

Agentic Hardware Framework for Device Controls using Low-Resource Natural Language

Pius Obilie
Department of Computer
Engineering
University of Ghana
Legon, Accra, Ghana
obliepius13@gmail.com

Latifah Abubakar
Department of Computer
Engineering
University of Ghana
Legon, Accra, Ghana
latifahabubakar216@gmail.com

Elsie Effah Kaufmann
Department of Biomedical
Engineering
University of Ghana
Legon, Accra, Ghana
eeffahkaufmann@ug.edu.gh

Godfrey A. Mills
Department of Computer
Engineering
University of Ghana
Legon, Accra, Ghana
gmills@ug.edu.gh

Abstract—Internet of Things (IoT) solutions using natural language and speech recognition systems have offered flexibility to users to directly control devices. Integration of voice assistant solutions such as Alexa and Google Assistant to IoT systems has made IoT solutions more user friendly in automation systems and smart homes. Incidentally, these natural language driven solutions are predominantly based on high-resource languages, leaving speakers of low-resource languages across developing countries unable to control embedded hardware devices in smart home environments using their natural languages. The available low-resource voice assistants that support smart home solutions mostly operate as a translation layer over the existing manufacturer's ecosystem and its interoperability constraints. This paper presents an Agentic Smart Home Assistant (ASHA) that connects commands from low-resource natural language (such as the Twi language spoken in Ghana), directly into the hardware controller level without using standard protocol for smart home device controls. The framework combines a fine-tuned automatic speech recognition system for the low-resource language, a large language model (LLM) reasoning layer, and a firmware library that exposes the hardware microcontroller bus operations as natural-language-addressable (NLA) functions. The framework was deployed and tested across a set of sensors and actuators connected to a commodity microcontroller device for a smart home environment. Tests results demonstrated that the framework was able to achieve correct task execution across all 8 trials spanning 4 communication bus types (7 live-speech, 1 typed command), despite a 62.6% word-error rate in the underlying Twi automatic speech recognition component (detailed in Section V). The results demonstrate the feasibility of a low-resource natural language-driven embedded hardware control in smart homes and other natural language control of physical devices.

Keywords—Internet of Things, large language models, low-resource languages, embedded systems, speech recognition, human-computer interaction

I. INTRODUCTION

Internet-of-Things (IoT) technology has changed the level of automation in homes and facilities with improvement in efficiency, user flexibility and convenience. In smart home environments, IoTs have been deployed to control operations of connected devices and systems. With the integration of artificial intelligence and machine learning technologies such as natural language in IoTs, these have further boosted the useability and functionality of IoTs in diverse domains and areas where natural language control of physical devices are required.

IoT-enabled smart homes consist of a network of peripheral devices such as sensors, actuators, appliances, and

an embedded Wi-Fi-enabled **hardware microcontroller node** that provides communication via the Internet. Low-cost, Wi-Fi-enabled microcontrollers such as ESP32 and Raspberry Pi have placed hardware device controls within the reach of the individual solution developers including large manufacturers [1]. The ability to integrate voice-assisted natural language interfaces like Alexa and Google Assistant with IoT networks provides flexibility and convenience in smart environments. However, as both peripheral devices and the microcontroller nodes multiply, the natural language used by the users to issue commands for device control becomes as consequential as the underlying hardware itself.

So far, the use of high-resource natural languages in device control operations is well-established. However, low-resource language solutions for physical device control operations in smart homes **in developing countries** is still under-developed. With the rapidly changing technological landscape with more enhanced solutions and automation drive, the need for such low-resource language in embedded solutions will be eminent. Already, smart home solutions using high-resource languages are becoming prevalent in many developing countries with non-existent local language-driven device control operations. With the growth in the market for smart home solutions in developing countries, local content will be essential to accelerate utilization rate.

Few local-language voice assistants have begun to emerge to address this low-resource language gap. **One such solution** is the Abena AI product [10], an offline-first voice assistant that supports the Ghanaian Twi language as well as other African languages for the control of specific devices within the smart home environments. However, the Abena AI only controls specific manufacturer's ecosystem rather than at the level of the underlying hardware bus. This means it can only interact with devices that have already been certified to that specific manufacturer's ecosystem.

