JURNAL ELTEK
Vol. 24, No. 2, October 2026, pp. 131~144
p-ISSN: 1693 – 4024 | e-ISSN: 2355-0740
DOI: 10.33795/eltek.v24i2.11288

Using edge computing and Large Language Models (LLM) for constructing a local agent

Setya Widyawan Prakosa[1], Naufal Ainur Rizal[2] , Tias Kurniati[3]
e-mail: setyawp@unej.ac.id, naufalrizal47@gmail.com, tias@cycu.org.tw
[1,2]Department of Electrical Engineering, Universitas Jember, Indonesia,
[3]Department of Information Management, Chung Yuan Christian University

Article Information

Article History

Received August 23, 2026
Revised October 23, 2026
Published October 30, 2026

***Keywords*:**

Edge Computing
Large Language Model
Quantization

ABSTRACT

The growing demand for artificial intelligence systems that are responsive, privacy-preserving, and capable of operating autonomously is driving the adoption of Large Language Models (LLMs) on edge devices. However, deploying LLMs as the core of agents on devices with limited resources still faces computational and memory constraints. This research develops an LLM-based local agent architecture that runs locally on a Raspberry Pi without relying on cloud services. The system integrates local LLM inference with agentic functions, including intent processing, reasoning, tool invocation, and interaction with hardware via GPIO. Several quantized Small Language Models (SLMs) were evaluated on a Raspberry Pi 5 with 16 GB of RAM using CPU-based inference. The evaluation included total latency, Time to First Token (TTFT), token generation rate, peak RAM usage, peak CPU utilization, and peak temperature. The results showed significant performance differences among the models. Qwen 2.5 0.5B Q3 produced the lowest total latency of 23.9 seconds, the lowest TTFT of 269.0 ms, and the highest generation rate of 12.1 tokens/s. However, the results indicate that inference speed alone is not sufficient to determine the most suitable model as the core of a local agent. Most models reached peak CPU utilization above 95%, while RAM usage and thermal characteristics varied across configurations. Based on a combination of inference performance and resource usage, models in the 1B–1.7B range specifically LLaMA 3.2 1B Q4, Qwen 2.5 1.5B Q4, and SmolLM 1.7B Q4 show potential as a balanced set of candidates between responsiveness and model capacity. The research results demonstrate the feasibility of deploying LLMs as local agents on low-cost edge devices and provide a basis for selecting models based on a balance between inference efficiency, resource constraints, and agentic capabilities.

Corresponding Author:
Setya Widyawan Prakosa,
Department of Electrical Engineering,
University of Jember,
Jl. Kalimantan No.37, Krajan Timur, Sumbersari, Kec. Sumbersari, Kabupaten Jember, Jawa Timur 68121, Indonesia
Email: setyawp@unej.ac.id

1. INTRODUCTION

The paradigm of Artificial Intelligence (AI) is undergoing a fundamental shift from centralized cloud-based infrastructures to decentralized Edge Computing. While cloud-centric architectures offer vast computational power, they introduce inherent challenges, including communication latency, reliance on

continuous network connectivity, bandwidth costs, and growing data privacy concerns. Processing data locally at the edge mitigates these vulnerabilities, making edge computing essential for real-time and privacy-sensitive applications [1], [2], [3].

Concurrently, the advancement of Large Language Models (LLMs) has catalyzed the transition from passive AI inference to Agentic AI. Agentic AI is an advanced form of artificial intelligence that possesses the autonomy to achieve a goal through reasoning, planning, decision-making, the use of external tools and resources, and the iterative evaluation of the results of its actions [4], [5], [6], [7].

Unlike traditional edge models limited to static classification or object detection, agentic systems possess high-order cognitive capabilities, including multi-step reasoning, dynamic task decomposition, long-term memory retention, and autonomous tool utilization. Operating within an iterative loop—perceiving the environment, planning actions, invoking local tools, and evaluating outcomes—Agentic AI transforms passive sensor nodes into active, autonomous decision-makers.

Deploying Agentic AI on low-cost, resource-constrained hardware such as the Raspberry Pi presents significant engineering challenges. Running parameter-heavy LLMs demands substantial memory bandwidth and processing capacity, which typically exceed the hardware boundaries of single-board computers (SBCs). However, recent breakthroughs in model quantization (e.g., 4-bit GGUF format), targeted matrix multiplication optimization, and the rise of high-efficiency Small Language Models (SLMs) have made on-device agentic execution increasingly viable [8], [9]

Several studies have explored the application of LLMs on edge devices; for example, Matsunani et al. evaluated local LLM inference on a Raspberry Pi Zero 2W and identified performance limitations due to limited computational resources. To address these issues, this study proposes a distributed prompt caching mechanism that enables the sharing of intermediate processing states among edge devices to reduce inference time [9].

In a broader exploration, Nguyen evaluated several local LLMs on 3 Raspberry Pi 4s, a Raspberry Pi 5, and an Orange Pi 5 Pro [10]. The study benchmarked 25 quantized LLMs using two inference runtimes Ollama and Llamafile and evaluated generation throughput, memory usage, and power consumption across various CPU configurations. In an industrial setting, Castelo et al. deployed a quantized Mistral-7B model on a Raspberry Pi 5 as part of a cognitive architecture integrated with various industrial protocols, such as OPC UA, MQTT, and REST API. The system demonstrated that on-premises LLMs can be used to provide access to operational information without relying on cloud services, while also reducing information access time compared to manual methods [11]. Another evaluation that focused more on model efficiency was conducted by Husom et al., who analyzed 28 quantized LLMs on a Raspberry Pi 4, taking into account energy consumption, output accuracy, and inference latency. The study showed that the quantization level presents a trade-off between energy efficiency, inference speed, and output quality, with some quantization configurations capable of reducing energy consumption by up to 79% and inference latency by up to 69% compared to the FP16 configuration [12]. Other studies have also explored the implementation of on-device LLMs on the Raspberry Pi for real-time interaction systems. Ardakani and Malekar, through PiCASO, developed a modular conversational AI system that integrates speech-to-text, quantized LLMs, and text-to-speech on a Raspberry Pi 5. The study evaluated several quantization levels to achieve a balance between accuracy and throughput, and demonstrated that compressed LLM models can support real-time inference on edge devices with limited resources [13].

However, these studies generally still focus on the inference and efficiency of LLMs on edge devices, while the local implementation of agentic AI remains relatively limited. Zappia et al. developed a distributed agentic AI architecture on edge devices for home surveillance, utilizing LLMs for tool-calling and distributing analysis tasks across multiple devices. The system also employs leader election to enhance resilience against device failures, demonstrating the potential for local deployment of LLM-based agents in edge environments [14]. Furthermore, Rüb and Gerhards proposed a modular architecture for embedded agent systems that differentiates agent deployment strategies based on hardware capabilities, ranging from autonomous on-device agents on edge devices capable of running SLM to cloud-connected agents on devices with limited resources. The research emphasizes local reasoning, deployment flexibility, and the trade-offs between latency, energy, privacy, and reasoning capabilities [15].

