information

MDPI

Article

Real-Time Speech-to-Text on Edge: A Prototype System for Ultra-Low Latency Communication with AI-Powered NLP

Stefano Di Leo, Luca De Cicco and Saverio Mascolo *

Dipartimento di Ingegneria Elettrica e dell'Informazione, Politecnico di Bari, 70125 Bari, Italy; stefano.dileo@poliba.it (S.D.L.); luca.decicco@poliba.it (L.D.C.)
* Correspondence: saverio.mascolo@poliba.it

Abstract

This paper presents a real-time speech-to-text (STT) system designed for edge computing environments requiring ultra-low latency and local processing. Differently from cloud-based STT services, the proposed solution runs entirely on a local infrastructure which allows the enforcement of user privacy and provides high performance in bandwidth-limited or offline scenarios. The designed system is based on a browser-native audio capture through WebRTC, real-time streaming with WebSocket, and offline automatic speech recognition (ASR) utilizing the Vosk engine. A natural language processing (NLP) component, implemented as a microservice, improves transcription results for spelling accuracy and clarity. Our prototype reaches sub-second end-to-end latency and strong transcription capabilities under realistic conditions. Furthermore, the modular architecture allows extensibility, integration of advanced AI models, and domain-specific adaptations.

Keywords: speech-to-text (STT); automatic speech recognition (ASR); edge computing; natural language processing (NLP); real-time communication; WebRTC; WebSocket; offline ASR engines; low-latency AI

Academic Editor: Arkaitz Zubiaga

Received: 31 May 2025
Revised: 29 July 2025
Accepted: 8 August 2025
Published: 11 August 2025

Citation: Di Leo, S.; De Cicco, L.; Mascolo, S. Real-Time Speech-to-Text on Edge: A Prototype System for Ultra-Low Latency Communication with AI-Powered NLP. *Information* **2025**, *16*, 685. https://doi.org/10.3390/info16080685

1. Introduction

Speech technology innovation, particularly in fields that require seamless and real-time human-to-machine interactions, is nowadays driven by the combined use of natural language processing (NLP), Artificial Intelligence (AI), and real-time communication platforms. Automatic speech recognition (ASR), text-to-speech (TTS), and language comprehension technologies are driving the progress of interactive voice-based applications in medical diagnosis, industrial automation, intelligent assistants, and multilingual communication frameworks.

Recent progress in self-supervised learning and transformer models has changed the fundamental techniques of speech recognition. For instance, Wav2Vec 2.0 proves remarkable performance with limited labeled data by leveraging plentiful unlabeled audio for pre-training and applying contrastive learning to predict masked latent speech characteristics [1]. Similarly, transformer models employing an attention mechanism (like BERT, GPT, and HuBERT) show improved capability to understand long-range dependencies in sequential data, enhancing both the recognition and generation phases of speech and language processing [2,3].

In addition to these enhancements at the model level, advancements in real-time transmission systems have enabled the use of these technologies in low-latency settings. *Web Real-Time Communication*, a crucial component of modern speech systems, offers

features for direct audio transmission, echo cancellation, jitter buffering, and adaptive bitrate control [4,5]. These functions are required in ultra-low-latency communications, where responsiveness is a key factor such as in the case of interactive web transcription services or voice-activated robotic systems.

Shifting from cloud-based processing to edge-centric inference reduces end-to-end latency as well as enforcing user privacy. Edge computing platforms, together with robust speech engines such as Vosk and Kaldi, allow local processing for speech tasks while conserving resources effectively [6,7]. This hybrid architecture—cloud-supported for model improvements and device management, edge-driven for inference—provides a scalable approach for speech interfaces that endure fluctuations in connectivity and server-side restrictions [4,6].

In this work, we propose a prototype system that integrates real-time speech capture and transcription capabilities with intelligent language correction, optimized for edge deployment. The system employs WebRTC for media acquisition, WebSocket for real-time communication, and an offline speech recognition engine (Vosk) for inference. An auxiliary NLP microservice performs orthographic correction, ensuring that the output is both accurate and human-readable. This hybrid architecture provides efficient, private, and reliable transcription, also when bandwidth fluctuates or in the case of privacy-sensitive environments.