Contemporary approaches to physical device control using low-resource natural languages present three unresolved challenges. First, current natural language interfaces remain restricted to the high-resource languages. This means speakers of the low-resource languages like the Ghanaian Twi have no equivalent means of issuing direct device control commands. Secondly, the issue of interoperability across heterogeneous sensors and actuators remains a problem to be solved. Standard protocols like Matter framework require individual devices to be certified, which leaves uncertified devices that communicate over general-purpose buses like I2C, SPI, and PWM unsupported. Thirdly, existing standard protocols like the Matter framework solutions lack context-aware reasoning,

which makes them execute commands as stateless instructions, repeating actions needlessly (such as attempting to turn on a light bulb that is already on, or watering a plant that was just watered) rather than recognizing the device's current state and responding appropriately.

Fortunately, recent advances in large language models (LLMs) and automatic speech recognition (ASR) present an opportunity to address these three challenges concurrently. Current LLMs can interpret open-ended natural language instructions and translate them into structured, executable actions without requiring a fixed command vocabulary [3]. When an LLM is paired with a fine-tuned ASR component for a low-resource language and a firmware layer that exposes bus-level operations as callable functions, this can create a standalone natural language-to-embedded control pipeline that operates independently without interventions from any manufacturer's proprietary standard.

In this paper, we present an Agentic Smart Home Assistant (ASHA) hardware abstraction framework, that connects low-resource language commands such as Twi language spoken in Ghana, for direct control operation of devices in a smart home. The natural language device control commands are issued via messaging platforms (WhatsApp, Telegram, and Discord) as voice notes or text messages directly into embedded hardware microcontroller-level for control operations. The framework was developed using FastAPI for the backend, Model Context Protocol (MCP) for the agent tools, MQTT for the agent-to-device communication, and a custom C++ firmware library. The framework was implemented on the ESP32-WROOM-32 embedded hardware and tested using multiple sensors and actuators in a smart home setting. The functional performance of the system was evaluated using task success rate, end-to-end latency, and speech recognition word error rate.

The rest of the paper is organized as follows. Section II examines recent solutions relating to command-driven LLMs for IoT applications. Section III presents the design and development of the proposed ASHA framework, while Section IV describes the experimental methodology. Section V presents the results and discussion, and Section VI concludes the paper and outlines directions for future work.

II. RELATED WORK

Voice-assistant solutions for smart home automation have gained widespread global adoption, with platforms such as Amazon Alexa, Google Assistant, and Apple Siri serving as primary user interfaces for consumer Internet of Things (IoT) ecosystems. Despite their commercial success, these platforms are engineered almost exclusively around high-resource languages that possess vast digital corpora and established acoustic training datasets [2]. Consequently, speakers of underrepresented, low-resource languages in developing regions remain largely excluded from voice-driven automation. While recent initiatives have begun addressing low-resource speech recognition, these systems are typically deployed as standalone transcription or translation tools rather than interactive control interfaces for physical hardware. Bridging this divide requires integrating natural language understanding directly with embedded hardware actuation.

Recent research has increasingly explored Large Language Models (LLMs) to overcome the rigid syntax and fixed vocabulary constraints of traditional voice interfaces. In the work of Zeng et al. [3], an LLM-driven security and

control assistant was proposed for heterogeneous IoT networks. Their framework deployed Chain-of-Thought (CoT) reasoning to decompose high-level, ambiguous natural language instructions into structured execution steps. Evaluated across simulated smart home clusters comprising lighting, environmental sensors, and smart locks, their system demonstrated that multi-step prompt reasoning significantly improved command parsing accuracy compared to direct keyword matching, achieving over 90% execution precision under complex instruction sets. However, their approach operated strictly at the software simulation layer, assuming standardized API endpoints rather than interfacing directly with uncertified microcontroller hardware buses.

Similarly, Yokotsuji et al. [4] investigated an LLM-based service platform designed to enhance interoperability among diverse smart home devices. Their methodology mapped real-time sensor streams including temperature, humidity, and motion readings into structured, descriptive natural language prompts. The LLM analyzed these ambient data inputs to infer environmental context and autonomously trigger appropriate actuator commands across connected appliances. Experimental evaluations on a multi-sensor testbed confirmed that the platform could dynamically formulate and execute context-aware control rules without requiring hardcoded automation scripts. Nonetheless, the architecture relied on pre-integrated, manufacturer-standardized communication gateways, limiting its applicability to off-the-shelf, uncertified components.

