Optimizing Large Language Models Assisted Smart Home Assistant Systems at the Edge: An Empirical Study

Krishna Sruthi Velaga[1], **Yifan Guo**[1]
[1]Department of Computer & Information Sciences, Towson University, Towson, MD 21204 USA
kvelaga1@students.towson.edu, yguo@towson.edu

Abstract

The last decade has witnessed the widespread adoption of AI-assisted smart home applications on the network edge, supported by improvements in edge device hardware accelerations and AI computing algorithms. Particularly, the surge of Large Language Models (LLMs) in 2022 pushes smart home applications to handle more complicated and multiple tasks, such as chat-bots, video surveillance, signal sensing, voice controls, etc. However, new changes have appeared in response precisions, delays, and power consumption with limited computation power and resources when utilizing LLM services in resource-constrained edge environments. To this end, in this study, we develop a testbed to evaluate the efficacy and latency of real-time responses and actions from on-device models directly in smart home environments. Based on it, we leverage lightweight and fine-tuned LLMs optimized for seamless integration with benchmark home assistant systems, a popular open-source platform for smart home automation, on resource-constrained edge devices like Raspberry Pis. Furthermore, we optimize the search engines for configured devices in system configurations, shortening the response delay further. In our evaluation, we have utilized four models to evaluate their real-time on-device performance, including a pre-trained model (serving as our baseline), e.g., the Home-1B model, and three customized and fine-tuned models, e.g., TinyHome, TinyHome-Qwen, and StableHome, based on a medium-sized synthetic smart home dataset tailored to smart home environments. Evaluation results show that our optimized models maintain high accuracy in understanding and executing user commands. More importantly, with optimizations, we reduce the response time by around 82%, from originally 45.1 seconds to 7.9 seconds, on average, for four models. `Our demo video can be reached with the link:` https://youtu.be/zukPKLNWR54.

Introduction

Large Language Models (LLMs) are advanced artificial intelligence systems based on transformer model architectures, designed to process and generate human-like language by identifying sequential patterns within tons of volumes of datasets. They excel in understanding context, generating coherent responses, and performing tasks such as text summarization, language translation, and conversational interactions. Smart home assistants, on the other hand, are AI-driven systems that manage connected devices like lights, thermostats, and security cameras. Integrated into devices such as smart speakers or smartphones, they use natural language processing to execute commands, automate tasks, and provide personalized control based on user behavior.

Integrating LLMs with smart home assistants enhances home automation by enabling more precise and intelligent interactions. These models improve the assistant's ability to interpret complex commands and personalize user experiences by analyzing preferences and behaviors to suggest optimized routines or recommend energy-saving practices. By unifying devices into a cohesive ecosystem, this integration ensures smooth interoperability, supports natural conversational interactions, and creates a smarter, more adaptive home environment. For instance, (Shi et al. 2024) and (King et al. 2024) introduced AwareAuto and Sasha, enabling accurate automation and open-ended communication interpretation. (Civitarese et al. 2024) and (Zeng et al. 2024b) developed ADL-LLM and GestureGPT for precise activity and gesture recognition using sensor data and common-sense reasoning. (Takeda et al. 2024) highlighted LLMs' effectiveness in zero-shot learning, while (Gao et al. 2024) and (Zeng et al. 2024a) improved efficiency with edge training.

However, when LLMs-enabled smart home assistant systems perform on low-power devices, low precision and long latency in responses still remain critical issues that prevent the broader utilization of LLMs in real smart home applications. Thus, how to fine-tune the model tailored for smart home environments with more precise responses and optimize the model architecture and the search engine of smart home assistants to reduce the overall latency has become our major concern. In this paper, we have developed a testbed with an LLM-enabled Home Assistant platform on Raspberry Pi 5 to evaluate the response precision and delay empirically. We finetune the model with post-training techniques like prompt tuning and LoRA to reduce inference latency with improved response performance tailored for smart home environments. Also, we optimize the search engine of configured devices, which significantly shortens the response time. Evaluation performance verifies the efficacy of our solutions, with more precise answers and significantly short delay.

The contributions of this paper are listed below:

- We have developed a testbed with an LLM-enabled Home Assistant platform on Raspberry Pi 5 to evaluate the on-device and real-time response precision and delay empirically.
- We finetune the model with post-training techniques like prompt tuning and LoRA on synthetic smart home datasets tailored to smart home environments to reduce inference latency with improved response performance. Also, we optimize the search engine of configured devices, which significantly shortens the response time.
- We testify the performance of four lightweight models on the edge device. Evaluation results show that our optimized models maintain high accuracy in understanding and executing user commands. More importantly, with optimizations, we reduce the response time by around 82%, from 45.1 seconds to 7.9 seconds, on average.

Related Work

The recent advancements in Enhanced User Programming (EUP) systems demonstrate significant progress in user-centered automation through the integration of LLMs. For instance, (Shi et al. 2024) introduced AwareAuto, which standardized user expressions and utilizes a two-step inference with LLMs to generate automation, achieving an impressive 91.7% accuracy in aligning with user intentions. Building on this foundation, (Rey-Jouanchicot et al. 2024) developed a proactive system that not only leveraged LLMs but also interacted with the environment, resulting in a 20-fold increase in operational speed and a 26.4% improvement in performance over larger, less specialized models.

