

AIVA: An Agentic Intelligent Voice Assistant with Hybrid Local and LLM Processing on the ESP32-S3

Prof. Amey R. Solanke

Department of Industrial IoT Engineering
Prof. Ram Meghe Institute of Technology and Research
Badnera-Amravati, Maharashtra, India

Yogeshwari Jitendra Mutyalu

Student
Department of Industrial IoT Engineering
Prof. Ram Meghe Institute of Technology and Research
Badnera-Amravati, Maharashtra, India

Atharva Sanjay Nakod

Student
Department of Industrial IoT Engineering
Prof. Ram Meghe Institute of Technology and Research
Badnera-Amravati, Maharashtra, India

Asmita Pramod Paraskar

Student
Department of Industrial IoT Engineering
Prof. Ram Meghe Institute of Technology and Research
Badnera-Amravati, Maharashtra, India

Pratik Manoharrao Deshmukh

Student
Department of Industrial IoT Engineering
Prof. Ram Meghe Institute of Technology and Research
Badnera-Amravati, Maharashtra, India

Abstract — Voice assistants normally rely on cloud based processing to understand requests, which creates network dependency and response delay, while microcontroller class embedded platforms lack the memory and computing power to run a complete conversational AI locally. This paper presents the design of AIVA (Agentic Intelligent Voice Assistant), a voice assistant built around the ESP32-S3 that follows one rule: do locally what the device can handle and use a large language model (LLM) only when deeper language reasoning is required. A digital I2S microphone feeds an on device pipeline for wake word detection and intent identification. A routing layer then sends simple supported requests to a command engine for immediate device control and forwards complex natural language requests to an LLM. Replies are delivered through a speaker and an OLED status display. We describe the system architecture, operational flow, hardware and software needs, a first order latency model of the hybrid design and a structured evaluation plan covering response time, reliability, accuracy and resource usage. The architecture keeps essential functions available without internet connectivity, avoids unnecessary cloud dependence and is modular enough to extend with more IoT devices, multilingual interaction and personalized memory.

Keywords — voice assistant, ESP32-S3, edge computing, large language model, hybrid architecture, TinyML, Internet of Things, human-computer interaction

I. INTRODUCTION

Voice has become one of the most natural interfaces between people and machines. Commercial voice assistants generally rely on cloud-based processing to interpret a request and generate a reply. Cloud processing offers strong language understanding, but it also makes every interaction depend on network availability, adds round-trip delay even to trivial commands such as switching on a light, and sends user requests to remote servers, which raises well-known privacy and security concerns [1].

Low-cost embedded platforms such as the ESP32-S3 now combine a dual-core processor, Wi-Fi connectivity and I²S

audio interfaces, which makes them appealing as always-on voice front ends and IoT controllers. Moving intelligence toward the network edge reduces latency and bandwidth use [2]. However, the limited memory and computing power of a microcontroller make it impractical to run a complete conversational AI system locally. The assistant must also be able to act on services such as IoT devices and Internet APIs.

The guiding principle of this project is: do what the device can handle; use the LLM when deeper language reasoning is required. The main contributions of this work are:

- 1) a hybrid local/LLM architecture for an MCU-class platform, in which the local-versus-LLM decision is an integral part of the design;
- 2) an intent and routing layer that sends simple supported commands to a local command engine and complex natural-language requests to an LLM;
- 3) a design that keeps essential assistant functions available when internet connectivity is lost; and
- 4) a first-order latency model and a structured evaluation plan covering response time, reliability, functional accuracy and resource usage.

The remainder of the paper is organized as follows. Section II reviews related work, Section III states the problem and objectives, Section IV presents the proposed system and methodology, Section V lists the system requirements, Section VI defines the evaluation plan, Section VII discusses the expected outcomes, Section VIII outlines the applications and Section IX concludes the paper.