In an effort to achieve autonomous multi-step physical task execution, Rivkin et al. [5] introduced the Smart Autonomous Goal Execution (SAGE) framework. SAGE utilized an LLM-powered agent to translate abstract, long-horizon user goals into sequences of grounded, executable subtasks within smart home environments. Evaluated across diverse simulated household automation scenarios, SAGE demonstrated an 85% task completion rate, effectively managing dependency ordering and state verification across multiple devices. However, SAGE was built upon pre-existing high-level device APIs and assumed English-only input modalities, leaving both low-resource language processing and raw bus-level peripheral communication unaddressed.

Device interoperability across heterogeneous hardware has traditionally been addressed through abstraction layers operating at different tiers of the computing stack. At the embedded operating system level, platforms such as RIOT [6] provide compile-time hardware abstraction layers (HALs) that decouple low-level microcontroller drivers from application logic, enabling uniform peripheral access across disparate silicon architectures. At the middleware level, service-oriented architectures encapsulate proprietary device protocols behind standardized web APIs to unify communication across vendor boundaries [7]. At the application standard level, the Web of Things framework applies semantic descriptors to describe IoT endpoints uniformly, while the Matter protocol [8] establishes unified data models and discovery clusters for certified commercial devices. More recently, edge-agent architectures such as ESP-Claw [9] have demonstrated dynamic hardware capability exposure using the Model Context Protocol (MCP) paired with on-chip Lua execution on ESP32 microcontrollers.

Despite these developments, a critical architectural gap remains. Standardized protocols like Matter [8] mandate strict manufacturer certification, excluding vast ecosystems of low-

cost, commodity sensors and actuators that interface via raw bus protocols such as Digital I/O, Analog, PWM, I2C, and SPI. Furthermore, existing edge-agent frameworks assume high-resource language interfaces, while low-resource language technologies remain detached from embedded control pipelines.

The proposed ASHA framework addresses the challenges of context-aware, low resource support, and interoperability support by offering an alternative approach to device-level communication to hardware microcontroller communication bus level. By exposing universal bus operations as callable tools within an LLM reasoning loop, ASHA enables plug-and-play control operations of arbitrary sensors and actuators without requiring any per-device firmware modifications or vendor certification. The contribution of this paper is twofold

- (1) **Bus-Level Hardware Interoperability:** A hardware abstraction framework implemented on commodity ESP32 microcontrollers that standardizes control at the raw communication bus layer (Digital, Analog, PWM, I2C, and SPI). This architecture eliminates the need for per-device driver development or vendor ecosystem certification, allowing uncertified commodity peripherals to be controlled through a unified software interface.
- (2) **Low-Resource Natural Language to Embedded Control:** An end-to-end integration combining a fine-tuned Ghanaian Twi automatic speech recognition (ASR) pipeline with an agentic LLM reasoning engine. This extends direct, context-aware hardware control to low-resource African language currently unsupported by mainstream voice-activated smart home platforms.

III. SYSTEM DESIGN

Fig. 1 presents the ASHA framework architectural diagram that was implemented. The framework is structured as a four-layer architecture designed to decouple multimodal user interactions from low-level microcontroller execution.

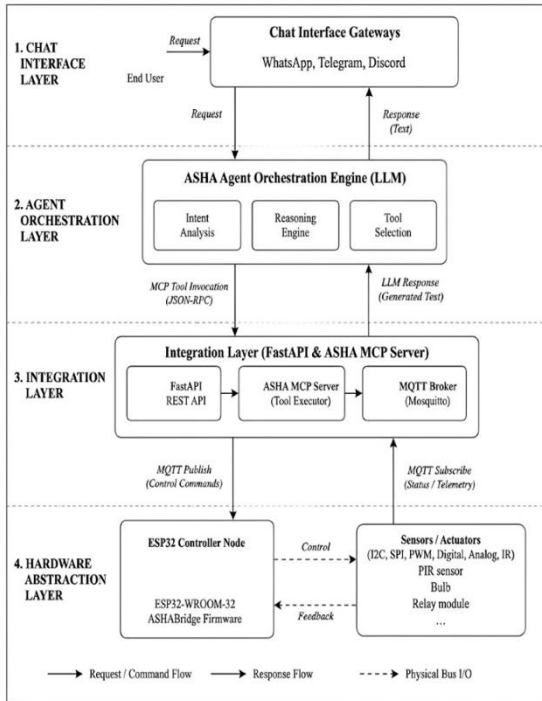


Fig. 1. ASHA system diagram.