The application of LLMs has also been expanded to activity recognition and smart home assistance. (Civitarese et al. 2024) implemented ADL-LLM that transforms raw sensor data into textual descriptions for zero-shot recognition of Activities of Daily Living (ADLs), with an option to boost accuracy via few-shot prompting. Similarly, (Zeng et al. 2024b) and (King et al. 2024) introduced GestureGPT and Sasha, respectively; the former integrated LLM's reasoning to analyze gestures within a triple-agent framework, achieving high accuracy in smart home tasks, while the latter used an LLM to interpret open-ended commands for natural user-device interaction. These systems underscored the versatility of LLMs in understanding and facilitating human-machine interaction in everyday environments. For models dealing with sparse data, (Takeda et al. 2024) discussed LLMs like GPT-3.5 and GPT-4 excel in zero- and few-shot learning, addressing early-stage activity recognition when labeled data were limited. Similarly, (Cleland et al. 2024)'s approach leveraged LLMs' contextual understanding and language modeling to process natural language descriptions derived from binary sensor data in smart home environments. Focusing on mental state monitoring, (Fan et al. 2024) explored integrating foundation models and images from vacuum robots to detect behaviors indicative of mental states, specifically focusing on identifying smoking- and drinking-alone behaviors.

In terms of deployment efficiency, significant strides have been made to optimize LLM deployment. For example, (Gao et al. 2024) and (Zeng et al. 2024a) discussed methods like parallelized randomized gradient estimation and collaborative edge training to enhance the efficiency of on-device AI training, reducing reliance on remote cloud processing. (Ur Rahman et al. 2023) described large-scale models that can be converted to an optimized FlatBuffer format for deployment on resource-limited edge devices, with evaluations focusing on latency, performance, and resource efficiency. (Yonekura et al. 2024) further explored leveraging LLMs to simulate human-like activities, utilizing their experiential knowledge and adaptability to enhance intelligence and responsiveness in smart home environments. Finally, (Paul et al. 2024) demonstrated how large language models (LLMs) could enhance small language models for device control tasks by developing an automated system that uses LLMs to generate device control planning data.

From a security perspective, several pioneered works have been conducted to safeguard sensitive operations. Particularly, (Woszczyk, Lee, and Demetriou 2021) and (Li et al. 2024) addressed the vulnerabilities of voice-controlled systems to acoustic attacks and model theft, proposing solutions like Sesame and CoreGuard to ensure robust security measures on edge devices. These innovations are crucial for maintaining the integrity and privacy of user data in increasingly automated environments.

System Implementation

Our Overall System Architecture

The system architecture in our developed testbed is meticulously designed to integrate various components that collectively enhance home automation and control, as illustrated in Fig. 1. It comprises four major components and their functions within the system: 1) Remote Monitoring Devices (Laptop/PC). They are connected wirelessly to the home network and serve as the primary user interface, facilitating remote access and command execution via SSH to a Raspberry Pi. 2) Edge Devices (Raspberry Pi 5). They host Home Assistant systems for smart device control and deploying and running LLMs to process commands. To ensure stable communication performance, a wired connection to the Wi-Fi router is used in our testbed. 3) Wi-Fi Routers. The routers are crucial, acting as the central hub that links the laptop, Raspberry Pi, and various smart devices like smart plugs, light bulbs, and cameras, ensuring a reliable internet connection and seamless data flow across the network. 4) Smart Devices (Smart Plug and Associated Devices like Light Bulb/Camera). These smart devices are managed wirelessly through commands from the Raspberry Pi, enabling sophisticated automation tasks such as power toggling and scheduling.

Edge Devices

We utilize Raspberry Pi 5 acting as our edge device in the system, which offers a 2-3x improvement in CPU performance over its predecessor, Raspberry Pi 4. Beneficial from its advantages in low cost, modularity, open design, and compatibility with HDMI and USB standards, it has a wide range of applications in robotics, home automation,