II. RELATED WORK

Voice assistants have evolved from command-based systems into conversational systems, but implementing voice intelligence on resource-constrained embedded devices remains challenging because both speech recognition and

conversational AI demand significant computation. Prior work can be grouped into TinyML-based speech processing and offline or privacy-focused voice assistants.

A. TinyML-Based Voice Recognition

Kwon and Park [3] studied hardware/software co-design for a TinyML voice-recognition application. By moving the Hann windowing step of the audio front end into hardware attached to the P2S sound module, they reduced the execution time of the front-end function by 8.06% and energy consumption by 3.27%. This is relevant to AIVA because an embedded voice system must use its limited resources efficiently; however, the work addresses efficient voice processing only, not LLM-based conversation or local/cloud task routing.

Zhang *et al.* [4] proposed Tiny Transducer, a lightweight phone-based transducer with a small decoding graph for edge devices. With only 0.9 M parameters after SVD compression, it achieved a relative improvement of 9.1–20.5% over a larger conventional hybrid system. The work shows that speech recognition can be compressed for edge deployment, but recognition alone provides neither conversational intelligence nor a decision between local and cloud processing.

Maayah *et al.* [5] presented LimitAccess, an on-device TinyML system on an Arduino Nano 33 BLE Sense that uses a lite CNN to classify the speaker's age group and so restrict children's access to speech-controlled vehicle functions, reporting an F1 score of 87.7%. It demonstrates that speech-related processing can run directly on constrained hardware and reduce cloud dependence, but broader conversational reasoning is outside its scope. Ray [6] reviewed the TinyML landscape (tools, enablers, frameworks and use cases) and identified open challenges in running machine learning on devices with very limited processor and memory. Gondi and Pratap [7] evaluated the performance of offline speech recognition on edge devices, which is the local-recognition setting AIVA builds on.

B. Offline and Local-LLM Voice Assistants

Lazzaroni *et al.* [8] proposed an embedded end-to-end voice assistant that works entirely offline by combining several deep-learning models. Using transfer learning for the Italian language, they report accuracy comparable to cloud-connected recognition in an automotive use case, with latency reduced by eliminating the cloud connection. In general, however, a fully offline system cannot match the breadth of language understanding of a large cloud-hosted language model.

Wibowo *et al.* [9] built a privacy-focused AIoT voice assistant for smart-building control using local speech and language models, with a Raspberry Pi Zero 2W audio node and a local GPU server running Whisper, Ollama-hosted LLMs and Piper, integrated with Home Assistant. Across 33 IoT entities the system reached a 96.67% execution success rate with an average response time of 5.5 s. Local processing improves privacy and removes reliance on external services, but running complete local language models needs far more capable hardware than an MCU-class platform. Birkmose *et al.* [10] likewise explored on-device LLMs for Home Assistant in the dual role of intent detection and response generation. These observations motivate a lighter hybrid architecture for the ESP32-S3.

C. Comparison of Existing Approaches

Table I summarizes the approaches and positions AIVA among them.

TABLE I. COMPARISON OF APPROACHES

Approach	Main advantage	Main limitation
Fully local voice processing	Low latency and offline operation	Limited computational capability
TinyML speech recognition	Embedded speech processing	Limited conversational intelligence
Cloud voice assistant	Strong language understanding	Internet dependency
Local LLM assistant	Greater privacy and independence	Requires powerful hardware
AIVA hybrid approach	Local speed plus LLM intelligence	Complex features require network

D. Identified Research Gap

The literature shows two main directions. Fully local processing gives low latency and offline capability but is limited by embedded resources. Cloud-based processing gives conversational intelligence but depends on network connectivity. TinyML research also stresses managing memory, computation, latency and energy on embedded devices [6]. There is therefore an opportunity for a task-based hybrid architecture in which simple device-level requests are handled locally and only complex language tasks are sent to an LLM. This is the research direction of AIVA.