The topmost layer is the Chat Interface Layer that statelessly ingests user inputs across messaging platforms. The second layer is the Agent Orchestration Layer that manages dialogue state, constructs context windows dynamically, and drives iterative LLM tool-calling loops. The third layer is the Integration Layer which provides backend services, authentication, Model Context Protocol (MCP) tooling, and MQTT message routing while the lowest layer, the Hardware Abstraction Layer (HAL), performs executions on the ESP32 hardware microcontroller to carry out physical bus-level peripheral operations and run dynamic edge scripts.

A. Chat Interface Layer

The chat interface layer serves as the user-ingression and egression gateway of the ASHA framework. Rather than requiring users to install proprietary mobile applications or interact with specialized voice-related hardware system such as a dedicated microphone, the ASHA framework allows users to directly use the commonly available messaging platforms comprising WhatsApp, Telegram, and Discord. The chat interface layer currently handles two primary modalities of user input: typed natural language text and spoken audio voice notes. The user messages are ingested asynchronously through platform web-hooks and decoupled from the backend execution to ensure that the interface remains stateless with respect to physical device actuation.

With the user text ingestion, typed commands bypass acoustic processing as in Fig. 2 and are forwarded directly to the agent orchestration layer alongside active user session identifiers. In the case of user audio ingestion, the inbound voice notes are buffered as raw audio payload and passed to a language identification classifier (MMS-LID), which distinguishes between the legacy resource language (English language) and the low-resource language (Ghanaian spoken Twi language). Inputs identified as English are transcribed directly using an English ASR model. When the low-resource language (the Twi language) is detected, the audio is transcribed into native text using the fine-tuned MMS Twi ASR model and translated into the English language via the NLLB-200. Both the original Twi transcript and the translated English text are bundled together and forwarded to the agent orchestration layer to provide full semantic context for downstream reasoning. The outbound responses from the agent orchestration layer are transmitted back to the user through the platform's API as conversational text or as voice notes generated by a fine-tuned TTS model, matching the user's initial input modality.

To ensure a responsive conversational user experience for smart home control, the framework was architected targeting an end-to-end interactive latency of under 30 seconds from message ingestion to user feedback.

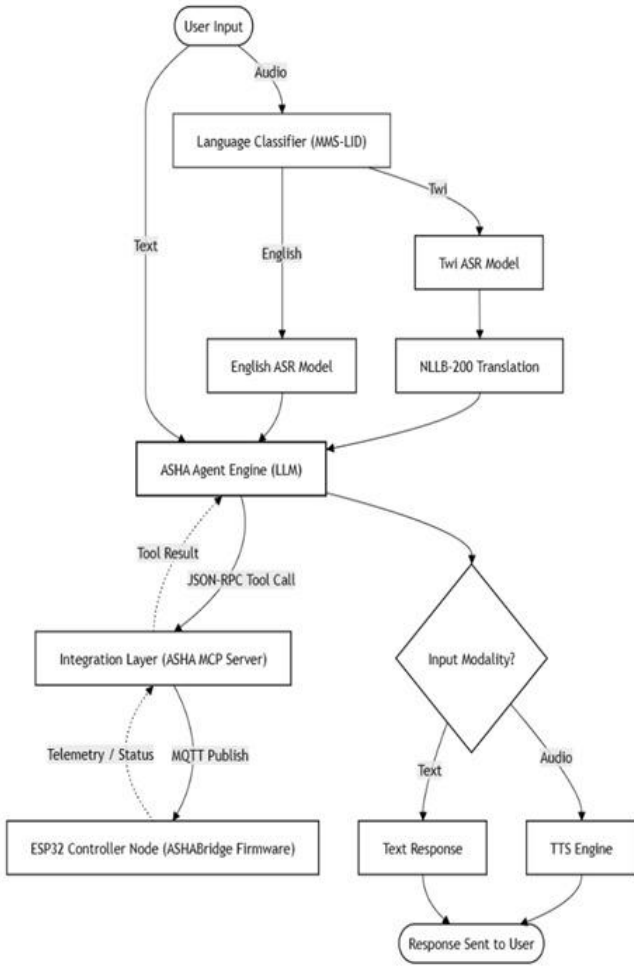


Fig. 2. Detailed ASHA operational flow diagram.

B. Agent Orchestration Layer

The agent orchestration layer serves as the core cognitive decision-making and task-planning engine of the ASHA framework, translating high-level, open-ended user intentions into deterministic hardware operations. It operates asynchronously to manage dialogue state, coordinate model inference, and orchestrate iterative tool-calling sequences.

