

A Bluetooth-based Embedded Platform for Real-time Audio Algorithm Validation

Gabriel Calazans D. Moura, Arysso Oliveira, Anderson Gabriel V. M. de Sá, Maria Augusta B. de Gois and
Tiago G. F. Barros

Abstract—This paper presents a real-time algorithm validation platform on a Raspberry Pi 5 with Bluetooth audio streaming. It supports unidirectional transport through the Advanced Audio Distribution Profile (A2DP) and bidirectional transport through the Hands-Free Profile (HFP), forming a testbed for evaluating algorithms under realistic wireless conditions. The software stack combines BlueZ over D-Bus and an ALSA audio layer. As proof of concept, a classic DSP algorithm (echo generation) and a speech translation pipeline were deployed; both achieved a real-time factor (RTF) below 1 (echo: negligible, translation: 0.97 on A2DP), confirming that the platform sustains computationally diverse workloads without violating real-time constraints.

Keywords—Real-time audio processing, Bluetooth (A2DP/HFP), Raspberry Pi, Algorithm validation platform, speech translation.

I. INTRODUCTION

Real-time audio processing is a critical component of modern communication systems, with applications ranging from speech enhancement and noise reduction to assistive audio technologies [1]. Developing and validating algorithms that operate on audio streams requires prototyping environments capable of handling bidirectional data flow under realistic operating conditions [2]. However, many existing platforms rely on specialized hardware or simulation-based approaches that do not accurately capture real-world constraints such as latency, wireless transmission variability, and limited computational resources [3].

Latency and buffering effects introduced by software layers and hardware heterogeneity can significantly degrade system performance in practice [4]. Bluetooth, in particular, introduces additional challenges including packet loss, limited bandwidth, and variable latency, complicating the evaluation of real-time systems that depend on wireless audio transport [5]. Bluetooth audio profiles serve complementary roles: the Advanced Audio Distribution Profile (A2DP) provides unidirectional streaming suited to high-fidelity playback, while the Hands-Free Profile (HFP) supports the simultaneous capture and playback required for interactive speech scenarios. A platform aimed at validating arbitrary audio algorithms over wireless links must therefore expose both.

Gabriel Calazans D. Moura, Center for Advanced Studies and Systems (CESAR), Recife-PE, Brazil, email: gcdm@cesar.org.br; Arysso Oliveira, Center for Advanced Studies and Systems (CESAR), Recife-PE, email: aso@cesar.org.br; Anderson Gabriel V. M. de Sá, CESAR School, Recife-PE, Brazil, email: agvms@cesar.school; Maria Augusta B. de Gois, CESAR School, Recife-PE, Brazil, email: mabg@cesar.school; Tiago G. F. Barros, CESAR School & CESAR Recife Center for Advanced Studies and Systems, Recife-PE, email: tiago.barros@cesar.school. This work was supported by CESAR and CESAR School.

Low-cost single-board computers have emerged as attractive alternatives for prototyping real-time algorithms [6]. Among these, the Raspberry Pi is widely adopted due to its versatility and compatibility with Linux-based audio frameworks such as Advanced Linux Sound Architecture (ALSA) [7] and PipeWire [8], which provide standardized interfaces for device control and stream management [9]. The integration of these software layers with wireless communication capabilities enables end-to-end systems that closely resemble real-world deployment conditions [10].

Several prior works have explored embedded platforms for real-time audio processing. Drakopoulos et al. [11] achieved sub-16 ms latency for DNN-based speech enhancement on a wired Raspberry Pi 3 setup using JACK. Sykulski and Jędrzejewski [12] implemented a wired, ALSA-based active noise cancelling system in C++ with sub-millisecond jitter. Gupta et al. [5] proposed a Bluetooth audio acquisition framework on an ESP32 for mobile robotics, achieving 90–95% packet delivery with sub-150 ms motor-command latency over a generic serial profile rather than a standardized audio profile. Osadchuk and Olenych [13] ran Whisper locally on a wired Raspberry Pi 4, reporting 82.3% transcription accuracy at 2.62 s per utterance for the Tiny model. A common limitation across these works is the absence of a unified platform combining bidirectional wireless audio streaming with the flexibility to deploy arbitrary algorithms, from lightweight DSP filters to demanding machine learning pipelines, under realistic operating conditions.

