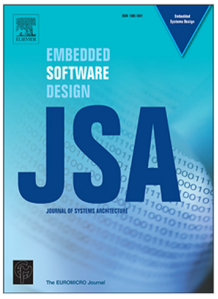
Journal Pre-proof



Cross-platform deployment and characterization of an on-device generative conversational agent

Pietro Firpo, Riccardo Berta, Francesco Bellotti, Luca Lazzaroni, Matteo Fresta, Alessandro Pighetti, Aydin Abedinia, Niloofar Yoonesi, Danilo Pietro Pau

PII: S1383-7621(26)00299-7
DOI: https://doi.org/10.1016/j.sysarc.2026.103981
Reference: SYSARC 103981

To appear in: *Journal of Systems Architecture*

Received date : 12 March 2026
Revised date : 23 July 2026
Accepted date : 22 August 2026

Please cite this article as: P. Firpo, R. Berta, F. Bellotti et al., Cross-platform deployment and characterization of an on-device generative conversational agent, *Journal of Systems Architecture* (2026), doi: https://doi.org/10.1016/j.sysarc.2026.103981.

Cross-Platform Deployment and Characterization of an On-Device Generative Conversational Agent

Pietro Firpo[a], Riccardo Berta[a], Francesco Bellotti[a], Luca Lazzaroni[a], Matteo Fresta[a], Alessandro Pighetti[a], Aydin Abedinia[a], Niloofar Yoonesi[a], Danilo Pietro Pau[b]

[a]*Department of Electrical, Electronic and Telecommunication Engineering (DITEN), University of Genoa, Genova, Italy*
[b]*System Research & Applications, STMicroelectronics, Agrate Brianza, Italy*

Abstract

Recent advances in Language Models (LMs) enable conversational systems that interpret natural-language requests, select application actions, and generate user-facing responses. However, existing studies typically either examine complete assistants with limited deployment evidence or benchmark individual components in isolation. This work advances the state of the art through a unified cross-platform characterization of a local speech-to-text-LM-text-to-speech pipeline combining Moonshine, Piper, and four quantized LMs ranging from 1.2B to 8B parameters. The pipeline is deployed on an Nvidia Jetson AGX Orin, a Raspberry Pi 5, and an STM32MP257F-EV1, and evaluated in thermostat and washing-machine use cases. Action-selection capability is assessed on 200 labelled explicit and implicit requests per use case, while deployment is characterized through metrics such as memory footprint, speech Real-Time Factor, decoding throughput, power consumption, etc. Granite 3.3 (8B) achieves the highest Macro F1-scores, whereas Qwen3.5 (4B) provides the best capability–resource trade-off. Speech processing is faster than real time on the Jetson and Raspberry Pi, while the STM32MP2 supports smaller LMs in memory but is unsuitable for responsive CPU-only conversation. LM decoding accounts for over 90% of turn duration, and memory-Roofline analysis identifies model footprint, runtime support, response length, and streaming execution as the main optimization levers.

Keywords: Edge AI, on-device inference, generative conversational agents, language models, embedded systems, speech processing, energy efficiency

1. Introduction

Transformer-based Language Models (LMs) increasingly support natural-language interaction, structured output generation, and tool-oriented reasoning across a broad range of applications [1–6]. Deploying these capabilities on edge platforms, however, is challenging because autoregressive inference imposes substantial memory, computational, and energy requirements [7]. Cloud-based execution can provide greater capacity but introduces dependence on network connectivity and raises concerns related to latency, scalability, energy consumption, and the exposure of user data to external services [8–10]. These limitations motivate the investigation of conversational generative systems whose core speech and language-processing functions execute locally.

Beyond text generation, instruction-tuned LMs can interpret indirect requests, select among available operations, produce structured action representations, and interact with external functions [11–16]. These capabilities support agentic workflows in which a model contributes to goal-directed decision processes rather than generating text alone. In the context of this work, agentic operation refers to the bounded capability to interpret a user request, select an application action, access predefined functions, maintain conversational context, propose an action plan, and obtain explicit user confirmation before execution.

Existing conversational assistants span cloud-based LM agents, locally executed systems, and conventional voice assistants based on fixed intent-classification pipelines [17–21]. State-of-the-art studies typically rely on cloud inference or report only aggregate latency, while local implementations generally provide limited information about key deployability aspects such as memory footprint, power consumption, component-level bottlenecks, and the effect of LM choice on task-level behaviour. On the other hand, studies of compact speech-to-text, text-to-speech, and language-model components typically examine individual models in isolation rather than as parts of a complete conversational workflow.

Consequently, there is only limited published evidence on the joint functional and deployment trade-offs of a fully local generative pipeline across heterogeneous edge platforms. Particularly, it is unclear how the selection of differently sized reasoning models affects the ability to infer device actions from explicit and implicit requests, whether the resulting models fit within different platform memory envelopes, and which pipeline stages dominate latency and energy consumption.

This work addresses this gap through the deployment

and characterization of a speech-enabled conversational generative pipeline applied to two household-appliance scenarios. The study investigates three questions: how the selected reasoning model affects action-selection performance and deployment cost; how memory capacity, processing resources, and runtime support affect feasibility across heterogeneous edge platforms; and which pipeline components determine conversation-turn latency and limit more responsive interaction.

The contribution of this work is a reproducible system-level deployment and design-space characterization of a fully local conversational generative pipeline. The pipeline integrates Moonshine for speech transcription, a replaceable quantized LM for action selection and response generation, and Piper for speech synthesis. Core STT, LM, and TTS inference is executed locally, while optional network access is limited to retrieving contextual application information not available on the device. More specifically, the work:

- deploys a common conversational pipeline on an Nvidia Jetson AGX Orin, a Raspberry Pi 5, and an STM32MP257F-EV1 embedded MPU platform;
- compares four quantized LMs ranging from 1.2B to 8B parameters through the same interaction workflow;
- evaluates action-selection performance on 200 labelled explicit and implicit natural-language requests for each of two appliance use cases;
- characterizes process memory footprint, STT and TTS Real-Time Factor, LM decoding throughput, power consumption, energy efficiency, and sequential conversation-turn duration; and
- analyzes the main deployment boundaries, including out-of-envelope swap behaviour, runtime–accelerator compatibility, and the memory-bandwidth limitation of autoregressive decoding.

The comparison therefore provides quantitative evidence on the capability–resource trade-offs associated with the evaluated model configurations and platforms. The resulting measurements identify LM decoding as the dominant latency contribution and quantify the gap between the present sequential implementation and an aspirational one-second interaction reference. They also show that advertised accelerator capability alone is insufficient to predict application performance, because runtime maturity, operator support, model compatibility, and workload granularity strongly affect the execution mapping.

The remainder of this paper is organized as follows. Section 2 reviews complete conversational assistants, edge-oriented speech and language models, and relevant deployment frameworks, and identifies the system-level evaluation gap addressed in this work. Section 3 presents the hardware–software architecture, interaction workflow, execution backends, and target platforms. Section 4 defines the two application use cases and their bounded functional scope. Section 5 describes the action-selection benchmark and the methodology adopted to measure memory footprint, processing performance, and power consumption. Section 6 presents and discusses the functional and deployment results, analyzes sequential conversation-turn latency, and develops a memory-Roofline bound for LM decoding. Section 7 summarizes the main findings, limitations, and directions for future research.

2. Related Work

This section reviews research on end-to-end conversational assistants alongside work on edge-based speech processing. It also examines compact language models used for action selection and tool interaction, as well as software frameworks that support on-device deployment. The comparison of end-to-end systems considers how users interact with them, how they perform reasoning, where execution takes place, which hardware they target, and what functional or deployment results they report. The discussion of individual pipeline components focuses on whether they can run locally, how large the models are, which runtimes they support, and how much latency they introduce.

2.1. End-to-End Approaches for Smart Assistants

End-to-end generative pipelines for assistants on edge devices are not entirely new. Existing literature mostly emphasizes functional aspects rather than details of edge deployability, which represents one of the main contributions of this paper. Table 1 summarizes the surveyed approaches, highlighting the adopted speech stack, processing strategies, hardware, and reported latency.

In [17], an LMBA for smart home control is presented with emphasis on NL communication and high-quality action planning. Although the platform has an NL interface, it is unclear whether it is textual or vocal, and no STT or TTS models are specified. The LMBA is tested with different reasoning cores, including proprietary and open-weight LMs. When OpenAI's GPT-3.5 and GPT-4 are used, inference is performed in the cloud through OpenAI's APIs; with Llama 2 [22], inference is run on a Hugging Face Inference Endpoint [23]. Among the surveyed LM-based assistants, [17] reports mean cloud execution latencies between 7 and 30 s, although the underlying hardware is not specified. Lazzaroni et al. [21], which relies on a conventional NLU module rather than a generative LM, reports an execution time of approximately 1 second for the complete local pipeline.

In [18], an LMBA capable of interacting with smartphone applications is described. The focus of the work is on understanding user inputs and generating action plans. GPT-4 is used as the reasoning core, and a vocal interface is mentioned, but no details are given on STT or TTS models or whether processing is local or cloud-based. LM inference relies on OpenAI's APIs, and no latency data are reported.

Table 1: Comparison of related conversational assistants. NR indicates that the corresponding information was not reported.

Reference	STT	TTS	Processing	Hardware	Latency
[17]	NR	NR	GPT-3.5, GPT-4, Llama2 (7B/13B)	Cloud (unspecified)	7-30 s
[18]	NR	NR	GPT-4	Cloud (unspecified)	NR
[19]	NR	NR	GPT-3.5T, GPT-4, Llama2 (7B/13B)	Cloud (100×Nvidia A40 GPUs)	NR
[20]	NR	NR	Llama3 (8B)	PC	NR
[21]	QuartzNet 15x5	Tacotron2 + MelGAN	NLU (intent classifier)	Nvidia Jetson AGX Xavier	1 s

The LMBA in [19] was also designed to interact with smartphone applications and was validated with payment operations on the Alipay platform. Although an NL interface is described, it is not specified whether it is vocal or textual. The agent is tested with GPT-3.5T, GPT-4, and Llama2 (7B and 13B), running on a cluster of 100 Nvidia A40 GPUs [24], without latency measurements.

A closer approach to edge deployment is proposed in [20], where an LMBA for smart home control with NL interface (unspecified if vocal or textual) is described. The focus is on user satisfaction, with no performance or latency information. The agent runs Llama3 [25] (8B) locally on a PC.

The study most similar to a complete end-to-end edge pipeline is presented in [21], where a system running on an Nvidia Jetson AGX Xavier [26] is presented as an automotive AI assistant. It uses QuartzNet 15x5 [27] for STT, Tacotron2 [28] with MelGAN [29] for TTS, and a Natural Language Understanding (NLU) module for action selection and entity recognition. The full pipeline execution time is reported at about 1 s, with expected power consumption between 10 and 30 W, but without detailed latency analysis. Differently from [21], the proposed pipeline replaces the fixed intent-classification and entity-recognition stage with a generative LM that jointly produces the device action and the natural-language response. This removes the need for a separately trained intent classifier and supports a broader range of linguistic formulations within the predefined application action space.

Existing systems show that conversational control is feasible, but they differ in their reasoning methods, execution environments, and evaluation scope. Cloud-based systems do not address embedded deployment. The closest comparable local voice assistant uses a fixed NLU pipeline and reports only aggregate execution time. As a result, the literature offers little evidence on how model size and hardware characteristics together affect functional accuracy, latency, memory use, and power consumption in a fully local generative pipeline.

2.2. Edge Speech Processing

The end-to-end assistants summarized in Table 1 provide limited information about their STT implementations. Lazzaroni et al.[21] adopts QuartzNet 15×5, a convolutional architecture designed for efficient speech recognition. More recent transformer-based systems, including Whisper and Moonshine, provide different trade-offs among transcription robustness, input handling, computational cost, and runtime availability.

More recent work focuses on transformer-based encoder-decoder models that improve both accuracy and deployability. Whisper [30], trained on 680,000 hours of multilingual audio-text pairs, has become a widely used benchmark for general-purpose STT. It combines a convolutional front-end with a transformer encoder-decoder that autoregressively generates text tokens. While robust and multilingual, its fixed 30-second input window requires zero-padding, introducing overhead and a latency floor that hampers edge deployment. Other approaches, such as [31], confirm the trade-off between accuracy and efficiency, often leaving edge requirements only partially addressed.

Moonshine[32] supports variable-duration audio without enforcing Whisper's fixed 30-s processing window and is distributed in a form compatible with local inference. The reported reduction in computational demand for short English utterances, together with ONNX Runtime support, makes it suitable for the voice-command workloads and heterogeneous platforms considered in this study. Moonshine was therefore selected as a practical STT component. Among the surveyed assistants, only one explicitly reports the TTS models used: The autoregressive acoustic generation of Tacotron 2 and the separate MelGAN vocoder introduce processing and integration costs that motivated the consideration of more recent compact alternatives. Tacotron2 relies on autoregressive spectrogram generation, resulting in high latency, while MelGAN adds further computational load. The two-stage pipeline therefore introduces a large memory footprint and delay, limiting suitability for real-time use on constrained hardware.

To overcome these issues, several lightweight TTS models optimized for edge execution have been proposed. Nix-TTS [33] distilled VITS [34] into a non-autoregressive model with 5.23M parameters and a 21.2 MB footprint, enabling real-time synthesis on a Raspberry Pi 3B. EfficientSpeech [35] reduces model size to 1.2M parameters by pairing a compact acoustic model with HiFi-GAN [36], achieving real-time performance on a Raspberry Pi 4B. FastStreamSpeech [37] integrates a streaming mechanism into a FastSpeech2 [38] + Multi-band MelGAN pipeline, enabling very low-latency synthesis on smartphone-class CPUs with high perceptual scores.

