Agentic LLMs on Raspberry Pi: Challenges and Opportunities

Edge device deployment of agentic LLMs leverages tool calling and quantization techniques to balance performance and resource demands, while supporting integration through API-based, message exchange, and executable code protocols.

Abstract

Agentic large language models on edge devices show promise for local inference and tool calling while provoking trade-offs between performance and resource demands. Studies describe several frameworks that implement agentic LLMs on systems such as Raspberry Pis, NVIDIA GPUs, and other resource-constrained platforms. For example, TinyAgent supports 1.1B and 7B parameter models with 80–85% function calling success scores using 4-bit quantization; Promp-tAI similarly applies quantization techniques to enable lightweight LLM deployment. Other systems, such as CoLLM and Less-is-More, report speed improvements (1.9–2.3× faster inference, up to 80% reduced execution time) by leveraging dynamic tool selection and adaptive load balancing.

Frameworks use diverse integration protocols. API-based designs, message exchange systems (as in AgentScope), containerized architectures, and executable Python code (as in CodeAct) serve to connect LLMs with local applications. Studies note that tool calling may reduce the need for further model optimization by offloading specialized tasks but also incur challenges in balancing model size, inference speed, and energy consumption. Variations in architecture (end-to-end, multi-agent, customizable, and cooperative) and quantification of resource efficiency reveal the need for standardized evaluation across differing approaches.

Paper search

Using your research question "Large Language Model Applied on rapberry pi to implement agentic llm for locally running llm. benefits and challeges. find research gap. instead of model optimizing more capabale through tool calling to overcome limitations on edge devices. what are the trade offs? to fulfill overall goal, what ist agentic llm. what potentials do they have and why do they help to realise the development of such an agent on edge devices? what open sourse frameworks are there to develop agentic llm, like langgrap, smolagent and autoAgent? compare them, in term of overcoming resource limitations to run llm on such small devices. what open source llms are suitable with tool calling abilities such as llama3-groq-tool-use. compare them. as the agentic llm app needs to communicate locally with other locally running python applications, which protocals are there? which is suitable?", we searched across over 126 million academic papers from the Semantic Scholar corpus. We retrieved the 50 papers most relevant to the query.

11 papers were uploaded to Elicit for screening.

Screening

We screened in papers that met these criteria:

- **Edge Device Implementation**: Does the study examine LLM deployment on Raspberry Pi or similar resource-constrained edge devices?
- **Agent Implementation**: Does the research present practical implementation details for agentic LLMs or autonomous AI agents?

- **Technical Optimization**: Does the study describe specific technical implementations for optimizing LLMs on edge devices, including tool-calling capabilities?
- **Framework Evaluation**: Does the study evaluate implementation of frameworks like LangChain, SmalAgent, or AutoAgent for edge deployment?
- **System Architecture**: Does the study include system architecture details for local edge AI deployment and execution?
- **Deployment Focus**: Does the study focus on local edge deployment rather than solely cloud-based implementation?
- **Open Source**: Does the study examine open-source LLM systems rather than exclusively closed-source or proprietary systems?
- **Implementation Context**: Does the study include practical deployment details rather than focusing exclusively on model training or compression?
- **llm tool calling**
- **local communication protocol**

We considered all screening questions together and made a holistic judgement about whether to screen in each paper.

Data extraction

We asked a large language model to extract each data column below from each paper. We gave the model the extraction instructions shown below for each column.

- **Framework/Platform for Agentic LLM on Edge Devices**:

  Identify and extract the specific framework or platform used for implementing LLM on edge devices. Include:

- Name of the framework
- Key architectural features
- Specific techniques for overcoming resource limitations
- Target edge device specifications (e.g., Raspberry Pi, specific hardware constraints)

If multiple frameworks are described, list all of them. If no specific framework is mentioned, write "Not specified".

Look primarily in the methods, introduction, and results sections for this information.

- **Model Size and Performance Characteristics**:

  Extract details about the language model's:

- Total number of parameters
- Model size (in MB or GB)
- Inference speed (tokens per second)
- Accuracy or performance metrics
- Specific optimization techniques used

Prioritize quantitative measurements. If ranges are provided, record the full range. If exact numbers are not available, note the closest approximation or qualitative description.