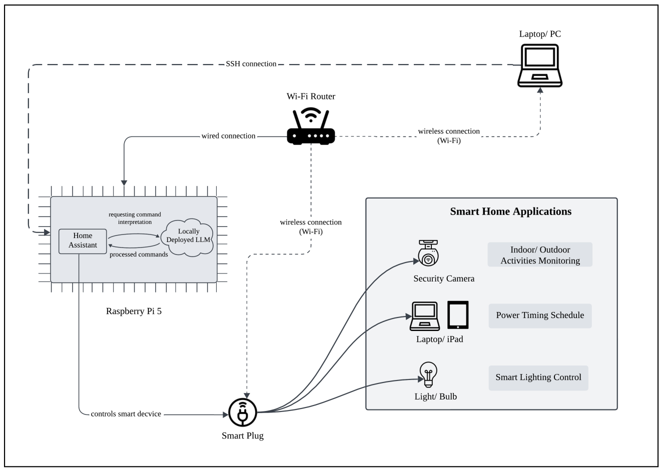


Figure 1: The System Architecture of Our LLM-Assisted Smart Home System

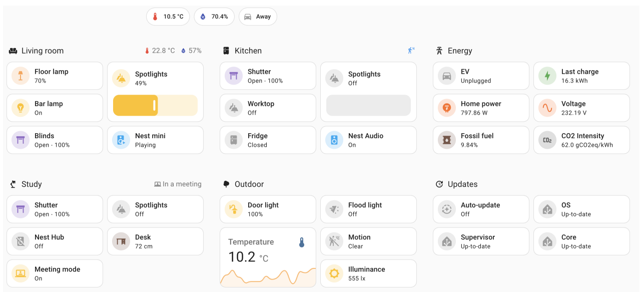
industrial automation, etc. The Raspberry Pi 5 is powered by a quad-core ARM Cortex-A76 processor, operating at 2.4 GHz, and equipped with 4 GB of LPDDR4X RAM. For graphics, it features the advanced VideoCore VII GPU, which supports dual 4K displays at 60Hz, enhancing its utility in high-definition applications. In terms of connectivity, it includes two USB 3.0 ports, two USB 2.0 ports, Gigabit Ethernet, Wi-Fi 5 (802.11ac), and Bluetooth 5.0. Storage solutions include a MicroSD card slot and a PCIe 2.0 x1 interface for external SSDs. The device operates with a USB-C power input that requires a 5V, 5A power supply, ensuring efficient energy usage while maintaining high performance.

The Home Assistant Platform

In our testbed, we utilize the Home Assistant[1] platform to enhance development efficiency and ensure seamless integration of various devices. It is a free, open-source software platform designed to manage smart home devices through an intuitive user interface. As an on-premise solution, Home Assistant offers significant security advantages by storing and processing data locally, without the need for external servers or cloud services. It has the following key features:

- *Broad Compatibility and Flexibility of Hardware Platforms.* Home Assistant supports a diverse range of hardware and operating systems, from single-board computers like Hardkernel ODROID, Raspberry Pi, Asus Tinkerboard, Intel NUC systems, etc., to traditional computing platforms, including Windows, macOS, and Linux, and can be installed on both virtual and physical storage systems.
- *Customized Control Dashboard.* Home Assistant features the Lovelace dashboard as shown in Fig. 2, which provides real-time monitoring and control of devices with a fully customizable interface through YAML code for tailored user experiences.

[1]https://github.com/home-assistant/core


10.5 °C
70.4%
Away

Living room 22.8 °C 57%
Floor lamp 70%
Spotlights 49%
Bar lamp On
Blinds Open · 100%
Nest mini Playing

Kitchen
Shutter Open · 100%
Spotlights Off
Worktop Off
Fridge Closed
Nest Audio On

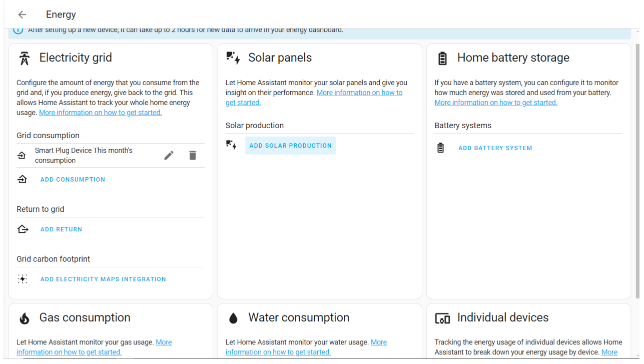
Energy
EV Unplugged
Last charge 16.3 kWh
Home power 797.86 W
Voltage 232.19 V
Fossil fuel 9.84%
CO2 Intensity 62.0 gCO2eq/kWh

Study In a meeting
Shutter Open · 100%
Spotlights Off
Nest Hub Off
Desk 72 cm
Meeting mode On

Outdoor
Door light 100%
Flood light Off
Temperature 10.2 °C
Motion Clear
Illuminance 555 lx

Updates
Auto-update Off
OS Up-to-date
Supervisor Up-to-date
Core Up-to-date

Figure 2: Home Assistant's Lovelace Dashboard


Energy

After setting up a new device, it can take up to 2 hours for new data to arrive in your energy dashboard.

Electricity grid

Configure the amount of energy that you consume from the grid and, if you produce energy, give back to the grid. This allows Home Assistant to track your whole home energy usage. More information on how to get started.

Grid consumption
Smart Plug Device This month's consumption
ADD CONSUMPTION

Return to grid
ADD RETURN

Grid carbon footprint
ADD ELECTRICITY MAPS INTEGRATION

Solar panels

Let Home Assistant monitor your solar panels and give you insight on their performance. More information on how to get started.

Solar production
ADD SOLAR PRODUCTION

Home battery storage

If you have a battery system, you can configure it to monitor how much energy was stored and used from your battery. More information on how to get started.

Battery systems
ADD BATTERY SYSTEM

Gas consumption

Let Home Assistant monitor your gas usage. More information on how to get started.

Water consumption

Let Home Assistant monitor your water usage. More information on how to get started.

Individual devices

Tracking the energy usage of individual devices allows Home Assistant to break down your energy usage by device. More

Figure 3: Home Assistant's Energy Dashboard

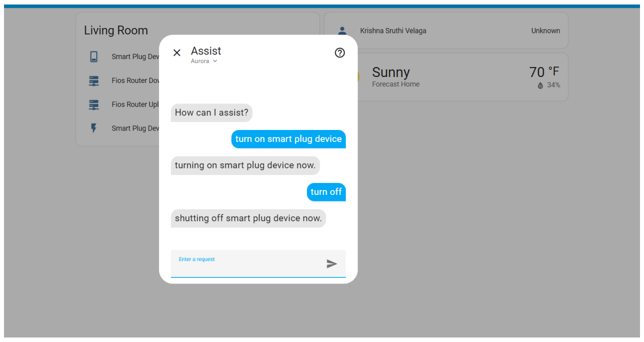
- *Integrated Automation and Monitoring System.* Home Assistant merges various devices and services into a unified system with a rule-based automation engine, enabling the creation of custom routines for efficient home management. It allows users to create custom routines based on specific triggers, conditions, and actions. Use cases include automating lighting, managing security alarms and video surveillance, and monitoring energy.