Piper [39] emerges as a more practical solution for edge deployment. Derived from VITS and optimized for the

Raspberry Pi 4, it combines high synthesis quality with efficient local inference. Rather than a single model, Piper offers a range of voices from lightweight 5M-parameter versions to higher-quality ones exceeding 30M. This flexibility enables balancing fidelity and resource usage while ensuring real-time or near real-time operation without cloud reliance. Piper was selected because it offers locally executable voices at different model sizes, ONNX compatibility, and established deployment support on ARM-based platforms. These properties simplify consistent deployment across the three evaluation platforms. The selection is essentially based on integration and deployability requirements.

2.3. Compact Language Models for On-Device Agents

As summarized in Table 1, existing assistants use reasoning components that range from fixed intent classifiers to proprietary and open-weight generative language models. Fixed natural language understanding pipelines produce predictable outputs within predefined intent and entity sets, but typically require separate task-specific classifiers, dialogue-management components, and response-generation logic. Generative language models can instead perform language interpretation, structured action generation, and user-facing response generation within a unified model. This flexibility, however, comes with substantially higher memory and computational requirements.

On-device deployment therefore requires balancing model capacity, instruction-following ability, structured-output reliability, memory consumption, and decoding speed. This study evaluates four open-weight, instruction-tuned models ranging from 1.2B to 8B parameters: LFM2.5-1.2B-Thinking [40], Qwen3.5-2B and Qwen3.5-4B [41], and Granite-3.3-8B-Instruct [42, 43]. The selected configurations and their roles in the comparison are summarized in Table 2.

All four models are evaluated using the same inference pipeline and quantization setting, enabling a direct comparison of task capability, latency, memory usage, and suitability for on-device deployment.

Table 2: Language models selected for the experimental comparison.

Model	**Params**	**Purpose in the comparison**
LFM2.5 Thinking	1.2B	Minimum resource configuration
Qwen3.5 2B	2B	Compact Qwen configuration
Qwen3.5 4B	4B	Larger Qwen configuration
Granite 3.3 Instruct	8B	Maximum capacity reference

2.4. On-Device Runtimes and Orchestration Frameworks

Frameworks such as TEN [44] provide general-purpose orchestration for real-time multimodal conversational applications, including reusable components and communication mechanisms for integrating speech and generative models. The present work addresses a different objective: controlled deployment characterization on platforms with markedly different memory and processing capacities. It therefore adopts a minimal application-specific orchestration layer, allowing the resource contribution of the STT, LM, and TTS stages to be measured separately.

Given the choice of Moonshine for STT and Piper for TTS, ONNX Runtime was selected for Moonshine and Piper because both models provide compatible representations and the runtime supports portable CPU execution across the evaluated Linux platforms [45]. Both models are distributed with ONNX support, and ONNX Runtime provides portable CPU execution and, subject to model and backend compatibility, access to platform-specific accelerators. Its kernels are optimized even for constrained devices, which is essential for edge deployment.

For LM inference, several frameworks are available, each offering different trade-offs between usability, performance, and flexibility. ONNX Runtime GenAI [46] extends ONNX Runtime with primitives for generative models, including graph optimizations and kernel fusion tailored to autoregressive decoding. Its strength lies in seamless integration with ONNX models and hardware acceleration, though maturity and support for emerging architectures remain evolving.

Another relevant solution is `llama.cpp` [47], a C++ inference framework optimized for running transformer-based models on commodity hardware. It supports multiple quantization schemes, SIMD vectorization, and hardware-specific optimizations, making it highly efficient on CPUs and suitable for constrained environments. Compared to more integrated solutions, it requires manual model management and pipeline setup.

Ollama [48] was selected as the LM-serving layer because it provides a common interface for model packaging and local execution while relying on llama.cpp for the underlying inference. The same interface can be retained across the three platforms, while CUDA acceleration was enabled only on the Jetson AGX Orin and CPU execution was used on the Raspberry Pi 5 and STM32MP2.

2.5. Synthesis and Research Gap

The reviewed literature provides three complementary but incomplete perspectives. Complete conversational assistants demonstrate functional feasibility but generally omit detailed cross-platform measurements of memory, power, and component-level latency. Research on STT, TTS, and compact LMs provides efficient individual models, but normally evaluates each component in isolation. General-purpose orchestration and inference frameworks simplify integration, but their availability does not establish whether a complete generative pipeline is practically deployable across heterogeneous edge platforms.

The gap addressed in this work is therefore the lack of a reproducible system-level characterization that considers functional action-selection performance and deployment cost across different LM scales and hardware classes.

The experimental study evaluates four reasoning models on three platforms using a common pipeline and reports action-selection performance, execution latency, memory footprint, power consumption, and power-normalized efficiency.

3. System Architecture

This section presents the hardware–software architecture adopted to deploy a fully local, speech-enabled conversational generative agent on heterogeneous edge platforms. The logical architecture comprises audio acquisition, speech-to-text transcription, LM-based action selection and response generation, access to task-specific external functions, text-to-speech synthesis, and local dialogue-state management. The same functional pipeline is deployed across the evaluated platforms, while the execution backends are selected according to the available processing resources and runtime support.

Hardware characteristics (including memory capacity, computational capability, accelerator compatibility, and power budget) are treated as first-class architectural constraints. The proposed organization therefore separates the platform-independent interaction workflow from its platform-specific execution mapping, enabling the same pipeline to be evaluated on a high-end edge-AI platform, a single-board computer, and an embedded MPU-based platform.

Fig. 1 presents a high-level block diagram of the proposed architecture, illustrating both the organization of the software pipeline and its interaction with the underlying hardware resources and runtime environments. The diagram highlights how the components for STT, LM-based reasoning, and TTS are integrated within a modular software stack, while remaining adaptable to different hardware configurations and acceleration capabilities.

Rather than focusing on isolated algorithmic optimizations, the proposed architecture addresses system-level challenges that arise when deploying conversational generative agents on constrained hardware, including resource efficiency, performance scalability, and energy awareness.

In the following subsections, we detail the requirements that guided the architectural design, the software organization of the pipeline, the operational workflow of the agent, and the hardware platforms used to experimentally characterize the impact of hardware constraints on end-to-end system behavior.

3.1. Architectural Requirements and Design Targets

The architecture was developed according to the following requirements and design targets. These targets guide the system design but are not necessarily satisfied by every hardware configuration; their achievement is assessed experimentally in Section 6.

- All STT, LM, and TTS inference shall be performed locally, without reliance on cloud inference services. Network access may be used only to retrieve application data that are unavailable locally, such as weather forecasts or dynamic tariffs.
- The interaction pipeline shall expose common interfaces among the STT, LM, TTS, and application-function components, while platform-specific runtime and accelerator backends are allowed.
- Dialogue history shall be maintained locally and bounded according to the available memory and context capacity.
- Speech processing should achieve real-time or near-real-time operation, corresponding to a Real-Time Factor (RTF) [49] close to or below one.
- The implementation should target a 30 W power envelope, compatible with embedded and edge operation.
- Potentially consequential device actions shall require explicit user confirmation before execution.
- The architecture shall support refinement of the planned operation across multiple dialogue turns without executing the action during the planning phase.

3.2. Software Architecture

Fig. 1 illustrates the system-level organization of the pipeline, including the software stack and its main runtime interfaces. The proposed pipeline adopts Moonshine as the STT module, Piper for TTS, and an LLM as the reasoning core. Specifically, `moonshine/base` with 61.5M parameters and `en_US-amy-medium` with 15.6M parameters were used. To evaluate different reasoning capacities and trade-offs, four candidate LMs were benchmarked: Granite 3.3 8B, Qwen 3.5 4B, Qwen 3.5 2B, and LFM2.5 1.2B. For LM inference, all models run in the `q4_K_M` quantized format. The inference settings and platform-specific context configurations are reported in Section 5.3. The components are coordinated by a Python orchestration layer. Audio samples acquired from the microphone or read from an input file are passed to Moonshine through ONNX Runtime. The resulting transcription, together with the relevant dialogue history and application-specific system prompt, is submitted to the LM through the Ollama interface. The LM output contains an internal representation of the planned operation and a user-facing natural-language response. Only the user-facing response is passed to Piper for speech synthesis, while the structured action information is kept by the orchestration layer for possible refinement or execution after confirmation. Application functions are exposed through use-case-specific interfaces, thereby separating conversational reasoning from device-control implementation. The software stack is designed to be modular and scalable: each component can be replaced or updated with limited effort, enabling rapid adaptation

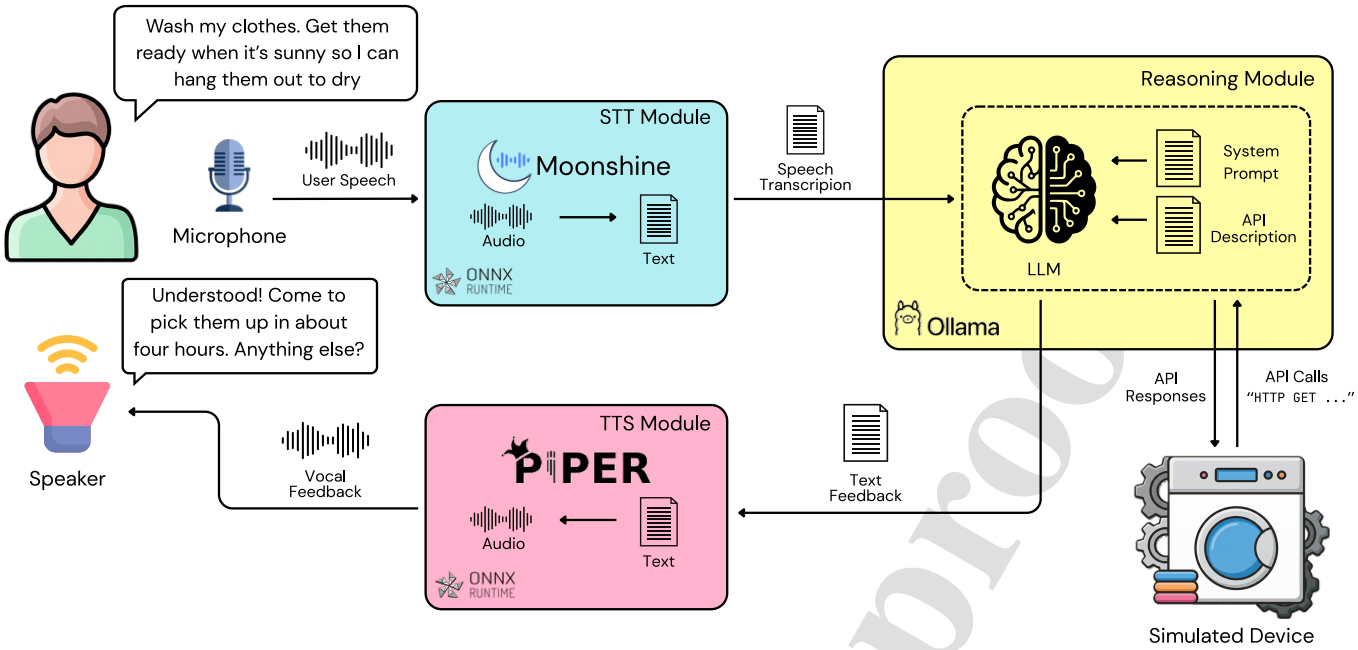


Figure 1: System architecture of the proposed edge conversational agent. Moonshine performs speech-to-text processing, a replaceable language model provides action selection and response generation, and Piper performs text-to-speech synthesis.

to new models or libraries as they evolve. This modularity also facilitates porting across heterogeneous platforms, from resource-constrained embedded devices to more capable single-board computers, thereby broadening the range of scenarios in which the pipeline can be deployed.

Fixed system-prompt templates were adopted for the two use cases and applied consistently across the four evaluated models. The prompts define the required response structure, the available application functions, and the confirmation policy. They were not individually optimized for each model, allowing the comparison to focus on the models' behavior under a common interaction protocol. The system prompt explicitly defines the conversational structure in two parts:

- the Action Plan presents a numbered list of planned operations;
- the User Response summarizes the expected outcome in simple terms.

To enforce robustness, the orchestration layer keeps the dialogue history locally and supplies the relevant bounded context to the LM at each turn. The system prompt instructs the model to preserve previously established constraints and modify only the steps affected by newly supplied information. This instruction set effectively constrains the LM to act as a stateful planner, capable of handling multi-turn dialogues where new information is incrementally integrated without disrupting prior commitments. As a conversational safeguard, the system prompt instructs the model not to initiate an action before explicit user confirmation, while still allowing the system to engage in natural, human-like conversation. In this way, the system prompt transforms the LM from a general conversational model into a task-oriented control agent, specialized for the application domains. Although the evaluation includes agentic capabilities, no dedicated exploration of task-specific prompt-engineering strategies was performed. Instead, minimal and standardized prompts were intentionally adopted to compare the models' action-selection and structured-response capabilities under a common prompting configuration.

Maintaining inference and dialogue history locally reduces the exposure of user speech and conversational data to external inference providers. However, local operation does not eliminate security risks. Stored dialogue histories, model files, and device-control interfaces require appropriate management (e.g., access control, protected storage, authenticated software updates, secure API). In addition, user utterances and information retrieved from external services must be treated as untrusted inputs because they may induce incorrect or malicious model behavior. The present prototype establishes the local-processing architecture, while comprehensive security hardening and prompt-injection resistance are left as deployment-level requirements.