Despite these developments, existing studies have primarily addressed LLM inference efficiency or specific edge-agent architectures, while empirical evidence of integrating conversational interaction, persistent memory, and physical hardware control within a single, fully local edge device remains limited. Furthermore, the computational requirements of different quantized language models must be considered when deploying such systems on resource-constrained hardware. Therefore, this study investigates the feasibility of implementing a local LLM-based agent on a Raspberry Pi 5 and evaluates the inference performance and resource utilization of different quantized model configurations.

This study makes two primary contributions. First, it presents and demonstrates a locally deployed LLM-based agent integrating conversational interaction, conversation memory management, and GPIO-based

hardware control without relying on cloud-based inference. Second, it provides a comparative evaluation of quantized language model configurations in terms of inference responsiveness and computational resource utilization on the Raspberry Pi 5. The evaluation focuses on functional system implementation and computational performance rather than a quantitative assessment of reasoning, planning, or tool-calling reliability.

2. RESEARCH METHOD

2.1 Proposed Local Agent Architecture

The proposed local agent architecture is illustrated in Figure 1. Users submit natural-language requests through a web-based interface, which communicates with a local backend implemented using Express.js. The backend processes incoming requests and directs them to the corresponding interaction path, including ordinary conversation, memory-related operations, or tool execution. For conversational requests, the locally hosted language model generates the response, which is returned to the user through the interface. This arrangement keeps inference and agent-side processing on the Raspberry Pi without requiring a cloud-based LLM service.

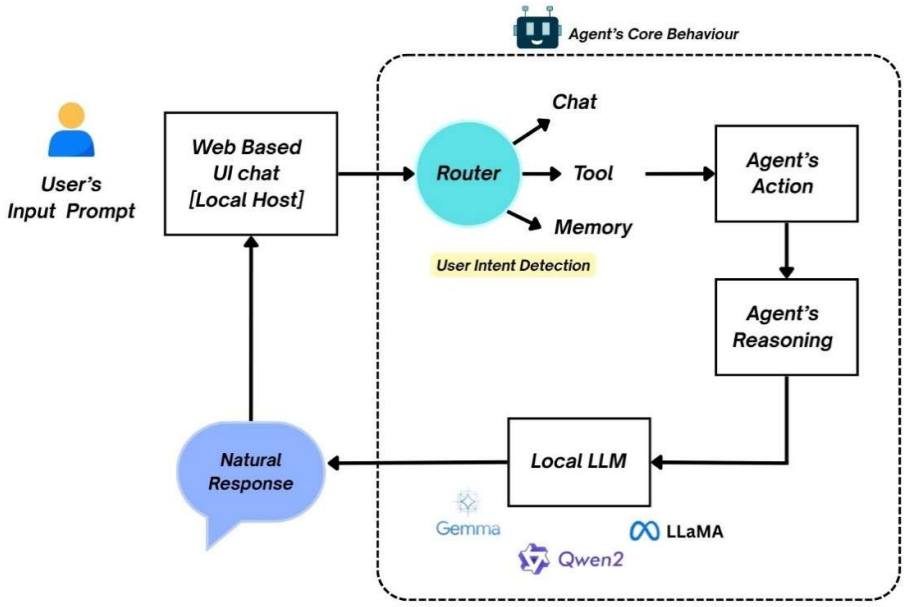


Figure 1. Proposed Local Agent Architecture

In addition to conversational interaction, the system supports persistent conversation memory, as illustrated in Figure 2. A conversation checkpoint is stored locally and contains the full interaction history, associated metadata, and a condensed memory capsule summarizing earlier exchanges. When a saved conversation is restored, relevant memory information can be reintroduced into the agent context to support continuity across sessions without requiring the entire conversation history to be included in every inference request.

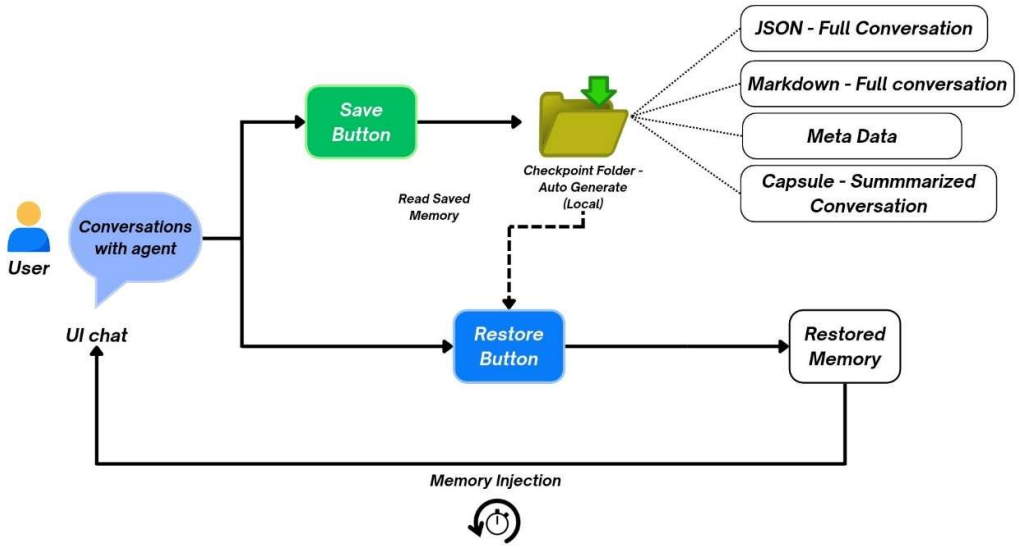


Figure 2. Conversation Memory Saving and Restoring Mechanism

The agent also supports local hardware interaction through GPIO control. Figure 3 illustrates the workflow for interpreting a natural-language hardware request and forwarding it to a GPIO command-processing component. The hardware-control pathway is separated from ordinary conversational responses so that a requested action can be translated into a structured operation for the Raspberry Pi.