Home Assistant User Interfaces

Home Assistant inherently provides user interfaces (UIs), add-ons, and statistical dashboards for configured devices, such as smart plugs and the Fios router, to track energy consumption and daily activities, as well as more personalized automation options. It displays detailed information about each device, allowing users to check their status quickly.

Energy Monitoring and Management UI. Energy management is an important feature of Home Assistant, which offers a dashboard that tracks energy usage statistics for configured devices as illustrated in Fig. 3. By displaying real-time and historical energy data, the system gives insights into how much energy each device consumes, along with grid usage. This allows users to manage and optimize energy efficiency across their homes, helping them make informed decisions about device usage.

Voice Assistant UI. One practical use in this tested is the integration of the voice assistant UI with a locally deployed model. For example, we created a custom voice assistant named "Aurora", shown in Fig.4, which was connected to the local LLM and had access to various device entities in



How can I assist?

turn on smart plug device

turning on smart plug device now.

turn off

shutting off smart plug device now.

Figure 4: An Illustration of Interactions with Voice Assistant “Aurora” based on Locally Deployed LLMs

the system. This allowed Aurora to interact intelligently with the smart home, providing natural language control over devices. It enables precise actions and responses based on local models as a “personal assistant” without relying on cloud services. Our goal is to enable our conversation agent, Aurora, to understand user commands correctly and process commands locally with relatively low latency, allowing for real-time, natural language interactions within the system.

The Logbook feature. This feature is designed to track all activities and changes within the system after it has been activated. It records interactions with smart devices and the commands executed by the deployed language model, as well as a detailed history of automation activities and system modifications. For example, the logbook could record multiple instances of a smart plug[2] controlled device being turned on and off, as well as changes made to the LLM model initiated by the user, and provide a detailed history of automation activities and system modifications, which is essential for monitoring and troubleshooting the smart home system.

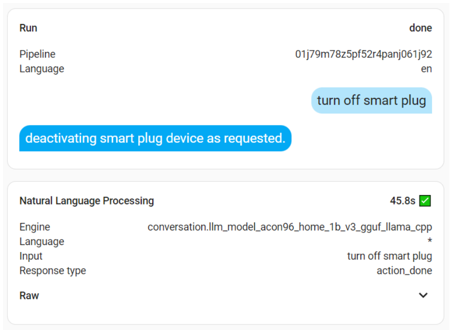
LLM Deployment on the Home Assistant Platform

We initially integrate the pre-trained `Home-1B-v3-GGUF`[3] model into our Home Assistant platform using the Llama.cpp[4] backend, designed to function optimally on resource-constrained devices without dedicated GPUs, such as a Raspberry Pi. This model is also optimized for devices with limited memory, such as those with under 4GB of RAM. The Home Assistant has provided the `Local LLM Conversation` functions (under the conditioning `Devices and Services` section) to run models locally using Llama.cpp as part of the system. It also provides the quantization level setting, which provides an ideal balance between performance and compatibility, allowing the model to use system resources efficiently. In our practices, we also find that to improve response speed, we should consider limiting the number of entities that the model processes, as this can help it run more smoothly.

[2]We utilize the Kasa KP125M Smart Wi-Fi Plug as our smart control device, which is seamlessly integrated with Home Assistant API in energy monitoring and voice control.

[3]https://huggingface.co/acon96/Home-1B-v3-GGUF

[4]https://github.com/ggerganov/llama.cpp



Run done

Pipeline 01j79m78z5pf52r4panj061j92
Language en

turn off smart plug

deactivating smart plug device as requested.

Natural Language Processing 45.8s ✓

Engine conversation.llm_model_acon96_home_1b_v3_gguf_llama_cpp
Language *
Input turn off smart plug
Response type action_done

Raw

Figure 5: High Latency in Responses

Limitations of Current System Designs

Based on the results from our implemented testbed, we have identified two significant challenges that impede performance during on-device deployment: non-precise responses and high latency. Firstly, the responses from the pre-trained model tend to be generic and lack the ability to adapt to individual user behavior patterns, limiting their effectiveness in personalized scenarios. Secondly, the response time is considerably extended, primarily due to the time consumed by model inference on the device and device searching, which affects the overall user experience and system efficiency. For instance, as demonstrated in Fig. 5, processing a straightforward command like “turn off smart plug” takes approximately 45 seconds. This significant delay underscores the critical need to optimize local LLM inference on power-constrained edge devices.

System Optimization

To address these two issues, we implement a two-fold optimization strategy for our system. First, we enhance the model by applying post-training techniques, such as prompt tuning and LoRA, using synthetic smart home datasets tailored to smart home environments. This approach aims to reduce inference latency and improve response performance, ensuring that interactions are tailored to the preferences of each individual customer. Additionally, we optimize the search engine settings in querying configured devices, which significantly accelerates the response speed.