3.3. Workflow

As shown in Fig. 2, the system implements an iterative perception–planning–confirmation workflow. Audio is first acquired from a microphone or input file and transcribed by the STT module. The transcription and the locally retained dialogue context are then provided to the LM together with the use-case-specific system prompt and available application functions.

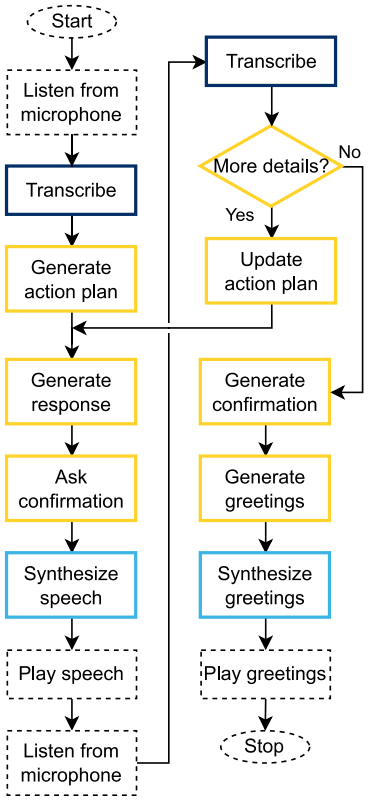


Figure 2: Workflow of the proposed edge AI agentic system. The process involves speech-to-text transcription (blue outlines), LM-based reasoning (yellow outlines), and text-to-speech synthesis (cyan outlines).

The LM generates two outputs: an internal structured action plan and a concise user-facing response. The internal plan is retained by the orchestration layer, whereas the user-facing response is synthesized by Piper and presented to the user. When the user supplies additional constraints, the new utterance is appended to the dialogue context and the LM generates an updated complete plan while preserving unaffected prior constraints.

When explicit confirmation is received, the approved application action is executed and a final confirmation response is generated. Otherwise, the planning loop continues until the user confirms, revises, or cancels the request. This organization separates conversational planning from action execution and provides an explicit human-in-the-loop control point.

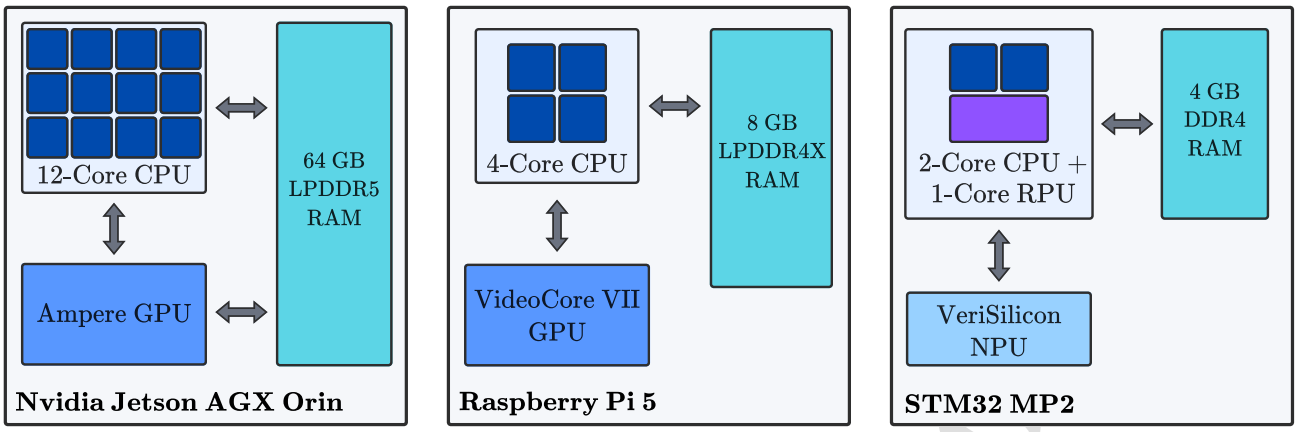
3.4. Hardware Platform

The architecture was deployed on three commercially available edge platforms selected to span substantially different memory, computational, acceleration, and power characteristics: the Nvidia Jetson AGX Orin, the Raspberry Pi 5, and the STM32MP257F-EV1 (Fig. 3). These platforms represent, respectively, a high-end GPU-accelerated edge-AI system, a CPU-oriented single-board computer, and a low-power embedded MPU platform.

The Nvidia Jetson AGX Orin Developer Kit (Fig. 3a) is a highly integrated platform combining a 2048-core Nvidia Ampere GPU with 64 Tensor Cores, a 12-core ARM Cortex-A78AE CPU, and 64 GB of LPDDR5 memory shared between CPU and GPU, delivering as much as 275 TOPS of AI performance according to the manufacturer [50]. Declared power consumption is up to 60 W, and the module also integrates dedicated hardware accelerators such as dual NVDLA engines and a vision accelerator to offload deep learning and perception tasks. Moreover, it is fully compatible with the CUDA ecosystem which can be exploited from both Ollama and ONNX Runtime. While often presented as an edge AI platform, this classification can be imprecise, since power requirements can reach several tens of watts. Nonetheless, it was selected in this work as a development platform because it remains significantly less power-hungry than a conventional consumer PC equipped with a discrete GPU, while still providing ample computational resources for real-time AI workloads.

The Raspberry Pi 5 [51] Fig. 3b has a 2.4 GHz quad-core ARM Cortex-A76 CPU paired with 8 GB of LPDDR4X-4267 RAM. While typical power consumption is much lower, the manufacturer suggests the use of a 15 to 25 W power supply, making it significantly less power-hungry than conventional computer desktop systems while still providing relatively ample computing resources. The board integrates dual HDMI 2.1 outputs supporting 4Kp60, PCIe 2.0 lanes for external devices connection, and high-speed I/O including USB 3.0, Gigabit Ethernet, and multiple camera and display connectors. This combination of relatively high performance, moderate power budget, and broad expandability positions the Raspberry Pi 5 as a practical bridge between ultra-low-power embedded boards and full-scale consumer PCs, suitable for prototyping and deployment of edge-oriented applications.

The STM32 MP257F-EV1 by STMicroelectronics [52] Fig. 3c is powered by the STM32MP257FAI3 MPU, integrating two Cortex-A35 application cores and a Cortex-M33 real-time core, paired with a memory configuration including two 16-Gbit DDR4, 512-Mbit NOR flash, and 32-Gbit eMMC. The device also integrates a Neural Processing Unit (NPU) that, according to the manufacturer, can achieve up to 1.35 TOPS, extending the platform's suitability for edge AI workloads [53]. Power consumption measurements performed under realistic workloads by the manufacturer [54] indicate levels substantially lower than the typical 15 W supply recommendation, suggesting that actual power draw in operational scenarios is moderate and well within embedded system budgets. For expandability, the board offers rich I/O support, including triple Gigabit Ethernet with TSN, dual CAN FD, USB Type-C DRP, USB Host, mini-PCIe, micro-SD, camera and display connectors, and GPIO/mikroBUS headers supported by an onboard STLINK-V3EC debugger. Software support includes OpenSTLinux, a Linux distribution developed by the MCU manufacturer, with development tools such as Yocto, Buildroot, and STM32CubeIDE available.



(a) (b) (c)

Figure 3: Hardware architectures of the three edge platforms used for deployment and evaluation.

Table 3: Processing resources and acceleration backends used across devices.

Item	Nvidia Jetson AGX Orin	Raspberry Pi 5	STM32 MP257F-EV1
CPU	12 × Cortex-A78AE	4 × Cortex-A76	2 × Cortex-A35 + Cortex-M33
Accelerator	Ampere GPU, Tensor Cores	Broadcom VideoCore VII	VeriSilicon GC8000UL
Memory	64 GB LPDDR5	8 GB LPDDR4X	4 GB DDR4
LM backend	CUDA GPU	CPU	CPU
STT/TTS backend	CPU (lower latency for short inputs)	CPU (no compatible mature backend)	CPU (models unsupported by NPU)

In contrast to the other two platforms considered in this work, which rely on active cooling with a fan, the STM32 MP257F-EV1 is entirely passively cooled. Not only does this design choice reduce mechanical complexity and potential points of failure, but also lowers acoustic noise, improves reliability in continuous-operation scenarios, and enables integration in environments where fan-based cooling would be impractical.

On all the three devices used, hardware acceleration was employed only where supported by the platform and where it reduced latency. On the Jetson AGX Orin, CUDA was used through Ollama, leveraging the platform's native GPU support for LM inference. ONNX Runtime, used for deploying the TTS and STT models selected in this work, also supports CUDA acceleration. However, for short audio files (approximately 10-20 seconds) typical of the considered scenario, the kernel-launch and data-transfer overhead outweighed the benefits of GPU offload, and CPU-only execution achieved lower end-to-end latency. On the Raspberry Pi 5, the integrated VideoCore VII GPU offers limited general-purpose GPU (GPGPU) acceleration and lacks a mature deployment path for the adopted models within the available toolchains; consequently, STT, TTS, and LM tasks were executed on the CPU. On the STM32 MP257F-EV1, Ollama does not natively support the on-chip VeriSilicon GC8000UL NPU, and LM inference was performed exclusively on the CPU. Although ONNX Runtime is available in the manufacturer's software stack, exploiting the NPU requires meeting specific constraints, which the models employed by this work do not satisfy. Therefore, on the STM32 MP2 all inference tasks were executed on the CPU [55]. On all the three platforms, CPU tasks were run using all the available cores. The main processing resources of the three platforms, together with the hardware acceleration effectively employed for LM, STT, and TTS inference, are summarized in Table 3.

To achieve a complete implementation of the pipeline with NL voice interaction, the audio input and output were managed through low-cost USB peripherals: a Seeed Technology Mini USB Microphone [56] and a Maikii Eye Bluetooth speaker [57]. The STT–LM–TTS processing chain was executed on all three platforms. Live microphone acquisition and speaker playback were integrated on the Jetson AGX Orin, whereas the Raspberry Pi 5 and STM32MP257F-EV1 were evaluated using prerecorded audio inputs and generated audio files. Consequently, all computational stages were characterized on every platform, while live end-user interaction was demonstrated on the Jetson configuration.

4. Use Cases and Functional Scope

Two household-appliance use cases were selected to instantiate the same conversational architecture under different

Table 4: Functional characteristics of the two application scenarios used for action-selection and deployment evaluation.

Use case	Control pattern	Contextual information	Main action categories	Main language challenge
Thermostat	Immediate control	Room temperature, weather forecast	Temperature increase/decrease, explicit set point, eco mode, status	Mapping implicit comfort expressions to control actions
Washing machine	Parameterized and deferred control	Electricity and water prices, PV production, weather	Power on/off, start/stop, program, water temperature, spin speed, status	Integrating user constraints with operating and scheduling choices

control and contextual requirements. The thermostat represents a relatively compact and immediate-control scenario, in which natural-language requests must be mapped to temperature and operating-mode actions. The washing-machine assistant represents a more articulated scenario involving multiple device parameters, deferred execution, and contextual information such as energy tariffs and photovoltaic availability.

The two use cases provide simple, repeatable application scenarios for evaluating the ability of the reasoning models to map explicit and implicit natural-language requests to predefined device actions, together with the deployment cost of the complete STT–LM–TTS pipeline. The available actions and external functions are bounded and specified by the application; the LM interprets the user request, selects or parameterizes an appropriate operation, and generates a user-facing explanation.

All speech processing and LM inference are performed locally. Application information is accessed through task-specific APIs, which expose either local device state and control functions or contextual data that may originate from external services. The system prompt instructs the model not to initiate any device action before explicit user confirmation.

The main functional characteristics of the two use cases, including their control patterns, contextual information, action categories, and language challenges, are summarized in Table 4.

4.1. Smart-Thermostat Use Case

The thermostat use case evaluates the interpretation of natural-language requests concerning indoor comfort and energy-aware operation. The available application functions provide access to the current room temperature, weather information, thermostat status, and temperature or operating-mode controls.

User requests may directly identify the required action, for example by specifying a desired temperature, or may express an implicit comfort need, such as feeling cold or planning to leave the house. The reasoning model must map these formulations to the appropriate action category and, where required, determine the associated parameter. The supported action set includes increasing or decreasing the current set point by 2 °C, setting an explicit temperature, enabling or disabling eco mode, and requesting the current system status.

The agent may use the available contextual information to formulate and explain a proposed operation, while the user supervises its execution through the confirmation step. Fig. 4a illustrates a multi-turn interaction in which an initial comfort request is subsequently refined by information about the user's planned absence.

4.2. Smart-Washing-Machine Use Case

The washing-machine use case extends the interaction beyond immediate device control by combining operating parameters, scheduling preferences, and contextual information. The available functions provide access to the appliance state and washing programs, together with water and electricity prices, photovoltaic production, and weather information. The supported device actions include switching the appliance on or off, starting or stopping a cycle, selecting a washing mode, setting the water temperature and spin speed, and requesting the current status. User requests may specify these operations directly or express higher-level constraints, such as completing the cycle before the user returns or avoiding prolonged storage of wet laundry in the drum. Contextual data can support the generation of a suitable schedule, for example by identifying periods associated with lower tariffs or greater photovoltaic availability. However, the quantitative evaluation in Sections 5.1 and 6.1 focuses on correct action selection rather than on proving the global optimality of the generated schedule. Fig. 4b illustrates how an initial washing request can be refined with an additional timing constraint before confirmation.