III. PROBLEM STATEMENT AND OBJECTIVES

A. Problem Statement

Modern voice assistants rely heavily on internet connectivity to understand and answer requests. This reliance can cause response delays, reduce functionality when the network is unavailable, and send simple requests to remote servers. On the other hand, embedded devices such as the ESP32-S3 have limited memory and computing resources, making a complete conversational AI system impractical to run locally. The problem addressed in this paper is: how can an embedded voice assistant handle requests locally for fast and reliable response while using an LLM for complex conversational tasks?

B. Aim

To develop an ESP32-S3-based intelligent voice assistant that performs voice commands and supported device operations locally and uses an LLM for conversational requests, balancing embedded response with advanced AI capabilities.

C. Objectives

- 1) To develop an ESP32-S3-based voice assistant for human–computer interaction.
- 2) To implement wake-word detection and basic voice commands.
- 3) To control supported device functions locally with quick response.
- 4) To integrate an LLM for complex questions and conversational interaction.
- 5) To develop a routing mechanism that selects local or LLM processing according to the type of request.

- 6) To keep essential assistant functions available when internet connectivity is unavailable.
- 7) To evaluate the system in terms of response time, reliability, functionality and resource usage.

IV. PROPOSED SYSTEM AND METHODOLOGY

8) System Overview

The proposed AIVA system includes an ESP32-S3, a microphone, a command engine, an intelligent routing layer, an LLM layer, audio output and an OLED status display (Fig. 1). The microphone captures the user's voice and sends digital audio to the ESP32-S3. After wake-word detection, the system identifies the request and decides whether it can be handled locally. A simple supported command is executed directly by the command engine, which reduces latency and lets essential functions work without internet connectivity. A request that needs natural-language understanding is sent to the LLM, which generates a response that is returned to the system and spoken to the user. The methodology follows this pipeline: voice input → wake word → intent check → local processing or LLM → action or response → audio output.

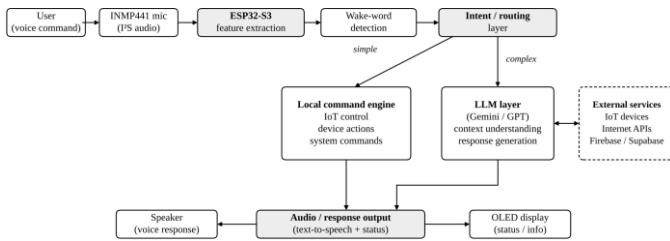


Fig. 1. Block diagram of AIVA, the Agentic Intelligent Voice Assistant.

B. Functional Blocks

User. The user gives commands or questions through natural voice interaction.

Microphone. The INMP441 is an I²S MEMS microphone [11]; it captures the user's speech and provides digital audio to the ESP32-S3.

ESP32-S3 controller. The ESP32-S3 [12] is the controller. It performs preprocessing, feature extraction, wake-word processing, routing and supported device control.

Wake-word detection. The assistant listens for a predefined activation word such as "Hey AIVA". When that word is detected, the assistant begins listening for a command and processes commands only after it has been addressed.

Routing layer. This layer examines the request and decides whether it can be handled locally or requires the LLM.

Local command engine. The engine runs device commands, such as IoT control, device actions and system commands, on the embedded platform without cloud processing.

LLM layer. The LLM, for example Gemini or GPT, handles natural-language questions and conversations that need deeper language understanding, context, response generation and multilingual interaction in Hindi, English and Marathi.

External services. The LLM path can use services such as IoT devices, Internet APIs (for example weather or general knowledge) and a cloud database such as Firebase or Supabase for memory and user data.

Audio output. The assistant converts responses to speech and plays them through a speaker driven by the MAX98357A amplifier [13].

OLED display. The SSD1306 OLED shows visual status information and feedback about the system.

C. Operational Flow and Routing Logic

The assistant listens continuously for the wake word. After activation, voice input is processed on the ESP32-S3 (preprocessing and feature extraction) and the user's intent is identified. A decision block then asks whether the request is simple. If so, the local command engine handles it; if not, the LLM processing layer performs context understanding, reasoning and response generation. Both paths converge at response generation or execution, followed by audio and OLED feedback. Algorithm 1 summarizes the routing logic and Fig. 2 gives the corresponding flowchart.

