning engine for the Semantic Web of Everything. *Journal of Web Semantics*, 73:100709, 2022.
 - [142] Michele Ruta, Floriano Scioscia, Eugenio Di Sciascio, Domenico Rotondi, and Salvatore Piccione. Semantic-based knowledge dissemination and extraction in smart environments. In *2013 27th International Conference*

- on *Advanced Information Networking and Applications Workshops*, pages 1289–1294. IEEE, 2013.
- [143] Michele Ruta, Floriano Scioscia, Giuseppe Loseto, and Eugenio Di Sciascio. Semantic-based resource discovery and orchestration in home and building automation: A multi-agent approach. *IEEE Transactions on Industrial Informatics*, 10(1):730–741, 2013.
 - [144] Michele Ruta, Floriano Scioscia, Giuseppe Loseto, Agnese Pinto, and Eugenio Di Sciascio. Machine learning in the Internet of Things: A semantic-enhanced approach. *Semantic Web*, 10(1):183–204, 2019.
 - [145] Mutaz Ryalat, Hisham ElMoaqet, and Marwa AlFaouri. Design of a smart factory based on cyber-physical systems and Internet of Things towards Industry 4.0. *Applied Sciences*, 13(4):2156, 2023.
 - [146] Eveline R Sacramento, Vânia MP Vidal, José Antonio F de Macêdo, Bernadette F Lóscio, Fernanda Lígia R Lopes, and Marco A Casanova. Towards automatic generation of application ontologies. *Journal of Information and Data Management*, 1(3):535–535, 2010.
 - [147] Waddah Saeed and Christian Omlin. Explainable AI (XAI): A systematic meta-survey of current challenges and future opportunities. *Knowledge-based systems*, 263:110273, 2023.
 - [148] Fouad Sakr, Riccardo Berta, Joseph Doyle, Alessandro De Gloria, and Francesco Bellotti. Self-learning pipeline for low-energy resource-constrained devices. *Energies*, 14(20):6636, 2021.
 - [149] Deepti Saraswat, Pronaya Bhattacharya, Ashwin Verma, Vivek Kumar Prasad, Sudeep Tanwar, Gulshan Sharma, Pitshou N Bokoro, and Ravi Sharma. Explainable AI for healthcare 5.0: opportunities and challenges. *IEEE Access*, 10:84486–84517, 2022.
 - [150] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

- [151] Nikolaos Schizas, Aristeidis Karras, Christos Karras, and Spyros Sioutas. TinyML for ultra-low power AI and large scale IoT deployments: A systematic review. *Future Internet*, 14(12):363, 2022.
- [152] Guus Schreiber and Fabien Gandon. RDF 1.1 XML Syntax. Recommendation, W3C, February 2014. <http://www.w3.org/TR/rdf-syntax-grammar/>.
- [153] Guus Schreiber and Fabien Gandon. RDF-star and SPARQL-star, June 2023. https://w3c.github.io/rdf-star/cg-spec/editors_draft.html.
- [154] Floriano Scioscia, Ivano Bilenchi, Michele Ruta, Filippo Gramegna, and Davide Loconte. A multiplatform energy-aware OWL reasoner benchmarking framework. *Journal of Web Semantics*, 72:100694, 2022.
- [155] Floriano Scioscia and Michele Ruta. Building a Semantic Web of Things: issues and perspectives in information compression. In *2009 IEEE International Conference on Semantic Computing*, pages 589–594. IEEE, 2009.
- [156] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. Serverless Computing: A Survey of Opportunities, Challenges, and Applications. *ACM Computing Surveys*, 54(11s), nov 2022.
- [157] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.
- [158] Abhigya Pote Shrestha and Chen-Hsiang Yu. ECG data analysis with IoT and machine learning. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0323–0327. IEEE, 2022.
- [159] Manan Shukla and Oshani Seneviratne. MentalHealthAI: Utilizing Personal Health Device Data to Optimize Psychiatry Treatment. In *AMIA Annual Symposium Proceedings*, volume 2023, page 641. American Medical Informatics Association, 2023.