Unlike conventional systems that rely on cloud-based STT engines (e.g., Google Speech API or Microsoft Azure Speech), our solution is designed for minimal dependencies, using lightweight models deployable on constrained hardware such as Raspberry Pi or mobile edge devices. In addition, the integration of NLP techniques for post-processing differentiates this system by improving the syntactic quality of the transcription in real time, an important feature in practical use cases ranging from education to multilingual business environments.

The rest of the paper is organized as follows: Section 2 presents the background information and discusses the related work; Section 3 describes the proposed system architecture and its building blocks; Section 4 focuses on the implementation details; Section 5 validates the proposed system's functionality and provides a performance evaluation; Section 6 discusses the obtained results; and, finally, Section 7 concludes the paper, also highlighting future work and limitations.

2. Background and Related Work

The domain of ASR, TTS, and NLP technologies has experienced a notable transformation, driven by both the integration of deep learning approaches and unsupervised training techniques, and by the recent introduction of real-time communication standards such as the Web Real-Time Communication (WebRTC) initiative. Conventional ASR systems mainly relied on statistical models such as Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs), in addition to Mel-frequency cepstral coefficients for representing acoustics [7]. These systems, though effective in regulated settings, were not optimal in unplanned speech situations and demonstrated poor adaptability across languages and domains.

The emergence of end-to-end models has driven a major methodological change. Recurrent Neural Networks (RNNs), along with their variants such as LSTMs and Gated Recurrent Units (GRUs), introduced a method to capture long-term temporal relationships in audio sequences [8]. Nevertheless, their sequential characteristics presented scalability challenges, especially for real-time uses. The transformer models—grounded in self-attention—addressed these limitations, allowing effective parallel processing and improved context representation [2].

One of the most significant contributions is Wav2Vec 2.0, which employs contrastive pretraining and quantization on continuous audio signals to create strong acoustic representations. When fine-tuned with limited supervision, these embeddings perform very well in terms of Word Error Rates (WERs) even on intricate multilingual datasets [1]. Wav2Vec 2.0 has been successfully employed for low-resource languages and specific vocabularies through interlingual fine-tuning methods [9].

Toolkits such as Kaldi and Vosk operationalize these models in practical settings. Kaldi offers a research-grade ASR development framework supporting FST-based decoding, speaker adaptation, and discriminative training pipelines [6]. Vosk, built atop Kaldi's core principles, offers lightweight, language-flexible ASR modules optimized for embedded and offline scenarios [7]. It supports over 20 languages, adapts to domain-specific lexicons, and integrates with RESTful and WebSocket APIs, making it particularly suitable for real-time deployment in enterprise and industrial applications [7].

Advancements in NLP have affected the efficiency of speech systems. Contemporary NLP methods differentiate between Natural Language Understanding (NLU) and Natural Language Generation (NLG). While NLU handles tasks such as syntactic analysis, semantic tagging, and intent identification, NLG, on the other hand, focuses on generating coherent and contextually suitable replies [3]. These procedures are crucial for comprehensive dialogue systems and smart voice assistants, where speech is transcribed, comprehended, and replied to in real-time.

Today, WebRTC is a key technology and considered the de facto standard for real-time transmission in speech-enabled web and mobile interfaces. It facilitates direct media exchange using peer-to-peer channels while ensuring low-latency through built-in mechanisms for jitter buffering, packet loss concealment, and adaptive bitrate control [4,5]. The integration with Web Speech API further enhances the user experience by enabling direct STT and TTS interaction layers within browsers, bypassing the need for heavyweight client-side software. These features have been employed in scenarios such as multilingual translation, live subtitling, and interactive tutoring, where responsiveness and reliability are important [4,5,10].

Cloud–edge hybrid deployment has also gained traction, particularly in latency-sensitive use cases such as healthcare diagnostics and industrial monitoring. Real-time speech interfaces built on edge inference reduce dependency on network throughput and improve resilience. Projects such as *Screevo* show how ASR systems integrated into private intranets can operate with high accuracy even in noisy industrial environments, while preserving data security and meeting regulatory requirements [6].