Synthetic Smart Home Dataset Generation

The dataset generation process for a smart home assistant involves compiling diverse interaction sets that accurately mirror real-world user requests and the corresponding responses of the assistant within a smart home setting[5]. These interactions are crucial for training the assistant to accurately interpret and respond to user commands, address a variety of queries, and effectively manage edge cases. The dataset is structured in English and adopts a medium-sized format in the “ShareGPT” configuration, comprising 34,250 training examples and 2,532 test examples. This comprehensive dataset is essential for enhancing the assistant’s performance in realistic scenarios.

[5]https://github.com/acon96/home-llm/tree/develop/data

The generation process begins by loading essential data components, known as "piles", which supply the contextual data necessary for generating a diverse range of smart home interactions. These piles include: a catalog of device names organized by types, such as lights, thermostats, and media players; templated actions that use placeholders like `<device_name>` and `<temperature>` to simulate commands for various devices; specific actions that are direct and straightforward; standard responses and common queries about device status; and other components like lists of conjunctions, media titles, durations, and to-do items. These components are controlled by the language parameter to ensure the dataset aligns with the linguistic and contextual requirements of the target examples. It also offers five different languages to generate the system prompts, including English, German, French, Spanish, and Polish. In this case, English is our configured language.

Each generated prompt encompasses three primary components designed to train the smart home assistant effectively: 1) *Static Actions.* They consist of direct commands such as "Turn on the living room light," which require the assistant to respond directly, covering basic commands for straightforward interactions. 1) *Templated Actions.* They feature dynamic placeholders that fill in specifics like device names, states, and values, for example, transforming the template "Set the thermostat to `<temperature>`" into "Set the thermostat to 72 degrees." 3) *Status Requests*. They involve inquiries about device conditions, such as "Is the front door locked?" teaching the assistant to accurately handle and respond to specific status queries. Additionally, the dataset incorporates Direct Preference Optimization (DPO) examples that simulate incorrect responses, including mistakes like wrong arguments or unnecessary service calls, which are crucial for training the assistant to recognize and correct common errors and to ignore irrelevant commands.

In terms of presentation style, each prompt is formatted in a natural, conversational style using the ShareGPT conversational format to mimic realistic user-assistant interactions. When the system role is enabled, the conversation begins with a system prompt listing available devices (e.g., "Devices: light, thermostat") and services (e.g., "Services: turn on, adjust temperature"). Each example consists of a user command (e.g., "Turn on the living room light") followed by the assistant's response (e.g., "Turn on the living room light now"). If the response includes a specific action, a JSON-style code block is appended, detailing the service call (e.g., action turn_on, target device living room light, and necessary parameters).

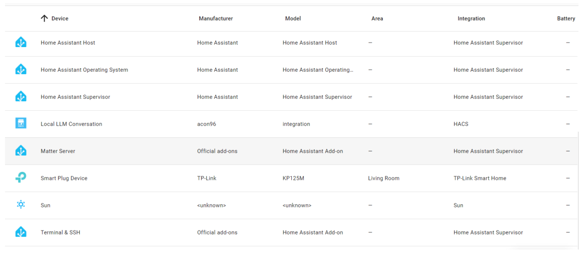
We showcase a sample system prompt generated below. In this example, the AI assistant "Al" is tasked with controlling smart home devices through a natural language interface. The conversation begins with a prompt showing the assistant's name, purpose, current date and time, and a list of available services, including commands like `cover.close_cover` and `light.turn_on` for various device types. Devices are listed along with their current states, such as: `light.front_lounge_ge 'Front Lounge Light' = on` providing an overview of each device's status in the home environment. When the user asks "Al" to "lower the kitchen blinds," the assistant correctly interprets this as a request to close the kitchen blinds (`cover.kitchen`). It uses the appropriate `cover.close_cover` service to perform the action and responds with the confirmation *"closing the blinds as you requested."* This confirmation ensures that the user knows the request was understood and completed. Similarly, When the user asks AI to "increase the fan speed in the guest room," the assistant interprets this as a request to raise the speed of the fan in the guest room (`fan.guest_room`). It uses the appropriate `fan.increase_speed` service to execute the action and responds with the confirmation *"increasing speed of Guest Room."* This confirmation reassures the user that the request was understood and fulfilled.

Example System Prompt

You are `'Al'`, a helpful AI Assistant that controls the devices in a house. Complete the following task as instructed or answer the following question with the information provided only.
The current time and date is 7:13 AM on Monday, October 14, 2024.
Services: cover.close_cover, cover.open_cover, cover.stop_cover, cover.toggle, fan.decrease_speed, fan.increase_speed, light.turn_on, light.turn_off.
Devices: cover.close_cover = on; cover.kitchen = on; fan.guest_room 'Guest Room Fan' = on; light.front_lounge_ge 'Front Lounge Light' = on.
User: Lower the kitchen blinds.
Assistant: Closing the blinds as you requested.
Action:
```
homeassistant:
{
  "service": "cover.close_cover",
  "target_device": "cover.kitchen"
}
```
User: Increase the fan speed in the guest room.
Assistant: Increasing speed of Guest Room.
Action:
```
homeassistant:
{
  "service": "fan.increase_speed",
  "target_device": "fan.guest_room"
}
```

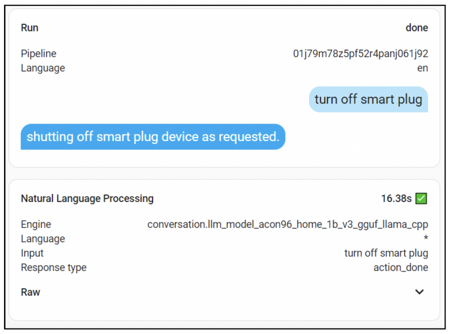
Model Fine-Tuning

To locally fine-tune the models, we consider both prompt tuning (Lester, Al-Rfou, and Constant 2021) and LoRA (Low-Rank Adaptation) (Hu et al. 2021) techniques based on our generated prompts from the synthetic smart home dataset. Meanwhile, to better fit into edge environments, we also involve 4-bit and 8-bit model quantization in our model re-training process. We finetune the models using generated training sample prompts and save the model's state (checkpoint) with the highest evaluation performance on testing prompts. Particularly, in LoRA, we consider low-rank matrix adaption to be at a rank of 64 with a scaling factor of 128 for low-rank updates (as the adapter's contribution).



