Operating System And Artificial Intelligence: A Systematic Review

Xidong Wang
School of Data Science
The Chinese University of HongKong (ShenZhen)
Email: 223040239@link.cuhk.edu.cn

***Abstract*—In the dynamic landscape of technology, the convergence of Artificial Intelligence (AI) and Operating Systems (OS) has emerged as a pivotal arena for innovation. Our exploration focuses on the symbiotic relationship between AI and OS, emphasizing how AI-driven tools enhance OS performance, security, and efficiency, while OS advancements facilitate more sophisticated AI applications. We delve into various AI techniques employed to optimize OS functionalities, including memory management, process scheduling, and intrusion detection. Simultaneously, we analyze the role of OS in providing essential services and infrastructure that enable effective AI application execution, from resource allocation to data processing. The article also addresses challenges and future directions in this domain, emphasizing the imperative of secure and efficient AI integration within OS frameworks. By examining case studies and recent developments, our review provides a comprehensive overview of the current state of AI-OS integration, underscoring its significance in shaping the next generation of computing technologies. Finally, we explore the promising prospects of Intelligent OSes, considering not only how innovative OS architectures will pave the way for groundbreaking opportunities but also how AI will significantly contribute to advancing these next-generation OSs.**

1. Introduction

Operating systems (OSes) have long been central to computer systems, efficiently managing hardware resources and providing secure environments for application execution. However, the increasing complexity of modern OSes, the rapid diversification of hardware, and the continuous evolution of Artificial Intelligence (AI) present new opportunities to explore AI's potential throughout the lifespan of OSes, spanning both development and runtime. As the AI wave continues to surge, AI systems grow increasingly massive and complex, necessitating optimization and efficiency enhancements at the lower layers of the stack, particularly within the operating system.

This review delves into the intricate relationship between AI and OS, examining how AI-driven tools enhance OS performance, security, and efficiency. Conversely, we explore how advancements in OS design facilitate the deployment and optimization of AI applications. By scrutinizing various AI techniques employed to augment OS functionalities—such as memory management, process scheduling, and intrusion detection—this review provides a comprehensive overview of the current state of AI-OS integration. Additionally, we discuss the critical role of the OS in providing essential services and infrastructure necessary for effective AI application operation, from resource allocation to data processing. The article also addresses challenges and future research directions in this domain, emphasizing the imperative for secure and efficient integration of AI capabilities within OS frameworks. Through systematic examination of case studies and recent developments, this review underscores the significance of the AI-OS nexus in propelling the next generation of computing technologies.

In this paper, we embark on a comprehensive exploration of the intersection between AI and OS. Our approach involves collecting and analyzing 108 primary studies in the field, aiming to uncover key insights and trends. The rest of this paper is organized as follows. We provide the research background.Then we propose our research questions, report on the paper selection process, and analyze the distribution of research popularity. Collected papers in the fields of AI4OS and OS4AI are classified and summarized in Sections 4, 5 and 6. Furthermore,we discuss how and what novel OS architectures may create opportunities for AI4OS in section 7 and how LLM can create oppotunies for OS in section 8.

Subsequently, in Section 9, we consolidate these insights by outlining prospective opportunities for future work, particularly emphasizing the synergy between Library OS paradigms and LLM-driven approaches in the pursuit of Intelligent OS. This section underscores the immense potential for synergistic collaborations between these technologies, which could pave the way for highly modular, adaptable, and self-aware operating systems. Finally, we bring this comprehensive exploration to a close in Section 10, synthesizing our findings and offering a compelling conclusion.

2. Background

Operating systems have been providing a crucial layer of abstraction between applications and hardware resources. The design of an OS can significantly impact hardware

resource management efficiency and, consequently, the performance of all applications running on it. Additionally, the development of OSes requires extensive engineering efforts from experts. And the complexity of modern OSes poses challenges for developers to effectively improve and optimize performance. The scale and dynamics of computing systems further contribute to this complexity.Moreover, the escalating scale and intricacy of AI systems themselves necessitate optimizations from the OS. As the complexity increases and the need for efficient resource management grows, the integration of AI techniques has emerged as a promising avenue for enhancing OS functionality and performance.

In recent years, researchers have explored various dimensions of AI integration in operating systems, leading to notable advancements.

3. Methodology

We present the research questions, introduce the paper selection process, and statistically analyze the selected papers in this section.

3.1. Research questions

The definition of research questions is the core innovation of a secondary study, as they clearly convey the authors' perspective on the subject under investigation and the study's goal.We characterize the purpose, the major research topics, and the scope of this paper as follows.