As communication technologies progress toward 6G, with promises of sub-millisecond latencies and AI-native infrastructure, future speech systems will likely incorporate federated learning, zero-touch orchestration, and context-aware service adaptation [11,12]. However, current challenges remain in areas such as model bias, energy efficiency, hallucination mitigation in non-speech audio [13], and multilingual robustness, which continue to motivate new lines of research.

3. Materials and Methods

3.1. System Architecture Overview

The proposed system utilizes a layered, modular framework designed to provide real-time speech-to-text capabilities within the limitations of edge computing settings. This design is particularly tailored for low latency, data security, and operational independence. The architecture consists of three main layers: (1) the client-side interface, (2) server-side coordination and processing, and (3) supporting AI microservices for speech recognition and natural language processing.

A browser-based client at the frontend handles user audio capture, starts session events, and displays both partial and complete transcriptions. This interface is built with standard web technologies and browser APIs, allowing cross-platform compatibility without the need for additional plugins. The client creates a continuous communication link with the backend via WebSocket, enabling two-way, low-latency data transfer fine-tuned for real-time engagement.

The server layer, built with Node.js, acts as the main coordinator. It receives audio segments from the client, transforms them into the correct format, and sends them to the STT engine. The backend additionally oversees session management, data buffering, and communication protocols. Its event-driven design guarantees non-blocking execution, making it suitable for the simultaneous management of numerous concurrent sessions.

Two microservices—one for offline speech recognition and another for orthographic correction—compose the core backend. The speech recognition microservice uses the Vosk engine to infer transcriptions from audio data, while the NLP microservice processes the raw output to ensure grammatical correctness and lexical normalization. This separation of concerns adheres to microservice principles and facilitates independent scaling or upgrading of system components.

This distributed but tightly coupled architecture enables low-latency STT processing that remains functional even in bandwidth-limited or privacy-sensitive deployment scenarios. Figure 1 illustrates the interaction flow among components, from speech capture to transcription rendering.

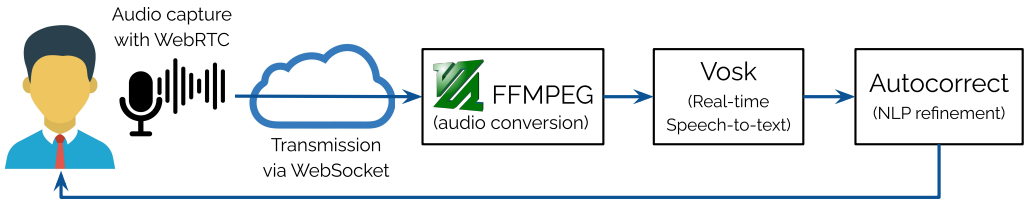

Figure 1. Overview of the system architecture, showing the interaction between the client interface, WebSocket communication channel, server-side processing modules (audio conversion, STT, NLP), and the final display of the transcribed text.

3.2. Component Description

The system's components are organized into discrete functional units, each responsible for a specific aspect of the transcription pipeline. This modularization enhances maintainability and allows for targeted optimization of individual stages.

3.2.1. Client-Side Application

The client application is implemented using JavaScript and leverages WebRTC's *getUserMedia()* API to access microphone input. Audio is captured in real time and segmented into short-duration chunks using the MediaRecorder API [14], a W3C standard feature supported by most modern browsers. These segments are compressed into WebM format and sent over a WebSocket connection to the backend. The client is event-aware, able to initiate and terminate recording sessions, display partial transcription updates as they arrive, and finally render the corrected full transcript.

In terms of user experience (UX), the client also includes visual indicators of session status, transcription activity, and error feedback mechanisms, which collectively contribute to a seamless interactive flow.

3.2.2. Backend Server (Node.js)

The server, built using Node.js [15] and Socket.IO, manages session lifecycles and orchestrates the data flow. When the server receives audio chunks from the client, it

converts them from WebM to WAV format using the ffmpeg tool [16]. This conversion is necessary since the Vosk engine operates on linear raw PCM audio data.

Audio is temporarily stored in session-specific buffers that support incremental processing. During an active session, the backend submits these buffers to Vosk for partial transcription. It compares successive outputs to isolate new tokens and sends only the delta to the client as part of a *partialTranscription* event. At the end of the session, the complete audio buffer is passed to Vosk for final transcription, ensuring that any discrepancies or missed segments during real-time updates are corrected.