This paper presents an embedded real-time algorithm validation platform with wireless Bluetooth audio streaming, built on a Raspberry Pi 5. The system provides bidirectional audio transport: signals are captured via a microphone, made available for processing, and transmitted over a Bluetooth connection, while incoming streams are simultaneously received, processed, and rendered through an output device. This architecture serves as a general-purpose testbed where arbitrary algorithms can be deployed and evaluated under realistic wireless conditions. To facilitate system control, the platform incorporates Bluetooth HFP [14] functionalities for volume management and connection handling, complemented by physical push-button controls.

The main contribution of this work is demonstrating that a general-purpose embedded platform supporting both A2DP and HFP Bluetooth audio profiles can serve as an effective and flexible environment for validating real-time algorithms over wireless audio links. Experimental results confirm the feasibility of the approach in terms of latency, stability, and processing performance.

II. HARDWARE DESCRIPTION

The hardware platform is based on the Raspberry Pi 5 (quad-core ARM Cortex-A76 @ 2.4 GHz, 4 GB LPDDR4X, integrated Bluetooth 5.0), running Raspberry Pi OS (Bookworm), chosen for its computational capability and Linux compatibility. The platform provides flexible interfacing through USB and GPIO, facilitating rapid prototyping of real-time audio systems.

Audio acquisition and playback are performed through an external USB audio adapter since the Raspberry Pi 5 does not include a native analog audio interface. The adapter provides both microphone input and headphone output via standard 3.5 mm connectors and relies on USB Audio Class drivers natively supported by the Linux kernel, requiring no additional driver development.

User interaction is implemented via three physical push buttons connected to GPIO pins 17, 22, and 27, configured as digital inputs with internal pull-up resistors enabled through the *lgpio* library [15]. GPIO 17 serves as the pairing control, while GPIO 22 and GPIO 27 handle volume increase and decrease, respectively. This minimal interface enables field operation without requiring additional peripheral devices. The complete hardware configuration is illustrated in Figure 1.

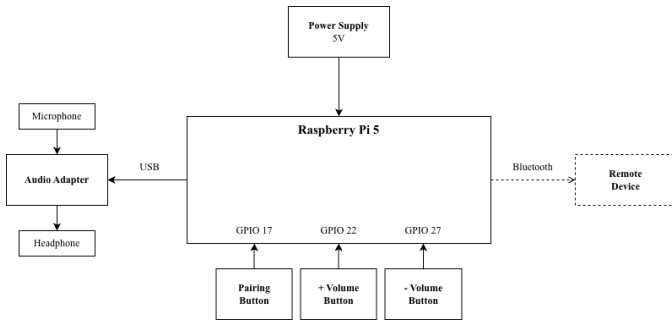


Figure 1
HARDWARE DIAGRAM

III. SOFTWARE ARCHITECTURE

The software stack is organized as four cooperating layers, illustrated in Figure 2: a Bluetooth control layer that manages pairing, profile negotiation, and connection persistence through BlueZ [16]; a BlueALSA layer that bridges Bluetooth audio profiles (A2DP and HFP) to standard ALSA PCM devices; an ALSA audio layer that exposes both the USB adapter and the Bluetooth transports through a uniform PCM interface; and an audio processing engine responsible for real-time capture, algorithm execution, and playback. Control-plane interaction with the BlueZ daemon is carried over D-Bus through the GLib/GIO asynchronous framework, while audio data flows through BlueALSA [17]. Of these layers, the Bluetooth control layer, the audio processing engine, and the GPIO button handlers were implemented by the authors; BlueZ, BlueALSA, and ALSA are off-the-shelf, as are the frameworks underlying the validation algorithms (whisper.cpp, CTranslate2, and Piper).