Algorithm 1: Request routing

1. Listen continuously for the wake word.
2. On wake word: capture the utterance; extract features on the ESP32-S3.
3. intent ← classify(utterance)
4. **if** intent is a supported command **then**
 5. Execute the command locally; give audio and OLED feedback
6. **else if** network is available **then**
 7. Forward the request to the LLM and receive the response
 8. Play the response through the speaker; update the OLED
9. **else**
 10. Inform the user that the request needs connectivity
11. **end if**
12. Return to step 1.

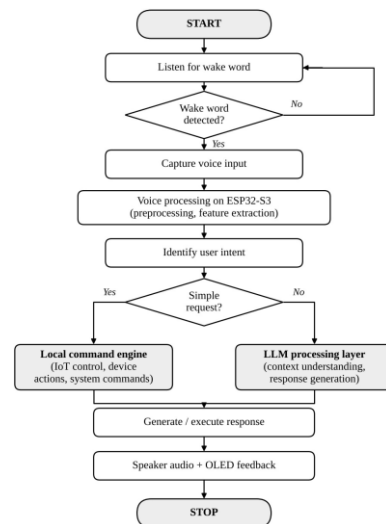


Fig. 2. Operational flowchart of AIVA.

D. First-Order Latency Model

To understand the advantage of routing, let p be the fraction of requests that the assistant handles locally and let t represent the time taken by each stage. The total latency for the local path and the LLM path are:

$$T_{loc} = t_{cap} + t_{feat} + t_{rt} + t_{act} \quad (1)$$

$$T_{llm} = t_{cap} + t_{feat} + t_{rt} + t_{net} + t_{inf} + t_{tts} \quad (2)$$

In these equations t_{cap} is the capture time, t_{feat} is the on-device feature extraction and intent identification time, t_{rt} is the routing decision time, t_{act} is the local action and feedback time, t_{net} is the network round-trip time, t_{inf} is the LLM inference time and t_{tts} is the speech synthesis and playback startup time.

The expected latency of the system and its improvement over a baseline that sends every request to the LLM are:

$$E[T] = p T_{loc} + (1 - p) T_{llm} \quad (3)$$

$$\Delta T = p (T_{llm} - T_{loc}) \quad (4)$$

Equation (4) assumes that the capture and feature-extraction times are similar on both paths. It shows that the latency improvement grows linearly with p and with the cost of the LLM round trip, and that the improvement disappears only if the local path is not faster than the LLM path. This model gives the evaluation in Section VI a target: measure each stage, estimate p from a representative workload, and compare the predicted latency with the observed latency.

E. Offline Operation and Graceful Degradation

The device can handle all supported commands when Wi-Fi or the internet is not available, meeting Objective 6. When a request needs the LLM, the assistant informs the user through audio and the OLED display rather than failing quietly. This approach reflects the limitation noted in Table I: complex features need the network, while essential local functions do not.

V. SYSTEM REQUIREMENTS

A. Hardware Requirements

The core hardware is the ESP32-S3 system-on-chip. It has a dual-core Xtensa LX7 processor, Wi-Fi and Bluetooth LE, and vector instructions that speed up neural-network tasks [12]. The prototype also includes the INMP441 MEMS microphone with digital I²S output [11] and the MAX98357A I²S-input Class-D audio amplifier [13]. Table II lists all components.

TABLE II. HARDWARE COMPONENTS AND THEIR PURPOSE

Component	Purpose
ESP32-S3	Main embedded controller
INMP441 I ² S microphone	Voice input
MAX98357A amplifier	I ² S audio amplification
Speaker	Voice output
OLED (SSD1306)	Visual status and feedback
Relay / supported device	Local device control
Power supply	System power
Breadboard	Prototype development
Jumper wires	Component connections

B. Software Requirements

The firmware is written in C/C++ using the Arduino IDE or PlatformIO with the ESP32-S3 development framework. The software stack also contains voice-processing code for wake-word detection, an LLM API for complex requests, a text-to-speech system for spoken replies, Wi-Fi connectivity and the required API and backend services.

