IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

Impact Factor: 7.53

Volume 4, Issue 4, April 2024

AI based Operating System

Ashutosh Kumar[1], Ankit Kumar Singh[2], Ashima Mehta[3]
Students, Department of Computer Science and Engineering[1,2]
Professor, Department of Computer Science and Engineering[3]
Dronacharya College of Engineering, Gurugram, India

Abstract: *This review paper examines the potential applications of artificial intelligence (AI) in the development of an operating system (OS) that not only provides functionalities for software and hardware management, as well as common system services, but also integrates intelligent management capabilities. Advanced AI techniques such as expert systems, neural networks, pattern recognition, fuzzy logic prediction, and other AI features can be leveraged in the creation of such an AI-based OS. Features of an AI-based OS may include abstraction, associative AI thinking, perceptual intelligence, contextual imagination, context-specific search, context priming, and various other AI methodologies. Integrating and using smart agents based on large language models (LLMs) face tough challenges that affect how well they work. These challenges include problems like not scheduling and sharing resources efficiently for agent requests on the LLM, difficulties in keeping track of context when agents interact with the LLM, and the complexity of blending different agents with various skills and specialties. The rising number and complexity of agents often cause problems like bottlenecks and inefficient use of resources. To tackle this, our paper introduces AIOS, a unique operating system that incorporates a large language model. This innovation gives the operating system a kind of "intelligence" and brings us closer to achieving Artificial General Intelligence (AGI). AIOS is designed to better manage resources, help agents switch between tasks smoothly, allow multiple agents to work at the same time, offer tools for agents, and control access to the system. We explain how AIOS works, highlight the main issues it solves, and provide a basic overview of its design and implementation.*

Keywords: AIOS, LLM, fuzzy logic prediction, intelligence management system

I. INTRODUCTION

Human-computer interaction has evolved in a notable manner over time. Initially, users struggled to meet the expectations of computers, grappling with technologies like punch cards and command-line interfaces that demanded precision in commands. The emergence of GUIs (Graphical User Interfaces) represented a significant advancement, making computers more accessible to those without technical expertise.

However, GUIs have only partially closed the gap between humans and computers, as users still navigate menus and application structures, essentially adhering to the predefined logic set by developers. Many individuals encounter frustration when searching for specific functions within these interfaces. GUIs require users to adopt a developer-centric mindset, thereby restricting the user experience and imposing certain behavioural norms.

The introduction of LLMs marks a phase where computers can adapt fully to users. These conversational interfaces necessitate no change in user behaviour, facilitating a genuinely intuitive and personalized interaction. LLMs excel in understanding and manipulating language and data. An LLM-based operating system can dynamically respond to natural language, rendering traditional apps and the App Store obsolete. LLMs can create personalized applications on-demand, removing installation needs, and effortlessly manage various data formats, making specific apps redundant.

LLM (Language Model) based operating systems represent an intriguing concept that leverages natural language understanding and generation capabilities to interact with users in a more intuitive and human-like manner.

Nevertheless, substantial obstacles remain to be overcome. Addressing issues such as local execution to safeguard privacy, optimizing memory and performance, and establishing a local vector database for long-term memory are crucial components that require attention. By confronting these challenges directly, we can lay the groundwork for a future in which Large Language Models (LLMs) seamlessly integrate into our operating systems. This integration has the potential to revolutionize the manner in which we interact with computers, ushering in a new era of personalized and intelligent digital experiences.

OS-APP Ecosystem	AIOS-Agent Ecosystem	Explanation
Kernel	LLM	The core of AIOS, providing supportive services for Agent Applications (AAPs).
Memory	Context Window	Short-term memory of the current session, such as the prompt-response history of this session.
Memory Management	Context Selection and Management	Select relevant context for the session, and manage the context such as adding, deleting and changing context information.
File	External Storage	Long-term storage of the AIOS's history sessions, user profiles, factual knowledge, etc.
File System	Retrieval-augmentation	Retrieve relevant information from long-term storage.
Devices/Libraries	Hardware/Software Tools	Help systems interact with the external world (both physical and virtual/digital world).
Driver/API	Tool-Driver/Tool-API	Serves as the interface for LLM/Agents to access and use the tools, usually in the form of prompts.
OS SDK	AIOS SDK	Assist users in developing Agent Applications.
User Interface (UI)	User Prompt/Instruction	Instruction(s) given by user to the system in (sometimes semi-structured) natural language (NL) to perform specific tasks. The NL instruction may be converted from users' non-NL instructions such as clicks.
Application (APP)	Agent Application (AAP)	A diverse world of AI Agents.

Comparison of OS-APP Ecosystem and AIOS-Agent Ecosystem

III. AIOS IMPLEMENTATION

In this section, we start by providing an overview of the basic design and implementation of every module within the LLM kernel. Following that, we introduce the LLM system calls, which include crucial functions for each module. Finally, we delve into the exploration of the AIOS SDK, with the goal of simplifying the development process for agent developers.

Agent Scheduler:

Agent scheduler is designed to manage the agent requests in an efficient way. Consider the various agents (denoted as A, B, and C) , each of which has several execution steps. In the sequential execution paradigm, the agent tasks are processed in a linear order, where steps from a same agent will be processed first. This can lead to potential increased waiting times for tasks queued later in the sequence. The agent scheduler employs strategies such as First-In-First-Out (FIFO)7 , Round Robin (RR)8 , and other scheduling algorithms to optimize this process. Through concurrent execution, the scheduler significantly balances waiting time and turnaround time of each agent, as tasks from different agents are interleaved and executed in parallel. This concurrent approach is visualized through a timeline where tasks from different agents are processed in an interleaved manner (e.g., A1, B1, C1, B2, A2, A3, C2, C3), ensuring that no single agent monopolizes the processing resources and that idle times are minimized. Apart from implementing the traditional scheduling algorithms, more complex scheduling algorithms considering the dependency relationships between agent requests can also be incorporated, which can be considered in the future.

Context Manager:
The context manager oversees the context provided to the LLM and the generation process based on the given context. Its main responsibilities include two key functions: context snapshot and restoration, and context window management.

Context Snapshot and Restoration:
Given that scheduler algorithms may involve time quantum operations (e.g., Round-Robin), agent requests may be suspended even if the LLM has not fully generated a response. To address this, AIOS provides snapshot and restoration mechanisms in the context manager . These mechanisms preserve the state of the LLM's generation process, ensuring accurate resumption once resources are available again. For instance, using beam search process, the context manager captures and stores the current state of the LLM's beam search tree when the generation process is suspended. Upon resumption, the restoration function reloads the saved state, allowing the LLM to continue from the suspension point to reach the final answer. This ensures that the temporary suspension of an agent's request does not result in progress loss, optimizing resource use without compromising response quality or efficiency.

Context Window Management:
To handle challenges posed by long contexts exceeding the LLM's context window limit, the context manager must manage potential expansion of the window. AIOS's context manager supports basic text summarization and incorporates other expansion techniques to manage the context window. This enhances the LLM's ability to process and understand extensive contexts without compromising information integrity or relevance.

Memory manager:
the memory manager oversees short-term memory within an agent's lifecycle, ensuring that data is stored and accessible only while the agent is active—either waiting for execution or during runtime. In the current AIOS setup, each agent's memory is stored independently, with other agents having no direct access unless authorized by the access manager. More complex memory mechanisms, like shared memory pools among agents or hierarchical caches, could be considered and integrated into AIOS in the future. Compared to the storage manager discussed later, the memory manager allows for rapid data retrieval and processing. This capability facilitates quick responses to user queries and interactions without burdening AIOS storage.

Storage Manager:
In contrast, the storage manager handles the long-term storage of data. It oversees the storage of information that needs to be preserved beyond the active lifespan of any single agent. AIOS achieves this permanent storage using durable mediums like local files, databases, or cloud-based solutions, ensuring data integrity and availability for future reference or analysis. The storage manager also supports retrieval augmentation. By storing user preferences and maintaining historical interaction logs, it enriches agent knowledge updates and enhances the long-term user experience.

Tool Manager:
The tool manager in the AIOS system oversees a wide range of API tools that enhance the functionality of LLMs. The tool manager integrates commonly used tools from various sources and categorizes them into different groups, covering areas such as web search, scientific computing, database retrieval, and image processing. This approach allows the managed tools to encompass various modalities of input and output, including both image and text, thereby facilitating agent development within the AIOS ecosystem.

Access Manager:
The access manager coordinates access control operations among different agents by managing a dedicated privilege group for each agent. Agents excluded from an agent's privilege group are denied access to its resources, such as interaction history. Additionally, to improve system transparency, the access manager compiles and maintains auditing

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 4, April 2024

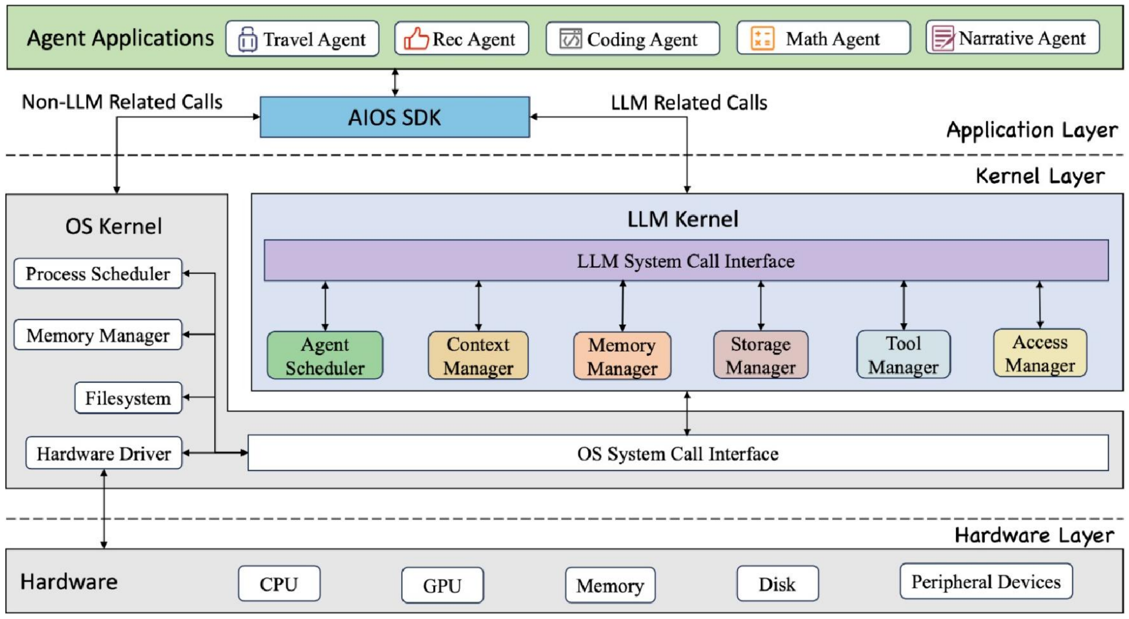
logs. These logs capture detailed information about access requests, agent activities, and any modifications to access control parameters. This helps safeguard against potential privilege attacks.

LLM System Call:
The LLM system call interface within the LLM kernel is crafted to provide fundamental LLM call operations. This interface serves as a connection between intricate agent requests and the execution of various kernel modules. Similar to OS system calls, the LLM system calls offer a range of basic functions that cover agent management, context handling, memory and storage operations, and access control. The list of LLM system calls can be extended in the future to support additional operations.

AIOS SDK:
The AIOS SDK is crafted to empower developers with a versatile toolkit for creating sophisticated agent applications within the AIOS environment. This SDK encompasses a wide range of functionalities, from initializing agents and managing their lifecycles to facilitating complex operations such as resource monitoring and generating plans for agent tasks. Similar to any operating system, enriching the SDK to be comprehensive and user-friendly is an ongoing effort. The current SDK functions supported in AIOS , and they will continue to be updated and expanded to meet the evolving needs of agent applications. This development endeavor aims to provide developers with the necessary tools to fully leverage the potential of their agent applications within the AIOS framework.

III. PROPOSED SYSTEM

This paper suggests the AIOS architecture, which aims to simplify the development and use of LLM-based agents, making the AIOS-Agent system work better together. The ideas and methods shared here add to the ongoing conversation in AI and system research, offering a practical solution to the challenges of integrating various AI Agents. There's room for more exploration, building on these ideas to enhance and expand the AIOS architecture, adapting to the changing needs of using LLM agents.



AIOS Layers:
As shown in Figure 2, our AIOS architecture consists of three main layers: the application layer, the kernel layer, and the hardware layer. This structured layout ensures that each layer has its own set of responsibilities within the system.

Each layer hides the complexities of the layers beneath it, making interaction easier through interfaces or specific modules. This approach enhances modularity and simplifies communication between different layers of the system.

- Application Layer: At the application layer, various agent applications like travel agents or math agents are created and put into use. Here, AIOS offers the AIOS SDK, which simplifies the development process for agent developers by providing a higher level of abstraction for system calls. This SDK provides a comprehensive toolkit that hides the complexities of lower-level system functions, allowing developers to concentrate on the core logic and features of their agents. This streamlined approach makes the development process more efficient.
- Kernel Layer: The kernel layer comprises two main parts: the OS Kernel and the LLM Kernel. Each component serves specific needs, with the OS Kernel handling non-LLM operations and the LLM Kernel focusing on tasks unique to LLMs. This setup allows the LLM Kernel to concentrate on essential LLM-related functions like context management and agent scheduling, which aren't typically handled by standard OS kernels. Our work primarily enhances the LLM Kernel without major changes to the existing OS kernel structure. The LLM Kernel includes key modules such as the LLM system call interface, agent scheduler, context manager, memory manager, storage manager, tool manager, and access manager. These modules cater to various execution requirements of agent applications, ensuring efficient management and execution within the AIOS framework.
- Hardware Layer: The hardware layer includes the physical parts of the system like the CPU, GPU, memory, disk, and peripheral devices. It's important to understand that the system calls from the LLM kernel don't directly communicate with the hardware. Instead, they interact with the OS's system calls, which then handle the hardware resources. This indirect interaction adds a layer of abstraction and security, allowing the LLM kernel to use hardware capabilities without needing to manage the hardware directly. This setup helps maintain the system's integrity and efficiency.

IV. FEATURES

- **Monitor and optimize their own performance:** The OS continuously monitors its own performance metrics and adjusts settings or allocates resources to optimize efficiency and effectiveness.
- **Smart Process Management:** Utilizing AI algorithms to manage and prioritize processes running on the system, ensuring efficient utilization of resources.
- **Smart Memory Management:** AI algorithms are used to dynamically allocate and manage memory resources, optimizing performance and preventing memory-related issues such as leaks or fragmentation.
- **Predicting results:** Using machine learning or predictive analytics to anticipate outcomes or behaviors based on historical data and current system state.
- **Using the right resources:** Automatically selecting and allocating appropriate system resources (such as CPU, memory, or storage) based on the requirements of running processes or tasks.
- **Making sure that resources are available by predicting before a process asks for it:** Anticipating resource needs before they are requested by processes, preemptively allocating resources to minimize delays or bottlenecks.
- **Suggesting alternative options to a problem:** Providing users with alternative solutions or courses of action based on analysis and evaluation of the problem at hand.
- **Search internally and externally for information to help themselves:** Accessing both internal and external data sources to gather information or assistance in resolving issues or executing tasks.
- **Deriving a solution to a problem by themselves:** Utilizing AI algorithms to autonomously analyze and solve problems without human intervention.
- **Healing themselves and getting immune to viruses after attack:** Employing AI-based cybersecurity measures to detect, respond to, and recover from security threats such as viruses or malware, potentially adapting over time to become more resilient.

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

Impact Factor: 7.53

Volume 4, Issue 4, April 2024

- **Communicate with other OS:** Facilitating communication and collaboration between different instances of the OS or with other systems and devices, enabling seamless integration and interoperability.

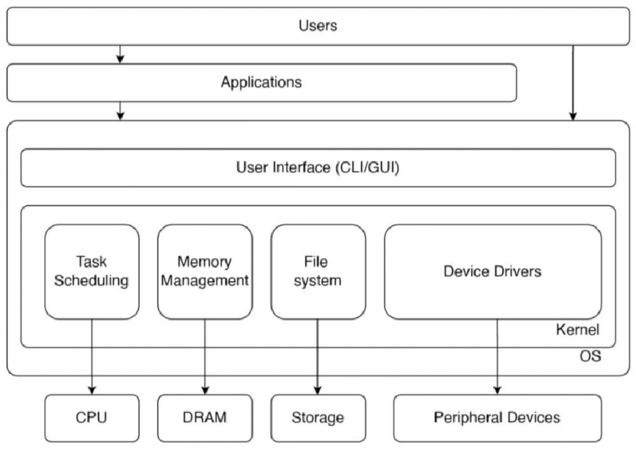
These features collectively represent a sophisticated level of AI integration into the operating system, enabling it to adapt, optimize, and secure itself while providing enhanced functionality and user experience.

V. CURRENT PROGRESS

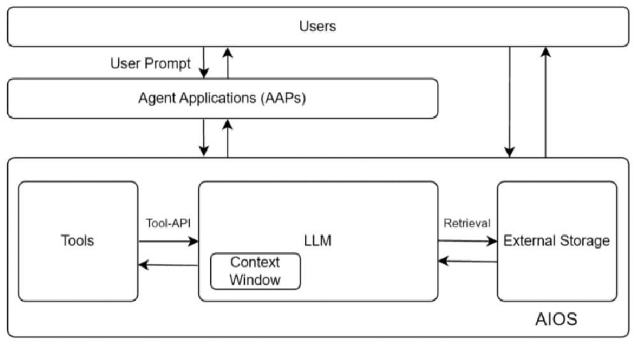
Several organizations have developed projects centered around AI-based operating systems:

- Black Swan: An Israeli software services developer has introduced a platform positioned as an enterprise AI operating system. It combines deep learning, machine learning, natural language processing, neural networks, and data operations.
- Algorithmia's AI OS: Algorithmia's CEO presented at the 2017 GeekWire Tech Cloud Summit about "Building an Operating System for AI."
- NeurOS: A neural network-based OS called NeurOS has been developed. It performs various functions such as perception, pattern learning and recognition, working memory, imagination, prediction, context priming, attention, abstraction, classification, associational thinking, and behavior. NeurOS applications are portable, scalable, networkable, extensible, and embeddable.
- Cognition: An AIOS-based operating system named Cognition has been developed. It serves as an intelligent assistant, electronic advisor, and offers engines and chatbots.

VI. BENEFITS



(a) Architecture of Operating System (OS).



(b) Architecture of Large Language Model as OS (LLMOS) for AIOS.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 4, April 2024

- User Interaction: LLM-based OS could revolutionise user interaction by allowing users to communicate with their devices using natural language, eliminating the need to memorize commands or navigate complex menus. This could make computing more accessible to a wider range of people, including those who are not tech-savvy.
- Personalization: Since language models can understand context and user preferences, an LLM-based OS could offer highly personalized experiences tailored to individual users. It could anticipate user needs and provide relevant information or assistance proactively.
- Workflow Optimization: By understanding user intents and tasks, an LLM-based OS could optimize workflows and automate routine tasks, thereby improving productivity and efficiency.
- Learning and Adaptation: Language models have the ability to learn from interactions, which means an LLM-based OS could continuously improve its understanding and responsiveness over time, providing a more seamless user experience.
- Integration with Other Systems: LLM-based OS could potentially integrate with other AI systems and services, such as virtual assistants, smart home devices, or enterprise software, to offer a unified and cohesive computing environment.

VII. FUTURE SCOPE

AIOS represents the future of operating systems. Its AI-based nature holds immense potential for assisting in defense systems, biomedical research, scientific endeavors, education, and various other fields.

The primary advantage of this OS lies in its continuous learning ability powered by AI. Similar to how humans learn from their mistakes, this OS will improve over time, becoming increasingly efficient as it evolves.

Furthermore, AI helps users complete complex tasks quickly. This makes modifying the operating system easier—you won't have to wait for updates or restart your computer. The OS updates itself automatically, reducing downtime. Plus, changing the AI won't cause crashes like it does in traditional operating systems.

The AIOS represents a significant upgrade from traditional operating systems. It can reduce operation time, manage parallel processes more efficiently, improve memory management, enhance security measures, and better understand user context.

VIII. CONCLUSION

This paper introduces a fresh perspective on the future of computing within the AIOS-Agent ecosystem, where the Large Language Model (LLM) serves as the core of AIOS. This innovative approach represents a significant departure from the traditional OS-APP setup, ushering in a new era in technology where AI seamlessly merges with conventional computing systems. The envisioned AIOS-Agent ecosystem isn't just an incremental change but a fundamental shift in how we engage with technology.

By placing the LLM at the system level, treating Agents as applications, Tools as devices/libraries, and Natural Language as the Programming Interface, we redefine the interaction between users, developers, and the digital world. This paradigm shift aims to democratize software development and access, enabling users and developers to program Agent Applications (AAPs) using natural language. This accessibility contrasts sharply with the traditional ecosystem, where software development is limited to those with specialized programming skills.

Furthermore, the discussion on single and multi-agent systems, as well as human-agent interactions, demonstrates the potential of AIOS in enhancing productivity, creativity, and decision-making across various domains. Looking ahead, the proposed strategic roadmap, inspired by the developmental trajectory of the traditional OS-APP ecosystem, offers a practical and systematic approach to the evolution of AIOS and its Agent Applications. This roadmap not only guides future development and research in this field but also anticipates the challenges and opportunities ahead.

REFERENCES

[1] Dennis M. Ritchie and Ken Thompson. The unix time-sharing system. Commun. ACM, 17(7):365–375, jul 1974.
[2] Charles Antony Richard Hoare. Monitors: An operating system structuring concept. Communications of the ACM, 17(10):549–557, 1974.

[3] Dawson R Engler, M Frans Kaashoek, and James O'Toole Jr. Exokernel: An operating system architecture for application-level resource management. ACM SIGOPS Operating Systems Review, 29(5):251–266, 1995.

[4] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1):46–61, 1973.

[5] Edsger W Dijkstra. Cooperating sequential processes. In The origin of concurrent programming: from semaphores to remote procedure calls, pages 65–138. Springer, 2002.

[6] Peter J Denning. The working set model for program behavior. Communications of the ACM, 11(5):323–333, 1968.

[7] Robert C Daley and Jack B Dennis. Virtual memory, processes, and sharing in multics. Communications of the ACM, 11(5):306–312, 1968.

[8] Mendel Rosenblum and John K Ousterhout. The design and implementation of a log-structured file system. ACM Transactions on Computer Systems (TOCS), 10(1):26–52, 1992.

[9] UW:CSE451. History of Operating Systems, 2023.