	↑ Device	Manufacturer	Model	Area	Integration	Battery
	Home Assistant Host	Home Assistant	Home Assistant Host	–	Home Assistant Supervisor	–
	Home Assistant Operating System	Home Assistant	Home Assistant Operating...	–	Home Assistant Supervisor	–
	Home Assistant Supervisor	Home Assistant	Home Assistant Supervisor	–	Home Assistant Supervisor	–
	Local LLM Conversation	acon96	integration	–	HACS	–
	Matter Server	Official add-ons	Home Assistant Add-on	–	Home Assistant Supervisor	–
	Smart Plug Device	TP-Link	KP125M	Living Room	TP-Link Smart Home	–
	Sun	<unknown>	<unknown>	–	Sun	–
	Terminal & SSH	Official add-ons	Home Assistant Add-on	–	Home Assistant Supervisor	–

Figure 6: Optimized Configured Devices in the Search Engine



Run done

Pipeline 01j79m78z5pf52r4panj061j92
Language en

turn off smart plug

shutting off smart plug device as requested.

Natural Language Processing 16.38s ✓

Engine conversation.llm_model_acon96_home_1b_v3_gguf_llama_cpp
Language *
Input turn off smart plug
Response type action_done

Raw

Figure 7: Reduced Latency after Search Engine Optimization

Search Engine Optimization

Given the large number of smart devices and associated APIs managed by the search engine in Home Assistant, excessive time spent searching for non-task-related devices can significantly impact overall response times. To address this, a task-oriented search engine configuration should be prioritized. We find that by narrowing the scope of entities exposed by the search engine to only the pre-configured devices necessary for core voice assistant functionalities, such as those illustrated in Fig. 6, response delays can be substantially reduced. For example, initially, when we exposed around 20 entities, it resulted in a 45-second delay, as in Fig. 5. By reducing the number of exposed entities to 8 (which are the core task-related functions) in Fig. 6, the response time for tasks such as turning on a smart plug is down to just 16 seconds as in Fig. 7. This approach demonstrates the efficiency gains through task-oriented and optimized search engine configurations.

Performance Evaluation

Model and Training Environment Setup

Besides the pre-trained Home-1B model serves as our baseline, we consider three other models, e.g., TinyHome, TinyHome-Qwen, and StableHome, for model fine-tuning and performance evaluation. Particularly, for TinyHome-Qwen and StableHome, they also adopt LoRA techniques for performance comparisons. All models are locally retrained on a workstation equipped with two Nvidea RTX 3090 GPUs. The initial learning rate is set as 2×10^{-5} for Home-1B, TinyHome, and TinyHome-Qwen and 10^{-5} for StableHome, respectively, followed by learning rate decay. The batch size is 128. For Home-1B, TinyHome, and TinyHome-Qwen, we use 4-bit quantization. For StableHome, we use 8-bit quantization.

For evaluation metrics, we focus on two key metrics: on-device response latency and precision. Latency is measured using the built-in functionality of the home Assistant system, including its logbook. Precision is assessed through a qualitative evaluation of the feedback by people, focusing on clarity and level of detail.

Evaluation Results

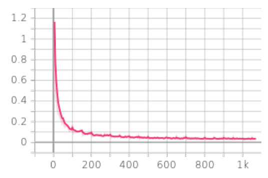
Learning Convergence. Fig. 8 shows the learning convergence curves for three different models, e.g., tinyhome, tinyhome-qwen, and stablehome, arranged from the first to the third row, respectively. Each row illustrates the model's training and evaluation dynamics through four key metrics: training loss, training gradient norm, training learning rate, and evaluation loss. In each plot, we observe the steady decreases in magnitudes for each metric, which showcases the learning convergence for each training and reflects our fine-tuned models' generalization capabilities with both decreased training and evaluation losses.

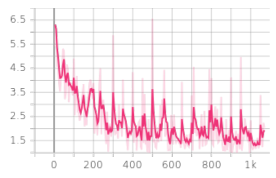
Reduced Latency and Enhanced Response Precision for On-device Inferences. Fig. 9 demonstrates the impact of model fine-tuning and search engine optimization on reducing latency and improving response precision in Home Assistant based on the Home-1B model. For Prompt 1, the system completes the task of turning on a smart plug in 7.68 seconds. In Prompt 2, it retrieves temperature data ("44 degrees Celsius") in 4.01 seconds, and for Prompt 3, it provides weather information ("partly cloudy") in 12.44 seconds. These examples highlight efficient task execution and data retrieval, showcasing the enhanced performance achieved through optimized natural language processing and search configurations, compared with the benchmark setting with an average of 45.1 seconds delay. We also evaluate the performance of two customized models, e.g., TinyHome and TinyHome-Qwen. TinyHome-Qwen (with both prompt tuning and LoRA) emerged as the most efficient model, achieving an average response time of 4.55 seconds, which outperforms Home-1B in 8.04 seconds and TinyHome in 11.18 seconds. In terms of response precision, we also discover that the responses from the fine-tuned model cover more details related to each individual user compared with those from the pre-trained model.