As presented in Fig. 1 and Fig. 2, when an incoming query is received from the chat interface layer, the orchestration engine retrieves the user's active dialogue history from an in-memory Redis data store. The context assembly and state management dynamically builds the context window for the LLM by combining the system prompt's behavioral guidelines with the user's dialogue history, the current query (including translated transcripts for Twi voice inputs), and the schemas of available MCP tools. To prevent the context window from saturation during extended interactions, the orchestration layer monitors the conversation length. When the dialogue history in the Redis reaches a threshold of 150 messages, the orchestration layer summarizes the previous exchanges, compacts them into a concise conversational summary, and replaces the message history with this summary alongside the active query before forwarding it back to the agent.

The assembled context is sent to the LLM through an OpenAI-compatible interface. The agent runs an iterative ReAct (Reasoning and Acting) loop to break down complex user

requests, check device states before and after taking action, and make sure commands run correctly on the hardware.

If a hardware manipulation or state verification is required, the LLM generates structured tool invocations encoded as JSON-RPC payloads targeting the appropriate device node. The orchestration engine then routes these JSON commands down to the integration layer for execution. The function looping captures the intermediate tool execution outputs and feeds them back into the context window, prompting the model to re-evaluate the system state until all subtasks are successfully resolved. Upon completing the execution loop, the orchestration layer synthesizes a final natural language confirmation or status report, forwarding the text or synthesized voice note back to the chat interface layer for presentation to the user.

C. Integration Layer

The integration layer serves as a bridge between the AI agent and the physical hardware. It combines FastAPI, the Model Context Protocol (MCP), and an MQTT broker to manage system settings, give the agent standard tools to control devices, and route messages across the network.

Backend Services and Device Management: A REST API handles user authentication and stores device details in a database. The database maintains records of all connected hardware, including specific pin assignments and communication buses, ensuring the agent retrieves accurate device parameters before taking action.

MCP Tooling Server: The MCP server provides the agent with structured tools to interact directly with the ESP32 hardware. These tools allow the agent to read device and bus configurations, send one-time control commands, run real-time tasks, check device states, and schedule automated actions.

Message Routing and Telemetry: An MQTT broker handles communication between the backend and hardware nodes using dedicated topics. The integration layer sends control commands to connected devices and receives sensor data or status updates in return. These updates are routed back to the agent, which evaluates the context and sends the final response to the user through the chat interface.

D. Hardware Abstraction Layer

The Hardware Abstraction Layer (HAL) consists of a C++ firmware library developed for the ESP32 and is designed around a core observation. While individual device models change rapidly, the underlying communication buses they use remain the same. A temperature sensor and a soil moisture sensor from different manufacturers can both use I2C. Similarly, a servo motor and an LED dimmer both use PWM. By working at the bus level rather than the individual device level, the HAL achieves broad compatibility across sensors and actuators without needing custom code for each new device. Currently, the HAL supports Digital, Analog, PWM, I2C, SPI, and Infrared (IR) interfaces.

To show the range of devices supported by this bus-level approach, Table I lists representative sensors and actuators tested across each communication bus type.

TABLE I. LIST OF SUPPORTED DEVICES BY BUS TYPE

Device	Bus	Category
HC-SR501 PIR Motion Sensor	Digital	Sensor
5V Single-Channel Relay Module	Digital	Actuator
LDR (Photoresistor)	Analog	Sensor
Capacitive Soil Moisture Sensor	Analog	Sensor
SG90 Micro Servo	PWM	Actuator
BME280 (Pressure/Temp/Humidity)	I2C	Sensor
MPU6050 (6-Axis IMU)	I2C	Sensor
MFRC522 (RFID Reader, 13.56 MHz)	SPI	Sensor
IR LED Transmitter (KY-005)	IR	Actuator

To support real-time edge device execution without facing challenges of communication responsiveness, in the current implementation, the firmware isolates the device operations across the ESP32's dual cores. Core 1 of the ESP32 handles one-time commands, network connectivity functionality and the MQTT command routing. Core 0 of the ESP32 on the other hand focuses on execution of real-time sensitive tasks through an embedded Lua script engine. This architecture allows the agent to dynamically deploy persistent, and event-driven scripts such as continuous sensor monitoring while ensuring the device remains responsive to subsequent network commands.

IV. EXPERIMENTS

The functional capabilities of the proposed ASHA framework were evaluated on a smart-home testbed deployed in a residential setting. All tests were executed using a consumer-grade laptop equipped with 16GB RAM and an Intel CPU, without dedicated GPU acceleration. Cloud-based LLM reasoning was routed via API to Claude 3.5 Sonnet through OpenRouter, while all speech and language components—including the MMS-LID language classifier, fine-tuned Twi ASR model, NLLB-200 translation model, and the TTS engine were executed locally on the laptop's CPU.