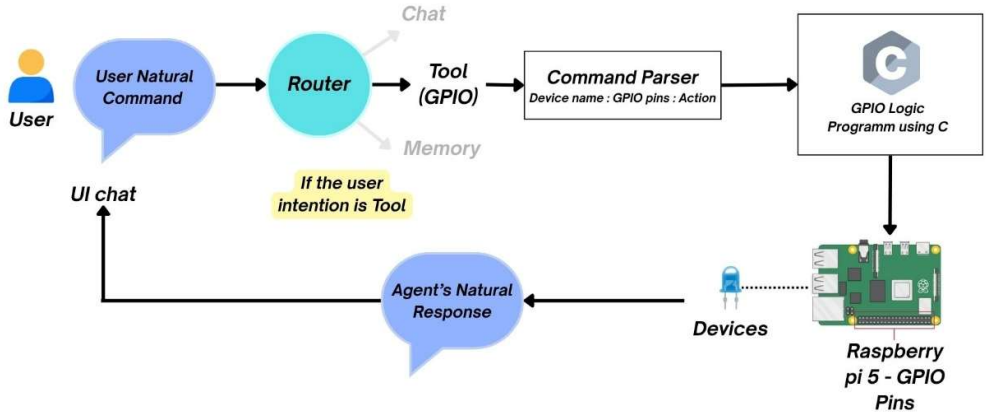


Figure 3. GPIO Command Processing and Hardware Control Workflow

For GPIO requests, the router directs the request to a command parser that identifies the intended device, GPIO pin, and requested state or action. The resulting command is passed to a local GPIO control program, which interacts with the attached hardware. The system then returns a user-facing response reflecting the processed action.

2.2 Experimental Environment and Configuration

The inference experiments were performed on a Raspberry Pi 5 equipped with 16 GB of RAM. The board uses a Broadcom BCM2712 quad-core 64-bit Arm Cortex-A76 processor operating at 2.4 GHz. The setup included a 512 GB NVMe SSD and active cooling, with the operating-system release identified in

the original setup notes as Trixie. LLM inference was performed on the CPU. The same edge device served as the deployment target for the local-agent demonstration. The principal hardware characteristics are summarized in Table 1.

Table 1. Experimental hardware and software configuration

Component	Specification
Edge device	Raspberry Pi 5
Processor	Broadcom BCM2712, quad-core Arm Cortex-A76, 2.4 GHz
Memory	16 GB LPDDR4X RAM
Storage	512 GB NVMe SSD
Cooling	Active cooling fan
Operating system	Trixie
Inference hardware	CPU-based execution
LLM Engine	llama.cpp

The compared model quantization configurations were selected from several locally executable small language model families with varying parameter sizes and architectures. The selection was primarily based on their compatibility with the Raspberry Pi 5 hardware constraints, particularly its limited computational resources and memory capacity. Models with different parameter scales were included to investigate how model size influences inference speed, responsiveness, and resource utilization in a CPU-based edge computing environment.

The Q3 and Q4 quantization levels were selected to reduce model memory requirements and computational overhead while maintaining the feasibility of local inference. Q3 provides greater compression and potentially lower memory consumption, whereas Q4 retains higher numerical precision at the cost of additional memory requirements. These configurations enable the evaluation of trade-offs between model size, quantization precision, inference performance, and computational resource utilization. The comparison was exploratory and focused on identifying computationally suitable model configurations for local agent deployment rather than exhaustively evaluating all available models and quantization levels.

2.3 Prompt Design and Experimental Procedure

The agent uses two complementary types of instructions: system prompts and user prompts. System prompts establish the agent's operating context and expected behavior, whereas user prompts specify the task requested during an interaction. The implementation distinguishes general conversational instructions from GPIO-specific instructions for hardware requests. Table 2 provides illustrative descriptions and paraphrased examples rather than exact reproductions of the deployed prompts.

The general system prompt context guides the agent to interpret natural-language requests and respond appropriately within the local application. When the request is routed to GPIO control, additional mode-specific instructions guide the interpretation of the target device, pin number, and requested action. The downstream command parser and local GPIO program perform the actual hardware operation, as described in Section 2.1, the language model's generated response alone is not evidence that a physical action was executed.

Table 2. System-prompt functions and illustrative examples

System-prompt context	Function	Illustrative prompt excerpt
General	Defines the agent's conversational role and task interpretation.	"You are a local assistant running on a Raspberry Pi. Interpret the user request and provide an appropriate response or select the supported system pathway."
GPIO-specific	Guides interpretation of hardware-control requests and structured handoff to the local GPIO executor.	"For a GPIO request, identify the target device, pin, and requested state. Provide the structured action to the GPIO command handler and report the execution result to the user."

For the experiment, user prompts were organized into three task-complexity categories: Simple, Medium, and Hard. In the examples reconstructed in Table 3, the Simple category requests a short direct

explanation; Medium requests a more developed but bounded response; and Hard requests a multi-step explanation of a forecasting workflow.

Table 3. Prompt-complexity categories with illustrative (reconstructed) examples

Prompt level	Definition	Illustrative user prompt
Simple	A short, direct request with a tightly constrained response length.	"Explain forecasting in one sentence."
Medium	A similar topic requiring a somewhat longer, explicitly bounded explanation.	"Explain forecasting in three sentences."
Hard	An open-ended procedural task likely to require multiple explanatory steps and a longer answer.	"Explain how to train a model for forecasting."

Each model quantization configuration was evaluated across all three prompt complexity levels, with three trials conducted per level, resulting in nine observations per model configuration. The nine observations were then averaged to obtain a single representative performance value for each model configuration.

The study comprises two complementary evaluation objectives:

- System Implementation Evaluation: Demonstrating the functionality of the implemented system, including conversational interactions, memory restoration and context injection, and GPIO-based hardware command execution.
- LLM Inference Performance Evaluation: Quantitatively evaluating LLM inference performance and Raspberry Pi resource utilization using the repeated benchmark measurements described above.

The first objective focuses on verifying the functional implementation of the proposed system through observed demonstrations, whereas the second objective focuses on comparing the computational performance and resource requirements of the evaluated model configurations.

2.4 Evaluation Metrics and Measurement Protocol

The quantitative evaluation covered inference responsiveness and computational resource utilization. The measurements were recorded separately for each trial and subsequently averaged to obtain a single representative value for each metric and model configuration, as described in Section 2.3. Table 4 defines the principal evaluation metrics and their interpretations. Resource measurements labeled as "peak" represent the highest recorded value within each individual trial, which was subsequently averaged across the trials and prompt categories.

Table 4. Definitions and interpretation of the reported evaluation metrics

Metric	Unit	Operational meaning / recorded quantity
Total latency	ms	Elapsed time from inference request to completion of the generated response
Time to First Token (TTFT)	ms	Elapsed time between inference request and the first generated output token.
Token generation rate	tokens/s	Per-trial generation throughput reported as token_per_sec. These recorded rates were averaged, not recomputed from pooled token counts.
Peak RAM usage	MB	Highest recorded RAM usage (MB) during an individual trial; reported values represent the average of trial-level RAM peaks across prompt categories.
Peak CPU utilization	%	Highest sampled CPU utilization during the trial charts report the mean of trial peaks.
Peak temperature	°C	Highest sampled device temperature per trial the plotted result averages these trial-level peaks.

3. RESULTS AND DISCUSSION

The experimental results are presented from three main perspectives. First, the implementation results are discussed to verify the functionality of the proposed local agent architecture, including intention routing, conversational interaction, GPIO-based tool execution, and conversation memory. Second, the inference performance of the evaluated LLMs is analyzed in terms of latency, time to first token (TTFT), and token generation rate. Third, the resource utilization of the Raspberry Pi is evaluated by measuring RAM consumption, CPU utilization, and peak temperature. These evaluations provide a comprehensive assessment of both the functional capability and the computational feasibility of deploying a local LLM-based agent on resource-constrained edge hardware.