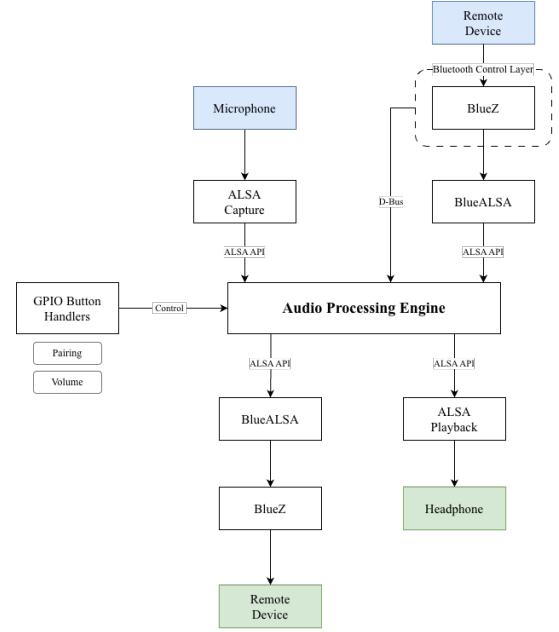


Figure 2
SOFTWARE ARCHITECTURE DIAGRAM

A. Bluetooth Control Layer

Bluetooth operations (adapter configuration, pairing, profile negotiation, transport resolution, property access) are exposed as `org.bluez` D-Bus objects, keeping each module loosely coupled and reactive to property and connection events without direct kernel or HCI access.

B. BlueALSA Bluetooth Layer

BlueALSA [17] is a userspace daemon that bridges Bluetooth audio profiles to the standard ALSA PCM interface, eliminating the need for profile-specific codec handling in the application. The platform configures both A2DP roles (a2dp-source, a2dp-sink) and both HFP roles (hfp-ag, hfp-hf), letting the Pi 5 act as either endpoint without reconfiguration. Once BlueZ negotiates a profile, BlueALSA exposes it as a standard-named virtual PCM, indistinguishable to the processing engine from the USB adapter.

C. ALSA Audio Layer

The audio data path is handled by ALSA [7], the Linux audio subsystem, which exposes every capture and playback device through a uniform PCM interface. The abstraction it provides decouples the processing code from transport-specific details since the same read/write primitives operate on wired and wireless endpoints and provides the buffering and rate control needed for real-time operation.

D. Audio Processing Engine

The audio processing engine orchestrates the capture, processing, and playback workflow on top of ALSA and acts as the real-time entry point for every algorithm hosted on the

platform. Each Bluetooth transport in use is driven by a dedicated three-thread pipeline scheduled under `SCHED_FIFO` (priority 10): a capture thread that reads PCM frames from the source ALSA device, a processing thread that hands each frame to the currently configured DSP algorithm, and a playback thread that writes the processed frames back to the destination ALSA device. The algorithm is pluggable: any frame-to-frame PCM routine integrates without modifying the surrounding infrastructure. This separation between transport, scheduling, and algorithm makes the platform a general-purpose validation testbed.

E. GPIO Button Handlers

Physical interaction relies on interrupt-driven callbacks registered via the *lgpio* library, dispatching the pairing-control and volume commands assigned to each button in section II. Switch bounce is filtered by press-duration measurement in software and by the library's hardware glitch filters.

IV. VALIDATION SCENARIOS

To evaluate the platform across the range of algorithmic loads it is intended to support, three scenarios were deployed as test cases, illustrated in Figure 3. Rather than a single worst-case benchmark, this set is designed to span the computational spectrum so the platform's behavior can be characterized at multiple operating points: a transport-only baseline, a lightweight deterministic DSP task, and a high-load pipeline. Running all three on the same hardware and audio path isolates the algorithm's contribution to resource usage and latency.

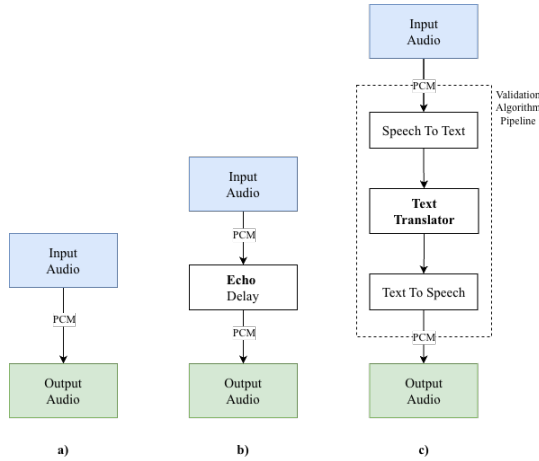


Figure 3

VALIDATION SCENARIOS: (A) BYPASS, (B) ECHO, AND (C) TRANSLATION PIPELINE

A. Bypass

The bypass scenario, illustrated in Figure 3(a), forwards each captured audio frame directly to the playback device without applying any processing. It serves as a reference baseline that measures the processing time of capture, buffering, and Bluetooth transport alone, establishing a lower bound on