Two individuals participated in the experimental evaluation: the system developer and an independent native Ghanaian Twi speaker. Testing was structured into four distinct experimental procedures to evaluate end-to-end performance under realistic operational conditions:

Speech Modality and Acoustic Channel Comparison: An initial trial comparing command recognition accuracy between live spoken audio and pre-recorded audio played back through an external loudspeaker.

Direct Bus-Level Hardware Actuation: Live-speech trials evaluating natural language translation and physical actuation across Digital, PWM, Analog, and SPI buses using the peripherals listed in Table I. (I2C interface firmware was implemented, but physical hardware trials were omitted during this evaluation cycle due to peripheral component availability).

Dynamic Real-Time Automation: An evaluation of persistent on-device edge script deployment, where an SPI RFID card scan event triggers a digital actuator via the embedded Lua engine on Core 0.

Fault Handling and Hallucination Resistance: Fault-injection scenarios testing system behavior under simulated edge failures (reading an unpowered hardware node) and ambiguous acoustic input (non-speech audio payloads).

V. RESULTS

A. Live Speech vs. Recorded Playback

An initial comparison was conducted to assess whether the audio delivery method affected transcription accuracy. Three commands were issued as pre-recorded audio played back through a phone speaker into the receiving phone microphone, compared against five commands spoken live.

Recorded playback produced correct outcomes in only 1 of 3 trials, including one trial in which the agent executed an incorrect action due to a garbled transcription. In contrast, live spoken speech produced correct task outcomes in all 5 trials.

This difference in accuracy between live speech and playback occurs because loudspeaker distortion and room reverberation degrade the acoustic signal before it reaches the microphone. These results confirm that live spoken input provides a much more accurate and realistic testing condition for real-world smart home interactions.

B. Direct Bus-Level Hardware Actuation

Table II summarizes the experimental results across all bus types tested, noting input modality for each trial. Seven of the eight trials were conducted using live speech; the SPI bus trial was conducted via a typed WhatsApp command. Because per-bus trial counts are small (1 to 4 trials), these results serve as a proof-of-concept feasibility demonstration rather than a broad statistical evaluation. Similarly, the human-subject evaluation involved a limited participant cohort (N = 2) comprising the system developer and one independent native speaker in a non-blinded setting. While the 100% task success rate establishes end-to-end technical feasibility under controlled conditions, it does not constitute a broad statistical generalization across diverse speaker demographics, regional Twi dialects, or unconstrained domestic acoustic environments.

Similarly, the human-subject evaluation involved a limited participant cohort (N = 2) comprising the system developer and one independent native speaker in a non-blinded setting. While the 100% task success rate establishes end-to-end technical feasibility under controlled conditions, it does not constitute a broad statistical generalization across diverse speaker demographics, regional Twi dialects, or unconstrained domestic acoustic environments.

A trial was scored as correct if the physical hardware action matched the user's intended outcome, regardless of whether the intermediate ASR transcript was word-perfect. In several trials, the Twi ASR component introduced transcription errors. For example, the phrase "tu ɔpono no" (meaning "close the door") was transcribed in place of the intended command "to ɔpono no". Despite this acoustic error, the LLM reasoning layer used surrounding dialogue context to infer the true user intent and triggered the correct hardware action.

TABLE II. TASK SUCCESS RATE BY BUS TYPE

Bus Type	Input Modality	Trials	Correct
Digital	Live Speech	1	1
PWM	Live Speech	4	4

Analog	Live Speech	2	2
SPI (RFID Reader)	Typed (WhatsApp)	1	1
Overall	Both Typed and Live Speech	8	8

Two additional live trials (recorded-playback comparison excluded) resulted in the agent correctly declining to act rather than guessing at an ambiguous or unclear transcription; these are discussed separately in Section V-A and not counted as task failures, as declining to act on unclear input is the intended, safe behavior.

The SPI trial recorded in Table II evaluated the system’s real-time, on-device automation path via a typed WhatsApp command rather than live speech. The user instructed the agent in English to turn on a light whenever a connected RFID reader detected a card scan. This required the agent to deploy a persistent Lua script to the hardware node rather than issue a one-time action, exercising the event-driven execution path described in Section III-D. The script was correctly generated and deployed within approximately 15 seconds, and the light activated correctly on the subsequent card scan, confirming the real-time automation path functions correctly over the SPI bus.