C. Functional Requirements

The system shall:

- detect the wake word;
- capture voice input;
- process supported commands locally;
- determine whether a request requires LLM processing;
- send complex requests to the LLM;
- generate an appropriate response;
- control supported local devices;
- provide audio output;
- display system status; and
- continue essential local functions during internet unavailability.

VI. EVALUATION PLAN

The prototype will be evaluated on response time, reliability, accuracy and resource usage. To isolate the benefit of the hybrid design, the same hardware will also run in a configuration in which every request is sent to the LLM (that is, $p = 0$ in (3)). Table III defines the metrics and how each is measured.

TABLE III. EVALUATION METRICS AND MEASUREMENT METHOD

Metric	Definition	Measurement
Response latency	Time from end of user speech to the device action (local) or start of the audio reply (LLM)	Firmware timestamps at each pipeline stage
Routing accuracy	Share of utterances sent to the correct path	Labeled test set; confusion matrix
Functional accuracy	Share of commands executed correctly	Labeled commands; success/failure count
Wake-word performance	Detection rate and false activations	Repeated utterances at varied distance and noise; idle-time false triggers
Offline availability	Success rate of supported local commands without internet	Repeat local tests with Wi-Fi disabled
Reliability	Failures per session; recovery from Wi-Fi or LLM errors	Long-duration sessions; error log
Resource usage	Free heap, CPU load and power draw	Runtime monitoring; inline current measurement

Five test scenarios are planned: (i) supported local commands with the network connected; (ii) the same commands with the network disconnected; (iii) complex natural-language queries served by the LLM; (iv) a mixed workload of both request types to measure routing accuracy and p ; and (v) varied acoustic conditions (distance and background noise) for wake-word and command accuracy. Each scenario will be repeated over several trials and speakers, and results will be reported as mean and standard deviation.

VII. EXPECTED OUTCOMES AND DISCUSSION

A. Expected Outcomes

The proposed project is expected to produce a working hybrid voice assistant on the ESP32-S3 with the following outcomes:

- fast local handling of basic voice commands;
- advanced conversational responses to complex questions through an LLM;
- essential local functions that remain available during internet loss;

- a functional hybrid local–cloud architecture suitable for an embedded platform;
- reduced unnecessary dependence on cloud processing;
- intelligent routing between local and LLM processing;
- measurable evaluation of response latency, reliability, functional accuracy and resource usage; and
- a modular architecture that can be extended with additional commands and IoT devices.

B. Discussion

Latency versus coverage. Equation (4) implies that the more requests the local engine can resolve, the larger the latency gain. The local command set, however, is bounded by the memory of the microcontroller, so it should be chosen from the most frequent device-level requests, leaving open-ended language to the LLM.

Routing quality. Sending a simple command to the LLM wastes time, whereas sending a complex request to the local engine yields a failed or wrong action. The routing layer is therefore the critical component, and routing accuracy is evaluated explicitly in Table III.

Privacy. Requests resolved locally do not leave the device, and only requests that genuinely need deeper reasoning are forwarded to the LLM, which reduces exposure relative to a cloud-only assistant [1]. Local-LLM designs such as [9] go further but need far more capable hardware than an MCU-class platform.

Limitations. The proposed system still needs network access for complex requests, inherits the latency, cost and availability of the chosen LLM service, and depends on third-party APIs. These limitations are inherent to the hybrid trade-off and are the reason essential functions are kept local.

VIII. APPLICATIONS

The hybrid design suits smart-home environments and IoT control systems, personal assistance, educational systems, kiosks, and other embedded applications that require voice interaction. The project lies at the intersection of artificial intelligence, embedded systems, the Internet of Things, voice processing, edge computing and human–computer interaction.

IX. CONCLUSION AND FUTURE WORK

This paper presented AIVA, a voice assistant that handles simple supported commands locally on the ESP32-S3 and delegates complex natural-language requests to an LLM. By making the local-versus-LLM decision part of the architecture, the design combines embedded response and offline availability of essential functions with the conversational intelligence of a large language model. We described the architecture, operational flow, requirements, a first-order latency model and an evaluation plan for the prototype.

Future work includes measuring the prototype against the plan in Section VI and extending the architecture with more IoT devices, multilingual interaction (Hindi, English and Marathi), personalized memory through a cloud database, advanced automation and other AI services. Because voice assistants expose attack surfaces [14], hardening the audio and API paths is another direction.

ACKNOWLEDGMENT

The author thanks Prof. Amey R. Solanke for guidance and support.

REFERENCES

- [1] T. Bolton, T. Dargahi, S. Belguith, M. S. Al-Rakhami, and A. H. Sodhro, "On the security and privacy challenges of virtual assistants," *Sensors*, vol. 21, no. 7, Art. no. 2312, 2021, doi: 10.3390/s21072312.
- [2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.
- [3] J. Kwon and D. Park, "Hardware/software co-design for TinyML voice-recognition application on resource frugal edge devices," *Appl. Sci.*, vol. 11, no. 22, Art. no. 11073, Nov. 2021, doi: 10.3390/app112211073.
- [4] Y. Zhang, S. Sun, and L. Ma, "Tiny Transducer: A highly-efficient speech recognition model on edge devices," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2021, arXiv:2101.06856.
- [5] M. Maayah, A. Abunada, K. Al-Janahi, M. E. Ahmed, and J. Qadir, "LimitAccess: On-device TinyML based robust speech recognition and age classification," *Discover Artif. Intell.*, vol. 3, no. 1, pp. 1–19, 2023, doi: 10.1007/s44163-023-00051-x.
- [6] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 4, pp. 1595–1623, 2022, doi: 10.1016/j.jksuci.2021.11.019.
- [7] S. Gondi and V. Pratap, "Performance evaluation of offline speech recognition on edge devices," *Electronics*, vol. 10, no. 21, Art. no. 2697, 2021, doi: 10.3390/electronics10212697.
- [8] L. Lazzaroni, F. Bellotti, and R. Berta, "An embedded end-to-end voice assistant," *Eng. Appl. Artif. Intell.*, 2024.
- [9] F. Wibowo, S. Suheri, F. Faisal, and F. Rolansa, "Privacy-focused AIoT: Implementing an offline voice assistant for smart building management using local LLMs," *G-Tech: Jurnal Teknologi Terapan*, vol. 10, no. 2, pp. 652–665, Apr. 2026, doi: 10.70609/g-tech.v10i2.9342.
- [10] R. Birkmose, N. M. Reece, E. H. Norvin, J. Bjerva, and M. Zhang, "On-device LLMs for Home Assistant: Dual role in intent detection and response generation," 2025, arXiv:2502.12923.
- [11] TDK InvenSense, "INMP441: Omnidirectional microphone with bottom port and I²S digital output," datasheet. [Online]. Available: <https://invensense.tdk.com/digital/inmp441>
- [12] Espressif Systems, "ESP32-S3 series datasheet," Shanghai, China. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf
- [13] Analog Devices, "MAX98357A/MAX98357B datasheet," Wilmington, MA, USA. [Online]. Available: <https://www.analog.com/en/products/max98357a.html>
- [14] C. Yan, X. Ji, K. Wang, Q. Jiang, Z. Jin, and W. Xu, "A survey on voice assistant security: Attacks and countermeasures," *ACM Comput. Surv.*, vol. 55, no. 4, pp. 1–36, 2023, doi: 10.1145/3527153.