5. Experimental Methodology

This section defines the methodology adopted to evaluate the proposed system along two complementary dimensions. First, the functional capabilities of the reasoning module are assessed through its ability to select the appropriate device action from explicit and implicit natural-language requests. Second, the deployment characteristics of the complete pipeline are investigated through component-level measurements of memory footprint, execution performance, and power consumption on the three target platforms.

User: I feel cold in the house. Can you warm the room a bit?

[Internal Action Plan - not spoken aloud]

1. Access current room temperature.
2. Check weather forecast for location.
3. If the current room temperature is below the forecasted average and there are no conflicts, initiate a gradual increase in the setting by 2 °C.
4. Set a reminder to check again in 15 minutes to avoid overshooting.

Agent: Your room will start to feel warmer as I slowly increase the temperature by 2 degrees. I aim to reach a comfortable reference temperature based on today's forecast, ensuring we don't overheat. Would you like to provide any additional preferences or should I proceed with these settings?

User: I'm out in a couple of hours and I'll be back tomorrow morning. I want the home to be warm when I'm back, but I don't need to heat the house when I'm away.

Agent: Understood! I will maintain this temperature for your return, ensuring a cozy environment without unnecessary energy use. Enjoy your time out and rest assured, your home will welcome you with comfortable warmth tomorrow morning.

(a)

User: I need to wash my cotton clothes. Can you schedule the washing machine for me?

[Internal Action Plan - not spoken aloud]

1. Select the "Cotton 40 °C" program with standard spin.
2. Query the water price API to identify the cheapest tariff in the next 12 hours.
3. Query the electricity price API and PV production API to locate the best overlap between low cost and high PV availability.
4. Check the weather forecast API to ensure that drying conditions are favorable at cycle completion.
5. Schedule the wash to finish shortly before the optimal drying window, while minimizing both water and electricity costs.

Agent: Your cotton load will be washed at 40 °C and ready just before the afternoon sun, so you can hang it out to dry at the lowest cost. Would you like to add details or proceed?

User: I'm leaving home soon and will be back in the evening. I want the laundry done when I return, but I don't want it sitting wet for too long.

Agent: Understood! I will delay the start so that the cycle finishes right before you return. This way, your clothes will be freshly washed and ready without staying damp in the drum. You can enjoy your day knowing your laundry will be perfectly timed.

(b)

Figure 4: Representative multi-turn interactions produced by Granite 3.3 for the thermostat (4a) and washing-machine (4b) use cases. Internal action plans are kept by the orchestration layer and not synthesized as speech. The examples illustrate plan refinement and confirmation.

5.1. Agentic Action-Selection Evaluation

For a task-level functional assessment, we evaluated the capability of the LM reasoning module to select the device action corresponding to explicit and implicit natural-language requests. The benchmark contains 200 labelled queries for each use case. The queries include direct commands, such as "Set the thermostat to 20 °C", as well as indirect formulations requiring contextual interpretation, such as "I'm freezing" or "I'm leaving home soon."

For the thermostat use case, each query is associated with one of six target actions: increasing or decreasing the temperature by 2 °C, setting an explicit temperature, enabling or disabling eco mode, and requesting the current status. For the washing-machine use case, the supported actions comprise power on/off, cycle start/stop, washing-mode selection, water-temperature setting, spin-speed setting, and status request.

The same labelled queries and system prompt were used for all four LMs, without task-specific fine-tuning. Each model output was mapped to one of the supported action labels through rule-based output parsing. Outputs that did not identify a supported action, produced malformed calls, or selected multiple incompatible actions were assigned to the UNKNOWN category and counted as incorrect predictions.

Performance was evaluated using the Macro F1-score, which assigns equal importance to all action classes independently of their frequency. Confusion matrices were also computed to identify systematic errors between semantically related actions. The evaluation used textual queries directly, and therefore isolates LM action selection from speech-recognition errors.

The queries were generated and then manually reviewed to assign ground-truth labels, resulting in a balanced distribution across all classes.

5.2. Deployment Metrics

For each component of the pipeline and for each device considered, three quantities were evaluated: the peak process memory footprint, a performance indicator, and the average power consumption. For Piper and Moonshine, the performance indicator chosen was the RTF [49]. The RTF is defined as the ratio between the processing time required to synthesize (Piper) or transcribe (Moonshine) and the duration of the corresponding audio:

$$\mathrm{RTF} = \frac{t_{\mathrm{proc}}}{t_{\mathrm{audio}}} \tag{1}$$

An RTF = 1 indicates real-time operation, an RTF<1 denotes faster-than-real-time processing, while an RTF>1 corresponds to slower-than-real-time processing. For LM inference, the chosen performance metric is throughput, denoted by r_{LM} and measured in tokens/s.

Regarding power characterization, the average power consumption of each device was measured both in idle state and under task execution. Net power consumption, defined as the difference between the average power under load and the idle baseline, is reported for each task and device:

$$P_{\mathrm{net}} = P_{\mathrm{load}} - P_{\mathrm{idle}} \tag{2}$$

where P_{load} is the average power measured during task execution and P_{idle} is the platform idle-power baseline.

For the STT and TTS components, energy efficiency is characterized through the incremental energy required per second of processed audio:

$$E_{\mathrm{audio}} = \mathrm{RTF} \cdot P_{\mathrm{net}} \tag{3}$$

E_{audio} is expressed in (J/s). Its numerical value is identical to that of the previously reported RTF $\cdot P_{\mathrm{net}}$ metric, but its interpretation explicitly accounts for both instantaneous power consumption and processing time. Lower values indicate greater energy efficiency.

For LM inference, energy efficiency is defined as

$$\eta_{\mathrm{LM}} = \frac{r_{\mathrm{LM}}}{P_{\mathrm{net}}} \tag{4}$$

where (r_{LM}) is the decoding throughput in tokens/s. The resulting unit is tokens/J, which is numerically equivalent to tokens/s/W. Higher values of (η_{LM}) indicate greater energy efficiency.

Net power quantifies the additional power associated with task execution and therefore excludes the platform idle baseline. For the speech components, multiplying net power by RTF gives the incremental energy required per second of processed audio; lower values indicate greater energy efficiency. For LM decoding, throughput divided by net power gives the number of generated tokens per joule; higher values indicate greater energy efficiency. These metrics complement absolute RTF and throughput.

5.3. Experimental Procedure

Component-level deployment measurements were collected using minimal dedicated scripts in order to isolate the resource and performance contribution of each pipeline stage and support bottleneck attribution. These measurements exclude orchestration and inter-component communication overheads; complete conversation-turn timing is evaluated separately in Section 6.3. To ensure complete reproducibility, the experiments were executed using Ollama version 0.31.18 and ONNX Runtime version 1.20.0. For CPU inference, the number of threads was set by default to the number of available physical cores, whereas a single dispatch thread was used for GPU execution. Furthermore, the Nvidia Jetson was configured in its 60W power mode to avoid performance capping. For each component, model, and hardware-platform configuration, five preliminary warm-up runs were performed and excluded from the statistical analysis. These runs allowed runtime initialization, memory allocation, cache state, and hardware acceleration backends to stabilize. Model-loading time was explicitly excluded from the measurements after this warm-up phase. Ten subsequent measurement runs were then recorded. Unless otherwise specified, the values reported in the figures correspond to the arithmetic mean over these ten recorded runs, while variability is expressed through the corresponding standard deviation.

Peak process memory footprint was determined by periodically sampling both the resident memory and the process-attributed swap usage through the psutil library. The reported value corresponds to the maximum sum of resident and swapped memory observed during each run.

For the TTS and STT components, the Real Time Factor was calculated independently for each recorded run by dividing the processing time by the duration of the corresponding audio file. Audio duration was extracted using the `wave` Python library [58], while processing time was measured through the `time` Python library [59]. For LM inference, decoding throughput was obtained from the Ollama runtime as the number of generated tokens divided by the model-evaluation duration. Model-loading and prompt-processing times were excluded. Fig. 6 reports the mean RTF or throughput, together with the corresponding standard deviation across the ten recorded runs.

All measurements were carried out under reproducible workload conditions. For TTS, the evaluation used input texts of approximately 150 characters (roughly 30 words), producing approximately 20 seconds of synthesized speech. For STT, English language audio files of around 20 seconds in duration (16-bit WAV, 16 kHz sample rate) were transcribed. For LM inference, LFM 2.5 Thinking (1.2B), Qwen 3.5 (2B), Qwen 3.5 (4B), and Granite 3.3 (8B) were evaluated using the `q4_K_M` quantization format. The input prompts contained approximately 150 tokens and generated 50 output tokens on average. The generation parameters adopted for each evaluated local model, including a context length of 2048 and an output limit set to 200 tokens, are detailed in Table 5. The same workload configuration was retained across the repeated measurements of each model-platform combination.

Power measurements were carried out using different methodologies depending on the hardware platform, but they all describe an equivalent system boundary: for all devices, the reported values represent the whole-board power consumption, including any connected USB peripherals. On the Nvidia Jetson and the RPi 5, a purely software-based approach was feasible, as both boards integrate the

Table 5: Generation parameters for the evaluated local models. The reported values correspond to the default settings of each model, and the column labels indicate the exact Ollama repository tags.

Parameter	lfm2.5-thinking:1.2b	qwen3.5:2b	qwen3.5:4b	granite3.3:8b
temperature	0.05	1.0	1.0	0.8
top_k	50	20	20	40
top_p	0.9	0.95	0.95	0.9
presence_penalty	0.0	1.5	1.5	0.0
repeat_penalty	1.1	1.1	1.1	1.1
num_ctx	2048	2048	2048	2048

circuitry required to monitor power consumption without the need for external instruments. On the Jetson, the `jetson-stats` library [60] (specifically the `jtop` utility) was employed to monitor the *total power* reading. On the RPi, the `vcgencmd pmic_read_adc` subcommand [61] was used to query all internal power domains, returning their respective voltage and current consumption, which were then summed to obtain the total board power. In contrast, for the STM32 MP257F board, whole-board power was measured externally using a Joy-IT JT-UM120 USB multimeter [62] connected in-line with the device's power supply. Across all platforms, power readings were sampled at 1 Hz for LM inference and at 10 Hz for TTS and STT.

For each workload-platform configuration, five warm-up executions were performed before power data collection. Power consumption was then measured through ten separate recordings, each having an averaging interval of 10 seconds. The idle-measurement duration was also 10 seconds to establish the baseline. The average power observed within each 10-second window was treated as one measurement sample; the values reported in Table 6 and Fig. 7 correspond to the mean and standard deviation calculated across the ten recorded samples. For tasks completing within the observation window, the workload was executed repeatedly to maintain the device under load throughout the measurement. Longer-running tasks remained active for the entire recording window. Net power consumption was calculated as the difference between the mean power measured under load and the mean idle baseline. The normalized metrics reported in Fig. 8 were calculated as per Eq. 3 for the speech components and as per Eq. 4 for the language models. Lower values indicate better efficiency for E_{audio}, whereas higher values indicate better efficiency for η_{LM}.

Complete conversation-turn timing was assessed over ten representative interactions. A turn was defined as the sequence comprising STT processing, LM inference, and TTS synthesis. The duration of each turn was obtained by directly measuring the integrated pipeline.

6. Results and Discussion

This section jointly examines the functional and deployment results. Section 6.1 compares the action-selection capability of the four reasoning models. Section 6.2 analyzes memory footprint, processing performance, power consumption, and energy efficiency across the three platforms. Finally, Section 6.3 evaluates sequential conversation-turn latency and uses a memory-Roofline model to identify the principal bottlenecks and quantify first-order performance bounds.

Table 6: Idle power consumption for each edge platform.

	Nvidia Jetson Orin AGX	Raspberry Pi 5	STM32 MP257F-EV1
Value	8.138 W	2.136 W	0.573 W
Std. dev.	0.55 W	0.18 W	0.035 W

6.1. Agentic Action-Selection Result

Fig. 9 reports the Macro F1-scores obtained by the four reasoning models on the two action-selection benchmarks. For the thermostat use case, Granite 3.3 (8B) and Qwen 3.5 (4B) achieve Macro F1-scores of 0.993 and 0.961, respectively. Comparable results are obtained for the washing-machine use case, with scores of 0.990 and 0.976. The smaller configurations show lower performance: Qwen 3.5 (2B) obtains approximately 0.86 on both tasks, while LFM 2.5 Thinking (1.2B) obtains 0.824 and 0.791 for the thermostat and washing-machine tasks, respectively.

These results reveal a marked performance transition between the tested 2B and 4B configurations, while the 4B configuration provides action-selection performance close to that of the 8B model.

The confusion matrices in Fig. 10 and Fig. 11 show that LFM 2.5 Thinking and Qwen 3.5 (2B) more frequently confuse semantically related actions or produce UNKNOWN outputs. Conversely, Qwen 3.5 (4B) and Granite 3.3 (8B) produce almost entirely diagonal matrices and identify the target action consistently across the considered queries. Inspection of the errors indicates that many of them concern implicit or ambiguous formulations rather than direct commands.

This evaluation quantifies the models' ability to map natural-language requests to device actions, which is a necessary element of the proposed agentic workflow.