end-to-end latency and resource usage against which the other scenarios can be compared.

B. Echo

The echo scenario, illustrated in Figure 3(b), applies a deterministic delay-and-mix effect: incoming samples are buffered in a circular delay queue, and each output frame is produced by summing the current input with an attenuated copy of previous frames. The algorithm represents a typical lightweight DSP workload with fixed memory footprint and bounded per-sample cost, allowing the evaluation of moderate real-time processing overhead without the variability introduced by learned models.

C. Translation Pipeline

The translation scenario, illustrated in Figure 3(c), exercises the platform under its heaviest load. Captured speech is transcribed to text, translated into a target language, and synthesized back into audio for playback. This workload exercises CPU-bound inference, memory allocation, and sustained Bluetooth transport simultaneously, establishing an upper bound on the platform's real-time processing capability.

1) *Speech-to-Text*: The Speech-to-text (STT) stage uses *whisper.cpp* [18], a C/C++ inference framework, with the pretrained *tiny.en* model from OpenAI's Whisper [19]. The captured audio is transcribed locally, enabling lightweight recognition even on the Raspberry Pi's constrained resources.

2) *Translation*: The translation stage employs *CTranslate2* [20] with the *OPUS-MT-TC-Big-EN-PT* model [21], [22], translating the transcribed English text to Portuguese. *CTranslate2* provides optimized low-latency inference suitable for near real-time execution, while *OPUS-MT-TC-Big* offers strong translation quality for this language pair.

3) *Text-to-Speech*: The Text-to-speech (TTS) stage employs *Piper* [23], a fast neural TTS engine implemented in native C++ and explicitly supported on the Raspberry Pi. The translated Portuguese text is synthesized into an audio waveform that is streamed back over the Bluetooth link for playback.

V. EXPERIMENTS

To validate the proposed platform, the complete system was assembled and the validation algorithms described in section IV were executed over Bluetooth. This section describes the experimental setup and the observed behavior during operation.

A. Experimental Setup

All experiments were conducted using the hardware described in section II and the software described in section III; a frame size of 50 ms was employed throughout the tests. Each profile operated at its default audio format: 44.1 kHz, 16-bit stereo for A2DP (SBC) and 8 kHz, 16-bit mono for HFP (CVSD).

In both scenarios, performance metrics were collected from the testbed after each run.

- 1) **A2DP test scenario.** To minimize external variability and improve reproducibility, the audio input was supplied by an Android device acting as a Bluetooth audio source. A Python benchmark running on a host PC orchestrated the phone's audio output via ADB (Android Debug Bridge) [24], instructing it to stream a fixed sequence of Harvard sentences [25] (~ 17 s total) to the platform over Bluetooth. The benchmark also synchronized start/stop with the audio processing engine on the Pi.
- 2) **HFP test scenario.** Because HFP is intended for telephony scenarios, the test required emulating a call: three machines were used. A Mac coordinated the other two machines. An Ubuntu machine injected the Harvard sentences directly into the phone's audio input path, bypassing the microphone. The phone, acting as the Audio Gateway, transmitted the audio over the call to the testbed, which acted as the Hands-Free unit and processed the received audio. The HFP narrowband baseline codec was negotiated by default, ensuring consistent codec behavior across runs.

Between runs, the Bluetooth connection was reset to ensure a clean link state.

The figures reported in Table I correspond to a single representative run per scenario and profile. Within each run, CPU usage and RAM were sampled once per second for the full duration of the run; the tabulated CPU and RAM values are the time-averages of these samples, and the standard deviation across samples is reported alongside each value. Inference time is averaged over all algorithm invocations observed during the run, and end-to-end latency over all captured frames.

