iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XII **Month of publication:** December 2025

DOI: https://doi.org/10.22214/ijraset.2025.76302

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XII Dec 2025- Available at www.ijraset.com

A Comprehensive Literature Survey on Large Language Model-Driven Operating Systems

Dr. S Gunasekaran[1], Ms. Kavitha S[2], Abhinav A R[3], Aadith Krishnan K[4], Reyzon Joseph K R[5]
[1]*Professor in CSE, Ahalia School of Engineering and Technology, Palakkad, Kerala*
[2]*Assistant Professor in CSE, Ahalia School of Engineering and Technology, Palakkad, Kerala*
[3, 4, 5]*Ahalia School of Engineering and Technology, Palakkad, Kerala*

Abstract: ***Recent progress in artificial intelligence has reshaped the design philosophy of modern operating systems, with Large Language Models (LLMs) emerging as a key enabling technology. Existing operating systems primarily rely on predefined graphical and command-line interactions, which limits their ability to adapt to user preferences, understand context, and respond to complex intent. Early natural language–based interfaces attempted to address these limitations; however, they lacked robust contextual reasoning and system-level intelligence required for dynamic computing environments. As workloads and user expectations continue to grow in complexity, there is an increasing need for operating systems capable of understanding and acting upon natural language instructions in a reliable and transparent manner. This paper examines operating system architectures that embed LLMs within core system components to facilitate intent-driven interaction and intelligent automation. The study surveys representative frameworks such as AIOS, PEROS, Compressor–Retriever architectures, and Herding LLaMaS, emphasizing how these designs translate linguistic input into executable operating system functions. Key capabilities explored include continuous context retention, dynamic allocation of computational resources, and semantic coordination between software services and underlying hardware. These design choices aim to enhance usability while reducing the cognitive burden placed on end users. The discussion further addresses essential system considerations, including interpretability of model-driven decisions, user-specific adaptation, protection of sensitive data, and scalability across diverse computing platforms. Emphasis is placed on explainable reasoning mechanisms that help users and developers understand system behavior, particularly in safety- and security-critical scenarios. By integrating adaptable intelligence with transparent control logic, LLM-enhanced operating systems can improve trust and operational efficiency.***

Keywords: Large Language Models, Intelligent Operating Systems, Natural Language Interaction, Explainable Artificial Intelligence, Adaptive Computing, Human–Computer Interaction

I. INTRODUCTION

The operating system forms the central coordination layer of modern computing systems, enabling communication between users, software applications, and physical hardware components. Traditional operating systems primarily employ graphical and command-line interfaces that require users to issue structured commands or navigate predefined menus. Such interaction mechanisms often limit accessibility for non-technical users and constrain the system's ability to adapt dynamically to varying user intentions and operational contexts. Recent breakthroughs in Large Language Models (LLMs) have opened new possibilities for transforming how users interact with computing platforms. By supporting natural language communication, LLMs enable users to express goals and requests in an intuitive manner, allowing systems to shift from explicit command execution toward intent-aware operation. This capability allows operating systems to infer contextual meaning, interpret user objectives, and translate high-level language input into concrete system-level actions without requiring detailed procedural knowledge from the user.

Current research trends focus on embedding LLM capabilities directly into operating system architectures, thereby enabling more adaptive and intelligent system behaviour. LLM-driven operating systems are designed not only to process user instructions but also to learn from interaction patterns, optimize resource utilization, and coordinate across diverse hardware and software environments. Such systems promise improved efficiency, personalization, and usability in increasingly complex computing scenarios.

Despite these advantages, several challenges remain. The opaque decision-making processes of LLMs raise concerns related to transparency, reliability, and system security. Ensuring that operating system decisions are interpretable and trustworthy is essential, particularly in critical or multi-user environments. This paper reviews existing work on LLM-based operating systems and analyses the role of explainable mechanisms, user-centric personalization, and architectural design choices in developing dependable and intelligent operating platforms.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XII Dec 2025- Available at www.ijraset.com

II. LITERATURE REVIEW

This section reviews significant research contributions that form the foundation of LLM-driven operating systems.

A. *AIOS – LLM Agent Operating Systems*

The AIOS (LLM Agent Operating System) study explores a new approach to operating system design aimed at supporting the large-scale deployment of LLM-driven agents in a secure and efficient manner. Rather than permitting agents to access language models and system resources in an ad hoc fashion, AIOS introduces a structured framework in which LLM agents are managed similarly to traditional operating system processes and governed by a kernel-level control layer. The study highlights limitations in existing agent-based systems, including inefficient resource management, absence of scheduling mechanisms, and inconsistent execution latency, all of which restrict their ability to scale. In response to these challenges, AIOS adopts established operating system principles to provide controlled resource allocation, execution isolation, and coordinated parallelism. This architecture enables the reliable and concurrent operation of a large number of agents while maintaining predictable performance and system stability.

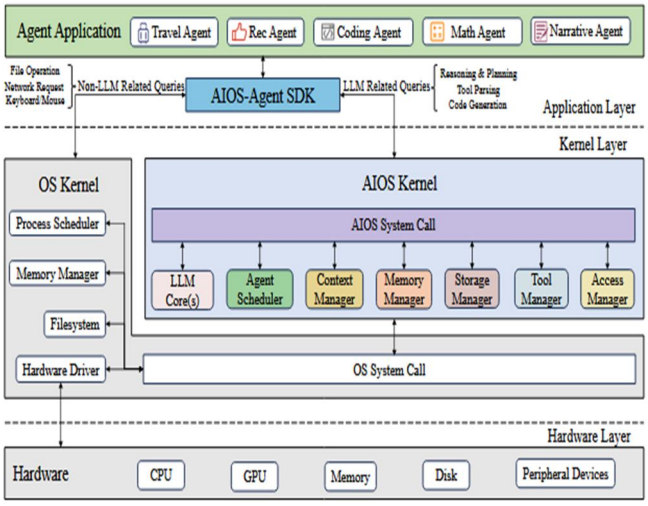


Fig. 1. Overview of the AIOS architecture and agent–kernel interaction

The proposed system follows a multi-layered architectural approach that distinctly separates high-level agent behavior from underlying resource management functions. At the upper layer, the platform provides software development interfaces that enable developers to implement agent logic independently of the complexities associated with hardware and system resources. The lower, kernel-oriented layer oversees critical functions such as controlled access to resources, execution isolation, and task scheduling. In addition, a dedicated context management component supports saving and restoring the internal state of LLM inference, allowing agent execution to be suspended and later resumed without loss of information. Through this structured design, AIOS manages LLM-centric workloads in a manner analogous to how traditional operating system kernels coordinate processes and threads.

1) Resource Decomposition and Agent Serving: Agent requests are broken down into smaller, well-defined operating system–level operations that can be scheduled independently. Execution threads are associated with these operations, allowing agent tasks to be handled in a modular and controlled manner. This fine-grained structuring enables more consistent and predictable management of computational resources, including processing power, memory usage, and contextual state assigned to each agent during execution.

2) Scheduling and Memory Management: The AIOS kernel applies classical scheduling policies, including First-In–First-Out and Round Robin mechanisms, to coordinate the concurrent execution of multiple agents in a fair manner while avoiding starvation. When an agent exhausts its allocated execution time, a context switch is triggered, mirroring the multitasking behavior of conventional operating systems. To manage memory efficiently, the system incorporates an eviction strategy inspired by Least Recently Used (LRU) policies, transferring agent memory segments to secondary or external storage when predefined resource thresholds are reached. Additionally, agent interaction histories and accumulated knowledge are maintained in a vector-based storage system, supporting persistence, retrieval, and long-term reuse across agent executions.

©IJRASET: All Rights are Reserved

3) Performance Evaluation and Metrics: Experimental evaluations indicate that AIOS delivers as much as a 2.1-fold improvement in agent-serving throughput while maintaining, and in some cases enhancing, performance accuracy on benchmark suites such as HumanEval, MINT, GAIA, and SWE Bench-Lite. The system is capable of managing approximately 2,000 agents executing concurrently, exhibiting near-linear growth in response latency as load increases. Performance metrics including success rate, execution delay, and overall throughput are analyzed to show that adopting an operating-system–inspired control framework does not compromise the reasoning effectiveness of LLM-based agents.

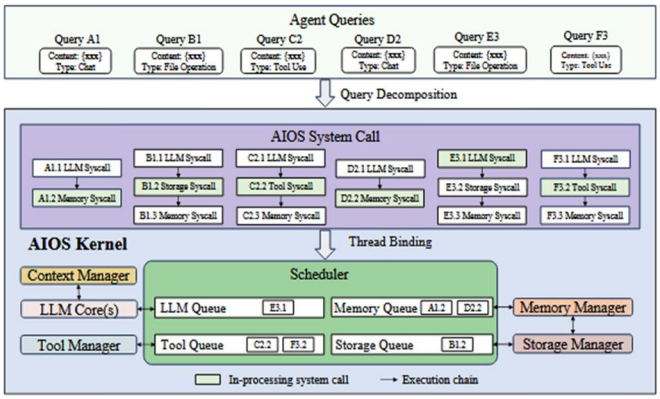


Fig. 2. AIOS system calls are dispatched and scheduled

In summary, the AIOS study establishes a solid conceptual and technical basis for incorporating LLM-based agents within an operating system–style framework. It demonstrates that managing language models, contextual state, and memory at the kernel level can markedly enhance system scalability and operational stability. As a result, the work serves as a valuable point of reference for research and development efforts focused on building operating systems powered by large language models.

B. PEROS – Personalized Self-Adapting Operating Systems in the Cloud

The PEROS study investigates the design of an operating system that emphasizes personalization, autonomous adaptation, and strong privacy guarantees, particularly within cloud-centric and multi-device computing settings. Conventional operating system architectures are largely static, offering limited capability to adjust to individual user behavior or support natural language interaction as a core system feature. PEROS contends that the growing scale of connected devices, software services, and machine learning workloads necessitates operating systems capable of modeling user context and preferences while dynamically optimizing their own configuration and behavior.

The authors advocate for moving away from inflexible interaction models toward declarative interfaces powered by large language models, allowing users to specify desired outcomes using natural language rather than precise system commands. Within the PEROS framework, LLMs are employed to analyze user intent and convert high-level requests into corresponding kernel-level policies and operating system configuration actions.

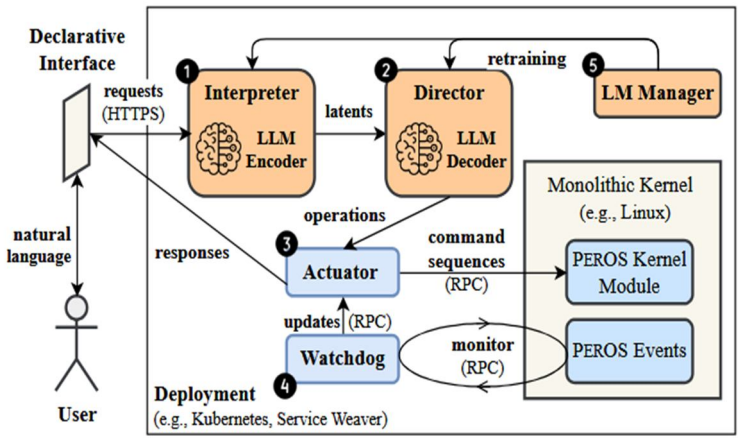


Fig. 3. Prototype of PEROS with the declarative interface powered by LLMs

1) Architecture and Self-Adaptive Kernel: PEROS adopts a microkernel-based design that limits the responsibilities of the core kernel to essential system functions, promoting modularity and fault isolation. Personalization mechanisms and operating system services are implemented as independent components, strengthening both system resilience and security. By continuously monitoring user interactions and workload characteristics, the kernel dynamically refines scheduling decisions, resource distribution, and system settings. This adaptive capability enables the operating system to prioritize performance goals such as low latency, reduced power consumption, or higher throughput in response to changing usage conditions.

2) Cloud Integration and Thin Clients: The architecture targets cloud-oriented deployment models in which core operating system intelligence resides within centralized infrastructure, while user devices function primarily as lightweight clients. PEROS components are deployed on containerized or serverless platforms, such as Kubernetes, enabling elastic scaling to support large user populations. End devices transmit natural language requests to the backend services, where the PEROS system processes intent, formulates execution plans, and carries out the corresponding operating system–level actions.

3) Privacy and Security Mechanisms: One of the primary innovations introduced by PEROS is a data handling framework centered on user privacy. Personal information is maintained within user-governed repositories, referred to as "Databoxes," which are frequently stored outside cloud infrastructure. All system interactions are designed to be verifiable through cryptographic mechanisms, providing transparency and accountability. In addition, the use of privacy-preserving machine learning techniques allows the system to deliver personalized functionality without directly revealing sensitive user data. This approach supports robust personalization while maintaining user trust and compliance with data protection regulations.

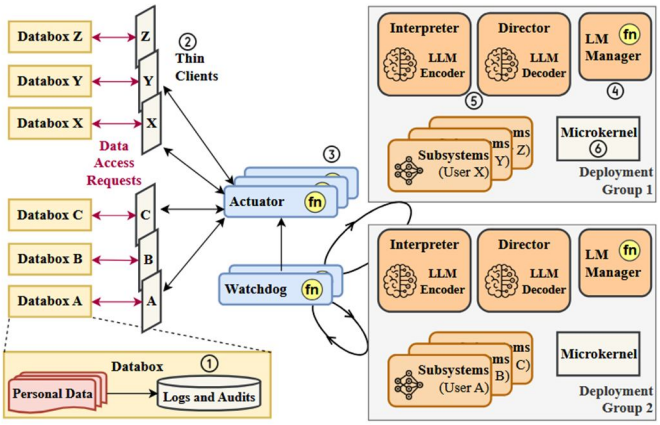


Fig. 4. PEROS architecture overview

Experimental prototypes demonstrate that PEROS is capable of dynamically modifying kernel-level configurations during execution, leading to enhanced user experience and sustained system responsiveness across varied workload conditions. By integrating personalization mechanisms, natural language–based interaction, and modular operating system architecture, the work offers an important contribution to the development of user-focused operating systems built around large language models.

C. Compressor–Retriever Architecture for LLM-Based Operating Systems

In contrast to the architectural and personalization emphases of AIOS and PEROS, the Compressor–Retriever work targets a fundamental constraint of large language models—their limited context capacity. LLM-based operating systems often need access to extensive user histories, system logs, and evolving state information, which cannot be accommodated within a single model invocation. To address this challenge, the study introduces a model-independent compression and retrieval framework that allows LLM-enabled operating platforms to preserve and reason over long-term contextual information without surpassing memory or prompt-size limitations. The authors point out that simplistic strategies, such as supplying the language model with the complete interaction history, lead to excessive computational overhead and are not practical in real systems. To overcome this limitation, the paper proposes a layered context compression strategy in which historical interactions are converted into compact embedding representations and maintained within an expandable memory repository. When a new request is issued, the system retrieves only the most relevant compressed context segments and presents them to the LLM for reasoning.

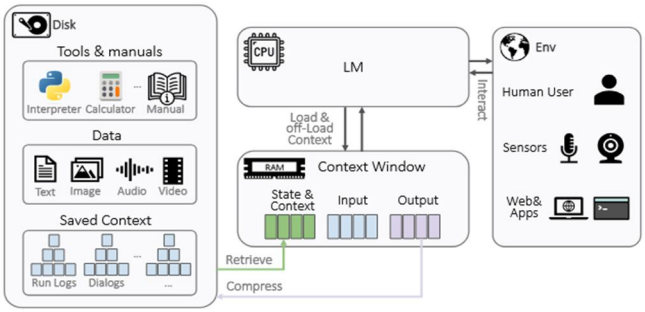


Fig. 5. Architecture showing context life cycle management in a language-model-based OS

1) Context Compression: Historical system events, user interactions, and preference data are divided into segments and processed through an encoding model to produce compact latent representations. These embeddings serve as semantic abstractions of the original content rather than raw text. The compression process is organized hierarchically, allowing the system to retain both detailed short-term information and higher-level long-term summaries within its memory store.

2) Sparse Retrieval and Attention: During inference, when the operating system generates a response, a retrieval component evaluates the similarity between the current input and the stored embeddings to identify the most relevant memory segments. Hierarchical storage structures and segmented attention mechanisms ensure that the model focuses only on the most informative portions of its stored knowledge while filtering out irrelevant data. As a result, the effective context size remains bounded even as the total accumulated history continues to expand.

3) Experimental Evaluation: The proposed framework is assessed using in-context learning reasoning benchmarks, where results demonstrate that the compression-and-retrieval approach retains approximately 75% of the accuracy achieved by a full-context baseline. This indicates that a large fraction of useful information can be maintained through compressed representations without repeatedly supplying the model with complete historical data. Additionally, the evaluation shows that the method scales efficiently with available hardware resources and supports long-term contextual reasoning without requiring additional specialized subsystems.

Mode	Accuracy
0-shot	0.250
6-shot	0.578
Compressor -Retriever	0.429

Fig. 6. Accuracy comparison between full-context models and compressor–retriever models on ICL tasks

D. Paper 4: Herding LLaMaS – LLMs as Operating System Modules

The Herding LLaMaS study addresses the challenge of managing increasingly diverse hardware environments and enabling operating systems to accommodate new devices without extensive manual configuration. Conventional operating systems depend on fixed drivers and carefully tuned heuristics to handle device scheduling, data placement, and performance optimization. However, as modern infrastructures span data centers, cloud platforms, and edge devices, maintaining these handcrafted rules becomes difficult and error-prone. To overcome this limitation, Herding LLaMaS introduces an approach in which large language models function as operating system components capable of understanding hardware characteristics through semantic descriptions. Rather than embedding rigid policies into the OS, system designers or hardware vendors supply natural language specifications describing device properties, such as performance traits or memory behavior. These descriptions are transformed into embedding representations that inform a downstream decision-making model, which subsequently determines memory allocation, task scheduling strategies, and data movement across heterogeneous devices.

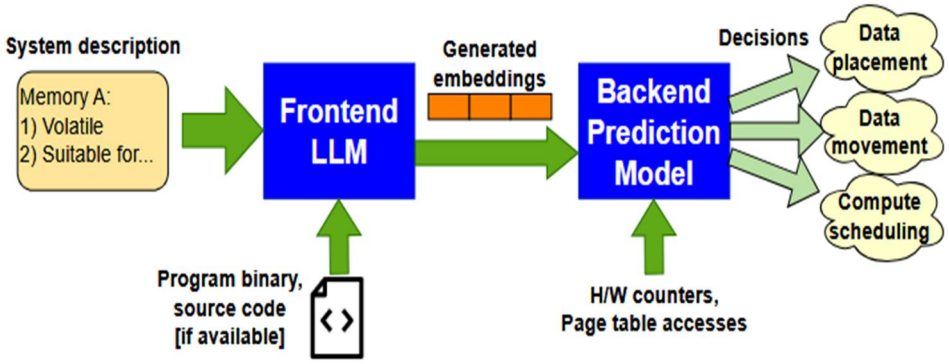


Fig. 7. LLaMaS system design: frontend LLM, backend prediction model, and device decision pipeline

1) Frontend LLM for Semantic Understanding: The frontend language model processes textual descriptions of individual hardware components or device collections and converts this information into embedding representations that reflect functional capabilities, limitations, and appropriate usage scenarios. By relying on these semantic encodings, the operating system can infer the properties of previously unseen devices directly from natural language specifications, eliminating the need for handcrafted scheduling rules or device-specific logic.

2) Backend Prediction and OS Decisions: The generated embeddings are supplied to a backend inference module that combines semantic information with dynamic runtime signals, such as performance counters and memory access patterns. Using this combined input, the system determines suitable strategies for data placement, task scheduling, and data migration across heterogeneous hardware resources. This approach allows the operating system to adapt its behavior to both familiar and newly introduced devices.

3) Experimental Outcomes: Experimental results indicate that the framework can effectively assign tasks and allocate memory regions based on device suitability, even when operating on hardware not encountered during training. This adaptability reduces the effort required for system administration and configuration, as new devices can be integrated without extensive manual tuning.

Together, the Herding LLaMaS framework demonstrates how large language models can function as integral decision-making components within an operating system, particularly for hardware management tasks. When considered alongside AIOS, PEROS, and Compressor–Retriever approaches, it contributes to a comprehensive vision of LLM-driven operating systems that combine conversational interaction, user adaptation, and semantic awareness of diverse hardware platforms.

III. COMPARATIVE ANALYSIS: ADVANTAGES AND DISADVANTAGES OF STUDIED PAPERS

A. Paper 2 – "AIOS: LLM Agent Operating System"

The AIOS study serves as a concrete architectural example of how operating systems can natively support and regulate large numbers of LLM-driven agents.

Whereas the foundational survey primarily discusses the motivation for LLM-enhanced operating systems and outlines broad challenges such as scheduling, context management, and safety at a conceptual level, AIOS translates these ideas into a practical system design. It introduces a layered architecture that clearly distinguishes high-level agent execution from kernel-style resource coordination.

The work details mechanisms for breaking agent requests into operating system calls, scheduling thousands of concurrent agents, and maintaining long-term agent state through vector-based memory storage. In doing so, AIOS demonstrates that established operating system mechanisms—such as task scheduling, context switching, and memory management—can be adapted effectively to language-model workloads, with validation provided through evaluations on HumanEval, MINT, GAIA, and SWE Bench–style benchmarks.

Despite these strengths, the scope of AIOS remains largely centered on performance, concurrency, and system safety at the kernel level. It gives limited attention to user-oriented concerns, including personalization, privacy protection, and modeling long-term user behavior across multiple devices—key aspects emphasized in the broader vision of LLM-driven operating systems.

Moreover, the sophistication of its kernel-centric approach and the computational demands of managing large agent populations may reduce its suitability for deployment in resource-limited environments, such as edge platforms.

Overall, AIOS makes a substantial contribution to the engineering foundations of LLM-based operating systems, while leaving higher-level user interaction and privacy-aware system design to complementary research efforts.

B. Paper 3 – "PEROS: Personalized Self-Adapting Operating Systems in the Cloud"

Relative to the foundational survey, which identifies personalization as a desirable feature of future LLM-enabled operating systems, PEROS delivers a more concrete and user-focused implementation of this objective. The system abandons traditional graphical and command-line interaction in favor of a declarative, language-based interface driven by large language models, allowing user intent to directly influence operating system behavior.

Through the analysis of historical usage data, PEROS demonstrates how kernel policies and system configurations can be adjusted automatically to better reflect individual preferences. Its architectural combination of microkernel design, cloud-oriented deployment, and lightweight client devices illustrates how LLM-based operating system logic can be scaled across large user populations while preserving modularity and fault isolation. A notable contribution of PEROS lies in its strong emphasis on privacy and security. Mechanisms such as user-controlled Databox storage, cryptographically auditable system interactions, and privacy-preserving learning techniques extend the discussion of data protection well beyond the high-level treatment provided in the base survey. These features position PEROS as a compelling example of how personalization can coexist with regulatory and trust requirements.

This paper PEROS is primarily architectural and exploratory. Although the prototypes validate the feasibility of adaptive kernel behavior and privacy-aware personalization, the work does not provide the same depth of experimental analysis seen in AIOS, particularly with respect to large-scale concurrency, agent scheduling, or throughput evaluation. In addition, reliance on cloud infrastructure and the separation of computation and data introduce latency and system complexity. PEROS also does not explicitly address challenges related to long-term context management or heterogeneous hardware support. Overall, the work strengthens the personalization and privacy dimensions of the LLM-driven operating system vision, while placing less emphasis on low-level performance optimization and lifelong system memory.

C. Paper 4-"Compressor–Retriever Architecture for Language Model OS"

The Compressor–Retriever approach responds directly to a key challenge identified in the foundational survey: the limited context capacity of large language models and the absence of persistent state in LLM-based operating systems. While the survey highlights the need to move beyond short, session-specific interactions, this work introduces a model-independent and fully differentiable pipeline that enables long-term contextual reasoning. Historical interactions are transformed into hierarchical embedding representations and selectively retrieved during inference, allowing LLM-driven system components to access deep historical context without exceeding prompt size constraints.

Through this mechanism, the study provides a concrete implementation of persistent memory for LLM-enhanced operating systems. Experimental results show that the compressed retrieval framework retains approximately 75% of the effectiveness achieved by full-context baselines, indicating that most relevant information can be preserved at substantially lower computational cost. This result gives practical form to the concept of lifelong operating system memory that was previously discussed only at a conceptual level in the survey.

On the other hand, the contribution of this work is intentionally focused. The evaluation centers on in-context learning and reasoning benchmarks rather than comprehensive operating system workloads such as task scheduling, personalization, or heterogeneous device management. In addition, the overhead introduced by compression and retrieval processes, along with the need for careful model tuning, presents challenges for deployment in environments with strict latency requirements. As a result, while the Compressor–Retriever framework significantly strengthens the memory and context management foundation of LLM-based operating systems, it does not by itself constitute a complete OS solution. Instead, it is best viewed as a complementary component that, when integrated with system-level architectures such as AIOS or user-centric designs like PEROS, can contribute to a fully functional and scalable LLM-driven operating system.

©IJRASET: All Rights are Reserved

Table. 1. Performance Matrix of All Papers

Paper / System	Domain	Key Metric	Best Reported Result
AIOS	LLM OS Kernel / Multi-Agent Serving	Speedup, Concurrency, Task Success	~2.1× speedup; up to ~2000 concurrent agents with near-linear latency; maintains or improves benchmark success rates (HumanEval, MINT, GAIA, SWE Bench-Lite).
PEROS	Personalized, Privacy-Aware OS	Prototype Responsiveness, User Satisfaction, Scalability	Prototype demonstrates adaptive kernel configuration and scalable multi-user support with strong privacy guarantees.
Compressor–Retriever	Context & Memory	Context Recall, Reasoning Accuracy	≈75% window recall relative to full-context baseline in in-context learning tasks; enables effectively "infinite-length" context handling.
Herding LLaMaS	Semantic Hardware Integration	Device Allocation Quality, Adaptation to New Devices	Correctly allocates memory and maps tasks for novel devices using only textual descriptions; reduces manual tuning effort.

IV. CONCLUSION

The AIOS framework shows that foundational operating system mechanisms—such as task scheduling, concurrency management, and memory handling—can be effectively reengineered to support large-scale execution of LLM-based agents. This work primarily advances system-level performance and scalability. In contrast, PEROS places emphasis on the user-facing dimension, demonstrating how adaptive kernels and natural language interfaces can deliver personalized experiences while enforcing strong privacy protections. The Compressor–Retriever approach addresses a fundamental limitation of LLM-based systems by enabling persistent, long-term context handling, thereby supporting reasoning beyond isolated interaction sessions. Meanwhile, Herding LLaMaS highlights the role of semantic understanding in hardware management, illustrating how LLMs can simplify integration and coordination across heterogeneous computing devices. A comparative analysis reveals that these approaches offer complementary contributions. AIOS excels in kernel-level control and concurrency but pays limited attention to personalization.

Overall, existing research points toward a future in which operating systems are built around natural language interaction, adaptive reasoning, and interpretable decision processes. The insights from these studies underscore the importance of developing operating systems that are not only technically robust but also transparent and user-centric. Advancing explainability, efficiency, and trustworthiness will be critical in realizing LLM-driven operating systems that can reliably interpret user intent, manage resources intelligently, and communicate system behavior in ways that users can readily understand.

V. FUTURE WORK

Future research should concentrate on designing holistic and interpretable operating system frameworks in which large language models are tightly integrated with core system functionality. Such frameworks should unify natural language interaction, intent interpretation, system-level reasoning, and secure execution within a single architectural design. Rather than positioning LLMs as peripheral tools or external assistants, forthcoming studies should explore their role as fundamental operating system components that collaborate directly with kernel mechanisms such as scheduling, memory management, and resource allocation. Equally important is the advancement of long-term memory and context-handling strategies that enable LLM-powered operating systems to retain, summarize, and selectively access user state over extended periods. Approaches inspired by compression-and-retrieval mechanisms should be adapted for real operating system environments, supporting persistent personalization across sessions and devices while minimizing computational overhead.

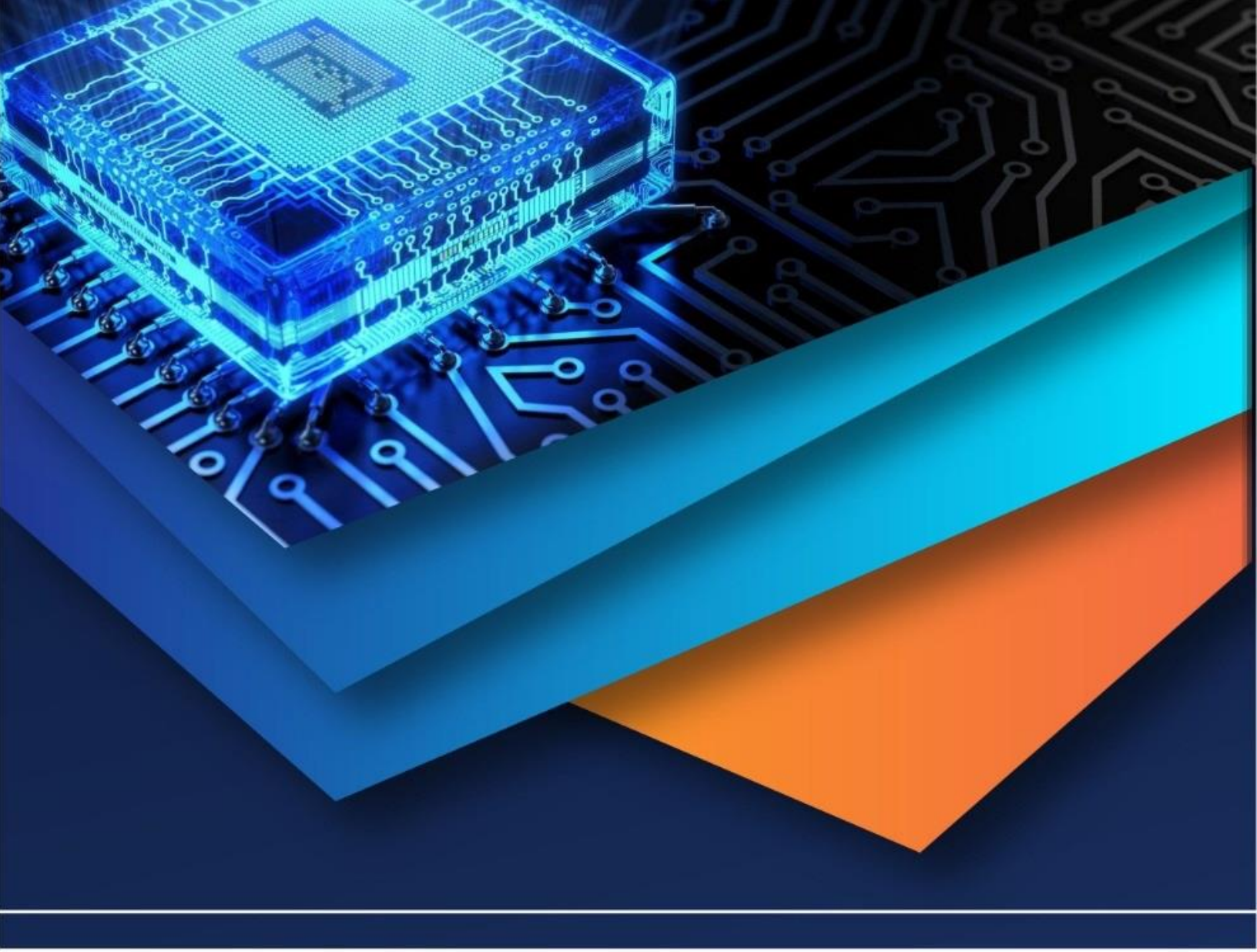
©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The field also requires the establishment of standardized benchmarks, datasets, and evaluation frameworks tailored to LLM-driven operating systems. Such benchmarks should assess not only conventional system metrics like performance, scalability, and latency, but also factors such as personalization effectiveness, interpretability, privacy safeguards, and resistance to misuse. Including trust- and explainability-focused metrics will support meaningful comparisons and reproducible experimentation. A natural next step is the construction of a prototype LLM-enabled operating system that demonstrates end-to-end capabilities, spanning natural language interaction, adaptive system control, transparent decision-making, and secure execution. Developing such a system would illustrate how advanced intelligence can be embedded into operating systems while ensuring accountability and clarity, ultimately enabling platforms that are both technically sophisticated and worthy of sustained user trust.

REFERENCES

[1] Lyu, H., Jiang, S., Zeng, H., Xia, Y., Wang, Q., Zhang, S., & Luo, J. (2023). LLMRec: Personalized Recommendation via Prompting Large Language Models. arXiv:2307.15780.
[2] Li, L., Zhang, Y., Liu, D., & Chen, L. (2023). Large Language Models for Generative Recommendation: A Survey and Visionary Discussions. arXiv:2309.01157.
[3] Tan, Z., Zeng, Q., Tian, Y., Liu, Z., Yin, B., & Jiang, M. (2024). Democratizing Large Language Models via Personalized Parameter-Efficient Fine-Tuning. arXiv:2402.04401.
[4] Wu, L., Zheng, Z., Qiu, Z., Wang, H., Gu, H., Shen, T., Qin, C., Zhu, C., Zhu, H., Liu, Q., Xiong, H., & Chen, E. (2023). A Survey on Large Language Models for Recommendation. arXiv:2305.19860.
[5] Netflix Foundation Model Team. (2025). Foundation Model for Personalized Recommendation. Netflix Tech Blog.
[6] Spurlock, K. D., Acun, C., Saka, E., & Nasraoui, O. (2024). ChatGPT for Conversational Recommendation: Refining Recommendations by Reprompting with Feedback. arXiv:2401.03605.
[7] Dao, H., et al. (2024). Demonstration-augmented Prompt Learning for Conversational Recommendation. SIGIR 2024.
[8] Li, L., Zhang, Y., & Chen, L. (2022). Personalized Prompt Learning for Explainable Recommendation. arXiv:2202.07371.
[9] Li, Z., Ji, J., Ge, Y., Hua, W., & Zhang, Y. (2024). PAP-REC: Personalized Automatic Prompt for Recommendation Language Models. arXiv:2402.00284
[10] Zou, J., Sun, A., Long, C., & Kanoulas, E. (2024). Knowledge-Enhanced Conversational Recommendation via Transformer-based Sequential Modelling (TSCRKG). ACM Digital Library.
[11] Chen, B., Zhang, Z., Langrené, N., & Zhu, S. (2023-2025). Unleashing the Potential of Prompt Engineering for Large Language Models. arXiv:2310.14735.
[12] Chen, Q., et al. (2025). Improving Optimal Prompt Learning through Multilayer Integration and Global Attention Mechanisms. Frontiers in Robotics and AI.
[13] Ai, J., et al. (2025). An Explainable Recommendation Algorithm Based on Content Summarization and Linear Attention Mechanism. Neurocomputing.
[14] Xian, Y., et al. (2019). Reinforcement Knowledge Graph Reasoning for Explainable Recommendation. arXiv:1906.05237.
[15] Li, L., Zhang, Y., & Chen, L. (2021). Personalized Transformer for Explainable Recommendation (PETER). ACL 2021, arXiv:2105.11601.
[16] Montagna, A. (2023). Graph-Based Explainable Recommendation Systems. CEUR Workshop Proceedings.
[17] Rong, Z., et al. (2024). Enhanced Knowledge Graph Recommendation Algorithm Based on Multi-Hop Reasoning and Multi-Level Contrastive Learning. Nature Scientific Reports.
[18] Khan, H.U., Naz, A., Alarfaj, F.K., & Almusallam, N. (2025). A Transformer-based Architecture for Collaborative Filtering in Personalized Recommender Systems. Scientific Reports.
[19] Chen, F., et al. (2023). KHGCN: Knowledge-Enhanced Hierarchical Graph Capsule Network for Personalized Recommendation. PMC.
[20] Zhang, M. (2025). Design and Implementation of Intelligent Library Personalized Recommendation Model Based on Reinforcement Learning. Journal of Combination Mathematics and Computing Informatica.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)