3.1. System Implementation Result

This subsection describes observed functional demonstrations of the local-agent architecture deployed on the Raspberry Pi 5. The demonstrations concern user interaction through the web interface, access to saved conversation context, and execution of a GPIO control request. They are intended to show the implemented functions operating together, rather than to quantify reasoning accuracy or tool-calling success rates across repeated commands.

Figure 4 demonstrates the success of the main architecture shown in Figure 1, particularly in meeting the user's need to engage in conversations that are responded to directly by an Agent based on a local LLM.

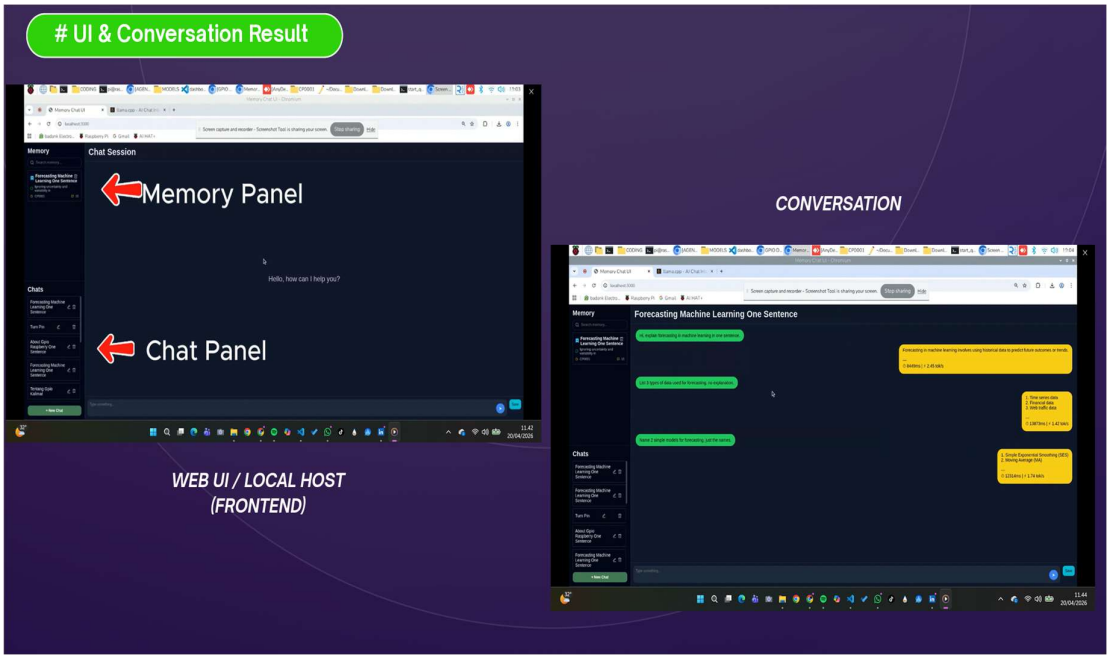


Figure 4. Local Agent User Interface and Conversation Results

Figure 5 illustrates the memory injection mechanism used by the Agent. The "active memory" status indicates that information from previous conversations has been successfully loaded and is available to the Agent throughout the interaction. The system provides two memory usage modes. In Full Mode, the context provided to the Agent consists of a summarized memory capsule and the last 10 messages, so the Agent receives both a concise summary and the most recent conversation context. Meanwhile, Fast Mode uses only the summarized memory capsule, thereby reducing the amount of context that needs to be processed. These results show that the implementation not only successfully provides conversational capabilities using a local LLM, but has also integrated memory and context injection mechanisms into the Agent's interaction cycle. As a result, the Agent can retain important information from previous sessions without having to include the entire conversation history in every inference process which could place an excessive burden on edge device resources and result in overly long contexts.

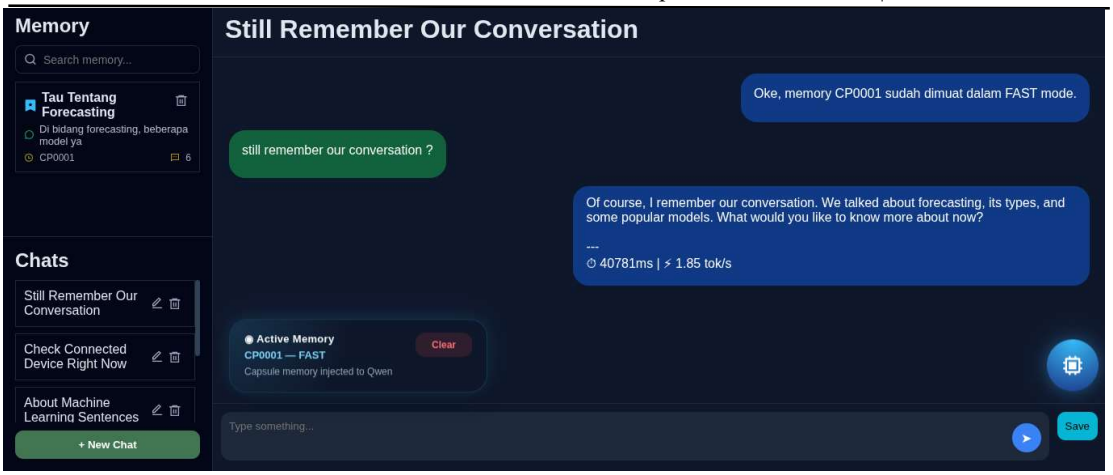


Figure 5. Conversation Memory Injection Result

Next, regarding the agent's control of the tool, Figure 6 illustrates the successful implementation of the local tool execution capability on the Agent. In this test, the user issued a command via the conversational interface to activate a specific GPIO pin. The command was processed by the Router and forwarded to the Command Parser to determine the device, GPIO number, and action to be performed. Next, the command is forwarded to the GPIO program running locally on the Raspberry Pi 5 to control the device connected to that pin.

The success of the execution is indicated by the GPIO dashboard, which is updated in real time and directly linked to the status of the GPIO pins on the Raspberry Pi 5. Thus, status changes on the dashboard represent changes in the GPIO status of the physical device, not merely status changes on the interface.