3.2.3. Speech Recognition Microservice (Vosk)

The Vosk engine [17] runs as a standalone Python microservice and is invoked through an internal API. It accepts WAV audio input and returns JSON-formatted transcriptions. Vosk is chosen due to its lightweight resource profile, real-time capabilities, and multilingual support, making it suitable for deployment on low-power edge devices.

3.2.4. NLP Correction Microservice (Flask)

Post-processing is performed by a Flask-based microservice responsible for orthographic correction. It uses the *autocorrect* Python library [18] to identify and fix spelling errors in the transcription. The correction strategy is based on token-level analysis using a frequency-based model, allowing the system to prioritize common language usage patterns. While more advanced NLP techniques such as neural grammar correction exist, this lightweight method achieves an effective trade-off between performance and computational cost.

3.2.5. Communication Protocols

WebSocket, implemenmted using the Socket.IO library [19], allows data exchange between the client and server. This guarantees event-based communication for transmitting audio data and obtaining transcription results. Compared to traditional HTTP-based polling mechanisms, WebSocket offers improved efficiency, especially in real-time systems requiring frequent state updates [20].

The architecture's modularity also enables future extensibility—e.g., replacing the Vosk recognizer with an advanced deep neural model or introducing additional NLP stages for punctuation or semantic analysis—without disrupting the core interaction pattern between components.

4. Implementation Details

The proposed system employs a streaming-based, modular architecture designed to support real-time, low-latency speech-to-text (STT) workflows for edge computing environments. The system has been designed to achieve (1) responsive audio acquisition, (2) incremental transcription, and (3) accurate correction without relying on cloud services.

4.1. Audio Workflow Overview

Upon session initiation, the browser-based client uses the *MediaRecorder* API to capture audio in small chunks (typically 200–500 ms). These chunks are encoded in WebM format and transmitted via a persistent WebSocket channel to the Node.js server. Each client session is tracked by a unique token, allowing parallel and isolated processing.

The server processes incoming audio chunks by converting them from WebM to WAV using *ffmpeg* [16], a step required for compatibility with the Vosk ASR engine [17], which operates on uncompressed linear PCM data. The audio is then processed in two stages: (1) partial transcription (incrementally during the session) and (2) complete transcription (after the session ends).

Figure 2 shows the overall data pipeline from media capture to client rendering. The figure highlights the sequential flow of audio chunks through conversion, transcription, and correction stages.