- [160] Bhagya Nathali Silva, Murad Khan, and Kijun Han. Internet of things: A comprehensive review of enabling technologies, architecture, and challenges. *IETE Technical review*, 35(2):205–220, 2018.
- [161] Vandana Singh, Sudip Kumar Sahana, and Vandana Bhattacharjee. A novel CNN-GRU-LSTM based deep learning model for accurate traffic prediction. *Discover Computing*, 28(1):38, 2025.
- [162] Alex Sinner and Thomas Kleemann. KRHyper—in your pocket. In *International Conference on Automated Deduction*, pages 452–457. Springer, 2005.
- [163] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.
- [164] Luke Steller and Shonali Krishnaswamy. Pervasive Service Discovery: mTableaux Mobile Reasoning. In *International Conference on Semantic Systems (I-Semantics)*, pages 93–101, Graz, Austria, 2008. TU Graz.
- [165] Liyilei Su, Xumin Zuo, Rui Li, Xin Wang, Heng Zhao, and Bingding Huang. A systematic review for transformer-based long-term series forecasting. *Artificial Intelligence Review*, 58(3):80, 2025.
- [166] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. Edge2Train: A Framework to Train Machine Learning Models (SVMs) on Resource-Constrained IoT Edge Devices. In *Proceedings of the 10th International Conference on the Internet of Things*, pages 1–8, 2020.
- [167] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. ML-MCU: A Framework to Train ML Classifiers on MCU-Based IoT Edge Devices. *IEEE Internet of Things Journal*, 9(16):15007–15017, 2021.
- [168] Bharath Sudharsan, Piyush Yadav, John G Breslin, and Muhammad Intizar Ali. Train++: An Incremental ML Model Training Algorithm to Create Self-Learning IoT Devices. In *2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart*

- City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, pages 97–106. IEEE, 2021.
- [169] Lanfang Sun, Xin Jiang, Huixia Ren, and Yi Guo. Edge-Cloud Computing and Artificial Intelligence in Internet of Medical Things: Architecture, Technology and Application. *IEEE Access*, 8:101079–101092, 2020.
 - [170] Tao Sun, Dongsheng Li, and Bao Wang. Decentralized federated averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4289–4301, 2022.
 - [171] Csaba Szepesvári. *Algorithms for reinforcement learning*. Springer Nature, 2022.
 - [172] W. Tai, J. Keeney, and D. O’Sullivan. COROR: a composable rule-entailment owl reasoner for resource-constrained devices. *Rule-Based Reasoning, Programming, and Applications*, pages 212–226, 2011.
 - [173] John R Talburt, Lisa Ehrlinger, and Justin Magruder. Automated data curation and data governance automation. *Frontiers in Big Data*, 6, 2023.
 - [174] Gunnar Teege. Making the difference: A subtraction operation for description logics. In *Principles of Knowledge Representation and Reasoning*, pages 540–550. Elsevier, 1994.
 - [175] Mehdi Terdjimi, Lionel Médini, and Michael Mrissa. HyLAR+ improving hybrid location-agnostic reasoning with incremental rule-based update. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 259–262, 2016.
 - [176] Daniel R Torres, Cristian Martín, Bartolomé Rubio, and Manuel Díaz. An open source framework based on Kafka-ML for Distributed DNN inference over the Cloud-to-Things continuum. *Journal of Systems Architecture*, 118:102214, 2021.