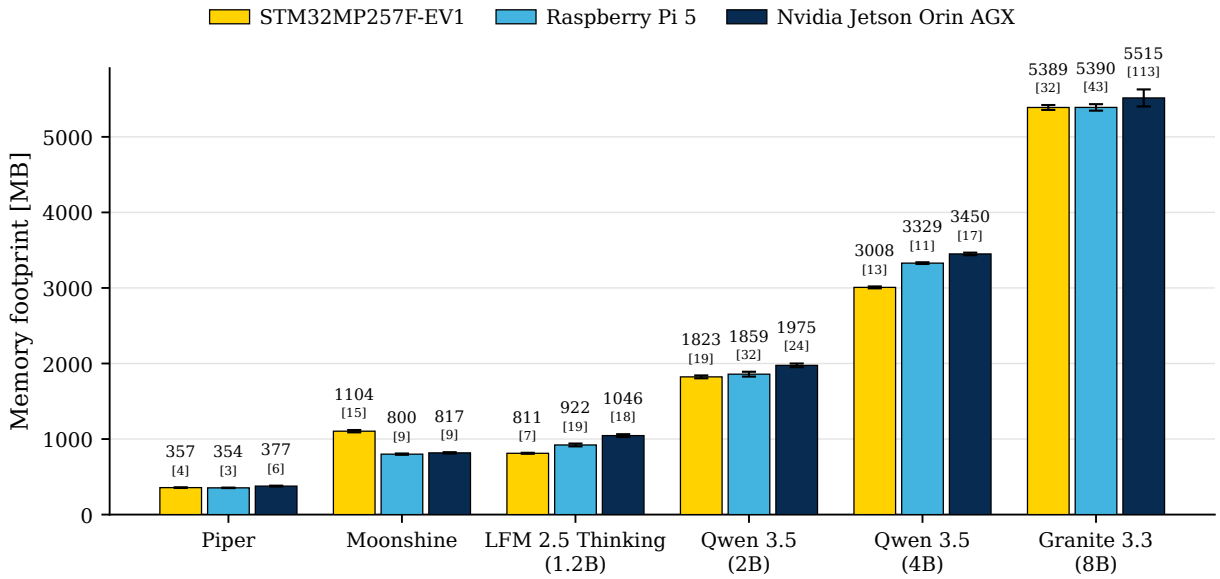


Figure 5: Mean peak process memory footprint, including resident and swapped memory where applicable. Bracketed values and error bars represent one standard deviation.

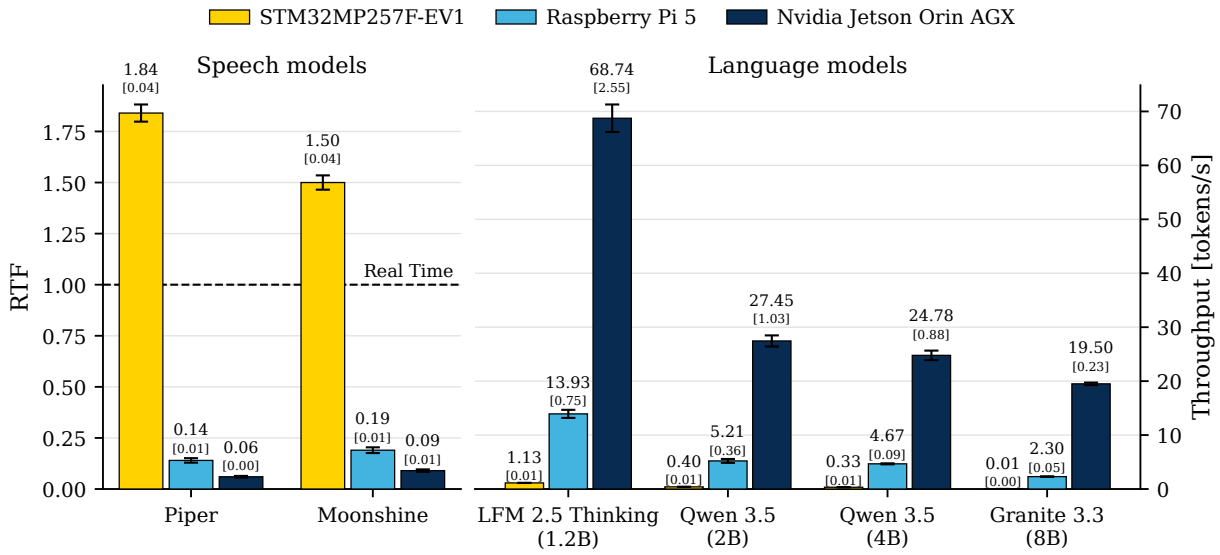


Figure 6: Mean and std performance metrics for each pipeline task and edge platform, along with the Real Time operation threshold for Piper and Moonshine. For RTF lower is better, for throughput higher is better. Bracketed values and error bars represent one standard deviation.

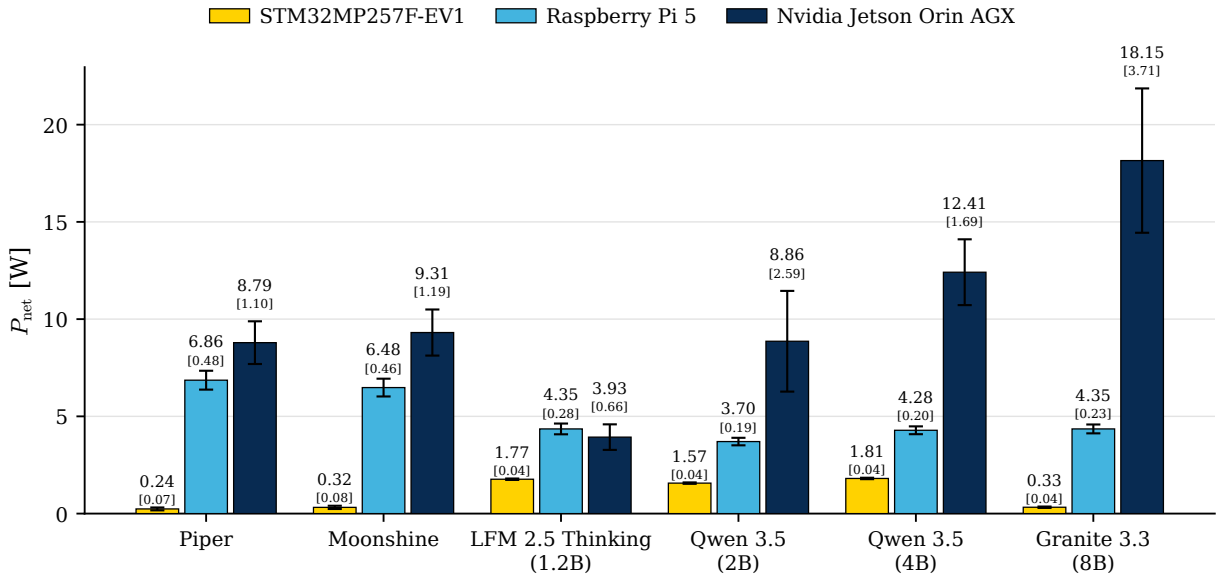


Figure 7: Mean and std net power consumption for each task and edge platform. Bracketed values and error bars represent one standard deviation.

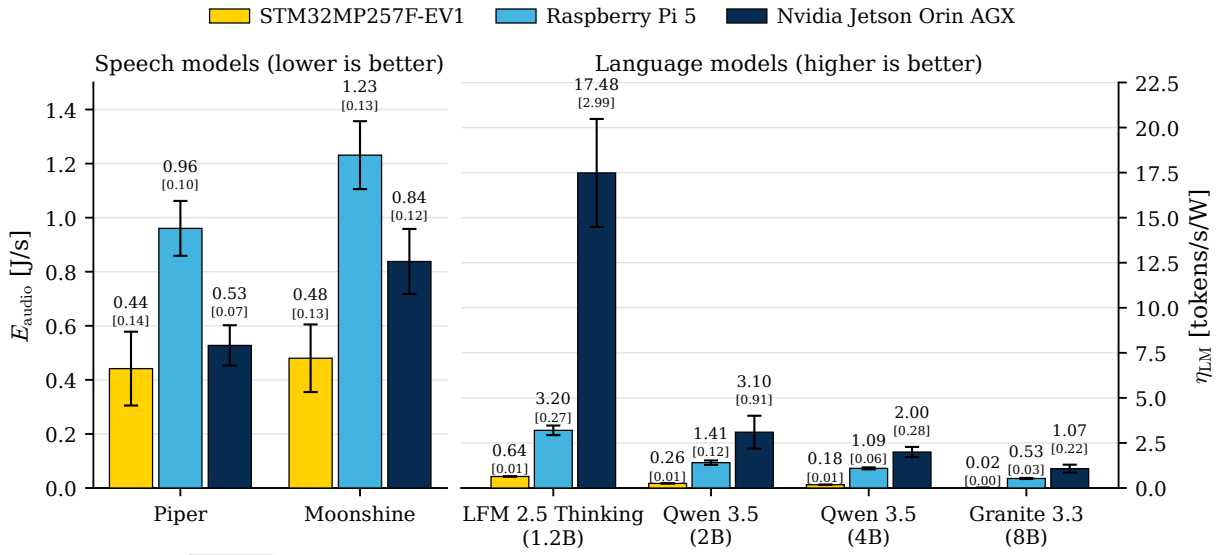


Figure 8: Mean and std performance metrics normalized with net power consumption for each task and platform. Bracketed values and error bars represent standard deviation.

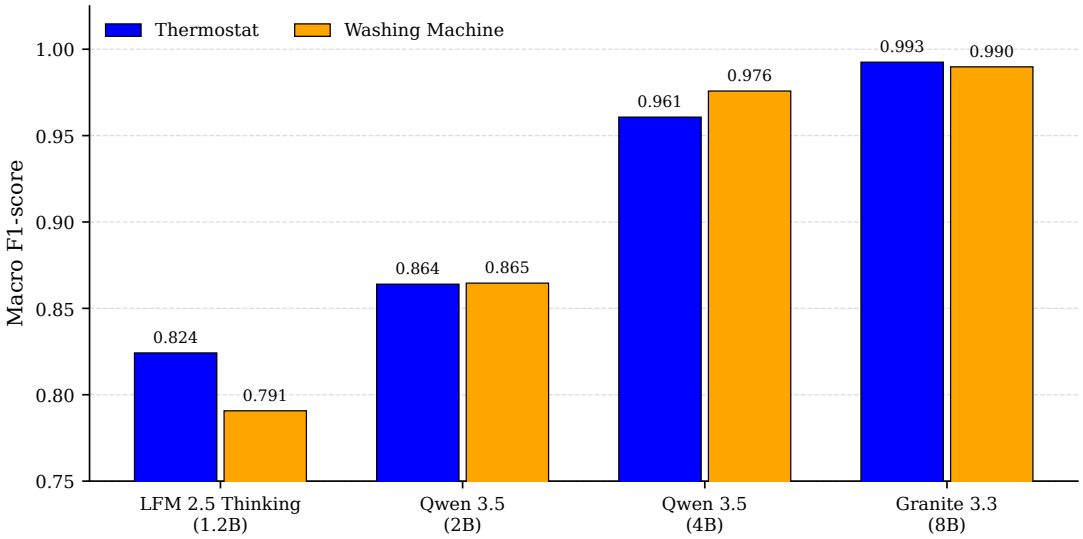


Figure 9: Macro F1-score performance across different reasoning models for the tested use cases.

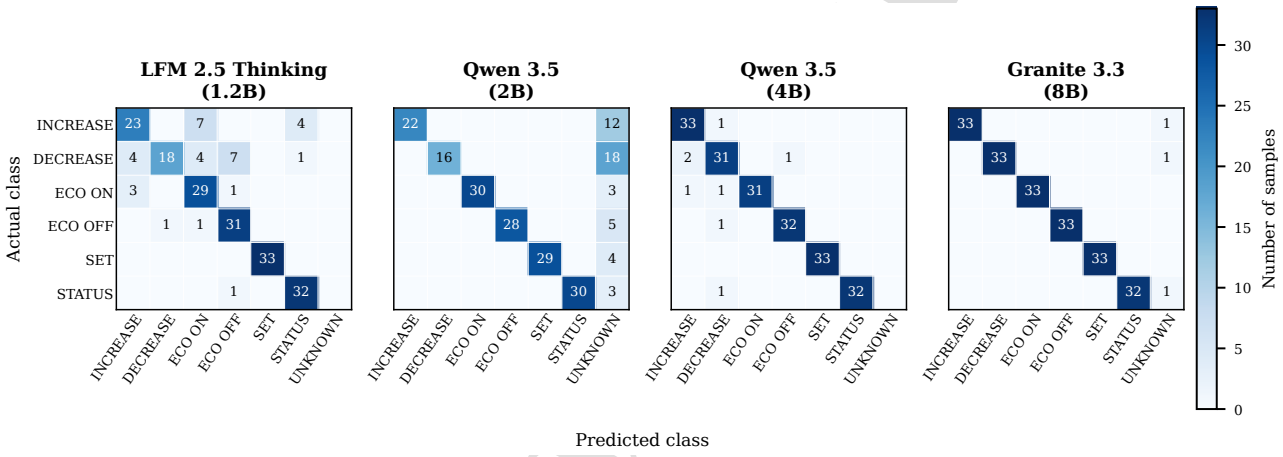


Figure 10: Confusion matrices for the thermostat use case across the four evaluated language models.