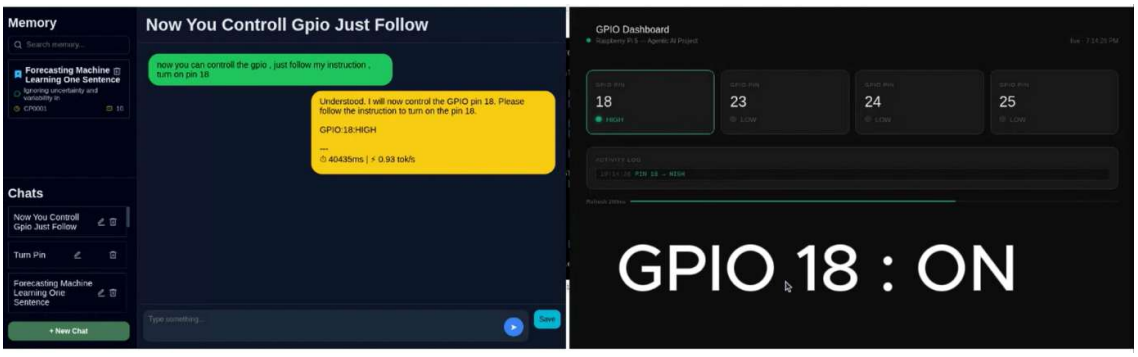


Figure 6. GPIO Command Execution Result

Table 5. Functional Implementation Results of the Proposed Local Agent

System Component	Evaluated Function	Implementation Result
Conversational Interface	Processing user prompts and generating responses through the locally deployed LLM.	Successfully demonstrated through conversational interactions using the web-based interface (Figure 4).
Conversation Memory	Saving, restoring, and injecting previous conversation context into agent interactions.	Successfully demonstrated through restoration and reuse of saved conversation context (Figure 5).
GPIO Tool Execution	Processing natural-language commands and executing GPIO control operations on the Raspberry Pi 5.	Successfully demonstrated through GPIO pin activation and the corresponding hardware-state display (Figure 6).

The results presented in Table 5 demonstrate the successful implementation and integration of the three main functional components of the proposed local agent. As illustrated in Figures 4–6, the system supports local conversational interactions, conversation memory management, and GPIO-based hardware control. These findings confirm the functional operation of the implemented architecture in the demonstrated scenarios.

3.2. LLM Inference Performance

To assess the computational suitability of locally deployed language models for edge-agent applications, we measured inference performance on the Raspberry Pi 5. This experiment evaluates responsiveness rather than directly testing the models' reasoning or autonomous decision-making ability. The three inference metrics are total latency, time to first token (TTFT), and token generation rate, as defined in Section 2.4.

The results reveal substantial performance differences among the evaluated models, demonstrating that model size and quantization configuration have a significant influence on local inference performance on the Raspberry Pi.

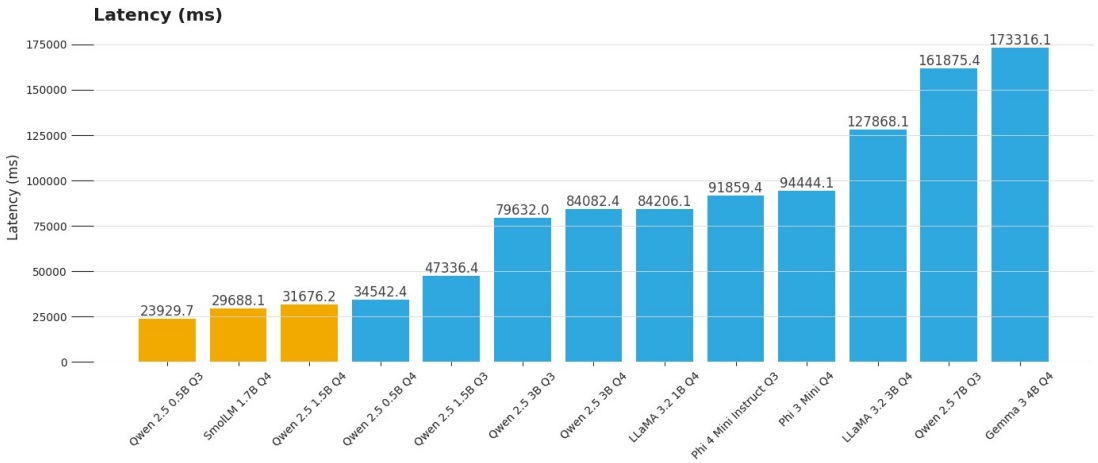


Figure 7. Total Inference Latency Across Evaluated Models

Based on the measured total latency, Qwen 2.5 0.5B Q3 achieved the lowest latency at 23,929.7 ms, followed by SmolLM 1.7B Q4 at 29,688.1 ms and Qwen 2.5 1.5B Q4 at 31,676.2 ms. In contrast, Gemma 3 4B Q4 exhibited the highest latency at 173,316.1 ms. The results generally indicate that smaller models tend to provide lower overall latency. However, the relationship is not strictly determined by parameter size alone, as quantization configuration and model architecture also influence inference efficiency.

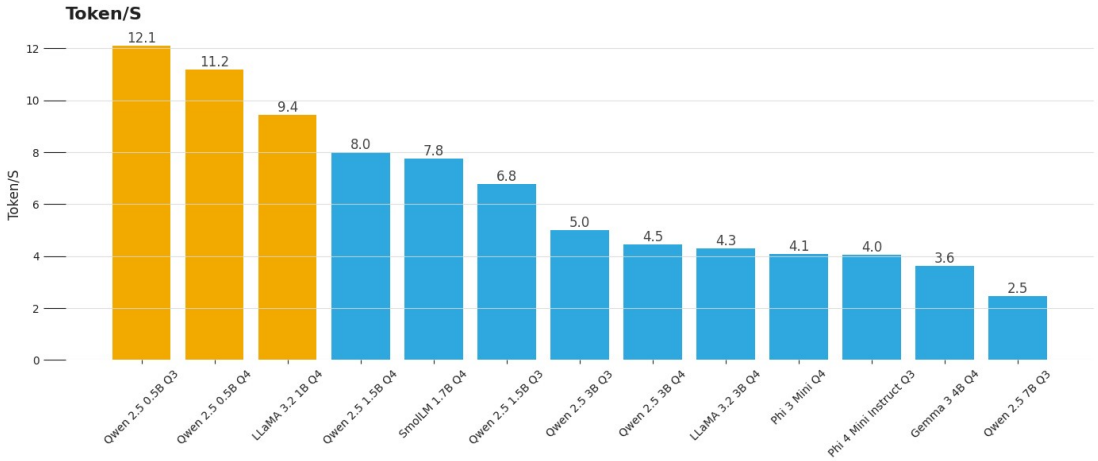


Figure 8. Token Generation Rate Across Evaluated Models

A similar trend can be observed in token generation rate. Qwen 2.5 0.5B Q3 achieved the highest generation rate at 12.1 tokens/s, followed by Qwen 2.5 0.5B Q4 at 11.2 tokens/s. Meanwhile, Qwen 2.5 7B Q3 produced only 2.5 tokens/s, representing the lowest generation rate among the evaluated models. These results indicate that the computational burden associated with larger models becomes increasingly significant when inference is performed on resource-constrained edge hardware.