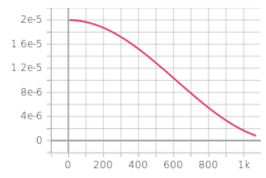
Energy Consumption. The platform also provides built-in visualization tools for tracking and managing the energy usage of the smart plug device over time. As indicated in Fig. 10, the device showed a peak consumption of approximately 8 watts at around 11:00 AM, followed by a sharp decline to 0 watts shortly after. We aim to involve the energy information as parts of prompts for fine-tuning in our future study to better learn customized and personalized models.

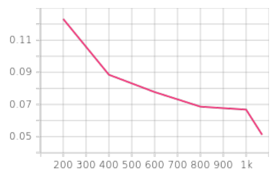
Final Remarks

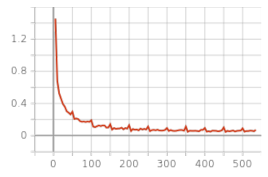
In this paper, we present a pilot study to evaluate the real-time performance of the edge deployment of LLMs on

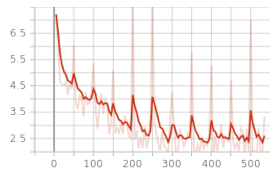

(a) Training Loss

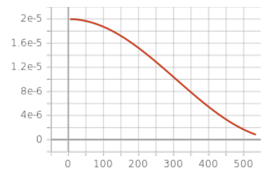

(b) Training Gradient Norm

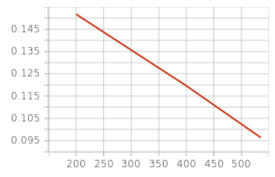

(c) Training Learning Rate

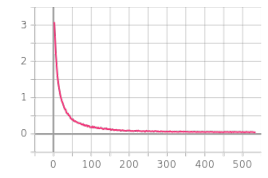

(d) Evaluation Loss

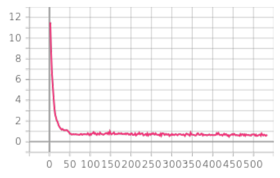

(e) Training Loss

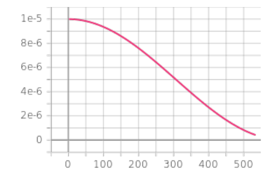

(f) Training Gradient Norm


(g) Training Learning Rate


(h) Evaluation Loss


(i) Training Loss


(j) Training Gradient Norm


(k) Training Learning Rate

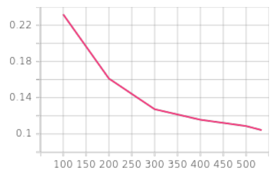
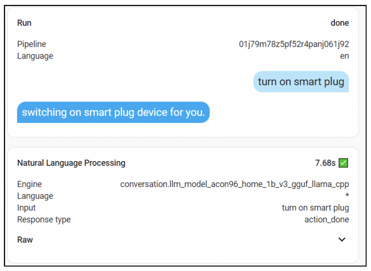

(l) Evaluation Loss

Figure 8: Learning Convergence Curves for Different Models (Results from models tinyhome, tinyhome-qwen, and stablehome are indicated from the first to the third row, respectively)

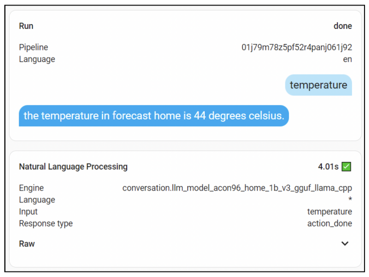

Run done
Pipeline 01j79m78z5pf52r4panj061j92
Language en

turn on smart plug

switching on smart plug device for you.

Natural Language Processing 7.68s ✓
Engine conversation.llm_model_acon96_home_1b_v3_gguf_llama_cpp
Language *
Input turn on smart plug
Response type action_done
Raw

(a) Prompt 1

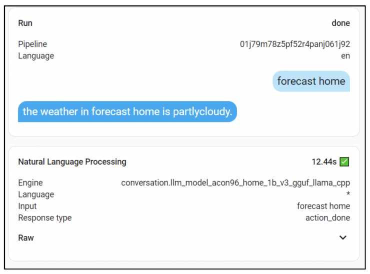

Run done
Pipeline 01j79m78z5pf52r4panj061j92
Language en

temperature

the temperature in forecast home is 44 degrees celsius.

Natural Language Processing 4.01s ✓
Engine conversation.llm_model_acon96_home_1b_v3_gguf_llama_cpp
Language *
Input temperature
Response type action_done
Raw

(b) Prompt 2


Run done
Pipeline 01j79m78z5pf52r4panj061j92
Language en

forecast home

the weather in forecast home is partlycloudy.