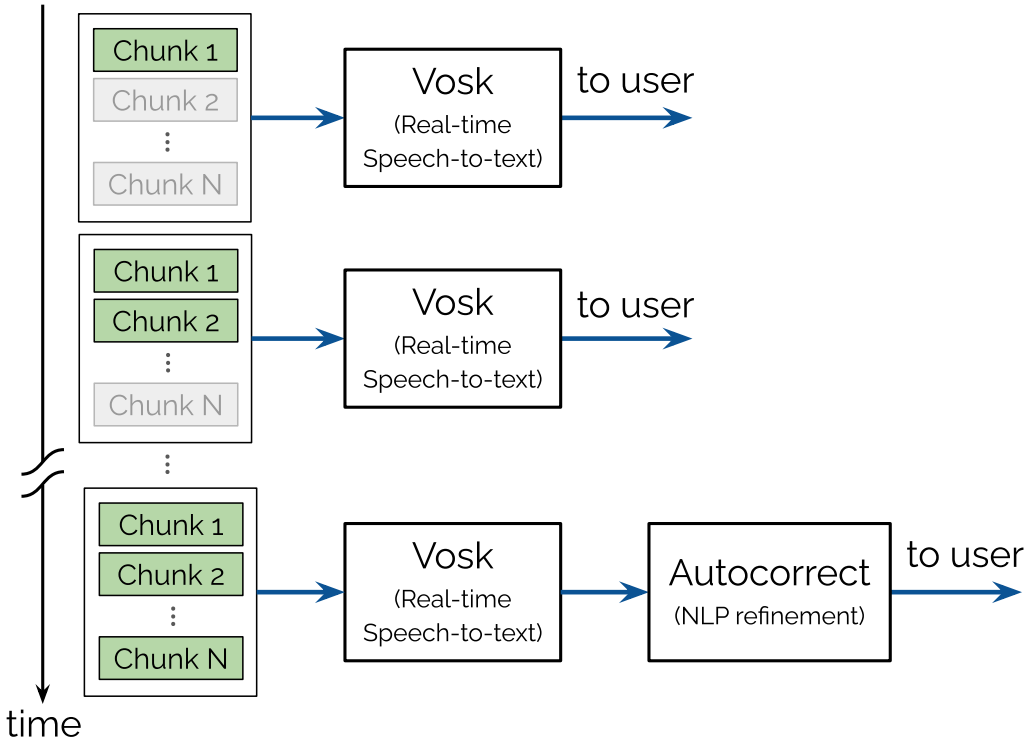

Figure 2. Progressive transcription pipeline: audio chunks are processed via Vosk for real-time speech-to-text conversion, with the final stage incorporating NLP-based orthographic correction for enhanced accuracy.

4.2. Buffer Management and Token Differentiation

A crucial aspect of the real-time flow is the differentiation of newly transcribed segments from previous results. The function `removeWordsUntilFirstOccurrence` compares the current transcription output with the previous one and strips redundant segments by identifying the last word of the previous transcription and extracting only the new part. This ensures that the user interface displays incremental updates without repeated text.

4.3. Orthographic Correction

The final transcription undergoes lightweight post-processing using a Flask-based microservice that employs the *autocorrect* Python library [18]. This module identifies and replaces likely spelling errors based on frequency and edit distance. While not as sophisticated as transformer-based grammar models, it provides effective real-time correction with negligible latency and minimal resource usage.

4.4. Communication and Protocol Design

The system uses WebSocket communication implemented via the *Socket.IO* library [19] to facilitate low-latency, event-driven message exchange between client and server. Audio data and transcription updates are exchanged asynchronously, with distinct events for partial and complete transcription states.

The modularity of the proposed architecture allows alternative STT or NLP components to be integrated without changing the communication layer, allowing its use in diverse application domains.

5. Validation and Performance Evaluation

5.1. Experimental Setup

The system was validated using both scripted and real-world audio inputs. For structured evaluation, we used samples from the Mozilla Common Voice dataset [21], which includes multiple speakers across various languages, accents, and noise conditions. The dataset provided ground truth for calculating transcription accuracy via WER and CER. In this case, the audio is fed directly to the audio processing pipeline, bypassing the WebRTC audio capturing that is instead used in the real-word audio scenario.

Additionally, the system was tested with user-recorded inputs under various conditions such as Italian-accented English, spontaneous speech, and background noise to simulate real-world performance.

5.2. Latency Analysis

Latency was measured at each stage of the STT pipeline:

- Audio Conversion (WebM to WAV): 83–107 ms;
- STT Inference (Vosk): 260–969 ms;
- Orthographic Correction: consistently under 30 ms.

All end-to-end processing cycles remained within a 1.1 s upper bound, confirming the system's suitability for real-time scenarios. (See Table 1).

Table 1. Latency metrics for a representative session.

Chunk	Conversion (ms)	STT Delay (ms)	Correction (ms)	Total (ms)
1	107	260	–	367
2	83	463	–	546
3	93	682	–	775
4	93	890	20	1003

5.3. Accuracy Metrics and Scenarios

The system's performance was evaluated using Word Error Rate (WER) and Character Error Rate (CER). Across test conditions, the system consistently achieved the following:

- WER: <5%;
- CER: 5–10%.

In the following, we provide detailed results obtained in each considered scenario:

- *Scenario A* (clean, native English): WER 2.4%, CER 6.3%;
- *Scenario B* (Italian-accented English, quiet background): WER 4.6%, CER 9.2%;
- *Scenario C* (noisy, spontaneous speech): WER 4.9%, CER 9.8%.