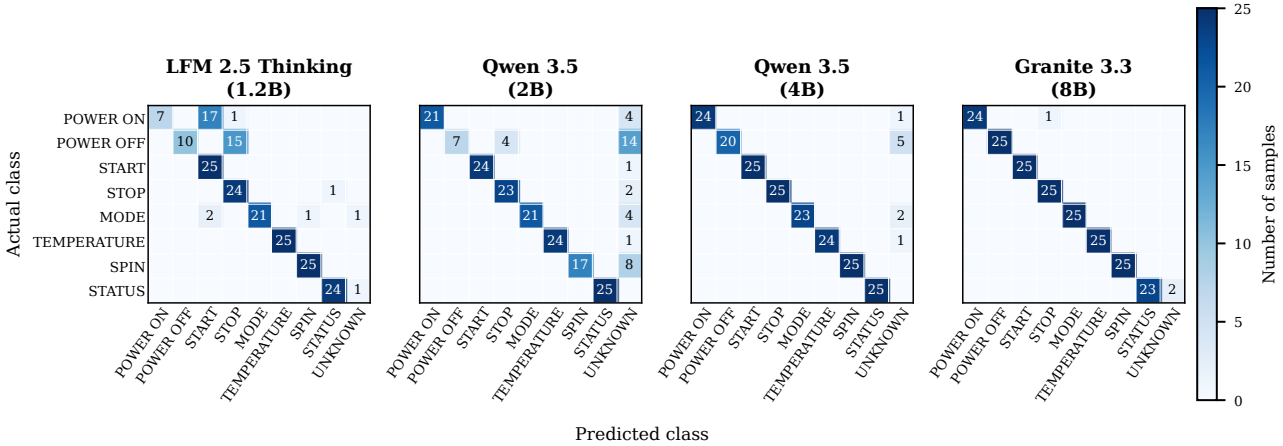


Figure 11: Confusion matrices for the washing machine use case across the four evaluated language models.

6.2. System Performance Assessment

Regarding memory usage, reported in Fig. 5, Piper and Moonshine exhibit relatively stable memory footprints across the three platforms, requiring approximately 354–377 MB and 800–1104 MB, respectively. For LM inference, memory usage instead increases consistently with model size. LFM 2.5 Thinking requires approximately 0.81–1.05 GB, Qwen 3.5 (2B) approximately 1.82–1.98 GB, Qwen 3.5 (4B) approximately 3.01–3.45 GB, and Granite 3.3 approximately 5.39–5.52 GB. Differences among hardware platforms are comparatively small with respect to those introduced by model scale. The Granite 3.3 memory footprint exceeds the 4 GB of physical RAM available on the STM32 MP2, making the use of swap memory unavoidable. The Qwen 3.5 (4B) process meets the 4 GB physical-memory boundary, but leaves only limited headroom for the operating system, runtime services, and the other pipeline modules. Its feasibility for the complete pipeline therefore depends on the adopted model-residency and sequential-execution strategy. Specifically, the LM must be kept permanently in memory, as loading its weights on-the-fly would introduce unacceptable latency overheads. Conversely, the STT and TTS models can be loaded into memory only when required and immediately unloaded after their execution. Granite 3.3, instead, exceeds physical memory even as an isolated LM process and consequently represents an explicitly out-of-envelope, swap-affected configuration.

Computation performance is reported in Fig. 6. For both Piper and Moonshine, the Jetson Orin and Raspberry Pi 5 achieve RTF values below one, thereby supporting faster-than-real-time execution. The STM32 MP257F-EV1 instead reports RTF values of 1.84 for Piper and 1.50 for Moonshine and therefore does not meet the real-time requirement for either speech component. Among the two platforms supporting real-time execution, the Jetson provides the lowest RTF values, reaching 0.06 for Piper and 0.09 for Moonshine. For LM inference, the Jetson Orin achieves the highest throughput for all four models, reaching 68.74 token/s for LFM 2.5 Thinking, 27.45 token/s for Qwen 3.5 (2B), 24.78 token/s for Qwen 3.5 (4B), and 19.50 token/s for Granite 3.3. The Raspberry Pi 5 provides intermediate performance, whereas the STM32 MP2 is substantially slower. On all platforms, throughput generally decreases as model size increases. The particularly severe degradation observed for Granite on the STM32 MP2, where throughput falls to approximately 0.005 token/s, is caused by the model footprint exceeding the available physical RAM and the resulting intensive use of swap memory.

Net power consumption is reported in Fig. 7. For the speech components, the STM32 MP257F-EV1 requires only 0.24–0.32 W, compared with 6.48–6.86 W on the Raspberry Pi 5 and 8.79–9.31 W on the Jetson Orin. A different trend emerges for LM inference. On the Jetson, net power increases substantially with model size, from 3.93 W for LFM 2.5 Thinking to 18.15 W for Granite 3.3. The Raspberry Pi remains within a narrower interval of approximately 3.70–4.35 W, while the STM32 MP2 requires approximately 1.57–1.81 W for LFM and the two Qwen models. Granite 3.3 represents an exception on the STM32 MP2, with a measured net power of only 0.33 W. This value should not be interpreted as evidence of superior efficiency. Because the model relies extensively on swap memory, the CPU spends a considerable fraction of the execution time waiting for data transfers and remains underutilized. The low instantaneous power is therefore accompanied by an extremely long execution time.

Performance normalized by net power consumption is reported in Fig. 8. For the speech components, the STM32 MP257F-EV1 achieves the lowest RTF×net-power values, with 0.44 J/s for Piper and 0.48 J/s for Moonshine. The Jetson Orin follows with 0.53 and 0.84 J/s, while the Raspberry Pi obtains 0.96 and 1.23 J/s. These results show that the STM32 platform provides favorable energy-normalized performance for lightweight speech workloads, although its absolute RTF remains above the real-time threshold. For LM inference, the Jetson Orin achieves the highest throughput per unit of net power for every evaluated model. It reaches 17.48 token/s/W for LFM 2.5 Thinking, 3.10 token/s/W for Qwen 3.5 (2B), 2.00 token/s/W for Qwen 3.5 (4B), and 1.07 token/s/W for Granite 3.3. The Raspberry Pi 5 provides intermediate values, whereas the STM32 MP2 obtains the lowest throughput-per-watt results. The smaller LFM model achieves the highest throughput-per-watt on all three platforms, but this deployment advantage must be considered together with its lower agentic performance reported in Fig. 9.

An important system-level result concerns the gap between nominal accelerator availability and effective model deployability. The Jetson GPU could be exploited for LM inference through the mature CUDA backend, but CPU execution was faster for the relatively short STT and TTS workloads because GPU launch and transfer overheads outweighed the computational gain. On the Raspberry Pi 5, the integrated GPU did not provide a mature general-purpose deployment path for the selected models. Similarly, the STM32MP2 NPU could not execute the adopted models because of runtime and operator-compatibility constraints.

Considering action-selection and deployment results jointly reveals a clear capability–resource trade-off. Granite 3.3 provides the highest Macro F1-score, but also requires the largest memory footprint and produces the lowest throughput among the in-memory configurations. LFM 2.5 Thinking offers the highest throughput and throughput per joule, but its action-selection performance is substantially lower. Qwen 3.5 (4B) achieves scores close to Granite 3.3 while reducing the LM memory footprint by approximately 2 GB and improving decoding throughput on every platform. It therefore represents the most balanced configuration among those evaluated. Nevertheless, its STM32MP2 execution time is incompatible with responsive conversation, and its memory footprint leaves lit-

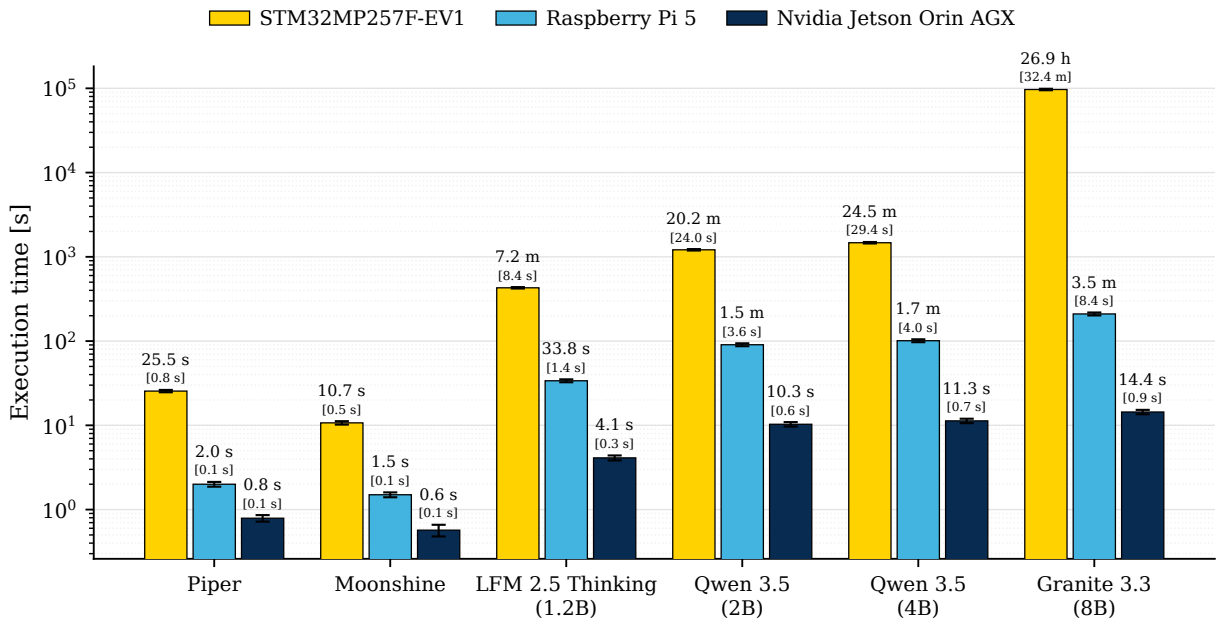


Figure 12: Mean execution time of each pipeline component and reasoning model on the three evaluated platforms. Values in brackets denote one standard deviation.

tle room for concurrent residency of the complete pipeline.

Direct end-to-end comparison with previous assistants is limited by substantial differences in models, hardware, processing boundaries, and reported metrics. Most related systems rely on cloud inference or do not separately report memory, power, and component-level latency. Subsystem results from other studies can provide useful contextual references, but they do not constitute controlled baselines for the complete pipeline evaluated here as they employ different models, quantization schemes, runtimes, and workload definitions.

6.3. Bottleneck Analysis for Real-Time Interaction

As a final step, we examine the responsiveness of the proposed system using 1 s as an aspirational interaction reference. Although previous human–robot interaction studies have associated longer response delays with reduced interaction quality [63], the considered appliance-control scenarios may tolerate multi-second responses without substantially compromising usability. For LMBA, a conversation turn consists of an STT operation, an LM processing cycle, and a TTS operation, whose contributions are reported in Fig. 12 and summarized in Table 7.

The values were averaged over ten conversations using Qwen 3.5 (4B), which provides the highest action-selection performance among the evaluated models without LM-level swap on all three platforms. Across the devices, LM inference accounts for more than 90% of the complete turn duration. The overall system responsiveness therefore depends primarily on the efficiency with which the LM is executed.

Table 7: Mean sequential conversation-turn duration and relative component contributions when using Qwen 3.5 (4B) as the reasoning module.

Device	Total turn duration (mean ± SD)	TTS	STT	LM Reasoning
Nvidia Jetson Orin AGX	12.7 ± 1.5 s	4.7%	3.6%	91.7%
Raspberry Pi 5	105.2 ± 12.6 s	1.9%	1.4%	96.7%
STM32 MP257F-EV1	25.1 ± 3.0 min	1.7%	0.7%	97.6%

A direct quantitative comparison with previous end-to-end assistants remains difficult. Among the few studies reporting latency, [17] relies on cloud inference, while [21] reports local measurements without separating the contributions of speech processing and reasoning. Instead of transferring speedup factors measured on heterogeneous models and hardware baselines, we derive a first-order performance bound directly from the characteristics of the model deployed in this work.

Autoregressive decoding at batch size one is generally dominated by data movement, since the model weights must be accessed repeatedly during token generation [64]. Under this assumption, the ideal decoding throughput of a model with weight footprint W running on a platform with peak memory bandwidth B can be approximated through a memory Roofline [65] as

$$R_{\text{roof}} = \frac{B}{W}. \tag{5}$$

The bandwidth values used in the analysis (Table 8)

correspond to the peak theoretical bandwidth of the main memory subsystem. When the aggregate bandwidth was directly reported by the manufacturer, the specified value was used. This applies to the Nvidia Jetson AGX Orin, Nvidia Jetson Thor, and AMD Alveo U55C, for which the reported bandwidths are 204.8 GB/s, 273 GB/s, and 460 GB/s, respectively [50, 66, 67]. For the Raspberry Pi 5 and STM32 MP257F-EV1, the bandwidth was derived from the memory transfer rate and the aggregate bus width as

$$B = f_{\mathrm{mem}} \frac{w_{\mathrm{bus}}}{8}, \tag{6}$$

where f_{mem} is the effective transfer rate in transfers per second and w_{bus} is the aggregate memory-bus width in bits. Accordingly, the Raspberry Pi 5 bandwidth was computed from its LPDDR4X-4267 memory and 32-bit interface as $4267 \times 32/8 = 17.07$ GB/s. For the STM32 MP257F-EV1, the two DDR4-2400 memory devices form an aggregate 32-bit interface, resulting in $2400 \times 32/8 = 9.60$ GB/s. Decimal units are used throughout the analysis, consistently with the bandwidth values reported in the hardware specifications.

For the Qwen3.5 4B q4_K_M model used in the experiments, the exact model footprint is $W = 3{,}389{,}990{,}592$ bytes. The evaluated conversations generated an average of $N_{\mathrm{out}} = 225$ tokens. The corresponding minimum decoding time is therefore

$$T_{\mathrm{dec,roof}} = \frac{N_{\mathrm{out}}}{R_{\mathrm{roof}}} = \frac{N_{\mathrm{out}} W}{B}. \tag{7}$$

For the experimentally evaluated platforms, the fraction of the Roofline reached by the measured implementation is computed as

$$\eta = \frac{R_{\mathrm{meas}}}{R_{\mathrm{roof}}}. \tag{8}$$

The measured decoding throughput and the corresponding memory-Roofline upper bounds are reported in Table 8 and visualized in Fig. 13. The Roofline values represent ideal upper bounds: they assume that one complete model-weight footprint is transferred for each decoded token and the entire nominal memory bandwidth is available for model-weight transfer. This does not account for runtime overhead due to KV-cache traffic, dequantization operations, memory contention, etc.