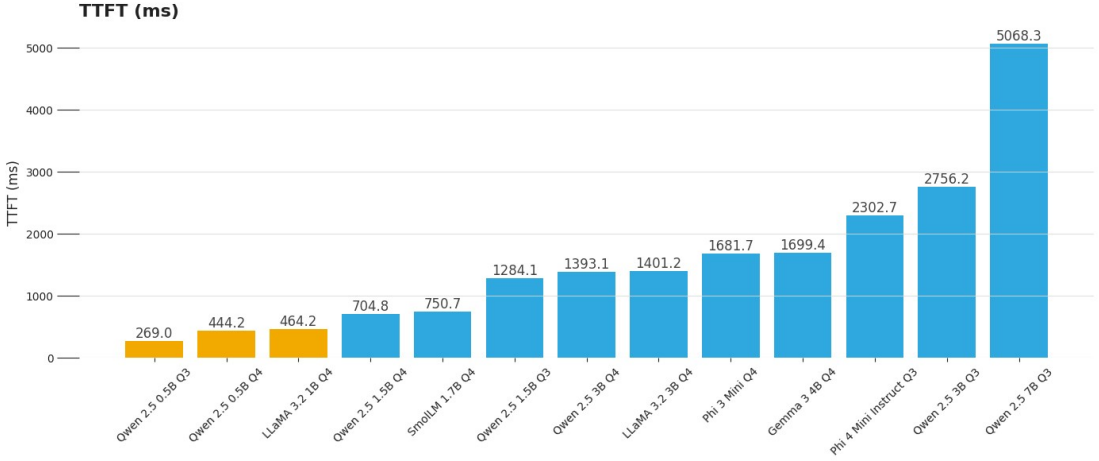


Figure 9. Time to First Token (TTFT) Across Evaluated Models

For TTFT, Qwen 2.5 0.5B Q3 again demonstrated the fastest initial response, requiring only 269.0 ms to generate the first token. Qwen 2.5 0.5B Q4 and LLaMA 3.2 1B Q4 followed with 444.2 ms and 464.2 ms, respectively. In contrast, Qwen 2.5 7B Q3 required 5,068.3 ms before producing the first token. The TTFT results are particularly relevant for interactive edge applications because they reflect how quickly the system can begin responding to a user request.

Overall, the evaluation results show that models with smaller parameter sizes tend to offer significant advantages in terms of inference speed on Raspberry Pi devices. Qwen 2.5 0.5B Q3 demonstrated the highest performance across all three inference metrics evaluated, making it one of the most efficient configurations from a response speed perspective. However, inference speed cannot be the sole criterion for determining the most suitable model for building a local agent. Larger models have the potential to provide better reasoning capabilities and instruction comprehension, while smaller models offer higher responsiveness with lower computational load. Therefore, model selection on edge devices must consider a balance between inference speed, device resource requirements, and the quality of the agent's capabilities. The results in this subsection will then serve as the basis for analyzing device resource usage for each model before determining the most suitable configuration for local agent deployment.

3.3. Resource Utilization

In addition to inference performance, the computational resource utilization of the Raspberry Pi was evaluated to assess the feasibility of running different LLMs under resource-constrained conditions. Three system-level metrics were considered peak RAM usage, peak CPU utilization, and peak temperature. These metrics provide an indication of the peak memory consumption, computational workload, and thermal behavior associated with local LLM inference.

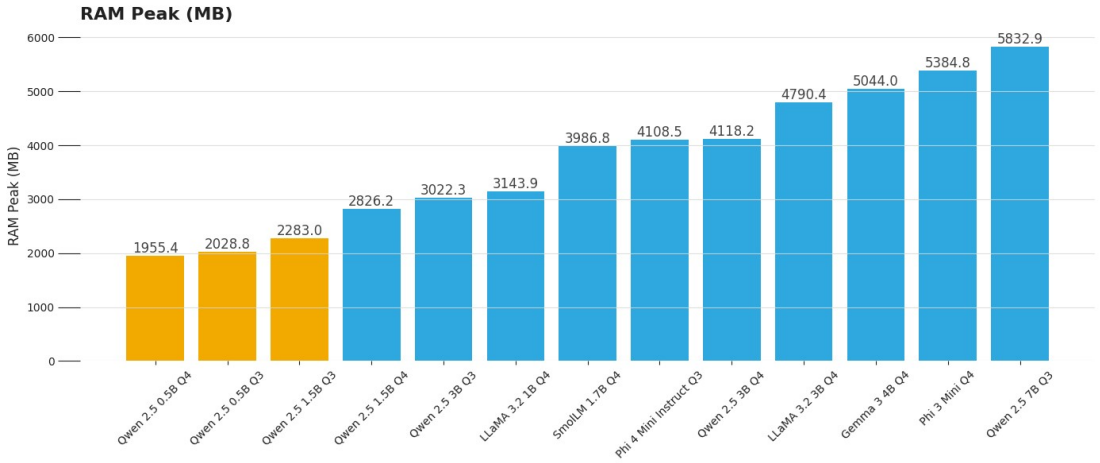


Figure 10. Peak RAM Usage Across Evaluated Models

The peak RAM measurements reveal differences in the absolute memory usage recorded during inference across the evaluated model configurations. Qwen 2.5 0.5B Q4 showed the lowest average peak RAM usage at 1,955.4 MB, followed by Qwen 2.5 0.5B Q3 at 2,028.8 MB and Qwen 2.5 1.5B Q3 at 2,283.0 MB. In contrast, Qwen 2.5 7B Q3 recorded the highest average peak RAM usage at 5,832.9 MB, followed by Phi 3 Mini Q4 at 5,384.8 MB and Gemma 3 4B Q4 at 5,044.0 MB. Overall, the results show that larger configurations generally require more memory, although peak RAM usage is not determined by parameter count alone. Model architecture and quantization settings may also contribute to the differences observed. These absolute peak readings provide a direct indication of the memory demand placed on the Raspberry Pi 5. The reported values are averages of the peak measurements recorded for individual trials.