RQ1: What OS sub-domains are researchers inclined to enhance using AI?

RQ2: How can OS be optimized to improve the efficiency of AI systems?

RQ3: How and which AI techniques are used to improve OS?

RQ4: How and what novel OS architectures create opportunities for AI4OS?

RQ5: How LLMs create opportunities for OS?

3.2. Paper Selection

We identify four AI-related terms and two OS-related terms, then combine these two categories of terms by logical ORs to create nine search strings. After specifying the range the papers were published: 2019–March 2024, we deliver these nine search strings to Google Scholar, to collect related papers in the intersection of AI and OS. The defined search terms are as follows:

AI **or** Machine Learning **or** Deep Learning **or** Large Language Model
And
OS **or** Operating System

We began with an initial pool of 212 papers related to the intersection of AI and Operating Systems (OS). To narrow down our focus, we applied the following inclusion criteria (IC) and exclusion criteria (EC):

IC1: Peer-Reviewed Research Paper: We considered only peer-reviewed research papers.

IC2: Primary Study: We included primary studies that directly addressed AI-OS integration.

IC3: Relevance to AI and OS: Papers needed to discuss AI techniques enhancing OS or propose novel approaches for improving AI systems using OS methods.

IC4: Publication Date: We restricted our selection to papers published within the past 5 years.

EC1: Avoiding Duplicates: Only the most comprehensive or recent version of each study was included.

EC2: Length Constraint: Papers shorter than 6 pages were excluded.

EC3: Language: We considered papers written in English.

As a result, 108 papers met our criteria and are within the scope of our research. Excluded papers merely mentioned the terms but did not delve into the actual intersection of OS and AI.

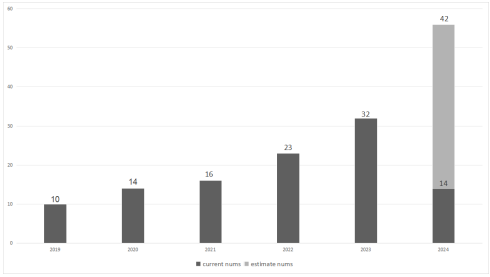

Figure 1. Year Distribution of Paper Published

3.3. Paper Analysis

3.3.1. The main venues. Figure 1 depicts the distribution of publication years for research 1. Notably, the number of papers published in the AI4OS and OS4AI domains indicates a growing trend, increased attention in this field. Notably, the number of papers published(including pre-print) in the field of integrating LLM and OS indicate a rapidly growing trend, 17 publications in this field since 2023, when powerful LLM emerged.

Journals and conferences are listed in Tables 1 , the selection criterion is how many times they appear in our paper pool. International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS) is the most prevalent conference.

We first divide all the papers into AI4OS(38), OS4AI(53), LLM AS OS(9) and LLM4OS(8).

4. RQ1: What OS sub-domains are researchers inclined to enhance using AI?

Currently, OSes employ global, static policies based on heuristics. However,AI techniques, adaptable to different application behaviors, hold promise for outperforming these traditional policies. This research question explores

TABLE 1. CONFERENCES DISTRIBUTION

Conference	Papers
ASPLOS	22
HPCA	11
USENIX ATC	11
OSDI	5
SOSP	2
HotOS	2
APsys	2
SOSP	2
DAC	1
CCS	1
MobiCom	1
ESEC/FSE	1
ACM TURC	1
ISSTA	1

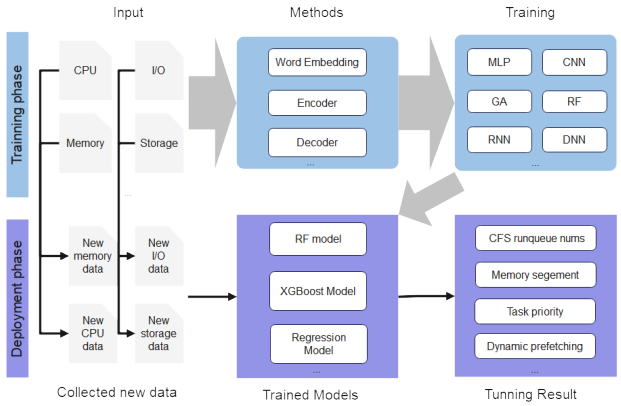

Figure 2. General AI for OS Tuning Workflow

and quantifies the diverse applications of AI approaches in enhancing or automating various OS tasks.