For each scenario, the audio samples were processed through the full system pipeline, including real-time chunk-based acquisition, conversion via ffmpeg, Vosk-based transcription, and orthographic correction. In *Scenario A*, samples were extracted from the Mozilla Common Voice dataset (native English speakers) and used in batch mode, bypassing WebRTC to isolate ASR performance. In *Scenario B*, recordings were made by Italian users speaking English in a quiet environment using standard laptop microphones via the WebRTC-enabled client interface, to simulate a typical remote collaboration setting. *Scenario C* involved live recordings of spontaneous speech with environmental noise (e.g., background conversation, typing), processed entirely through the system in real-time conditions. Each scenario was repeated multiple times (N = 10 per condition).

The system showed good results in handling neologisms and domain-specific terminology, thanks to the synergy between Vosk's ASR model and the autocorrect post-processing.

6. Discussion

The system demonstrates a viable and effective approach for real-time speech-to-text transcription leveraging edge-based infrastructure. Its architecture is specifically designed to operate independently of cloud services, relying instead on lightweight components. This deliberate design choice ensures privacy, making the system particularly well-suited for use in local or bandwidth-constrained environments. Moreover, the Node.js server's non-blocking architecture enabled multiple clients to stream audio and receive transcriptions in parallel without measurable performance degradation. This confirms its suitability for containerized deployments in edge or offline settings such as healthcare, education, and industrial environments.

One of the system's key strengths is its ability to deliver sub-second latency from speech input to transcribed text output. This level of performance is critical for interactive applications such as live captioning or real-time accessibility tools. In addition to low latency, the system maintains high transcription accuracy across a range of speaker conditions, including accented speech and informal language. Another advantage is the modular design, which allows for flexible substitution of the underlying components. Developers can, for instance, replace the Vosk STT engine or the orthographic correction module without altering the overall pipeline structure, allowing domain-specific customization.

Despite these strengths, the system does present limitations. As the length of the audio buffer increases, the inference delay grows, reflecting the batch-oriented nature of the Vosk engine. Furthermore, the current implementation does not support speaker diarization or handle overlapping speech, which restricts its use in multi-speaker scenarios such as meetings or panel discussions. The system lacks direct benchmarking against state-of-the-art alternatives like Whisper, DeepSpeech, or commercial cloud-based APIs, which would provide a clearer context for its relative performance.

Looking ahead, several improvements are planned. Future versions of the system will include comparative evaluations against established STT frameworks to quantify trade-offs in speed, accuracy, and resource use. There will also be a focus on integrating features such as punctuation, speaker identification, and context-aware error correction to increase the readability and semantic quality of the transcribed output.

7. Conclusions

This work presented the design, implementation, and validation of a modular, real-time speech-to-text (STT) system optimized for low-latency processing in constrained environments. The architecture relies on browser-based audio acquisition, local speech recognition using Vosk, and lightweight orthographic correction, all deployed without the use of cloud services.

The evaluation results confirmed that the system maintains end-to-end latencies under 1.1 s while achieving a Word Error Rate below 5% across a variety of test conditions, including accented speech and informal language. These results indicate that the system is suitable for real-time applications such as live captioning, accessibility tools, and embedded transcription interfaces.

The architecture is scalable and extensible, allowing future integration of more advanced components, such as transformer-based grammar correction or neural STT engines like Whisper. The use of open-source tools and standard web APIs ensures portability across platforms and ease of deployment.

Future work will focus on introducing punctuation and speaker diarization modules, and enhancing the user experience with advanced UI features and structured transcript formatting.