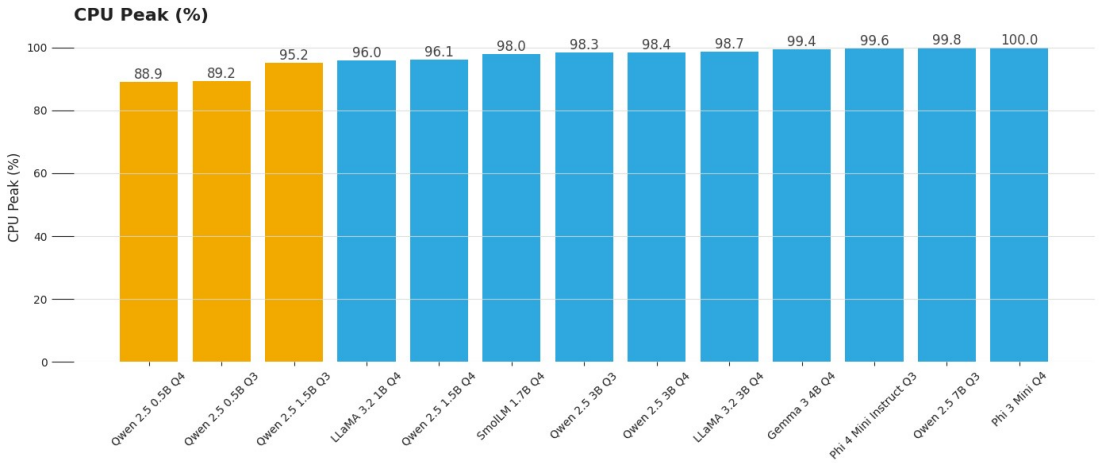


Figure 11. Peak CPU Utilization Across Evaluated Models

CPU utilization measurements show that local LLM inference places a very high computational load on the Raspberry Pi. Peak CPU utilization ranged from 88.9% to 100.0%, with Qwen 2.5 0.5B Q4 recording the lowest value at 88.9%, while Phi 3 Mini Q4 reached a maximum utilization of 100.0%. Most of the evaluated models showed peak CPU utilization above 95%, indicating that the Raspberry Pi's CPU is used intensively during the inference process.

Interestingly, the differences in CPU utilization among models are relatively smaller compared to the differences observed in inference metrics such as latency and token generation rate. Even models with relatively small parameter sizes still produced high CPU utilization. This suggests that low latency does not always equate to low CPU load during the inference process. Thus, these results demonstrate that local LLM inference is a CPU-intensive workload on the Raspberry Pi; therefore, model selection for edge deployment must consider not only response speed but also the computational load placed on the device.

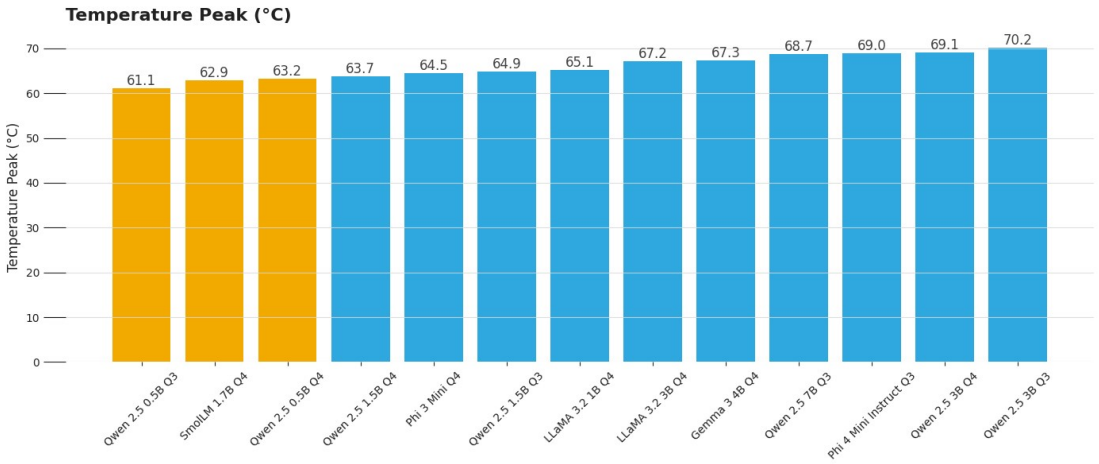


Figure 12. Peak Temperature Across Evaluated Models

The peak temperature measurements ranged from 61.1°C for Qwen 2.5 0.5B Q3 to 70.2°C for Qwen 2.5 3B Q3. The observed temperature increase indicates that sustained local inference generates a measurable thermal workload on the Raspberry Pi. Nevertheless, the recorded peak temperatures remained within the observed operating range during the experiments. The thermal results therefore complement the CPU measurements by showing the physical impact of sustained computational workloads on the edge device.
Overall, the resource utilization results demonstrate that model selection for edge-based Agentic AI should not be based solely on inference speed. A suitable model should provide a balanced trade-off between response latency, generation speed, memory consumption, CPU utilization, and thermal behavior.

3.4. Overall System Performance and Finding

Based on the system implementation results, inference performance, and Raspberry Pi resource utilization, this study demonstrates the feasibility of deploying a locally operated LLM-based agent on resource-constrained edge hardware. The implemented system successfully supports conversational interactions, conversation memory management, and GPIO-based hardware control, as demonstrated in Section 3.1. Meanwhile, the quantitative evaluation provides a comparison of inference efficiency and computational resource requirements across different model quantization configurations.Inference results show that Qwen 2.5 0.5B Q3/Q4 provides the highest responsiveness, with a TTFT of 269–444 ms, a generation rate of 11–12 tokens/s, and a latency of 23–34 seconds. However, this speed advantage alone is not sufficient to determine the best model, as agentic capabilities must also be considered. LLaMA 3.2 1B Q4, Qwen 2.5 1.5B Q4, and SmolLM 1.7B Q4 demonstrate a more balanced compromise between speed and model capacity. Conversely, models with 3B or more parameters exhibit higher latency and TTFT, as well as lower generation rates. In terms of resource usage, most models produce peak CPU utilization above 95%, while peak RAM usage and temperature vary across configurations, with the highest average peak RAM usage reaching 5,832.9 MB for Qwen 2.5 7B Q3 and the lowest reaching 1,955.4 MB for Qwen 2.5 0.5B Q4. Thus, based on inference performance and resource usage, the 1B–1.7B range can be considered an initial “sweet spot” for deploying a local agent on the Raspberry Pi 5. However, selecting the final model still requires evaluating the agent’s ability to understand intent, reason, call tools, and successfully execute actions on the hardware. Therefore, these results serve as a foundation for evaluating the agent’s capabilities and the reliability of tool calls in the next phase.

This study has several limitations that should be taken into account when interpreting the experimental results. First, the evaluation was conducted on only one hardware platform the Raspberry Pi 5 with 16 GB of RAM so the results may not necessarily be generalizable to other edge devices. Second, the inference testing used three levels of prompt complexity Simple, Medium, and Hard which do not fully represent the diversity of user requests in real-world conditions. Third, each model configuration was tested three times across each prompt category, so the number of experimental repetitions remains limited. Finally, the quantitative evaluation in this study focuses on inference performance and computational resource usage, without specifically measuring the quality of reasoning for each language model.

4. CONCLUSION

This research has explored the application of Edge Computing and Large Language Models to build local agents on resource-constrained devices using the Raspberry Pi 5. The developed system enables local LLM processing without reliance on cloud services and provides an agentic mechanism that allows the model to interact with external functions and hardware.