Check methods, results, and discussion sections for these details.

- **Tool Calling and Interaction Capabilities**:

  Identify and describe:

- Specific tool calling mechanisms
- Protocols for local application interaction
- Types of external tools or services the LLM can access
- Communication methods between LLM and local applications

Look for details about API design, communication protocols, and interaction frameworks in the methods and implementation sections.

If multiple interaction methods are described, list all of them.

- **Resource Optimization Strategies**:

  Extract strategies for managing computational resources on edge devices:

- Memory management techniques
- Computational load distribution methods
- Energy consumption optimization approaches
- Adaptive inference strategies

Prioritize concrete technical approaches over general statements. Look in methods, results, and discussion sections for specific optimization techniques.

- **Deployment and Practical Limitations**:

  Identify and describe:

- Specific challenges in edge device deployment
- Limitations of the proposed approach
- Comparative performance against cloud-based solutions
- Privacy or connectivity considerations

Examine discussion, limitations, and conclusion sections for these insights. If multiple limitations are noted, list them in order of significance.

Results

Characteristics of Included Studies

Study Focus	Implementation Type	Device Specifications	Key Findings	Full text retrieved
Edge General Intelligence via Large Language Models (LLMs)	Conceptual framework	No mention found	Identified three conceptual systems for LLM-empowered Edge General Intelligence: centralized, hybrid, and decentralized	Yes
Function calling at the edge	TinyAgent framework	MacBook Pro M3	Developed TinyAgent for deploying small language model agents on edge devices; achieved competitive performance with GPT-4-Turbo	Yes
Multi-agent platform	AgentScope framework	No mention found	Proposed AgentScope as a flexible multi-agent platform with message exchange as core communication mechanism	Yes
Customizable agent framework	ModelScope-Agent	No mention found	Introduced ModelScope-Agent as a customizable agent framework for real-world applications	Yes
Collaborative LLM inference	CoLLM framework	Graphics Processing Unit (GPU)-free devices (e.g., Raspberry Pis)	Proposed CoLLM for collaborative LLM inference on resource-constrained devices	No
LLM agent operating system	AIOS framework	NVIDIA RTX A5000 GPUs	Introduced AIOS as an LLM agent operating system for optimizing resource allocation	Yes

Study Focus	Implementation Type	Device Specifications	Key Findings	Full text retrieved
Generative AI on edge devices	Promp-tAI framework	Raspberry Pi 5 Cluster	Demonstrated feasibility of deploying lightweight LLMs on edge devices using quantization and optimization techniques	Yes
Function calling optimization	Less-is-More framework	NVIDIA AGX Orin board	Proposed Less-is-More framework for optimizing function calling on edge devices	Yes
Localized LLM inferencing	LLMEdge framework	Resource-constrained edge devices	Introduced LLMEdge framework for localized LLM inferencing on resource-constrained edge devices	No
LLM-based conversational systems	Empirical study	Raspberry Pi	Evaluated performance of LLM-based conversational systems on Raspberry Pi	No
LLM performance on constrained systems	Performance characterization	NVIDIA Jetson devices	Provided performance characterization of LLMs on resource-constrained systems	No
Executable code actions in LLM agents	CodeAct framework	No mention found	Proposed CodeAct framework for executable code actions in LLM agents	Yes

Study Focus	Implementation Type	Device Specifications	Key Findings	Full text retrieved
Cooperative embodied agents	CoELA framework	No mention found	Introduced CoELA framework for building cooperative embodied language agents	Yes

Our review identified 13 studies that reported on various aspects of Large Language Models (LLMs) and AI agents on edge devices. The studies covered a range of focus areas and implementation types:

- Study Focus :
  - Function calling (2 studies)
  - LLM inference (2 studies)
  - Edge general intelligence (1 study)
  - Multi-agent platforms (1 study)
  - Agent frameworks (1 study)
  - Agent operating systems (1 study)
  - Generative AI on edge (1 study)
  - Conversational systems (1 study)
  - LLM performance (1 study)
  - Code actions in agents (1 study)
  - Cooperative agents (1 study)
- Implementation Type :
  - Various frameworks (10 studies)
  - Conceptual framework (1 study)
  - Empirical study (1 study)
  - Performance characterization (1 study)
- Device Specifications :
  - We didn't find mention of device specifications for 5 studies in the available full texts or abstracts
  - Raspberry Pi (3 studies)
  - NVIDIA GPUs (3 studies)
  - MacBook Pro (1 study)
  - Unspecified resource-constrained devices (1 study)

Key observations:

- 10 out of 13 studies implemented frameworks for edge AI applications
- Studies explored diverse focus areas within edge AI and LLMs
- Raspberry Pi and NVIDIA GPUs were the most commonly specified devices (3 studies each)

Thematic Analysis

Edge Device Implementation Strategies

Framework	Resource Requirements	Tool Calling Support	Performance Metrics
TinyAgent	1.1B and 7B parameter models; 4-bit quantization	Function calling with 16 different functions	80.06% success rate (1.1B model), 84.95% success rate (7B model)
AgentScope	No mention found	ReAct algorithm-based tool usage module	No mention found
ModelScope-Agent	No mention found	Comprehensive tool-use framework	No mention found
CoLLM	GPU-free devices (e.g., Raspberry Pis)	No mention found	1.9x-2.3x improvement over current methods
AIOS	NVIDIA RTX A5000 GPUs	Standardized interface for tool invocation	No mention found
Promp-tAI	Raspberry Pi 5 Cluster; 4-bit quantization	No mention found	5 to 12 tokens per second for lightweight models
Less-is-More	NVIDIA AGX Orin board	Dynamic tool selection	Up to 80% reduction in execution time
LLMEdge	Resource-constrained edge devices	No mention found	No mention found
N/A (Sakai et al., 2024)	Raspberry Pi	No mention found	No mention found
N/A (Seymour et al.)	NVIDIA Jetson devices	No mention found	No mention found
CodeAct	No mention found	Executable Python code for tool calling	Approximately 9,000 tokens per second
CoELA	No mention found	Execution Module for primitive actions	No mention found
N/A (Chen et al., 2024)	No mention found	No mention found	No mention found

We found mention of specific resource requirements for 8 out of the 13 studies in the available full texts or abstracts. These included:

- Various parameter model sizes (1.1B to 7B)
- GPU-free devices
- Specific GPU models (NVIDIA RTX A5000)
- Edge devices (Raspberry Pi, NVIDIA Jetson, NVIDIA AGX Orin)

We found information on tool calling support for 7 out of 13 studies. Approaches included:

- Function calling
- ReAct algorithm-based modules
- Comprehensive frameworks
- Standardized interfaces
- Dynamic tool selection
- Executable Python code

We found performance metrics for 5 out of 13 studies. Metrics varied widely, including:

- Success rates (80.06% to 84.95%)
- Relative improvements (1.9x-2.3x)
- Tokens per second (5 to 12, and approximately 9,000)
- Execution time reduction (up to 80%)

Key observations:

- Diverse architectures and efficiency approaches observed across studies
- Each framework offers unique features and optimizations
- Lack of standardized evaluation metrics makes direct comparisons challenging

Agentic LLM Frameworks

Framework Name	Architecture Type	Key Features	Resource Efficiency
TinyAgent	End-to-end framework	Function calling, parallel function calling, novel tool retrieval method	4-bit quantization, reduced input prompt length
AgentScope	Multi-agent platform	Message exchange, fault tolerance, multi-modal data management	Actor-based distributed mode for parallel optimization
ModelScope-Agent	Customizable agent framework	Tool-use data collection, tool retrieval, tool registration, memory control	No mention found
CoLLM	Collaborative inference framework	LLM tensor parallelization, minimal latency algorithm, adaptive load balancing	Dynamic distribution of computational workloads
AIOS	LLM agent operating system	Tool manager, memory manager, context manager	K-LRU eviction policy, trie-based compression
Promp-tAI	Modular framework	API-based architecture	4-bit quantization, containerization
Less-is-More	Function calling optimization framework	Dynamic tool selection, similarity-based matching	Reducing available tools, smaller context windows
LLMEdge	Localized inference framework	Quantized LLMs, efficient localized inference	Minimizing power consumption
CodeAct	Executable code action framework	Python code execution, integration with Python interpreter	No mention found

Framework Name	Architecture Type	Key Features	Resource Efficiency
CoELA	Cooperative embodied agent framework	Modular design (Perception, Memory, Planning, Execution, Communication)	Parameter-efficient fine-tuning (LoRA)
N/A (Chen et al., 2024)	Conceptual framework	Centralized, hybrid, and decentralized systems	No mention found
N/A (Sakai et al., 2024)	Not a framework (empirical study)	N/A	No mention found
N/A (Seymour et al.)	Not a framework (performance characterization)	N/A	No mention found

We found information on 11 different LLM agent frameworks or related systems. The architectures of these frameworks varied widely:

- 9 frameworks had unique architecture types, including end-to-end, multi-agent, customizable, collaborative, operating system, modular, optimization-focused, localized, and cooperative approaches
- 1 framework was described as conceptual
- 2 entries were not actual frameworks but empirical studies or performance characterizations

Regarding resource efficiency:

- We found specific resource efficiency approaches for 8 of the 11 frameworks
- The most common approach was quantization, mentioned in 2 frameworks
- Other approaches included reduced input, parallel optimization, dynamic workload distribution, memory management, compression, containerization, reduced tools, smaller context windows, power optimization, and parameter-efficient tuning
- We didn't find specific resource efficiency information for 5 of the entries, including the 2 that were not actual frameworks

System Integration and Communication

The reviewed studies present various approaches to system integration and communication for agentic LLMs on edge devices:

- API-Based Integration : Several frameworks, including TinyAgent and Promp-tAI, utilize API-based architectures for seamless integration with existing systems and tools. This approach allows for flexibility in connecting LLMs with various applications and services.
- Message Exchange : AgentScope employs a message exchange mechanism as its core communication protocol, facilitating interaction between multiple agents and external tools. This approach enables more complex multi-agent systems and distributed architectures.
- Containerization : Studies like Nezami et al. (2024) use containerization techniques, such as Docker and Kubernetes (K3s), to manage and orchestrate LLM services across edge devices. This approach provides isolation and simplifies deployment across heterogeneous hardware.

- Executable Code Integration : The CodeAct framework introduces a novel approach by allowing LLMs to generate and execute Python code directly, enabling more dynamic and flexible interactions with local applications and tools.
- Standardized Interfaces : AIOS proposes a standardized interface for tool invocation, which helps in managing diverse tools under a single framework. This approach can simplify the integration of new tools and services.
- Local Application Interaction : TinyAgent demonstrates interaction with local Mac applications using predefined Apple scripts, showcasing a method for integrating LLMs with existing desktop software.
- Memory Management : Several frameworks, including ModelScope-Agent and AIOS, incorporate memory modules to manage contextual information and improve the efficiency of interactions between LLMs and local applications.

Key observations:

- Diverse approaches reflect complexity of integrating agentic LLMs with edge devices
- Each method offers trade-offs in flexibility, performance, and ease of implementation
- Lack of standardized protocols makes direct comparisons challenging

Performance and Resource Trade-offs

The deployment of agentic LLMs on edge devices involves significant performance and resource trade-offs, as highlighted by the reviewed studies:

- Model Size vs. Performance : Smaller models like TinyAgent-1.1B offer reduced resource requirements but may sacrifice some performance compared to larger models. The challenge lies in finding the right balance between model size and capability for specific edge applications.
- Quantization Effects : Many studies employ quantization techniques to reduce model size and computational requirements. For example, 4-bit quantization is used in frameworks like TinyAgent and Promp-tAI. While this significantly reduces memory usage and improves inference speed, it may lead to some degradation in model accuracy.
- Inference Speed vs. Energy Consumption : Frameworks like CoLLM aim to optimize both latency and energy consumption through adaptive load balancing. However, achieving high inference speeds often comes at the cost of increased energy usage, which is a critical consideration for battery-powered edge devices.
- Tool Selection and Context Window : The Less-is-More framework demonstrates that reducing the number of available tools and using smaller context windows can significantly improve execution time and power consumption. However, this approach may limit the model's capabilities in more complex scenarios.
- Local vs. Cloud Processing : Studies like Sakai et al. (2024) highlight the trade-off between locally-run models, which offer offline capabilities but higher computational demands, and API-based models, which are more efficient but require internet connectivity.
- Privacy vs. Performance : Edge deployment can enhance data privacy by reducing reliance on cloud services, but this often comes at the cost of reduced computational power compared to cloud-based solutions.

- Generalization vs. Specialization : Some approaches, like TinyAgent, focus on task-specific fine-tuning to improve performance on edge devices. While this can lead to better efficiency for specific tasks, it may reduce the model's generalization capabilities.
- Collaborative Inference : Frameworks like CoLLM propose collaborative inference across multiple devices to overcome individual device limitations. While this can improve overall performance, it introduces additional complexity in terms of communication and coordination.

These trade-offs highlight the complex decision-making process involved in deploying agentic LLMs on edge devices. Key observations:

- Complex decision-making process involved in deploying agentic LLMs on edge devices
- Optimal balance depends on:
  - Specific application requirements
  - Constraints of target edge devices

References

Chenliang Li, Hehong Chen, Mingshi Yan, Weizhou Shen, Haiyang Xu, Zhikai Wu, Zhicheng Zhang, et al. "ModelScope-Agent: Building Your Customizable Agent System with Open-Source Large Language Models." *Conference on Empirical Methods in Natural Language Processing*, 2023.

Dawei Gao, Zitao Li, Weirui Kuang, Xuchen Pan, Daoyuan Chen, Zhijian Ma, Bingchen Qian, et al. "AgentScope: A Flexible yet Robust Multi-Agent Platform." *arXiv.org*, 2024.

Handi Chen, Weipeng Deng, Shuo Yang, Jinfeng Xu, Zhihan Jiang, Edith C. H. Ngai, Jiangchuan Liu, and Xue Liu. "Towards Edge General Intelligence via Large Language Models: Opportunities and Challenges." *arXiv.org*, 2024.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, J. Tenenbaum, Tianmin Shu, and Chuang Gan. "Building Cooperative Embodied Agents Modularly with Large Language Models." *International Conference on Learning Representations*, 2023.

Jinrong Li, Biao Han, Sudan Li, Xiaoyan Wang, and Jie Li. "CoLLM: A Collaborative LLM Inference Framework for Resource-Constrained Devices." *International Conference on Innovative Computing and Cloud Computing*, 2024.

Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. "AIOS: LLM Agent Operating System." *arXiv.org*, 2024.

Koga Sakai, Yuji Uehara, and Shigeru Kashihara. "Implementation and Evaluation of LLM-Based Conversational Systems on a Low-Cost Device." *IEEE Global Humanitarian Technology Conference*, 2024.

Liam Seymour, Basar Kutukcu, and S. Baidya. "Large Language Models on Small Resource-Constrained Systems: Performance Characterization, Analysis and Trade-Offs," 2024.

Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Hooper, G. Anumanchipalli, Kurt Keutzer, and A. Gholami. "TinyAgent: Function Calling at the Edge." *Conference on Empirical Methods in Natural Language Processing*, 2024.

P. Ray, and Mohan Pratap Pradhan. "LLMEdge: A Novel Framework for Localized LLM Inferencing at Resource Constrained Edge." *2024 International Conference on IoT Based Control Networks and Intelligent Systems (ICICNIS)*, 2024.

Varatheepan Paramanayakam, Andreas Karatzas, Iraklis Anagnostopoulos, and Dimitrios Stamoulis. "Less Is More: Optimizing Function Calling for LLM Execution on Edge Devices," n.d.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. "Executable

Code Actions Elicit Better LLM Agents." *International Conference on Machine Learning*, 2024.
Zeinab Nezami, Maryam Hafeez, Karim Djemame, and S. A. R. Zaidi. "Generative AI on the Edge: Architecture and Performance Evaluation." *arXiv.org*, 2024.