- [177] Daniele Tovazzi, Francescomaria Faticanti, Domenico Siracusa, Claudio Peroni, Silvio Cretti, and Tommaso Gazzini. GEM-Analytics: Cloud-to-Edge AI-Powered Energy Management. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 57–66. Springer, 2020.
- [178] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *International Joint Conference on Automated Reasoning (IJCAR)*, pages 292–297, Berlin, Germany, 2006. Springer.
- [179] Mathias UsLAR, Sebastian Rohjans, Christian Neureiter, Filip Prösl Andrén, Jorge Velasquez, Cornelius Steinbrink, Venizelos Efthymiou, Gianluigi Migliavacca, Seppo Horsmanheimo, Helfried Brunner, et al. Applying the smart grid architecture model for designing and validating system-of-systems in the power and energy domain: A European perspective. *Energies*, 12(2):258, 2019.
- [180] Theo Van Veen. Wikidata: from “an” identifier to “the” identifier. *Information technology and libraries*, 38(2):72–81, 2019.
- [181] William Van Woensel and Syed Sibte Raza Abidi. Optimizing Semantic Reasoning on Memory-Constrained Platforms Using the RETE Algorithm. In *Extended Semantic Web Conference (ESWC)*, pages 682–696. Springer, 2018.
- [182] William Van Woensel and Syed Sibte Raza Abidi. Benchmarking semantic reasoning on mobile platforms: Towards optimization using OWL2 RL. *Semantic Web*, 10(4):637–663, 2019.
- [183] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [184] Ruben Verborgh and Jos De Roo. Drawing conclusions from Linked Data on the Web: the EYE reasoner. *IEEE Software*, 32(3):23–27, 2015.

- [185] Silviu Vert, Bogdan Dragulescu, and Radu Vasiu. LOD4AR: Exploring Linked Open Data with a Mobile Augmented Reality Web Application. In *ISWC (Posters & Demos)*, pages 185–188, 2014.
- [186] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana, and Rajiv Ranjan. Osmotic Computing: A New Paradigm for Edge/Cloud Integration. *IEEE Cloud Computing*, 3(6):76–83, 2016.
- [187] Binh Vu, Craig Knoblock, and Jay Pujara. Learning semantic models of data sources using probabilistic graphical models. In *The World Wide Web Conference*, pages 1944–1953, 2019.
- [188] Bernhard Walzl and Roland Vogl. Explainable artificial intelligence the new frontier in legal informatics. *Jusletter IT*, 4:1–10, 2018.
- [189] Jiaqi Wang, Enze Shi, Huawei Hu, Chong Ma, Yiheng Liu, Xuhui Wang, Yincheng Yao, Xuan Liu, Bao Ge, and Shu Zhang. Large language models for robotics: Opportunities, challenges, and perspectives. *Journal of Automation and Intelligence*, 4(1):52–64, 2025.
- [190] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(2), 2009.
- [191] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–105, 1991.
- [192] Daniel L Weller, Tanzy MT Love, and Martin Wiedmann. Interpretability versus accuracy: a comparison of machine learning models built using different algorithms, performance measures, and features to predict E. coli levels in agricultural water. *Frontiers in artificial intelligence*, 4:628441, 2021.
- [193] Ian H Witten and Eibe Frank. Data mining: practical machine learning tools and techniques with Java implementations. *Acm Sigmod Record*, 31(1):76–77, 2002.

- [194] William Wolberg, Olvi Mangasarian, and Nick Street. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1993. DOI: <https://doi.org/10.24432/C5DW2B>.
- [195] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.
- [196] David Wood, Marsha Zaidman, Luke Ruth, and Michael Hausenblas. *Linked Data*. Manning publications co., 2014.
- [197] Lars Wulfert, Johannes Kühnel, Lukas Krupp, Justus Viga, Christian Wiede, Pierre Gembaczka, and Anton Grabmaier. AIfES: A Next-Generation Edge AI Framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [198] Lars Wulfert, Christian Wiede, and Anton Grabmaier. TinyFL: On-Device Training, Communication and Aggregation on a Microcontroller for Federated Learning. In *2023 21st IEEE Interregional NEWCAS Conference (NEWCAS)*, pages 1–5. IEEE, 2023.
- [199] Jiangchao Yao, Shengyu Zhang, Yang Yao, Feng Wang, Jianxin Ma, Jianwei Zhang, Yunfei Chu, Luo Ji, Kunyang Jia, Tao Shen, et al. Edge-cloud polarization and collaboration: A comprehensive survey for ai. *IEEE Transactions on Knowledge and Data Engineering*, 35(7):6866–6886, 2022.
- [200] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
- [201] Roberto Yus, Carlos Bobed, Guillermo Esteban, Fernando Bobillo, and Eduardo Mena. Android goes semantic: DL reasoners on smartphones. In *Proceedings of 2nd International Workshop on OWL Reasoner Evaluation (ORE’13), Ulm (Germany)*, pages 46–52, 2013.