To evaluate the Infrared (IR) bus, a dedicated hardware test was conducted using a 32-inch Samsung Smart TV as the target appliance. A typed command ("turn on the TV") was issued to the agent via WhatsApp, which successfully synthesized the appropriate IR signal and powered on the television. This trial is excluded from Table II because it utilized typed input rather than live spoken audio, but it confirms IR bus functionality independent of the acoustic pipeline.

C. Automatic Speech Recognition Evaluation

The automatic speech recognition (ASR) component was evaluated on the held-out test split of the Mozilla Common Voice Twi corpus, consisting of 46 speech utterances resampled to 16 kHz mono. Prior to scoring, ground-truth reference texts and model hypotheses were normalized by converting text to lowercase, removing punctuation, and collapsing whitespace, while explicitly preserving language-specific orthographic graphemes (ɛ and ɔ).

Word Error Rate (WER) was formally computed via word-level Levenshtein dynamic programming alignment according to (1):

$$WER = \frac{S + D + I}{N} \times 100\% \quad (1)$$

where S is the number of word substitutions (words incorrectly transcribed), D is the number of deletions (words present in reference but omitted in hypothesis), I is the number of insertions (extraneous words introduced), and N represents the total number of words in the ground-truth reference transcript ($N = S + D + I + C$, where C denotes the count of correct words).

Under this benchmark evaluation, the fine-tuned Twi ASR model achieved an overall WER of 62.6%. While this error rate reflects the inherent challenges of low-resource African language speech recognition with limited fine-tuning corpora, qualitative error analysis reveals that errors were overwhelmingly word substitutions (S) driven by dialectal

spelling variations (e.g., Asante vs. Akuapem orthography), compound word boundaries, and vowel length differences, rather than acoustic omissions (D). Crucially, because phonemic roots of spoken commands were preserved, the downstream LLM reasoning layer (ReAct loop) demonstrated substantial semantic resilience, successfully mapping phonetically distorted commands (e.g., resolving near-miss transcriptions) to valid tool execution arguments and achieving 100% actuation success across the hardware trials.

TABLE III. SAMPLE TWI ASR TRANSCRIPTIONS

#	Reference Transcript	Model Hypothesis	Observation
1	ohaw bere mu titiriw na wohia adamfo <i>(It is especially in times of trouble that you need a friend)</i>	ohal bre mu titire na wohia adamfo	Phonetic alignment with dialectal suffix variation (adamfo vs. adamfo)
2	mekoo fam nkakrankakra senea ebye a mimpira asase no <i>(I went down slowly so that I wouldn't hurt/hit the ground.)</i>	mekoo fam nkakra nkakraa sedee beye a nempira asaase no	Dialectal synonym (sedee for senea) and vowel lengthening (asaase)
3	ankye koraa na kwabena hyiaa abena <i>(It wasn't long at all before Kwabena met Abena.)</i>	nkyekra na kwabena hyia bena	Spoken contraction (nkyekra); proper nouns (kwabena, abena) preserved
4	meboa no <i>(I will help him/her)</i>	neboa no	Single nasal phoneme substitution (n for m)

D. Fault Handling

Two fault-injection scenarios were tested against the Analog sensor path. In the first, the hardware node was powered off before a sensor-read command was issued; the agent correctly reported that no reading had been received rather than fabricating a value, though its wording ("the sensor may be flaky or out of range") did not explicitly state that the device was unreachable. In the second, a deliberately unclear voice note (non-speech audio) was submitted; the agent correctly declined to take any action and requested clarification rather than guessing at an intended command. In both trials, the agent did not hallucinate a false success, which was the primary behavior under test.

E. Latency

End-to-end latency, measured from voice note submission to confirmed physical action or spoken response, ranged from under 15 seconds (typed command, no ASR involved) to approximately 110 seconds for the slowest voice-based trial. Live speech trials generally completed within 30-75 seconds. While typed commands comfortably satisfied this requirement (under 15 seconds), voice-based trials exceeded the 30-second target defined in the system's non-functional requirements (Section III-A) due to CPU-only inference on consumer hardware rather than the GPU-accelerated or optimized deployment environment the system is intended for. Latency is expected to improve significantly under GPU-

backed inference or with a smaller, quantized local model, which remains as future work.