B. Performance Metrics

Three timing metrics were collected per scenario, alongside the CPU usage, RAM, duration, and call-count figures reported directly in Table I. *End-to-end latency* is the elapsed time between sample capture and playback (via `CLOCK_MONOTONIC`), with jitter reported as its standard deviation. *Inference time* is the wall-clock duration spent inside the algorithm per invocation, measured per stage for the translation pipeline. The *real-time factor* (RTF) is the ratio of processing time to audio duration; values below unity confirm the platform sustains the input rate.

C. Results and Discussion

Using the setup described in section II, the bypass baseline's measured end-to-end latency was 49.25 ms for A2DP and 23.83 ms for HFP, with jitter of 15.16 ms and 24.48 ms, respectively.

Table I reports per-algorithm resource utilization and timing for the three deployed algorithms. For the bypass and echo scenarios, inference time remained near zero (~ 0 ms and 0.08 ms respectively), CPU usage stayed below 0.4% on both profiles, and memory consumption remained below 9 MB. The corresponding A2DP RTF was below 0.001 for both algorithms, confirming that lightweight DSP workloads impose effectively no real-time burden on the platform; perceptual

evaluation confirmed no artifacts were introduced by the processing path for both scenarios.

The translation pipeline is substantially more costly: inference time averaged 5081.40 ms (A2DP) and 4939.38 ms (HFP) across the cascaded STT, MT, and TTS stages; CPU usage reached 21.39% (A2DP) and 134.95% (HFP, exceeding 100% because inference threads span multiple cores); and RAM reached 494.50 MB (A2DP) and 1050.81 MB (HFP). The A2DP RTF nonetheless remained at 0.97, just below unity. HFP's on-demand SCO link introduces a variable establishment delay that prevents reliable alignment between the captured window and the processing interval, so RTF is not reported for that profile, though its resource metrics are.

The windowed nature of the speech-translation pipeline introduces an inherent delay between the user's speech and the translated output, since each window must be fully captured before transcription can begin [26]. A 3 s window was used.

Separately, STT accuracy was observed to degrade under HFP, attributable to the narrowband (8 kHz) CVSD encoding, which limits speech bandwidth compared to the 44.1 kHz A2DP input.

TABLE I
PERFORMANCE COMPARISON BETWEEN PROCESSING STAGES ON BOTH
BLUETOOTH PROFILES

Metric	Profile	Bypass	Echo	Translation
Duration (same both profiles)	—	13.73 s	18.56 s	17.46 s
Processing calls	A2DP	110	148	3
	HFP	245	247	8
Latency (avg $\pm$ jitter)	A2DP	49.25 ms $\pm$ 15.16 ms	43.63 ms $\pm$ 10.95 ms	45.43 ms $\pm$ 12.38 ms
	HFP	23.83 ms $\pm$ 24.48 ms	25.24 ms $\pm$ 25.89 ms	37.12 ms $\pm$ 40.10 ms
CPU Usage (avg $\pm$ std)	A2DP	0.09% $\pm$ 0.13%	0.09% $\pm$ 0.11%	21.39% $\pm$ 28.18%
	HFP	0.17% $\pm$ 0.17%	0.34% $\pm$ 0.24%	134.95% $\pm$ 59.56%
RAM (avg $\pm$ std)	A2DP	7.12 MB $\pm$ 1.26 MB	7.18 MB $\pm$ 1.27 MB	494.50 MB $\pm$ 65.02 MB
	HFP	8.46 MB $\pm$ 1.44 MB	8.94 MB $\pm$ 1.37 MB	1050.81 MB $\pm$ 190.69 MB
Average Inference Time	A2DP	~ 0 ms	0.08 ms	5081.40 ms
	HFP	~ 0 ms	0.08 ms	4939.38 ms
RTF	A2DP	<0.001	<0.001	0.97

Although the translation pipeline achieves a real-time factor of 0.97, this ratio does not capture the windowing delay: the end-to-end latency perceived by the user is instead dominated by the 3 s window plus the sum of the inference times of the cascaded STT, MT, and TTS stages. An RTF below unity is therefore a necessary but not sufficient condition for interactive responsiveness.