For the Raspberry Pi 5, the measured throughput corresponds to approximately 93% of the memory Roofline. Within the assumptions of the model, this indicates that decoding is predominantly constrained by external-memory traffic and that limited benefit can be expected from compute-only optimization. The Jetson AGX Orin reaches approximately 41% of the theoretical ceiling. This larger gap indicates that further gains may be obtained by improving kernel implementation, quantized-weight access, and runtime utilization, although the ideal value remains unattainable in practice. The STM32 MP257F-EV1

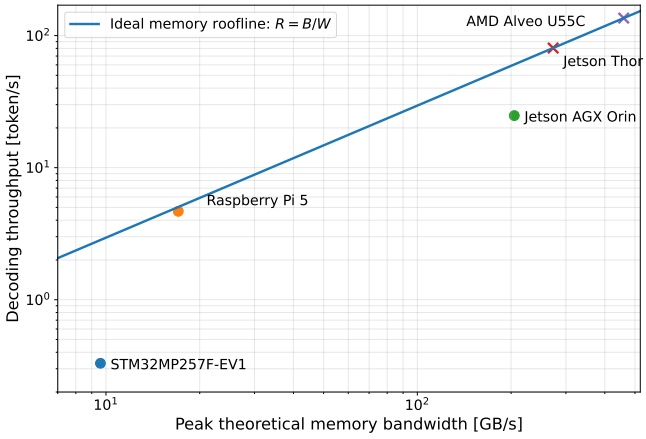


Figure 13: Memory-Roofline analysis for Qwen3.5 4B decoding.

reaches approximately 12% of its Roofline. This indicates that the Cortex-A35 implementation is unable to fully exploit the nominal DDR bandwidth, because of limited processing capability, dequantization cost, non-ideal access patterns, and runtime overheads.

Hardware evolution provides higher memory bandwidth, but the Roofline analysis shows that this improvement alone is insufficient to close the complete performance gap. Nvidia Jetson Thor provides a peak memory bandwidth of 273 GB/s [66], corresponding to an ideal decoding throughput of 80.53 token/s and a minimum decoding time of 2.79 s. For comparison, if Thor preserved the 41% Roofline utilization measured on the Jetson AGX Orin, the expected throughput would be approximately 33.03 token/s, corresponding to 6.81 s for the average output. This estimate is more conservative than directly applying the vendor-reported overall AI-performance improvement, because the latter includes computational advances that do not necessarily translate proportionally to memory-bound decoding. Applying the AGX Orin utilization ratio to Thor provides an illustrative conservative scenario, because architecture, kernels, and runtime efficiency may differ.

To evaluate the potential of a high-bandwidth FPGA implementation, the AMD Alveo U55C is considered as an architectural reference. The board provides 16 GB of HBM2 and a peak bandwidth of 460 GB/s [67], sufficient to store the complete quantized model and its working data. Its ideal memory Roofline corresponds to 135.69 token/s, or 1.66 s for the average output. The U55C is a datacenter accelerator rather than a low-power consumer platform and is therefore not proposed as a direct deployment target. It is included to quantify the benefit obtainable from an FPGA architecture equipped with a substantially higher-bandwidth memory subsystem. Actual performance would additionally depend on the hardware mapping of quantized operators, on-chip buffering, and host-device communication.

Under the strict assumption that the complete 225-

Table 8: Measured decoding performance and memory-Roofline upper bounds for Qwen3.5 4B. Decoding times refer to the average output length of 225 tokens. Values for Jetson Thor and Alveo U55C are theoretical ceilings derived from their specified peak memory bandwidth and have not been experimentally validated in this work.

Platform	B [GB/s]	R_{meas} [token/s]	R_{roof} [token/s]	η [%]	$T_{\text{dec,meas}}$	$T_{\text{dec,roof}}$
Nvidia Jetson AGX Orin	204.80	24.78	60.41	41.0	9.08 s	3.72 s
Raspberry Pi 5	17.07	4.67	5.04	92.7	48.18 s	44.68 s
STM32 MP257F-EV1	9.60	0.33	2.83	11.7	11.36 min	79.45 s
Nvidia Jetson Thor	273.00	–	80.53	–	–	2.79 s
AMD Alveo U55C	460.00	–	135.69	–	–	1.66 s

token LM output must be decoded within 1 s, and ignoring all non-decoding overheads, the corresponding ideal effective bandwidth requirement is

$$B_{1\text{s}} = N_{\text{out}} W = 762.75 \text{ GB/s}. \tag{9}$$

This value is a lower bound for decoding alone; a complete one-second conversation turn would leave less than one second for decoding after accounting for STT, prefill, orchestration, and initial TTS processing. The peak bandwidth of both of the prospective platforms considered is exceeded. The result indicates that replacing the current hardware with a moderately faster platform, while leaving the model, output length, and sequential pipeline unchanged, cannot by itself achieve the 1 s full-output target. However, in a streaming implementation, perceived latency is determined primarily by time to first token, time to first audio, and the ability to sustain generation at or above the speech-consumption rate. Consequently, shorter responses and overlapped LM–TTS execution may provide substantial interaction benefits even when full-response generation takes more than one second.

Further progress requires the joint optimization of multiple aspects. More compact or domain-specialized models reduce W and consequently improve the memory-Roofline throughput, while shorter structured outputs reduce N_{out}. Streaming execution may also reduce perceived latency by starting speech synthesis before completion of the entire LM response. Dedicated architectures such as Compute-in-Memory can further reduce repeated weight movement [68, 69]; however, their operation violates the external-memory traffic assumption of Eq. 5, and a device-specific latency projection cannot be derived using the present model.

The presented Roofline values bound single-sample sequential decoding of Qwen 3.5 (4B) under an external-memory traffic model and show that memory bandwidth, runtime efficiency, response length, and pipeline organization must be addressed jointly. Hardware acceleration remains a fundamental direction, but it must be combined with model specialization and streaming interaction to approach human-compatible conversational responsiveness on resource-constrained edge platforms.

7. Conclusions and Future Work

This work presented the deployment and systematic characterization of a fully local, voice-enabled conversational generative pipeline integrating speech-to-text transcription, LM-based action selection and response generation, application-function access, and text-to-speech synthesis. The same functional pipeline was evaluated on three heterogeneous edge platforms using four quantized language models ranging from 1.2B to 8B parameters. The assessment concerned action-selection performance and deployment-oriented metrics.

The results demonstrate a clear trade-off between functional capability and deployment cost. Within the evaluated model set, Granite 3.3 (8B) achieved the highest action-selection scores but required the largest memory footprint and the longest execution time. LFM2.5 Thinking (1.2B) provided the highest throughput and throughput per joule, but showed lower action-selection performance, particularly for implicit and ambiguous requests. The most balanced configuration, among those evaluated, was given by Qwen 3.5 (4B), achieving performance close to Granite 3.3, but with a substantial reduction in memory requirements and increase in decoding throughput.

The experiments also identify important platform-dependent feasibility boundaries. The speech components operated faster than real time on the Jetson AGX Orin and Raspberry Pi 5, whereas the STM32MP257F-EV1 could not keep real time under the adopted CPU-only implementation. Lightweight LMs could be executed in physical memory on the STM32MP2, but did not provide conversationally responsive performance. Granite 3.3 exceeded the platform's available physical memory and consequently its performance is strongly affected by swaps. Moreover, although Qwen 3.5 (4B) fit when profiled independently, its limited memory headroom must also accommodate the operating system, runtime services, and speech-processing components.

A further systems-level finding concerns the difference between nominal accelerator availability and effective application acceleration. CUDA support allowed the Jetson GPU to be used for LM inference, while CPU execution was faster for the short STT and TTS workloads because GPU-launch and transfer overheads outweighed the ben-

efits of offloading. On the other hand, the Raspberry Pi GPU and STM32MP2 NPU could not be exploited with the selected models and available runtime toolchains. We thus argue that Edge-AI platform selection should consider several aspects besides advertised peak accelerator performance, such as runtime support and maturity, model-conversion support, memory organization, overall system workload.

Across all three platforms, LM processing accounted for more than 90% of the sequential conversation-turn duration, establishing autoregressive decoding as the principal performance bottleneck. The memory-Roofline analysis further indicates that increasing external-memory bandwidth alone is insufficient to generate the complete average 225-token response within the adopted 1 s reference interval. The Roofline results represent first-order decoding bounds, but they show that responsive interaction requires the joint optimization of model footprint, output length, runtime efficiency. In particular, shorter responses and streaming LM-to-TTS execution can reduce perceived latency without requiring the complete response to be generated before speech synthesis begins.

Local execution reduces the need to transmit user audio and conversational history to an external inference provider, but it does not by itself guarantee privacy or security. Dialogue histories, model files, software updates, and device-control interfaces require protected storage, access control, authenticated updates, and secure API management. Furthermore, information obtained from external services may expose the agent to prompt injection or other forms of malicious manipulation. These aspects require explicit security mechanisms beyond the local-inference architecture considered in this work.

The functional assessment conducted in this study focuses on the selection of predefined device actions from explicit and implicit natural-language requests. It therefore evaluates an important component of agent operation, but does not constitute a complete validation, which would cover complete multi-turn dialogues, API-call and argument correctness, task completion, user-perceived quality, and security under erroneous or adversarial inputs. The deployment measurements also characterize the processing stages primarily in isolation to support bottleneck attribution, while full orchestration, model-residency policies, and speech-recognition error propagation require further investigation.

Future work building on these findings should consequently address two main directions. First, total and perceived interaction latency could be reduced by optimizing streaming generation, incremental speech synthesis, response structuring, and hardware acceleration. Second, domain-specific training and distillation should be investigated to reduce memory and computational costs. These developments will help transform the quantitative baselines established in this work into effectively deployable conversational-agent architectures for resource-constrained edge systems.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 30, Curran Associates, Inc., 2017, pp. 5998–6008.
URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.
URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[3] M. A. K. Raiaan, M. S. H. Mukta, K. Fatema, N. M. Fahad, S. Sakib, M. M. J. Mim, J. Ahmad, M. E. Ali, S. Azam, A review on large language models: Architectures, applications, taxonomies, open issues and challenges, IEEE Access 12 (2024) 26839–26874. doi:10.1109/ACCESS.2024.3365742.

[4] H. Ghaemi, Z. Alizadehsani, A. Shahraki, J. M. Corchado, Transformers in source code generation: A comprehensive survey, Journal of Systems Architecture 153 (2024) 103193. doi:https://doi.org/10.1016/j.sysarc.2024.103193.
URL https://www.sciencedirect.com/science/article/pii/S1383762124001309

[5] X. Sun, Q. Wang, F. Xie, Z. Quan, W. Wang, H. Wang, Y. Yao, W. Yang, S. Suzuki, Siamese transformer network: Building an autonomous real-time target tracking system for uav, Journal of Systems Architecture 130 (2022) 102675. doi:https://doi.org/10.1016/j.sysarc.2022.102675.
URL https://www.sciencedirect.com/science/article/pii/S1383762122001849

[6] K. Ye, Q. Yang, Z. Lu, H. Yu, T. Cui, R. Bai, L. Shen, Llm4netlist: Llm-enabled step-based netlist generation from natural language description, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 15 (2) (2025) 337–348. doi:10.1109/JETCAS.2025.3568548.

[7] Y. C. Kim, S. K. Yoon, S. S. Han, C. W. Park, J. O. Park, J. H. Ko, H. Kim, Accelerating language giants: A survey of optimization strategies for llm inference on hardware platforms, Journal of Systems Architecture 172 (2026) 103690. doi:https://doi.org/10.1016/j.sysarc.2026.103690.
URL https://www.sciencedirect.com/science/article/pii/S1383762126000081

[8] T. Bai, Y. Feng, S. Fu, Safeguarding user data privacy in online large language model services, Journal of Systems Architecture 168 (2025) 103555. doi:https://doi.org/10.1016/j.sysarc.2025.103555.
URL https://www.sciencedirect.com/science/article/pii/S1383762125002279

[9] M. Luo, H. He, W. Yang, S. Yuan, Z. Guan, An efficient privacy-preserving transformer inference scheme for cloud-based intelligent decision-making in aiot, Journal of Systems Architecture 172 (2026) 103687. doi:https://doi.org/10.1016/j.sysarc.2026.103687.
URL https://www.sciencedirect.com/science/article/pii/S1383762126000056

[10] A. Jaiswal, K. C. S. Shahana, S. Ravichandran, K. Adarsh, H. B. Bhat, B. K. Joardar, S. K. Mandal, Halo: Communication-aware heterogeneous 2.5-d system for energy-efficient llm execution at edge, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 14 (3) (2024) 425–439. doi:10.1109/JETCAS.2024.3427421.

[11] J. Huang, K. C.-C. Chang, Towards reasoning in large language models: A survey, in: A. Rogers, J. Boyd-Graber, N. Okazaki (Eds.), Findings of the Association for Computational Linguistics: ACL 2023, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 1049–1065. doi:10.18653/v1/2023.findings-acl.67.
URL https://aclanthology.org/2023.findings-acl.67/

[12] D. B. Acharya, K. Kuppan, B. Divya, Agentic ai: Autonomous intelligence for complex goals—a comprehensive survey, IEEE Access 13 (2025) 18912–18936. doi:10.1109/ACCESS.2025.3532853.