- [202] Yingfeng Zhang, Cheng Qian, Jingxiang Lv, and Ying Liu. Agent and Cyber-Physical System Based Self-Organizing and Self-Adaptive Intelligent Shopfloor. *IEEE Transactions on Industrial Informatics*, 13(2):737–747, 2016.
- [203] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

List of publications

The author's contributions are detailed according to the CRediT taxonomy.

International peer-reviewed journal articles

1. **Davide Loconte**, Saverio Ieva, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. Expanding the cloud-to-edge continuum to the IoT in serverless federated learning. *Future Generation Computer Systems*, 155:447–462, 2024

Contributions: Methodology, Software, Investigation, Data curation, Visualization, Writing – original draft.

2. **Davide Loconte**, Saverio Ieva, Filippo Gramegna, Ivano Bilenchi, Corrado Fasciano, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, Michele Ruta, and Eugenio Di Sciascio. Serverless microservice architecture for cloud-edge intelligence in sensor networks. *IEEE Sensors Journal*, 25(5):78757885, 2024

Contributions: Methodology, Software, Investigation, Data curation, Visualization, Writing – original draft.

3. Saverio Ieva, **Davide Loconte**, Giuseppe Loseto, Michele Ruta, Floriano Scioscia, Davide Marche, and Marianna Notarnicola. A Retrieval-Augmented Generation Approach for Data-Driven Energy Infrastructure Digital Twins. *Smart Cities*, 7(6):3095–3120, 2024

Contributions: Methodology, Validation, Investigation, Resources, Writing – original draft, Visualization.

4. Ivano Bilenchi, **Davide Loconte**, Floriano Scioscia, and Michele Ruta. Evaluating Correctness, Performance and Energy Footprint of Semantic

Reasoners in Mobile Edge Computing. *Journal of Systems and Software*, 233(112696), 2026 (in press; accepted 2025)

Contributions: Writing – review & editing, Writing – original draft, Visualization, Validation, Resources, Investigation, Formal analysis.

5. **Davide Loconte**, Saverio Ieva, Grazia Mascellaro, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. On-device explainable artificial intelligence for the semantic web of everything. *Future Generation Computer Systems*, page 108310, 2025. In press

Contributions: Writing – review & editing, Writing – original draft, Conceptualization, Methodology, Software, Investigation, Resources, Visualization.

International peer-reviewed conference papers

1. Giuseppe Loseto, Ivano Bilenchi Filippo Gramegna, **Davide Loconte**, Floriano Scioscia, and Michele Ruta. Tiny-ME Wasm: Description Logics. In *Current Trends in Web Engineering: ICWE 2023 International Workshops: BECS, SWEET, WALs, Alicante, Spain, June 6–9, 2023, Revised Selected Papers*, page 114. Springer Nature, 2024

Contributions: Software, Validation, Investigation, Visualization, Writing original draft

2. Giuseppe Loseto, Floriano Scioscia, Michele Ruta, Filippo Gramegna, Saverio Ieva, Corrado Fasciano, Ivano Bilenchi, **Davide Loconte**, and Eugenio Di Sciascio. A Cloud-Edge Artificial Intelligence Framework for Sensor Networks. In *9th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, pages 149–154, 2023

Contributions: Software, Investigation, Data curation, Writing – original draft preparation, Visualization.

3. **Davide Loconte**, Saverio Ieva, Agnese Pinto, Filippo Gramegna, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. A Serverless

Cloud-to-Thing Framework for TinyMLOps Workflows. In *1st ECAI Workshop on Machine Learning Operations (MLOps)*. CEUR-WS, oct 2025

Contributions: Writing – review & editing, Writing – original draft, Visualization, Validation, Resources, Investigation, Formal analysis.