These measurements also delimit the platform's operating envelope: the translation pipeline already approaches the Raspberry Pi 5's resource ceiling, with 134.95% mean CPU load on HFP and an A2DP RTF immediately below unity. Heavier algorithms (larger models, higher sampling rates, or additional stages) would likely push the RTF past unity and cause buffer underruns, while available memory remains bounded by the chosen Pi 5 variant (4–16 GB) and sustained multi-core load risks thermal throttling under passive cooling. Characterizing this ceiling systematically is left as future work.

A direct quantitative comparison with the works discussed in section I is not fully meaningful, since each targets a different task, platform, and audio path: Drakopoulos et al. [11]

report a theoretical 8–16 ms latency for DNN inference alone on a wired soundcard; Sykulski and Jędrzejewski [12] measure a 520–550 μ s wired acoustic round trip for a single fixed algorithm; Gupta et al. [5] report 105–142 ms for motor commands, not audio, over a serial ESP32 profile; and Osadchuk and Olenych [13] report 2.62 s per utterance and 355 MB for Whisper Tiny on a wired Pi 4. The 49.25 ms (A2DP) and 23.83 ms (HFP) reported here instead aggregate capture, transport, and playback with a 0.97 RTF under active Bluetooth transport, a combination none of the above evaluates and itself a limitation of the current literature.

VI. CONCLUSIONS

This work presented an embedded validation platform based on the Raspberry Pi 5 capable of executing real-time audio algorithms through Bluetooth streaming. Experimental results demonstrated that the platform sustained both a lightweight DSP workload (echo generation) with negligible RTF and a computationally expensive speech translation pipeline with a real-time factor of 0.97, without buffer underruns on either profile. These results confirm the platform’s viability as a general-purpose validation environment for a broad spectrum of audio algorithms.

As future work, the platform may be extended to support volume-aware algorithms, such as automatic gain control, dynamic range compression, or amplitude-adaptive speech enhancement, that respond to user-initiated volume changes. Further work should characterize the trade-off between window size, translation quality, and end-to-end performance, and quantify STT accuracy degradation under HFP narrowband encoding against a wideband baseline. Audio quality was only assessed informally as artifact-free; complementing this with instrumental measures (PESQ, POLQA) and MOS-style listening tests would quantify signal fidelity. Finally, since the reported end-to-end latency is an aggregate, decomposing it into its Bluetooth transmission, buffering, scheduling, and processing contributions would reveal which layer dominates the real-time budget.

ACKNOWLEDGEMENTS

The authors acknowledge CESAR (Centro de Estudos Avançados do Recife) and CESAR School for providing the institutional support and infrastructure that enabled this research.