[13] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z.-Y. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, J.-R. Wen, A survey on large language model based autonomous agents, Frontiers of Computer Science 18 (6) (2024) 186345. doi:10.1007/s11704-024-40231-1.

[14] X. Dong, X. Zhang, W. Bu, D. Zhang, F. Cao, A survey of llm-based agents: Theories, technologies, applications and suggestions, in: 2024 3rd International Conference on Artificial Intelligence, Internet of Things and Cloud Computing Technology (AIoTC), 2024, pp. 407–413. doi:10.1109/AIoTC63215.2024.10748304.

[15] X. Li, A review of prominent paradigms for LLM-based agents: Tool use, planning (including RAG), and feedback learning, in: O. Rambow, L. Wanner, M. Apidianaki, H. Al-Khalifa, B. D. Eugenio, S. Schockaert (Eds.), Proceedings of the 31st International Conference on Computational Linguistics, Association for Computational Linguistics, Abu Dhabi, UAE, 2025, pp. 9760–9779.
URL https://aclanthology.org/2025.coling-main.652/

[16] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Qin, Y. Zheng, X. Qiu, X. Huang, Q. Zhang, T. Gui, The rise and potential of large language model based agents: a survey, Science China Information Sciences 68 (2) (2025) 121101:1–121101:44. doi:10.1007/s11432-024-4222-0.
URL https://doi.org/10.1007/s11432-024-4222-0

[17] E. King, H. Yu, S. Lee, C. Julien, Sasha: Creative goal-oriented reasoning in smart homes with large language models, Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 8 (1) (2024) 1–38. doi:10.1145/3643505.
URL http://dx.doi.org/10.1145/3643505

[18] M. D. Vu, H. Wang, J. Chen, Z. Li, S. Zhao, Z. Xing, C. Chen, Gptvoicetasker: Advancing multi-step mobile task efficiency through dynamic interface exploration and learning, in: Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology, UIST '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 749–765. doi:10.1145/3654777.3676356.
URL https://doi.org/10.1145/3654777.3676356

[19] Y. Guan, D. Wang, Z. Chu, S. Wang, F. Ni, R. Song, C. Zhuang, Intelligent agents with llm-based process automation, in: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 5018–5027. doi:10.1145/3637528.3671646.
URL https://doi.org/10.1145/3637528.3671646

[20] Z. Yin, M. Zhang, D. Kawahara, Harmony: A human-aware, responsive, modular assistant with a locally deployed large language model (2025). arXiv:2410.14252.
URL https://arxiv.org/abs/2410.14252

[21] L. Lazzaroni, F. Bellotti, R. Berta, An embedded end-to-end voice assistant, Engineering Applications of Artificial Intelligence 136 (2024) 108998. doi:https://doi.org/10.1016/j.engappai.2024.108998.
URL https://www.sciencedirect.com/science/article/pii/S0952197624011564

[22] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. B. et al., Llama 2: Open foundation and fine-tuned chat models (2023). arXiv:2307.09288.
URL https://arxiv.org/abs/2307.09288

[23] Hugging Face, Inference endpoints - hugging face documentation, https://huggingface.co/docs/inference-endpoints/index, accessed: 2025-09-16 (2025).

[24] NVIDIA Corporation, NVIDIA A40 GPU Accelerator, product Brief PB-09976-001_v08 (Jan. 2021).
URL https://www.nvidia.com/en-us/data-center/a40/

[25] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. L. et al., The llama 3 herd of models (2024). arXiv:2407.21783.
URL https://arxiv.org/abs/2407.21783

[26] NVIDIA Corporation, NVIDIA Jetson AGX Xavier Module (Dec. 2018).
URL https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/

[27] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, Y. Zhang, Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions, in: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2020, pp. 6124–6128. doi:10.1109/ICASSP40776.2020.9053889.

[28] I. Elias, H. Zen, J. Shen, Y. Zhang, Y. Jia, R. Skerry-Ryan, Y. Wu, Parallel tacotron 2: A non-autoregressive neural tts model with differentiable duration modeling (2021). arXiv:2103.14574.
URL https://arxiv.org/abs/2103.14574

[29] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brébisson, Y. Bengio, A. C. Courville, Melgan: Generative adversarial networks for conditional waveform synthesis, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 32, Curran Associates, Inc., 2019, pp. 14910–14921.
URL https://proceedings.neurips.cc/paper_files/paper/2019/file/6804c9bca0a615bdb9374d00a9fcba59-Paper.pdf

[30] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. Mcleavey, I. Sutskever, Robust speech recognition via large-scale weak supervision, in: A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, J. Scarlett (Eds.), Proceedings of the 40th International Conference on Machine Learning, Vol. 202 of Proceedings of Machine Learning Research, PMLR, 2023, pp. 28492–28518.
URL https://proceedings.mlr.press/v202/radford23a.html

[31] A. S. Dhanjal, W. Singh, A comprehensive survey on automatic speech recognition using neural networks, Multimedia Tools and Applications 83 (8) (2024) 23367–23412. doi:10.1007/s11042-023-16438-y.
URL https://doi.org/10.1007/s11042-023-16438-y

[32] N. Jeffries, E. King, M. Kudlur, G. Nicholson, J. Wang, P. Warden, Moonshine: Speech recognition for live transcription and voice commands (2024). arXiv:2410.15608.
URL https://arxiv.org/abs/2410.15608

[33] R. Chevi, R. E. Prasojo, A. F. Aji, A. Tjandra, S. Sakti, Nix-tts: Lightweight and end-to-end text-to-speech via module-wise distillation, in: 2022 IEEE Spoken Language Technology Workshop (SLT), 2023, pp. 970–976. doi:10.1109/SLT54892.2023.10023322.

[34] J. Kim, J. Kong, J. Son, Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech, in: M. Meila, T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning, Vol. 139 of Proceedings of Machine Learning Research, PMLR, 2021, pp. 5530–5540.
URL https://proceedings.mlr.press/v139/kim21f.html

[35] R. Atienza, Efficientspeech: An on-device text to speech model, in: ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2023, pp. 1–5. doi:10.1109/ICASSP49357.2023.10094639.

[36] J. Kong, J. Kim, J. Bae, Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 17022–17033.

URL https://proceedings.neurips.cc/paper_files/paper/2020/file/c5d736809766d46260d816d8dbc9eb44-Paper.pdf

[37] V.-T. Nguyen, H.-C. Pham, D.-K. Mac, How to push the fastest model 50x faster: Streaming non-autoregressive speech synthesis on resouce-limited devices, in: ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2023, pp. 1–5. doi:10.1109/ICASSP49357.2023.10096329.

[38] Y. Ren, C. Hu, X. Tan, T. Qin, S. Zhao, Z. Zhao, T.-Y. Liu, Fastspeech 2: Fast and high-quality end-to-end text to speech (2022). arXiv:2006.04558.
URL https://arxiv.org/abs/2006.04558

[39] OHF-Voice contributors, piper1-gpl: Fast and local neural text-to-speech engine, gPL-3.0 licensed; accessed on 26 August 2025 (2023).
URL https://github.com/OHF-Voice/piper1-gpl

[40] Liquid AI, LFM2.5-1.2B-Thinking: On-device reasoning under 1gb, Liquid AI Blog (2026).
URL https://www.liquid.ai/blog/lfm2-5-1-2b-thinking-on-device-reasoning-under-1gb

[41] Qwen Team, Qwen3.5: Towards native multimodal agents (Feb. 2026).
URL https://qwen.ai/blog?id=qwen3.5

[42] Granite models documentation, accessed on 26 August 2025 (2025).
URL https://www.ibm.com/granite/docs/models/granite/

[43] IBM Granite Team, Granite-3.3-8B-Instruct model card, Hugging Face, https://huggingface.co/ibm-granite/granite-3.3-8b-instruct, accessed 2025-09-09 (2025).

[44] TEN Framework, Ten: Open-source framework for real-time multimodal conversational ai, https://theten.ai/, accessed: 2026-07-08 (2025).

[45] Microsoft, Onnx runtime, https://onnxruntime.ai/, accessed: 2025-09-09 (2025).

[46] Microsoft, Onnx runtime genai, https://onnxruntime.ai/docs/genai/, accessed: 2025-09-09 (2024).

[47] G. Gerganov, llama.cpp: Inference of llama and other transformer-based models in pure c/c++, https://github.com/ggerganov/llama.cpp, accessed: 2025-09-09 (2023).

[48] Ollama Team, Ollama: Run large language models locally, https://ollama.ai/, accessed: 2025-09-09 (2024).

[49] V. Pratap, Q. Xu, J. Kahn, G. Avidov, T. Likhomanenko, A. Hannun, V. Liptchinsky, G. Synnaeve, R. Collobert, Scaling up online speech recognition using convnets, in: Interspeech 2020, 2020, pp. 3376–3380. doi:10.21437/Interspeech.2020-2840.

[50] NVIDIA Corporation, Nvidia jetson agx orin developer kit technical brief, accessed on 26 August 2025 (2022).
URL https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf

[51] Raspberry Pi Ltd., Raspberry pi 5 product brief, accessed on 26 August 2025 (2023).
URL https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

[52] STMicroelectronics, Stm32mp257f-ev1 evaluation board, accessed on 26 August 2025 (2024).
URL https://www.st.com/en/evaluation-tools/stm32mp257f-ev1.html

[53] STMicroelectronics, STM32MP251C/F, STM32MP253C/F, STM32MP255C/F, STM32MP257C/F - Datasheet, accessed September 14, 2025 (2025).
URL https://www.st.com/resource/en/datasheet/stm32mp251c.pdf

[54] STMicroelectronics, An5730: Guidelines for measuring the system power consumption on the stm32mp25x line mpus, application Note, accessed on 26 August 2025 (2025).
URL https://www.st.com/resource/en/application_note/an5730-guidelines-for-measuring-the-system-power-consumption-on-the-stm32mp25x-line-mpus-stmicroelectronics.pdf

[55] STMicroelectronics, How to deploy your NN model on STM32MPU, sTM32 MPU ecosystem wiki (2025).
URL https://wiki.st.com/stm32mpu/wiki/How_to_deploy_your_NN_model_on_STM32MPU

[56] Seeed Technology, Mini usb microphone, accessed on 26 August 2025 (2019).
URL https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/338/114992002_Web.pdf

[57] Maikii S.r.l., Eye: 3 w bluetooth speaker with rechargeable 500 mah battery, accessed on 28 August 2025 (2022).
URL https://www.maikii.com/altoparlante-bluetooth/eye/

[58] Python Software Foundation, Python documentation — wave module, https://docs.python.org/3/library/wave.html, accessed: 2025-09-01 (2025).

[59] Python Software Foundation, Python documentation — time module, https://docs.python.org/3/library/time.html, accessed: 2025-09-01 (2025).

[60] R. Bonghi, jetson-stats: Simple python package to monitor and control your nvidia jetson, https://github.com/rbonghi/jetson_stats, accessed: 2025-09-01 (2025).

[61] Raspberry Pi Foundation, Raspberry pi documentation — vcgencmd, https://www.raspberrypi.com/documentation/computers/os.html#vcgencmd, accessed: 2025-09-01 (2025).

[62] Joy-IT, JT-UM120 USB-Multimeter Datasheet (2023).
URL https://joy-it.net/files/files/Produkte/JT-UM120/JT-UM120_Datasheet-EN_2023-10-19.pdf

[63] T. Shiwa, T. Kanda, M. Imai, H. Ishiguro, N. Hagita, How quickly should a communication robot respond? delaying strategies and habituation effects, International Journal of Social Robotics 1 (2) (2009) 141–155. doi:10.1007/s12369-009-0012-8.

[64] Z. Yuan, Y. Shang, Y. Zhou, Z. Dong, Z. Zhou, C. Xue, B. Wu, Z. Li, Q. Gu, Y. J. Lee, Y. Yan, B. Chen, G. Sun, K. Keutzer, Llm inference unveiled: Survey and roofline model insights (2024). arXiv:2402.16363.
URL https://arxiv.org/abs/2402.16363

[65] S. Williams, A. Waterman, D. Patterson, Roofline: An insightful visual performance model for multicore architectures, Communications of the ACM 52 (4) (2009) 65–76. doi:10.1145/1498765.1498785.

[66] NVIDIA Corporation, Jetson Thor Modules for Robotics and Edge AI – Datasheet, https://nvdam.widen.net/s/mdn8tjqrzn/robotics-datasheet-update-jetson-thor-modules-nvidia-us-4767587, accessed: 2026-02-23 (2024).

[67] Advanced Micro Devices, Inc., Alveo U55C Data Center Accelerator Cards Data Sheet, accessed: 2026-07-21 (2023).
URL https://docs.amd.com/r/en-US/ds978-u55c

[68] K. Asifuzzaman, N. R. Miniskar, A. R. Young, F. Liu, J. S. Vetter, A survey on processing-in-memory techniques: Advances and challenges, Memories - Materials, Devices, Circuits and Systems 4 (2023) 100022. doi:https://doi.org/10.1016/j.memori.2022.100022.
URL https://www.sciencedirect.com/science/article/pii/S2773064622000160

[69] M.-S. Li, J.-F. Ke, E.-M. Huang, Z.-W. Liu, Y.-G. Chen, C.-Y. Lee, Cim for transformer models: Enhancing large language model inference efficiency, in: 2025 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Vol. 1, 2025, pp. 1–6. doi:10.1109/ISVLSI65124.2025.11130222.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: