AIOS: LLM Agent Operating System

Ziyan Qiu (Elvira)

***Abstract*—AIOS, a Large Language Model (LLM) Agent operating system, embeds large language model into Operating Systems (OS) as the brain of the OS, enabling an operating system “with soul” -- an important step towards AGI. AIOS is designed to optimize resource allocation, facilitate context switch across agents, enable concurrent execution of agents, provide tool service for agents, maintain access control for agents, and provide a rich set of toolkits for LLM Agent developers. The project is open-source at https://github.com/agiresearch/AIOS.**

*Index Terms*—Large language model (LLM), agent operating system, resource allocation, context management, multi-agent execution, tool services, access control.

# I. Introduction

AIOS integrates LLMs into OS to optimize resource allo-cation, facilitate context management, and enable concurrent multi-agent execution. It includes modules like Agent Sche-duler, Context Manager, Memory Manager, Storage Manager, Tool Manager, and Access Manager. AIOS enhances LLM agent performance by combining LLM reasoning with OS-level actions, supporting complex, multi-modal tasks. Future work aims to improve agent-world integration, resource mana-gement, and multi-agent collaboration, making AIOS a pivotal platform for developing and deploying LLM agents.

The paper describes AIOS, an LLM agent operating system designed to integrate and optimize the use of large language models within an OS environment. AIOS addresses several core challenges in the deployment of LLM-based agents, including sub-optimal scheduling, resource allocation, context main-tenance, and the integration of heterogeneous agents. The architecture of AIOS is organized into three distinct layers: the application layer, the kernel layer, and the hardware layer, ensuring a clear delineation of responsibilities across the system.

# II. Related Work

## A. Evolution of Operating Systems

Operating systems have evolved from simple batch processing to advanced multi-tasking systems, incorporating process scheduling, memory, and filesystem management. The advent of GUIs (e.g., Macintosh, Windows, GNOME) made OS more user-centric. Modern OS ecosystems like Android Studio, XCode, and Cloud SDK support efficient application development and deployment. The integration of LLMs promises to enhance human-computer interaction, ushering in intelligent operating systems.

## B. Large Language Model Agents

LLM-based autonomous agents solve complex tasks using natural language instructions, classified into single-agent and multi-agent systems. Single-agent systems handle tasks like travel planning and recommendations, using external tools and APIs. Multi-agent systems involve interactions among agents, which can be cooperative, competitive, or a mix, to solve complex tasks. Examples include role playing, social simulations, and international conflict modeling. These agents operate in both digital and physical environments, leveraging various tools and methods.

日程表

中度可信度描述已自动生成

Fig. 1. An overview of the AIOS architecture.

The AIOS architecture is organized into three distinct layers: the application layer, the kernel layer, and the hardware layer. This structure ensures clear delineation of responsibilities and simplifies system interactions through modular design.

*1) Application Layer:* This layer is where agent applications (e.g., travel or math agents) are developed and deployed. AIOS provides an SDK that abstracts system calls, simplifying development by allowing developers to focus on the core logic and functionalities of their agents.

*2) Kernel Layer:* Divided into the OS Kernel and the LLM Kernel, this layer addresses non-LLM and LLM-specific operations, respectively. The LLM Kernel focuses on context management and agent scheduling, essential for LLM-related activities, and includes modules like the LLM system call interface, agent scheduler, context manager, memory manager, storage manager, tool manager, and access manager.

*3) Hardware Layer:* Comprising the CPU, GPU, memory, disk, and peripheral devices, this layer handles physical comp-onents. The LLM Kernel interfaces with the OS’s system calls to manage hardware resources indirectly, ensuring abstraction, security, and system integrity while leveraging hardware capabilities.

## C. Requirements Definition Work

To address the challenges mentioned, AIOS includes the following modules:

*1) Agent Scheduler:* Prioritizes and schedules agent requests to optimize LLM utilization.The agent scheduler efficiently manages agent requests using strategies like First-In-First-Out (FIFO) and Round Robin (RR) to optimize process execution. By enabling concurrent execution, it balances waiting and turnaround times, ensuring tasks from various agents are interleaved and executed in parallel, minimizing idle times and resource monopolization. Future improvements may include advanced scheduling algorithms that consider dependency relationships between agent requests.

*2) Context Manager:* Supports snapshot and restoration of intermediate generation status in LLMs and context window management.The Context Manager in AIOS handles context snapshot and restoration, ensuring that LLM processes can be paused and resumed without losing progress. It captures the current state of LLM’s generation process and restores it when resources are available. Additionally, it manages long contexts that exceed LLM’s window limit through summarization and other techniques, enhancing LLM’s efficiency in processing extensive information. This ensures optimal resource use and maintains response quality.

*3) Memory Manager:* Provides short-term memory for each agent’s interaction logs. The Memory Manager in AIOS handles short-term memory for each agent, storing data independently and ensuring accessibility only during the agent’s active lifecycle. Other agents cannot access this data without authorization from the Access Manager. Future enhancements may include shared memory pools and hierarchical caches. Unlike the Storage Manager, which deals with long-term data, the Memory Manager enables rapid data retrieval and processing, facilitating swift responses to user queries and interactions without overloading AIOS storage.

*4) Storage Manager:* Persists agent interaction logs to long-term storage for future retrieval. The Storage Manager in AIOS oversees long-term data preservation beyond the active lifespan of agents, using durable storage mediums like local files, databases, and cloud solutions. It ensures data integrity and availability for future use and supports retrieval augmentation. By maintaining user preferences and historical interaction logs, the Storage Manager enhances agent knowledge updates and long-term user experiences.

*5) Tool Manager:* Manages the calling of external API tools by agents.The Tool Manager in AIOS manages a variety of API tools enhancing LLM functionality, including web search, scientific computing, database retrieval, and image processing. These tools cover diverse input and output modalities, facilitating agent development. LLM system calls are categorized into agent, context, memory, and storage syscalls, each with specific functions to efficiently manage agents, context, memory, and data storage within the AIOS ecosystem.

*6) Access Manager:* Enforces privacy and access control policies between agents. The Access Manager in AIOS manages access control among agents by assigning dedicated privilege groups, denying access to resources like interaction history for unauthorized agents. It enhances transparency by maintaining audit logs detailing access requests, agent activities, and changes to access control parameters, safeguarding against privilege attacks.

# III. Conclusions

This paper presents the AIOS architecture, demonstrating its potential in developing and deploying LLM-based agents, fostering a cohesive, effective, and efficient AIOS-Agent ecosystem. The insights and methods showcased in this paper contribute significantly to the ongoing discourse in AI and system research, offering a viable solution to the integration challenges posed by the diverse landscape of AI agents. Future work can build on this foundation to explore innovative approaches to optimize and expand the AIOS architecture to meet the evolving needs of LLM agent development and deployment.

AIOS optimizes resource allocation, context switching, and concurrent multi-agent execution by providing a suite of modules, including the Agent Scheduler, Context Manager, Memory Manager, Storage Manager, Tool Manager, and Access Manager. These modules work together to ensure efficient system management and execution while maintaining data integrity and privacy. The Agent Scheduler balances waiting and turnaround times through concurrent execution, the Context Manager ensures continuity and efficiency of LLM processes through snapshot and restoration mechanisms, the Memory Manager and Storage Manager handle short-term and long-term data storage, respectively, the Tool Manager manages the invocation of various API tools, and the Access Manager enforces privacy and access control policies.

Experimental results show that AIOS performs excellently in concurrent multi-agent execution, ensuring consistency in LLM responses and efficient system scheduling. Compared to traditional sequential execution, AIOS's scheduling mechanisms significantly optimize waiting and turnaround times, especially when handling large LLMs. This highlights the importance of AIOS in accommodating parallel operations of multiple agents.

In conclusion, AIOS is an inspiring work that opens up a broad range of research opportunities. Each research direction can build upon the foundational elements of AIOS, driving progress across the entire field.

References

1. M. Wooldridge and N. R. Jennings, “Intelligent agents: theory and prac-tice,” *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115-152, June 1995, doi: 10.1017/S0269888900008122.
2. N. R. Jennings, K. Sycara, and M. Wooldridge, “A roadmap of agent research and development,” *Autonomous Agents and Multi-Agent Syst-ems*, vol. 1, no. 1, pp. 7-38, Mar. 1998, doi: 10.1023/A:1010090405266.
3. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, “Tropos: an agent-oriented software development methodology,” *Auto-nomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203-236, May 2004, doi: 10.1023/B:AGNT.0000018806.20944.ef.
4. OpenAI. “GPT-4.” OpenAI. https://openai.com/research/gpt-4 (accessed June 9, 2024).
5. Facebook. “Meta. introducing LLaMA: a foundational, 65-billion-para-meter large language model.” Facebook. https://ai.facebook.com/blog/ largelanguage-model-llama-meta-ai (accessed June 9, 2024).
6. Gemini Team Google, “Gemini: a family of highly capable multimodal models,” 2024, *arXiv:2312.11805*.
7. Y. Ge *et al.*, “OpenAGI: when LLM meets domain experts,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 5539-5568, Dec. 2023.
8. L. Ouyang *et al.*, “Training language models to follow instructions with human feedback,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 27730-27744, Dec. 2022.
9. H. W. Chung *et al.*, “Scaling instruction-finetuned language models,” 2022, *arXiv:2210.11416*.
10. H. Touvron *et al.*, “Llama 2: open foundation and fine-tuned chat models,” 2023, *arXiv:2307.09288*.
11. S. Geng, S. Liu, Z. Fu, Y. Ge, and Y. Zhang, “Recommendation as lang-uage processing (RLP): a unified pretrain, personalized prompt & pred-ict paradigm (P5),” in *Proc. 16th ACM Conf. Recommender Systems*, J. Golbeck *et al.*, Ed., Seattle, WA, USA, Sep. 2022, pp. 299-315, doi: 10. 1145/3523227.3546767.
12. T. Kojima, S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, “Large language models are zero-shot reasoners,” *Advances in Neural Information Proce-ssing Systems*, vol. 35, pp. 22199-22213, Dec. 2022.
13. E. Nijkamp *et al.*, “CodeGen: an open large language model for code with multi-turn program synthesis,” 2023, *arXiv:2203.13474*.
14. R. Taylor *et al.*, “Galactica: a large language model for science,” 2022, *arXiv:2211.09085*.
15. S. Hao *et al.*, “Reasoning with language model is planning with world model,” in *Proc. 2023 Conf. Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Ed., Singapore, Dec. 2023, pp. 8154-8173, doi: 10.18653/v1/2023.emnlp-main.507.
16. G. Kim, P. Baldi, and S. McAleer, “Language models can solve computer tasks,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 39648-39677, Dec. 2023.
17. S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, “The programmer’s assistant: conversational interaction with a large language model for software development,” in *Proc. 28th Int. Conf. Intelligent User Interfaces*, Sydney, Australia, Mar. 2023, pp. 491-514, doi: 10.1145/3581 641.3584037.
18. D. Driess *et al.*, “PaLM-E: an embodied multimodal language model,” in *Proc. 40th Int. Conf. Machine Learning*, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlet, Ed., Honolulu, HI, USA, July 2023, pp. 8469-8488.
19. A. Brohan *et al.*, “Do as I can, not as I say: grounding language in robotic affordances,” in *Proc. 6th Conf. Robot Learning*, K. Liu, D. Kulic, and J. Ichnowski, Ed., Auckland, New Zealand, Mar. 2023, pp. 287-318.
20. S. Yao *et al.*, “ReAct: Synergizing Reasoning and Acting in Language Models,” presented at the Eleventh Int. Conf. Learning Representations, Kigali, Rwanda, May 1-5, 2023.
21. N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, “Refle-xion: language agents with verbal reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 8634-8652, Dec. 2023.
22. X. Deng *et al.*, “Mind2Web: towards a generalist agent for the web,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 28091-28114, Dec. 2023.
23. University of Washington. “CSE451: introduction to operating systems.” University of Washington. https://courses.cs.washington.edu/courses/cse 451/16wi/readings/lecture\_readings/LCM\_OperatingSystemsTimeline\_Color\_acd\_newsize.pdf (accessed June 10, 2024).
24. D. M. Ritchie and K. Thompson, “The UNIX time-sharing system,” *Communications of the ACM*, vol. 17, no. 7, pp. 365-375, July 1974, doi: 10.1145/361011.361061.
25. C. A. R. Hoare, “Monitors: an operating system structuring concept,” *Communications of the ACM*, vol. 17, no. 10, pp. 549-557, Oct. 1974, doi: 10.1145/355620.361161.
26. D. R. Engler, M. F. Kaashoek, and J. O’Toole, “Exokernel: an operating system architecture for application-level resource management,” *ACM SIGOPS Operating Systems Review*, vol. 29, no. 5, pp. 251-266, Dec. 1995, doi: 10.1145/224057.224076.
27. C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprog-ramming in a hard-real-time environment,” *Journal of the ACM*, vol. 20, no. 1, pp. 46-61, Jan. 1973, doi: 10.1145/321738.321743.
28. E. W. Dijkstra, “Cooperating sequential processes,” in *The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls*, P. B. Hansen, Ed., New York, NY, USA: Springer, 2002, pp. 65-138, doi: 10.1007/978-1-4757-3472-0\_2.
29. P. J. Denning, “The working set model for program behavior,” *Commu-nications of the ACM*, vol. 11, no. 5, pp. 323-333, May 1968, doi: 10.1145/ 363095.363141.
30. R. C. Daley and J. B. Dennis, “Virtual memory, processes, and sharing in MULTICS,” *Communications of the ACM*, vol. 11, no. 5, pp. 306-312, May 1968, doi: 10.1145/363095.363139.
31. M. Rosenblum and J. K. Ousterhout, “The design and implementation of a log-structured file system,” *ACM Transactions on Computer Systems*, vol. 10, no. 1, pp. 26-52, Feb. 1992, doi: 10.1145/146941.146943.
32. M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, “A fast file system for UNIX,” *ACM Transactions on Computer Systems*, vol. 2, no. 3, pp. 181-197, Aug. 1984, doi: 10.1145/989.990.
33. Y. Ge, Y. Ren, W. Hua, S. Xu, J. Tan, and Y. Zhang, “LLM as OS, agents as apps: Envisioning AIOS, agents and the AIOS-agent ecosystem,” 2023, *arXiv:2312.03815*.
34. T. Schick *et al.*, “Toolformer: language models can teach themselves to use tools,” 2023, *arXiv:2302.04761*.
35. S. Yao and K. Narasimhan. “Language agents in the digital world: opportunities and risks.” Princeton NLP. https://princeton-nlp.github.io/la nguage-agent-impact/ (accessed June 10, 2024).
36. A. Parisi, Y. Zhao, and N. Fiedel, “TALM: tool augmented language models,” 2022, *arXiv:2205.12255*.
37. Q. Tang *et al.*, “ToolAlpaca: generalized tool learning for language models with 3000 simulated cases,” 2023, *arXiv:2306.05301*.
38. R. Nakano *et al.*, “WebGPT: browser-assisted question-answering with human feedback,” 2022, *arXiv:2112.09332*.
39. K. Zhang, H. Zhang, G. Li, J. Li, Z. Li, and Z. Jin, “ToolCoder: teach code generation models to use API search tools,” 2023, *arXiv:2305.04032*.
40. L. Fan *et al.*, “MineDojo: building open-ended embodied agents with internet-scale knowledge,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 18343-18362, Dec. 2022.
41. G. Wang *et al.*, “Voyager: an open-ended embodied agent with large language models,” presented at the NeurIPS 2023 Foundation Models for Decision Making Workshop, New Orleans, LA, USA, Dec. 10-16, 2023.
42. D. A. Boiko, R. MacKnight, and G. Gomes, “Emergent autonomous scientific research capabilities of large language models,” 2023, *arXiv: 2304.05332*.
43. A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schw-aller, “ChemCrow: augmenting large-language models with chemistry tools,” 2023, *arXiv:2304.05376*.
44. W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, “Language models as zero-shot planners: extracting actionable knowledge for embodied agents,” in *Proc. 39th Int. Conf. Machine Learning*, Baltimore, MD, USA, June 2022, pp. 9118-9147.
45. J. Xiang *et al.*, “Language models meet world models: embodied experi-ences enhance language models,” *Advances in Neural Information Proce-ssing Systems*, vol. 36, pp. 75392-75412, Dec. 2023.
46. G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, “CAMEL: communicative agents for ‘mind’ exploration of large language model society,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 51991-52008, Dec. 2023.
47. J. S. Park, J. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: interactive simulacra of human behavior,” in *Proc. 36th Annual ACM Symposium on User Interface Software and Technology*, S. Follmer, J. Han, J. Steimle, and N. H. Riche, Ed., San Francisco, CA, USA, Oct. 2023, pp. 1-22, doi: 10.1145/3586183.3606763.
48. S. Hong *et al.*, “MetaGPT: meta programming for a multi-agent collabo-rative framework,” 2023, *arXiv:2308.00352*.
49. C. Qian *et al.*, “ChatDev: communicative agents for software develo-pment,” 2024, *arXiv:2307.07924*.
50. Q. Wu *et al.*, “AutoGen: enabling next-Gen LLM applications via multi-agent conversation,” 2023, *arXiv:2308.08155*.
51. M. Josifoski *et al.*, “Flows: building blocks of reasoning and collaborating AI,” 2024, *arXiv:2308.01285*.
52. Y. Fu, H. Peng, T. Khot, and M. Lapata, “Improving language model negotiation with self-play and in-context learning from AI feedback,” 2023, *arXiv:2305.10142*.
53. Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, “Improving factuality and reasoning in language models through multiagent debate,” 2023, *arXiv:2305.14325*.
54. C. M. Chan *et al.*, “ChatEval: towards better LLM-based evaluators through multi-agent debate,” 2023, *arXiv:2308.07201*.
55. T. Liang *et al.*, “Encouraging divergent thinking in large language models through multi-agent debate,” 2023, *arXiv:2305.19118*.
56. W. Hua *et al.*, “War and peace (WarAgent): large language model-based multi-agent simulation of world wars,” 2024, *arXiv:2311.17227*.
57. A. Q. Jiang *et al.*, “Mistral 7B,” 2023, *arXiv:2310.06825*.
58. S. Biderman *et al.*, “Pythia: a suite for analyzing large language models across training and scaling,” in *Proc. 40th Int. Conf. Machine Learning*, Honolulu, HI, USA, July 2023, pp. 2397-2430.
59. S. Chen, S. Wong, L. Chen, and Y. Tian, “Extending context window of large language models via positional interpolation,” 2023, *arXiv:2306. 15595*.
60. B. Peng, J. Quesnelle, H. Fan, and E. Shippole, “YaRN: efficient context window extension of large language models,” 2023, *arXiv:2309.00071*.
61. G. Mialon *et al.*, “Augmented language models: a survey,” 2023, *arXiv: 2302.07842*.
62. GitHub. “LangChain.” GitHub. https://github.com/langchain-ai/langcha in (accessed June 10, 2024).
63. Rapid API. “Rapid API Hub.” Rapid API. https://rapidapi.com/hub (accessed June 10, 2024).
64. K. Thompson, “Reflections on trusting trust,” *Communications of the ACM*, vol. 27, no. 8, pp. 761-763, Aug. 1984, doi: 10.1145/358198.3582 10.
65. S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. R. Sadeghi, and B. Shastry, “Towards taming privilege-escalation attacks on android,” presented at the 19th Annual Network & Distributed System Security Symposium, San Diego, CA, USA, Feb. 5-8, 2012.
66. J. Banks and T. Warkentin. “Gemma: introducing new state-of-the-art open models.” Google. https://blog.google/technology/developers/gemm a-open-models/ (accessed June 10, 2024).
67. K. Papineni, S. Roukos, T. Ward, and W. J. Zhu, “BLEU: a method for automatic evaluation of machine translation,” in *Proc. 40th Annual Meeting of the Association for Computational Linguistics*, P. Isabelle, E. Charniak, and D. Lin, Ed., Philadelphia, PA, USA, July 2002, pp. 311-318, doi: 10.3115/1073083.1073135.
68. T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, “BERT-Score: evaluating text generation with BERT,” 2020, *arXiv:1904.09675*.