REFERENCES

- [1] A. V. Oppenheim and R. W. Schaffer, *Discrete-Time Signal Processing*, Pearson, 3rd edition, 2010.
- [2] Kowkutla Jeshwanth Reddy, “Noise reduction for multi-channel speech enhancement system,” *International Journal of Scientific Research in Engineering and Management*, vol. 09, no. 04, pp. 1–9, Apr 2025.
- [3] Rahul Singh Thakur, “Optimizing real-time audio filters for low-latency voice systems,” *European Modern Studies Journal*, vol. 9, no. 5, pp. 419–428, Sep 2025.
- [4] Alessio Balsini, Tommaso Cucinotta, Luca Abeni, Joel Fernandes, Phil Burk, Patrick Bellasi, and Morten Rasmussen, “Energy-efficient low-latency audio on android,” *Journal of Systems and Software*, vol. 152, pp. 182–195, 2019.
- [5] Sandeep Gupta, Udit Mamodiya, AKM Zakir Hossain, and Ahmed JA Al-Ghuri, “A framework for bluetooth-based real-time audio data acquisition in mobile robotics,” *Signals*, vol. 6, no. 3, pp. 31, 2025.
- [6] Luca Vignati, Stefano Zambon, and Luca Turchet, “A comparison of real-time linux-based architectures for embedded musical applications,” *Journal of the Audio Engineering Society*, vol. 70, no. 1/2, pp. 83–93, 2022.
- [7] Jaroslav Kysela, “Advanced linux sound architecture (alsa) the future for the linux sound!,” in *Journées d’Informatique Musicale-2000*, 2000, pp. 6–p.
- [8] Wim Taymans, “Pipewire: A low-level multimedia subsystem,” in *Proc. 18th Linux Audio Conference (LAC-20)*, 2020.
- [9] Juan Emilio Noriega-Linares and Juan Miguel Navarro Ruiz, “On the application of the raspberry pi as an advanced acoustic sensor network for noise monitoring,” *Electronics*, vol. 5, no. 4, pp. 74, 2016.
- [10] Argenox, “Bluez integration with alsa and linux audio: A deep dive,” <https://argenox.com/library/bluetooth-classic/bluez-integration-with-alsa-and-linux-audio-a-deep-dive>, 2026, Accessed: 2026-04-02.
- [11] Fotios Drakopoulos, Deepak Baby, and Sarah Verhulst, “Real-time audio processing on a Raspberry Pi using deep neural networks,” in *Proceedings of the 23rd International Congress on Acoustics (ICA 2019)*, Aachen, Germany, 2019, pp. 2827–2834.
- [12] Piotr Sykulski and Konrad Jędrzejewski, “Adaptive active noise cancelling system for headphones on Raspberry Pi platform,” in *Proceedings of the International Radar Symposium (IRS-2020)*, Warsaw, Poland, 2020, pp. 17–21, IEEE.
- [13] Oleh Osadchuk and Igor Olenych, “IoT system for real-time audio information processing,” *Advances in Cyber-Physical Systems*, vol. 10, no. 1, pp. 16–21, 2025.
- [14] Bluetooth Special Interest Group, “Hands-free profile specification,” <https://www.bluetooth.com/specifications/specs/hands-free-profile/>, 2019, Part of the Bluetooth Core Specification v5.x.
- [15] joan, “lg: Gpio library for linux single board computers,” <https://github.com/joan2937/lg>, Accessed: 2026-05-25.
- [16] Marcel Holtmann et al., “Bluez: The official linux bluetooth protocol stack,” <http://bluez.org>, 2001, Accessed: 2026.
- [17] Arkadiusz Bokowy, “BlueALSA: Bluetooth audio alsa backend,” <https://github.com/arkq/bluez-alsa>, Accessed: 2026-05-25.
- [18] Georgi Gerganov, “whisper.cpp: Port of openai’s whisper model in c/c++,” <https://github.com/ggerganov/whisper.cpp>, 2022, Accessed: 2026-05-06.
- [19] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever, “Robust speech recognition via large-scale weak supervision,” in *International conference on machine learning*. PMLR, 2023, pp. 28492–28518.
- [20] OpenNMT, “Ctranslate2: Efficient inference engine for transformer models,” <https://github.com/OpenNMT/CTranslate2>, 2022, Accessed: 2026-05-06.
- [21] Jörg Tiedemann, “The Tatoeba Translation Challenge – Realistic data sets for low resource and multilingual MT,” in *Proceedings of the Fifth Conference on Machine Translation*, Online, Nov. 2020, pp. 1174–1182, Association for Computational Linguistics.
- [22] Jörg Tiedemann and Santhosh Thottingal, “OPUS-MT — building open translation services for the world,” in *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal, 2020, pp. 479–480.
- [23] Michael Hansen and Rhasspy, “Piper: A fast, local neural text-to-speech system,” <https://github.com/rhasspy/piper>, Accessed: 2026-05-25.
- [24] Google, “Android debug bridge (adb),” <https://developer.android.com/tools/adb>, 2026, Accessed: 2026-05-28.
- [25] Institute of Electrical and Electronics Engineers, “IEEE recommended practice for speech quality measurements,” Tech. Rep. IEEE Std 297-1969, Institute of Electrical and Electronics Engineers, New York, 1969.
- [26] Sara Papi, Peter Polak, Dominik Macháček, and Ondřej Bojar, “How “real” is your real-time simultaneous speech-to-text translation system?,” *Transactions of the Association for Computational Linguistics*, vol. 13, pp. 281–313, 2025.