Author Contributions: Conceptualization, S.D.L., L.D.C. and S.M.; Methodology, L.D.C.; Software, S.D.L.; Validation, S.D.L.; Investigation, S.M.; Writing—original draft, S.D.L. and L.D.C.; Writing—review & editing, L.D.C. and S.M.; Supervision, S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been funded by the project "DARWIN—D2D system based on ultra-high speed (UHS) and 5g networks features for straight-to-process Analysis and user custom enRiched Web navIgation with ML and NLP algorithms" funded by "POR Puglia FESR-FSE 2014–2020 Asse III—Competitività delle Piccole e Medie imprese—Azioni 3.1, 3.5 e 3.7 Asse I—Ricerca, sviluppo tecnologico, innovazione—Azioni 1.1 e 1.3", Beneficiario Eulotech Srl, project id 3G98LE4.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Baevski, A.; Zhou, Y.; Mohamed, A.; Auli, M. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. In *Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 12449–12460.
2. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *arXiv* **2023**, arXiv:1706.03762. [CrossRef] [PubMed]
3. Khurana, D.; Koli, A.; Khatter, K.; Singh, S. Natural language processing: State of the art, current trends and challenges. *Multimed. Tools Appl.* **2022**, *82*, 3713–3744. [CrossRef] [PubMed]
4. WebRTC—Realtime Communication for the Open Web Platform. Available online: https://queue.acm.org/detail.cfm?id=3457587 (accessed on 7 August 2025).
5. De Cicco, L.; Carlucci, G.; Mascolo, S. Congestion Control for WebRTC: Standardization Status and Open Issues. *IEEE Commun. Stand. Mag.* **2017**, *1*, 22–27. [CrossRef]
6. Embedded Speech Technology. Available online: https://hdl.handle.net/20.500.12608/55116 (accessed on 7 August 2025).
7. Hasnat, A.; Mowla, J.; Khan, M. Isolated and Continuous Bangla Speech Recognition: Implementation, Performance and Application Perspective. 2007. Available online: http://hdl.handle.net/10361/331 (accessed on 7 August 2025).
8. Graves, A. Sequence Transduction with Recurrent Neural Networks. *arXiv* **2012**, arXiv:1211.3711. [CrossRef]
9. Wav2vec 2.0: Learning the Structure of Speech from Raw Audio. 2020. Available online: https://ai.meta.com/blog/wav2vec-20-learning-the-structure-of-speech-from-raw-audio/ (accessed on 7 August 2025).
10. Carlucci, G.; De Cicco, L.; Holmer, S.; Mascolo, S. Congestion Control for Web Real-Time Communication. *IEEE/ACM Trans. Netw.* **2017**, *25*, 2629–2642. [CrossRef]
11. Puspitasari, A.A.; An, T.T.; Alsharif, M.H.; Lee, B.M. Emerging Technologies for 6G Communication Networks: Machine Learning Approaches. *Sensors* **2023**, *23*, 7709. [CrossRef] [PubMed]
12. Mhedhbi, M.; Elayoubi, S.; Leconte, G. AI-based prediction for Ultra Reliable Low Latency service performance in industrial environments. In Proceedings of the 2022 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Thessaloniki, Greece, 10–12 October 2022; pp. 130–135. [CrossRef]
13. Barański, M.; Jasiński, J.; Bartolewska, J.; Kacprzak, S.; Witkowski, M.; Kowalczyk, K. Investigation of Whisper ASR Hallucinations Induced by Non-Speech Audio. In Proceedings of the ICASSP 2025—2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Hyderabad, India, 6–11 April 2025; pp. 1–5. [CrossRef]
14. Docs, M.W. MediaRecorder API. Browser API for Recording Media Streams. 2023. Available online: https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder (accessed on 7 August 2025).
15. Foundation, O. Node.js. JavaScript Runtime Built on Chrome's V8 Engine. 2023. Available online: https://nodejs.org (accessed on 7 August 2025).
16. Developers, F. FFmpeg. A Complete, Cross-Platform Solution to Record, Convert and Stream Audio and Video. 2023. Available online: https://ffmpeg.org/ (accessed on 7 August 2025).
17. Cephei, A. Vosk Speech Recognition Toolkit. Open-Source Offline Speech Recognition Toolkit. 2023. Available online: https://alphacephei.com/vosk (accessed on 7 August 2025).

18. Sedgewick, E. Autocorrect Python Library, Version 2.6.1. 2020. Available online: https://github.com/filyp/autocorrect (accessed on 7 August 2025).
19. Rauch, G.; Contributors. Socket.IO: Real-Time Bidirectional Event-Based Communication. 2023. Available online: https://socket.io (accessed on 7 August 2025).
20. Pimentel, V.; Nickerson, B.G. Communicating and Displaying Real-Time Data with WebSocket. *IEEE Internet Comput.* **2012**, *16*, 45–53. [CrossRef]
21. Mozilla. Common Voice Dataset. 2020. Available online: https://commonvoice.mozilla.org (accessed on 7 August 2025).