Natural Language Processing 12.44s ✓
Engine conversation.llm_model_acon96_home_1b_v3_gguf_llama_cpp
Language *
Input forecast home
Response type action_done
Raw

(c) Prompt 3

Figure 9: Reduced Latency and Enhanced Response Precision with Model Fine-Tuning and Search Engine Optimization

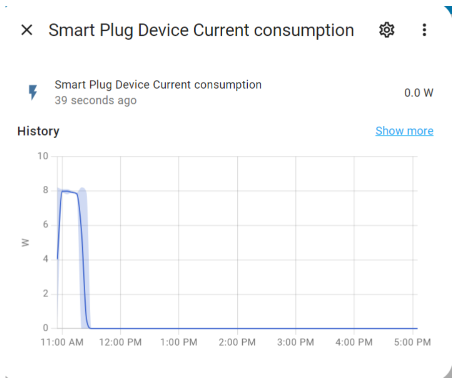


Figure 10: Energy Consumption Measurements

practical resource-constrained devices. Particularly, we develop a testbed with an LLM-enabled Home Assistant platform on Raspberry Pi 5 to evaluate the on-device and real-time response precision and delay empirically. Based on our observed issues in non-precise responses and high latency in benchmark system settings, we implement a two-fold optimization strategy for our system. We finetune the model with post-training techniques like prompt tuning and LoRA on synthetic smart home datasets to reduce inference latency with improved response performance tailored for smart home environments. Also, we optimize the search engine of configured devices, which significantly shortens the response time. By evaluating the performance of four lightweight models on the edge device, we find that our optimized models maintain high accuracy in understanding and executing user commands. More importantly, with optimizations, we reduce the response time by around 82%, from originally 45.1 seconds to 7.9 seconds, on average.

References

Civitarese, G.; Fiori, M.; Choudhary, P.; and Bettini, C. 2024. Large Language Models are Zero-Shot Recognizers for Activities of Daily Living. *arXiv preprint arXiv:2407.01238*.

Cleland, I.; Nugent, L.; Cruciani, F.; and Nugent, C. 2024. Leveraging Large Language Models for Activity Recognition in Smart Environments. In *2024 International Conference on Activity and Behavior Computing (ABC)*, 1–8.

Fan, Y.; Nie, J.; Sun, X.; and Jiang, X. 2024. Exploring Foundation Models in Detecting Concerning Daily Functioning in Psychotherapeutic Context Based on Images from Smart Home Devices. In *2024 IEEE International Workshop on Foundation Models for Cyber-Physical Systems Internet of Things (FMSys)*, 44–49.

Gao, L.; Ziashahabi, A.; Niu, Y.; Avestimehr, S.; and Annavaram, M. 2024. Enabling Resource-Efficient On-Device Fine-Tuning of LLMs Using Only Inference Engines. arXiv:2409.15520.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

King, E.; Yu, H.; Lee, S.; and Julien, C. 2024. 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): 1–38.

Lester, B.; Al-Rfou, R.; and Constant, N. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Li, Q.; Xie, Y.; Du, T.; Shen, Z.; Qin, Z.; Peng, H.; Zhao, X.; Zhu, X.; Yin, J.; and Zhang, X. 2024. CoreGuard: Safeguarding Foundational Capabilities of LLMs Against Model Stealing in Edge Deployment. arXiv:2410.13903.

Paul, S.; Zhang, L.; Shen, Y.; and Jin, H. 2024. Enabling Device Control Planning Capabilities of Small Language Model. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 12066–12070.

Rey-Jouanchicot, J.; Bottaro, A.; Campo, E.; Bouraoui, J.-L.; Vigouroux, N.; and Vella, F. 2024. Leveraging Large Language Models for enhanced personalised user experience in Smart Homes. arXiv:2407.12024.

Shi, Y.; Liu, X.; Yu, C.; Yang, T.; Gao, C.; Liang, C.; and Shi, Y. 2024. Bridging the gap between natural user expression with complex automation programming in smart homes. arXiv:2408.12687.

Takeda, N.; Legaspi, R.; Nishimura, Y.; Ikeda, K.; Plötz, T.; and Chernova, S. 2024. A Synergistic Large Language Model and Supervised Learning Approach to Zero-Shot and Continual Activity Recognition in Smart Homes. In *2024 9th International Conference on Big Data Analytics (ICBDA)*, 113–122.

Ur Rahman, M. W.; Abrar, M. M.; Copening, H. G.; Hariri, S.; Shao, S.; Satam, P.; and Salehi, S. 2023. Quantized Transformer Language Model Implementations on Edge Devices. In *2023 International Conference on Machine Learning and Applications (ICMLA)*, 709–716.

Woszczyk, D.; Lee, A.; and Demetriou, S. 2021. Open, Sesame!: Introducing Access Control to Voice Services. In *Proceedings of the 1st Workshop on Security and Privacy for Mobile AI*, MobiSys '21, 7–12. ACM.

Yonekura, H.; Tanaka, F.; Mizumoto, T.; and Yamaguchi, H. 2024. Generating Human Daily Activities with LLM for Smart Home Simulator Agents. In *2024 International Conference on Intelligent Environments (IE)*, 93–96.

Zeng, L.; Ye, S.; Chen, X.; and Yang, Y. 2024a. Implementation of Big AI Models for Wireless Networks with Collaborative Edge Computing. arXiv:2404.17766.

Zeng, X.; Wang, X.; Zhang, T.; Yu, C.; Zhao, S.; and Chen, Y. 2024b. GestureGPT: Toward Zero-shot Interactive Gesture Understanding and Grounding with Large Language Model Agents. arXiv:2310.12821.