VI. CONCLUSION AND FUTURE WORK

This paper presented an agentic hardware abstraction framework that connects a low-resource natural language command spoken in the Ghanaian Twi language directly via voice notes or the commonly used social media platforms such as WhatsApp, Telegram, or Discord, to the microcontroller for device control operations across multiple communication buses, without depending on standard manufacturer protocols or smart-home protocols. Unlike existing voice-controlled systems that are limited to high-resource languages or bound to single-vendor ecosystems, the ASHA framework combines the bus-level hardware interoperability with natural language processing for a low-resource African language. Experimental evaluation across Digital, PWM, Analog, and SPI buses demonstrated correct task execution **across all 8 trials (7 live-speech, 1 typed command)**, safe fault-handling behavior with zero observed hallucination during device-offline and noisy audio conditions, and functional end-to-end execution on consumer-grade CPU hardware.

These preliminary findings demonstrate the proof-of-concept feasibility of the ASHA framework for smart home applications and suggest the viability of physical device control using low-resource natural languages. Furthermore, the bus architecture solution can also be integrated into existing standard protocols, such as the Matter framework, to resolve interoperability constraints in smart home environments.

While the current solution has been successful in meeting the desired design requirements, refinement of the model using other low-resource languages may help strength the solution. **A useability study** by deploying the model in a real-world smart home environment will further strengthen the results. Future extensions will scale the framework across four main directions:

- 1) **Protocol and Peripheral Expansion:** Future work will validate physical I2C sensor arrays and integrate additional embedded buses, specifically UART to support serial peripherals such as GPS modules, and CAN bus to enable automotive and industrial applications.
- 2) **Cross-Platform Firmware Portability:** While currently targeted to ESP32 microcontrollers, the C++ hardware abstraction layer will be ported to broader platforms, including Arduino-compatible boards and Linux-based single-board computers such as the Raspberry Pi.
- 3) **Hardware Acceleration and Latency Optimization:** Deploying quantized local speech models and GPU-backed inference will reduce processing latency

toward real-time interactive levels (**well below the 30-second threshold**).

- 4) **Linguistic Generalization and Extended Evaluation:** Expanding testbed evaluations with larger speaker cohorts and generalizing the acoustic and reasoning pipeline to other low-resource regional languages with similar phonetic structures.

ACKNOWLEDGMENT

This work was supported in part by Google Outreach for Machine Learning through a grant award supporting machine learning research and training activities at the University of Ghana.

REFERENCES

- [1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.
- [2] "Automatic Speech Recognition (ASR) for African Low-Resource Languages: A Systematic Literature Review," *arXiv preprint arXiv:2510.01145*, 2025.
- [3] M. Zeng, X. Du, and M. Guizani, "Large Language Model-Driven Security Assistant for Internet of Things via Chain-of-Thought," *IEEE Internet of Things Journal*, vol. 12, no. 24, pp. 51832–51841, Dec. 2025.
- [4] R. Yokotsuji, D. Lin, and F. Uwano, "LLM-based Interoperable IoT Service Platform," in *Proc. 2024 IEEE/WIC Int. Conf. on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2024, pp. 438–444.
- [5] D. Rivkin, F. Hogan, A. Feriani, A. Konar, A. Sigal, X. Liu, and G. Dudek, "AIoT Smart Home via Autonomous LLM Agents," *IEEE Internet of Things Journal*, vol. 12, no. 3, pp. 2458–2472, Feb. 2025.
- [6] E. Baccelli, C. Gündoğan, O. Hahm, P. Kietzmann, M. S. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch, "RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4428–4440, Dec. 2018.
- [7] V. Issarny, G. Bouloukakakis, N. Georgantas, and B. Billet, "Revisiting service-oriented architecture for the IoT: a middleware perspective," in *Int. Conf. on Service-Oriented Computing*. Springer, 2016, pp. 3–17.
- [8] Connectivity Standards Alliance, *Matter 1.4 Core Specification*, Document 24-27349-006, 2024. [Online]. Available: https://csa-iot.org/wp-content/uploads/2024/11/24-27349-006_Matter-1.4-Core-Specification.pdf. [Accessed: Jul. 2026].
- [9] Espressif Systems, "ESP-Claw: Edge-Agent Architecture with Model Context Protocol and On-Chip Scripting for ESP32 Microcontrollers," GitHub repository, 2025. [Online]. Available: <https://github.com/espressif/esp-claw>
- [10] Abena AI, "Abena: Offline-First Voice Assistant for African Languages," 2024. [Online]. Available: <https://abena.ai/> [Accessed: Aug. 2026]