Inference evaluation results show a clear trade-off between model size and responsiveness. Qwen 2.5 0.5B Q3 delivers the highest inference performance with the lowest latency and TTFT, as well as the highest token generation rate. However, these results do not automatically make this model the best choice for the overall local agent, as model selection must also consider the model's capacity (its reasoning ability) to perform agentic tasks.

An evaluation of resource usage shows that local LLM inference places a heavy computational load on the Raspberry Pi. Most models reached peak CPU utilization above 95%, while average peak RAM usage ranged from 1,955.4 MB to 5,832.9 MB, while peak temperature also varied across models. These findings indicate that selecting an LLM for edge devices cannot be based solely on inference speed but must also consider resource efficiency and hardware characteristics.

Based on a combination of inference performance and resource usage results, models in the 1B–1.7B range namely LLaMA 3.2 1B Q4, Qwen 2.5 1.5B Q4, and SmolLM 1.7B Q4 can be considered initial "sweet spot" candidates for local agent deployment on the Raspberry Pi 5. This range offers a more balanced compromise between responsiveness and model capacity compared to both very small and very large models.

Overall, this study demonstrates that building LLM-based local agents on low-cost edge devices is feasible, but successful deployment is not determined solely by the ability to run the model locally. Model selection must consider a balance between inference performance, resource efficiency, and agentic capability.

5. REFERENCES

[1] Y. Zheng, Y. Chen, B. I. N. Qian, X. Shi, Y. Shu, and J. Chen, "A Review on Edge Large Language Models : Design , Execution , and Applications," *ACM Comput. Surv.*, vol. 57, no. 8, 2025, doi: 10.1145/3719664.

[2] Y. Qiao *et al.*, "An Effective Data Privacy Protection Algorithm Based on Differential Privacy in Edge Computing," *IEEE Access*, vol. 7, pp. 136203–136213, 2019, doi: 10.1109/ACCESS.2019.2939015.

[3] O. T. Modupe *et al.*, "REVIEWING THE TRANSFORMATIONAL IMPACT OF EDGE COMPUTING ON REAL-TIME DATA PROCESSING AND ANALYTICS," *Comput. Sci. IT Res. Journal, 5(3), pp. 693–702, 2024*, vol. 5, no. 3, pp. 693–702, 2024, doi: 10.51594/csitrj.v5i3.929.

[4] A. Bandi, B. Kongari, R. Naguru, S. Pasnoor, and S. V. Vilipala, "The Rise of Agentic AI : A Review of Definitions , Frameworks , Architectures , Applications , Evaluation Metrics , and Challenges," vol. 17, no. 9, pp. 1–50, 2025, doi: 10.3390/fi17090404.

[5] D. B. Acharya, K. Kuppan, and B. Divya, "Agentic AI : Autonomous Intelligence for Complex Goals — A Comprehensive Survey," *IEEE Access*, vol. 13, no. January, pp. 18912–18936, 2025, doi: 10.1109/ACCESS.2025.3532853.

[6] M. A. Ali, F. Dornaika, and J. Charafeddine, "Agentic AI : a comprehensive survey of architectures , applications , and future directions," *Artif. Intell. Rev.*, vol. 59, 2026, doi: 10.1007/s10462-025-11422-4.

[7] Y. Sun *et al.*, "IGAA : Intent-Driven General Agentic AI for Edge Services Scheduling using Generative Meta Learning," vol. 14, no. 8, pp. 1–17, 2026.

[8] M. Huang *et al.*, "EdgeLLM : A Highly Efficient CPU-FPGA Heterogeneous Edge Accelerator for Large Language Models," *J. IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 72, 2025, doi: 10.1109/TCSI.2025.354625.

[9] H. Matsutani, N. Matsuda, and N. Sugiura, "Accelerating Local LLMs on Resource-Constrained Edge Devices via Distributed Prompt Caching," *Conf. EuroMLSys 2026, pp. 232–237.*, pp. 232–237, 2026, doi: 10.1145/3805621.3807647.

[10] T. Nguyen and T. Nguyen, "An Evaluation of LLMs Inference on Popular Single-board Computers," 2024.

[11] F. Hidalgo-castelo, A. Guerrero-González, F. García-córdova, F. Lloret-abrisqueta, and A. Piñera-marín, "Local LLMs for Industrial Supervision and Control : An Edge AI Event-Driven Architecture for Proactive Operational Context Management in Real Industrial Environments," *Electronics*, vol. 15, 2026, doi: 10.3390/electronics15122547.

[12] E. J. Husom *et al.*, *Sustainable LLM Inference for Edge AI : Evaluating Quantized LLMs for Energy Efficiency , Output Accuracy , and Inference Latency*, vol. 6, no. 4. 2025. doi: 10.1145/3767742.

[13] A. Ardakani, J. Malekar, and R. Zand, *PiCASO : A Real-time Pi-Integrated Conversational AI System*

with Optimized LLMs, vol. 1, no. 1. Association for Computing Machinery. doi: 10.1145/3787109.3815259.

[14] L. L. Zappia, M. Lattuada, M. Bianchessi, and D. Pietro Pau, "Heterogeneous Low-Power Agentic Architecture for Edge Distributed Home Surveillance," *Electron. vol. 15, no. 14, Artic. 3175, 2026*, pp. 1–72, 2026, doi: 10.3390/electronics15143175.

[15] R. Marcus and M. Gehards, "Toward a Modular Architecture for Embedded Agent Systems at the Edge".

6. AUTHOR BIOGRAPHY



Setya Widyawan Prakosa received the B.Eng. (cum laude) degree from Jember University, Jember, Indonesia, in 2013, and the M.S. and Ph.D. degrees from the Department of Electronic and Computer Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, in 2018 and 2023, respectively. He was with the Industrial Technology Research Institute (ITRI), Hsinchu, Taiwan, and Askey Computer Corporation, Taipei, as an Associate Engineer and a Senior Engineer, respectively, from 2018 to 2022. In addition, he was with the National Taiwan University of Science and Technology, as a Research Fellow. He is currently with Universitas Jember as an Assistant Professor. He holds several industrial patents and academic publications. His research interests include pattern analysis, cloud and edge computing, and algorithm-on-chip development.



Naufal Ainur Rizal was born in Jember, East Java, Indonesia, in December 1998. He received his Master of Engineering degree from the University of Jember, Indonesia, in 2024. His research interests include machine learning, predictive modeling, data processing, and machine learning model evaluation. His current research interests also extend to artificial intelligence (AI) orchestration and AI agent systems, with a focus on integrating AI models, tools, and autonomous workflows for intelligent applications.



Tias Kurniati received her Master's degree from the National Taiwan University of Science and Technology, and her Ph.D. in Information Management from Tunghai University, Taiwan. She is currently an Assistant Professor at Chung Yuan Christian University, Taiwan. Her research interests include AI-driven applications in business and management, as well as information security. She is actively engaged in exploring the integration of intelligent systems into organizational decision-making and security frameworks, with a focus on practical and sustainable AI solutions for enterprise environments.