DAgent: A Multi-Agent System for Device-Aware Assistance

1st Given Name Surname
dept. name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—The limitations of large language models (LLMs) in interacting with local computing environments have hindered their applicability in device-aware tasks. This research addresses this gap by introducing DAgent, a modular, multi-agent system designed to provide personalized, context-aware assistance through integration with the user's system environment. The architecture comprises four specialized modules: the Tracer module for monitoring system activities, the RAG module for retrieval-augmented generation leveraging system logs, the Coding Agent for interacting with the environment to infer information or execute actions, and the Multi-Agent OS Assistant for orchestrating workflows. DAgent was evaluated on correctness, completeness, and clarity, scoring highly with the baseline 14B model: 4.87, 4.53, and 4.41 out of 5, respectively. Additionally, an ablation study removing the RAG module demonstrated the importance of the coding agent module, which achieved an average score of 4.45 when used independently. These findings highlight DAgent's ability to excel in device-aware tasks, providing actionable, comprehensible responses and enabling secure, supervised command execution. This work contributes to the advancement of intelligent, device-aware assistants, bridging the gap between language reasoning and real-world system interaction.**

***Index Terms*—Multi-Agent System, Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), Context-Aware Assistance, Personalized AI Assistant**

I. Introduction

The recent progress in Large Language Models (LLMs) such as OpenAI's GPT series and Meta's LLaMA has significantly expanded the capabilities of AI systems across a wide range of natural language tasks [1]. These models excel at text generation, question answering, and coding, driven by architectural advances and large-scale pretraining. However, despite their linguistic fluency, LLMs remain constrained by their limited access to external environments. Tasks requiring direct interaction with local devices—such as editing configs, running scripts, or monitoring systems—remain out of scope unless integrated with specialized toolchains [2], [3].

To overcome this, recent frameworks have begun treating LLMs as agents capable of structured interactions with APIs, tools, and services [4]–[6]. While this "agentic" shift has led to a wide range of applications, such as software documentation [7], most existing systems remain cloud-based, sandboxed, or detached from the user's actual computing environment. This leaves a critical gap in enabling personalized, device-level assistants that can safely execute commands, persist state across sessions, and adapt to individual user contexts.

This paper introduces DAgent, a modular, multi-agent system designed for personalized interaction with a user's local operating system. DAgent integrates LLMs into agents that can monitor system activity, retrieve personalized context, and execute commands under supervision. The key contributions of this work are:

- A multi-agent architecture that tightly integrates with the local device, enabling context-aware system interaction.
- A supervised command execution pipeline with persistent logging and traceability.
- A Tracer module for continuous system monitoring, providing historical context to improve agent decisions.
- A personalized Retrieval-Augmented Generation (RAG) pipeline using trace logs and file snapshots.
- A general evaluation framework for device-aware multi-agent systems, scoring them on correctness, completeness, and clarity.

The rest of this paper is organized as follows: Section II reviews related work in agentic LLMs, tool-augmented frameworks, and system-aware AI assistants. Section III details the architecture and components of DAgent. Section IV presents evaluation scenarios and performance benchmarks. Section V concludes with key takeaways.

II. Related Work

The evolution of AI assistants has progressed from rule-based systems to advanced multi-agent architectures, enabling more sophisticated and context-aware interactions (§II-A). Recent advances in AI agent systems have introduced innovative frameworks for personalization, multi-agent collaboration, and OS integration, which inform the design of DAgent (§II-B).

A. Evolution of AI Assistants

AI assistants have progressed from early rule-based systems like ELIZA [8] to today's deep learning models. The Transformer architecture [9] enabled the rise of large language models (LLMs), trained via next-token prediction and aligned with human preferences using reinforcement learning from human feedback (RLHF) [10]. In parallel, intelligent agent paradigms emerged [11], enabling goal-directed reasoning and collaborative multi-agent systems (MAS) [12].

Recent advances in multi-agent design have addressed persistent challenges like hallucination and lack of personalization [13]. Modular coordination strategies, such as Anthropic's Constitutional AI [14], offer greater adaptability and reliability. However, popular assistants like ChatGPT and Claude remain limited in their ability to reason over personalized environments or consistently produce factual, grounded responses [15].

To address these limitations, DAgent integrates retrieval-augmented generation with a modular, multi-agent framework—delivering personalized, context-aware assistance that interacts directly with the user's computing environment.

B. Recent Advances in AI Agent Systems

Recent innovations in AI agent systems have sought to overcome the contextual and functional boundaries of traditional assistants by embedding LLMs deeper into system environments. Notably, OS-integrated architectures like AIOS [16] and MemGPT [17] propose mechanisms for tool orchestration and virtual memory management inspired by operating systems. These approaches enable persistent and semantically coherent agent behavior by bridging short-term LLM contexts with long-term interaction history. DAgent extends these principles by introducing a system-level Tracer for live monitoring and deep personalization.

In the realm of multi-agent coordination, orchestration systems like Magentic-One [18] and OWL [19] highlight how specialized agents—managed via lead Orchestrators or hierarchical planners—can collaboratively solve complex tasks. DAgent adopts a similar modular approach while focusing specifically on tight integration with the user's OS. By combining virtual memory ideas from MemGPT, orchestration strategies from AIOS and Magentic-One, and modular task delegation as demonstrated in OWL, DAgent delivers a reliable and OS-aware agent system capable of personalized and trace-informed responses.

III. Methodology

DAgent employs a modular architecture to deliver personalized assistance. The Tracer module (§III-A) monitors system activities, providing contextual data. The RAG module (§III-B) retrieves personalized information, while the Coding Agent (§III-C) handles computational tasks. The Multi-Agent DAgent (§III-D) orchestrates these modules, ensuring coherent and contextually-aware responses. Figure 1 illustrates the system's architecture and workflow.

A. Tracer Module

The Tracer module enables real-time monitoring of OS-level events, ensuring that the assistant's responses are grounded in the actual system state. It bridges the gap between dynamic environmental context and the reasoning capabilities of the language model. Designed to be domain-agnostic and extensible, the Tracer supports multiple system domains such as file systems, and networking. Each domain-specific tracer captures events in a structured format and appends them to a shared persistent log. The current implementation includes a file system tracer that records metadata (e.g., file paths, timestamps, operation types) related to directory activities. This event log serves as a dynamic knowledge source for the RAG module (§III-B), enabling context-aware responses. Although the initial focus is on filesystem events, the architecture is built for easy extension to additional domains, enhancing the assistant's adaptability and system awareness.

B. RAG Module

The Retrieval-Augmented Generation (RAG) module serves as the personalized knowledge repository for DAgent, integrating device-specific data from the Tracer module (§III-A). It processes system logs into embeddings stored in a vector database, enabling efficient semantic retrieval. The module comprises two components: the ***RAG Engine***, which retrieves relevant data using similarity-based searches, and the ***Summarizer Agent***, which synthesizes retrieved information into coherent responses. These components ensure that user queries are addressed with personalized, contextually relevant information. Integrated with DAgent (§III-D), the RAG module enhances response generation by providing enriched context, enabling the system to deliver precise and user-specific assistance.

C. Coding Agent Module

The Coding Agent module addresses computational queries by generating, executing, and explaining code. It complements the RAG module (§III-B) by handling tasks requiring dynamic computation or real-time system data. The module comprises three components: the ***Code Generator***, which translates queries into executable code; the ***Code Runner***, which executes the code securely; and the ***Code Summarizer***, which provides user-friendly explanations of the results. Integrated with DAgent (§III-D), the Coding Agent enhances the system's ability to address both informational and computational queries, ensuring comprehensive assistance.

D. Multi-Agent DAgent

The Multi-Agent DAgent orchestrates the system's workflow, managing user inputs and coordinating interactions between modules. It employs a modular architecture to handle diverse queries by dynamically assembling processing pipelines. Key components include the ***Domain Analyzer***, which identifies relevant domains; the ***Context Retriever***, which gathers contextual data from the RAG module (§III-B); and the ***Query Classifier***, which determines the query type. The ***Information Generator*** and ***Command Generator*** produce responses or commands, while the ***Output Presenter*** formats the final output. By integrating with the RAG (§III-B) and Coding Agent (§III-C) modules, the assistant handles complex tasks, ensuring responses are both accurate and actionable.

IV. Results & Discussion

A. Evaluation Framework

Evaluating DAgent responses requires metrics beyond traditional NLP measures like BLEU or ROUGE, which fail

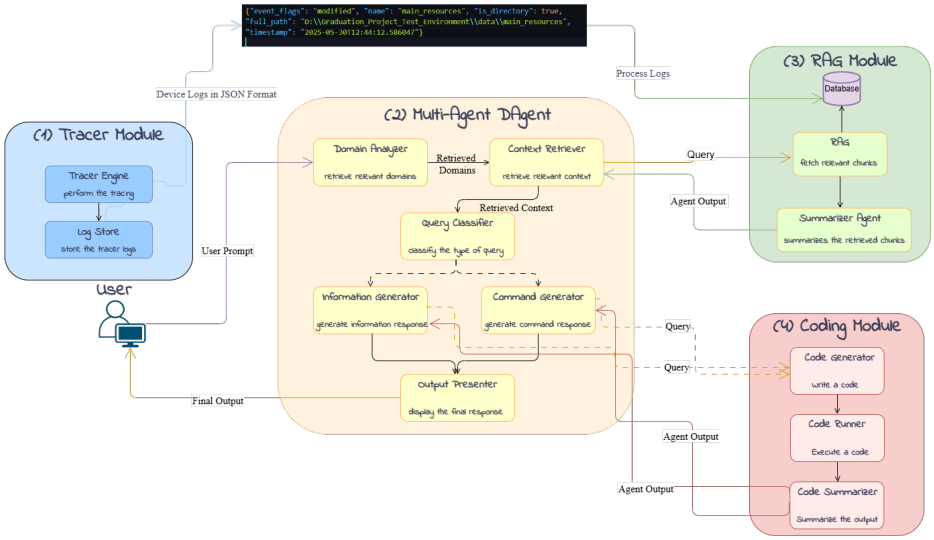

Fig. 1. DAgent Architecture: The system consists of four primary modules — Tracer, RAG, Coding Agent, and Main module. The workflow begins with user input processing, followed by domain identification, information retrieval, and determining whether to provide information or execute commands.

to capture functional accuracy or contextual relevance. While human evaluation offers depth, it lacks scalability and consistency. To address this, we adopt an automated framework where responses are rated by an LLM across three dimensions: **Correctness**, **Completeness**, and **Clarity**, each scored on a 0.0–5.0 scale using structured, prompt-based rubrics. Justifications generated by the LLM are manually reviewed to refine test design and expose framework limitations.

Correctness reflects technical accuracy and system executability, validated against Windows 10 behaviors, including file operations and scripting. Both CLI and GUI formats are accepted if operationally valid.

Completeness measures whether the assistant fully addresses the user's intent, executes available actions, and acknowledges any constraints.

Clarity assesses how clearly solutions are communicated—favoring structured, well-formatted responses with logical flow and concise language.

This LLM-driven approach enables reproducible, high-resolution evaluation of assistant outputs, balancing functional correctness with usability.

B. Experiment

1) Experiment Setup: **Data.** A total of 289 LLM-generated queries were created, from which 30 representative cases—covering command and information tasks—were manually selected for evaluation within the file system domain.

Environment. Experiments were conducted in a sandboxed Windows-style file system preloaded with dummy data and synthetic logs. This setup enabled safe execution and consistent verification of assistant outputs.

C. Evaluation Results

Five model variants—Qwen2.5-14B, 7B, 3B, 1.5B, and 0.5B—were evaluated in conjunction with Qwen2.5-Coder across the dimensions of **Correctness**, **Completeness**, and **Clarity**, using the rubric described in Section IV-A. All models were granted access to both the RAG and coding agent tools. Scoring was performed by an independent Qwen2.5-14B model serving as the evaluation judge.

TABLE I
EVALUATION SCORES ACROSS ALL MODELS (0–5 SCALE)

Model	Correctness	Completeness	Clarity
Qwen2.5-14B	**4.87**	**4.53**	**4.41**
Qwen2.5-3B	4.10	3.83	4.19
Qwen2.5-7B	2.93	2.63	3.56
Qwen2.5-1.5B	1.47	1.30	3.20
Qwen2.5-0.5B	1.10	1.20	3.00

Superior performance was exhibited by the Qwen2.5-14B model, which consistently produced accurate, comprehensive, and well-structured outputs. Competitive results were attained by the Qwen2.5-3B variant, attributed to more frequent tool utilization. Despite its larger size, the 7B model underperformed, likely due to infrequent tool invocation. Further inspection suggests that the 7B variant adopted a conservative approach to tool usage, frequently attempting to resolve complex queries through internal reasoning alone rather than leveraging available external tools, which ultimately contributed to its degraded performance. The smallest models (1.5B and 0.5B) were found to struggle with correctness and completeness, though moderate clarity was maintained. These findings suggest that effective tool usage can mitigate the limitations of smaller models, enhancing response quality in task-oriented assistant scenarios.

D. Ablation and Comparative Analysis

To quantify the contribution of tool integration, an ablation study was performed using the Qwen2.5-14B model under four

tool access configurations: no tools, RAG only, Coding Agent only, and both tools combined.

TABLE II
ABLATION STUDY: QWEN2.5-14B UNDER VARYING TOOL ACCESS

Configuration	Tools Used	Avg. Score
Qwen2.5-14B – No Tools (Baseline)	None	3.93
Qwen2.5-14B + RAG Only	RAG	3.22 (–0.71)
Qwen2.5-14B + Coding Agent Only	Coding Agent	4.45 (+0.52)
Qwen2.5-14B + Coding Agent + RAG	Both	**4.64 (+0.71)**

The highest score was obtained when both tools were enabled, confirming that hybrid tool access yields the greatest performance gains. The Coding Agent was found to contribute more significantly than RAG when used in isolation. Notably, the RAG-only configuration underperformed relative to the no-tools setting, suggesting that retrieval alone may not sufficiently support procedural reasoning. These results highlight the importance of execution capabilities and the synergistic benefit of tool integration in the Qwen2.5-14B agent system.

Model Comparison. To benchmark external models, the performance of StarCoder2 and Qwen2.5-14B was compared under full tool access conditions, with both the Coding Agent and RAG modules enabled.

TABLE III
EXTERNAL MODEL COMPARISON WITH FULL TOOL ACCESS

Model	Tools Used	Avg. Score
Qwen2.5-14B	Coding Agent + RAG	**4.64**
StarCoder2-15B + Phi-14B	Coding Agent + RAG	4.16

While StarCoder2-15B combined with Phi-14B demonstrated strong performance, it was outperformed by **Qwen2.5-14B** model.

V. CONCLUSION

This study has introduced DAgent, a modular, multi-agent system designed to bridge the gap between large language models and device-level interactions. By integrating components such as the Tracer module, RAG module, Coding Agent, and Multi-Agent DAgent, the system demonstrates the ability to provide personalized, context-aware assistance while ensuring secure and supervised command execution. The evaluation results highlight the system's effectiveness in addressing challenges related to correctness, completeness, and clarity. Specifically, the baseline 14B model achieved the highest scores across all dimensions, with an average correctness score of 4.87, completeness score of 4.53, and clarity score of 4.41. These results underscore the system's ability to deliver actionable, thorough, and comprehensible responses. Furthermore, the ablation study revealed that the coding agent tool alone achieved an average score of 4.45, demonstrating its critical role in enhancing task performance. The modular design and integration of retrieval-augmented generation and computational capabilities underscore DAgent's potential for real-world applications, including system diagnostics, automation, and personalized user support. This work represents a significant step toward enabling intelligent, device-aware assistants capable of enhancing productivity and system management.

REFERENCES

[1] T. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[2] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[3] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=Yacmpz84TH

[4] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, S. Zhang, X. Deng, A. Zeng, Z. Du, C. Zhang, S. Shen, T. Zhang, Y. Su, H. Sun, M. Huang, Y. Dong, and J. Tang, "Agentbench: Evaluating llms as agents," *arXiv preprint arXiv:2308.03688*, 2023. [Online]. Available: https://arxiv.org/abs/2308.03688

[5] LangChain, "Langchain: A framework for building multi-agent llm systems," 2023, https://docs.langchain.com.

[6] T. B. Richards, "Auto-gpt: An experimental open-source attempt to make gpt-4 fully autonomous," https://github.com/Torantulino/Auto-GPT, 2023, accessed: 2025-06-05.

[7] D. Yang *et al.*, "Docagent: A multi-agent system for automated code documentation generation," *arXiv preprint arXiv:2504.08725*, 2025.

[8] J. Weizenbaum, "ELIZA—A computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 5998–6008.

[10] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 4299–4307.

[11] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2010.

[12] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.

[13] J. Huang and K. C.-C. Chang, "Hallucination is inevitable: An innate limitation of large language models," *arXiv preprint arXiv:2401.11817*, 2024.

[14] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. Olsson *et al.*, "Constitutional ai: Harmlessness from ai feedback," *arXiv preprint arXiv:2212.08073*, 2022.

[15] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," 2023. [Online]. Available: https://arxiv.org/abs/2304.03442

[16] K. Mei, X. Zhu, W. Xu, W. Hua, M. Jin, Z. Li, S. Xu, R. Ye, Y. Ge, and Y. Zhang, "Aios: Llm agent operating system," 2025. [Online]. Available: https://arxiv.org/abs/2403.16971

[17] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez, "Memgpt: Towards llms as operating systems," 2024. [Online]. Available: https://arxiv.org/abs/2310.08560

[18] A. Fourney, G. Bansal, H. Mozannar, C. Tan, E. Salinas, Erkang, Zhu, F. Niedtner, G. Proebsting, G. Bassman, J. Gerrits, J. Alber, P. Chang, R. Loynd, R. West, V. Dibia, A. Awadallah, E. Kamar, R. Hosn, and S. Amershi, "Magentic-one: A generalist multi-agent system for solving complex tasks," 2024. [Online]. Available: https://arxiv.org/abs/2411.04468

[19] H. Guo, J. Yang, J. Liu, L. Yang, L. Chai, J. Bai, J. Peng, X. Hu, C. Chen, D. Zhang, X. Shi, T. Zheng, L. Zheng, B. Zhang, K. Xu, and Z. Li, "Owl: A large language model for it operations," 2024. [Online]. Available: https://arxiv.org/abs/2309.09298