4.1. AI for OS Tuning

Growing OS complexity challenges configuration and decision-making, prompting the exploration of AI and ML for enhancements in auto-tuning tasks, including scheduling, energy efficiency, and memory management.

Figure 2 showed the general workflow for AI tools to assist OS.

Scheduling. Scheduling, a core OS function, balances fairness, responsiveness, and throughput by efficiently managing resources. [2] [16] Advancements through machine learning can refine ticket distribution, enable adaptability, and adjust scheduling variables precisely.

Springborg et al. [86] introduce Chronus, a Python application that collaborates with the Simple Linux Utility for Resource Management (SLURM) scheduler, prevalent in global supercomputers. Chronus executes comprehensive benchmark tests on HPC clusters, varying parameters like core numbers, processor speeds, and hyperthreading. Throughout, it logs performance metrics and energy use for each setup. This data trains a machine learning model to recognize patterns, aiming to forecast the most energy-efficient configuration for specific jobs and systems. Proven through notable energy savings in benchmarks, this approach holds potential for broader application in HPC systems, promoting eco-friendliness and financial prudence without sacrificing computational power.

Chen et al. [3] presents a ML-based resource-aware load balancer for the Linux kernel with a low-overhead method for collecting training data,an ML model based on a multi-layer perceptron that imitates the Linux's Completely Fair Scheduler (CFS) load balancer based on the collected training data and an in-kernel implementation of inference on the model. The authors argue that CFS approach maximizes the utilization of processing time but overlooks the contention for lower-level hardware resources.Using eBPF for dynamic tracing, an MLP model replicates CFS decisions, with in-kernel inference for real-time balancing, the model achieves high accuracy and small latency increase, demonstrating effective, low-overhead load management.

Goodarzy et al. [14] also questioned CFS in its ability for proper allocation of CPU, memory, I/O, and network bandwidth. In response to this, the authors propose SmartOS, an operating system that automatically learns what tasks the user deems to be most important at that time. ased on the learned user preferences, SmartOS adjusts the allocation of system resources such as CPU, memory, I/O, and network bandwidth. It prioritizes the resources for the tasks that the user is currently focused on. The authors demonstrate an implementation of such a learning-based OS in Linux and present evaluation results showing that a reinforcement learning-based approach can rapidly learn and adjust system resources to meet user demands.

Storage. Storage systems, along with their associated OS components, are engineered to cater to a broad spectrum of applications and fluctuating workloads. The storage elements within the OS incorporate a range of heuristic algorithms, ensuring optimal performance and adaptability across diverse workloads.

Predictable latency can be very useful for data-center systems serving interactive applications such as messaging and search.Cortez et al. [6] introduce LinnOS, a novel approach to managing SSD performance by incorporating a lightweight neural network within the operating system's core. This neural network enables real-time inference of SSD performance at an extremely fine-grained level, specifically per-I/O operation, without requiring modifications to the underlying hardware or changes to file systems and applications. LinnOS profiles the latency of a large number of I/O operations submitted to the SSD, leveraging this data to train the neural network. By adopting a "black-box" perspective towards the device, LinnOS learns and infers the SSD's behavior to increase predictability, thereby empowering applications to anticipate whether their performance expectations will be met.

Wu et al. [85] introduce LearnWD, a novel approach that harnesses the synergies of machine learning and out-of-place updates to effectively mitigate the write disturbance (WD) problem in NVM. At its core, LearnWD employs clustering algorithms to systematically categorize stale data

based on their inherent error proneness. Upon receiving a write request, LearnWD meticulously assesses both the aggressiveness of incoming new data and the error vulnerability of the existing stale data. By thoughtfully leveraging this information, LearnWD strategically orchestrates write operations to proactively minimize the incidence of WD errors.

Wang et al. [87] introduce LearnedFTL,a page-level Flash Translation Layer design that harnesses the power of ML techniques to significantly enhance the random read performance of flash-based SSDs. This method effectively models the non-uniform yet largely linear relationships between logical page numbers and physical page numbers, overcoming the inherent irregularity in flash memory's address space distribution.LearnedFTL embeds the training of learned indexes within the GC process, allowing for the continuous updating and refinement of the learned models as the address mapping evolves. LearnedFTL also incorporates a bitmap prediction filter acting as a safeguard against potential model inaccuracies, validating the predicted PPNs and ensuring that only correct mappings are used for address translation. This innovative approach uniquely minimizes the occurrence of double reads resulting from address translation during random read accesses.

Memory. Traditional heuristic-based methods struggle due to the intricate hierarchy of memory levels, dynamic workloads, data-dependent behavior, and hardware constraints. These complexities make it difficult to optimize memory management effectively.

Zhang et al. [17] address the challenges of address mapping in 3D-stacking memory, focusing on technologies like High-Bandwidth Memory and Hybrid Memory Cube. They propose Software-Defined Address Mapping, allowing user programs to directly control memory hardware. SDAM enables fine-grained data placement while leveraging chunk-based address mapping management. The system employs machine learning to identify access patterns, resulting in significant speedups compared to fixed address mapping systems.

Lagar-Cavilla et al. [88] introduce a software-defined far memory system that has been successfully deployed across Google's data centers since 2016. The authors propose a proactive software-defined approach that compresses cold memory pages to create a far memory tier in software. This approach uses the Linux kernel's zswap mechanism to compress pages and store them in DRAM, effectively creating a far memory tier with low latency. The authors designed an autotuning system that employs a Gaussian Process Bandit to estimate the size of cold memory and the promotion rate under different configurations. This model is used to emulate the control algorithm offline and estimate the impact of different parameter settings. The authors conclude that their software-defined far memory system is effective in saving memory costs without impacting application performance.

Rocha et al. [15] propose PredG, a Machine Learning framework designed to enhance the performance of graph processing. PredG aims to find the ideal thread and data mapping on Non-Uniform Memory Access systems. One of the key features of PredG is its ability to be agnostic to the input graph. It uses the available features of the graphs to train an Artificial Neural Network to perform predictions as new graphs arrive. This is done without any application execution after being trained, which makes it a powerful tool for optimizing graph execution on NUMA machines. This work is part of a broader trend in the research community towards automated graph machine learning, which seeks to discover the best hyper-parameter and neural architecture configuration for different graph tasks/data without manual design.

4.2. AI for OS security

With the widespread application of deep learning in recent years, using deep learning technologies for OS security has emerged, and the effectiveness of threat detection has been dramatically improved.

Qin et al. [35]presents MSNdroid, a novel malware detector designed specifically for Android applications, leverages a combination of native API calls, permissions, system API call features, and a Deep Belief Network. By applying deep learning techniques to native code features, MSNdroid effectively detects Android malware. This approach involves extracting features from a comprehensive dataset comprising malicious applications and benign applications. Notably, MSNDroid achieves an impressive accuracy while maintains an impressively low false-negative rate.

De Wit et al. [39] emphasize the value of incorporating machine learning in malware detection strategies for Android platforms. By leveraging accessible hardware data and sophisticated classification techniques, the study demonstrates the feasibility of identifying malware with a reasonable degree of precision, highlighting the potential of app-specific metrics in enhancing detection rates. This research contributes to the growing body of knowledge on AI-integrated security measures within operating systems, particularly pertinent to the Android ecosystem.

4.3. Findings

In response to Research Question 1, which explores the inclination of researchers to utilize AI in enhancing specific OS sub-domains, our comprehensive analysis of existing literature highlights two prominent areas: auto-tuning and security. These domains have attracted considerable attention due to their potential for transformative improvements through AI integration.

Auto-tuning emerges as a key area where AI is being employed to overcome the shortcomings of conventional heuristic-based methods. These traditional approaches often struggle to cope with the ever-evolving demands of modern computing environments. Researchers are harnessing machine learning to dynamically adjust OS parameters, such as scheduling, energy management, memory allocation, and storage optimization, aiming to boost system performance and resource efficiency. AI's predictive abilities are particularly crucial in enhancing scheduling mechanisms to strike

a balance between fairness, responsiveness, and throughput, effectively managing critical resources, ensuring they can handle the intricate and fluctuating workloads common in today's computing landscapes.

The security domain has witnessed a surge in the adoption of deep learning technologies to combat malware threats, showcasing substantial advancements in detection efficacy. AI-powered malware detectors have proven superior in terms of accuracy and reduced false negatives compared to traditional machine learning models. Furthermore, AI frameworks are being developed to create adversarial malware, underscoring AI's dual role in fortifying OS security while also exposing vulnerabilities in AI-based detection systems.

5. RQ2: How can OS be optimized to improve the efficiency of AI systems?

AI accelerators are different from traditional hardware, affecting all aspects of system design, from data-center scale to single-chip scale. They also add high requirement for system architecture, management, and programming [13].

Previous work has shown AI jobs critically demand high-speed I/O and low-latency and high-bandwidth data communication [18] [21].Various attempt on hardware has been done to improve ML application performance,for exmaple,using newly NVMe SSD [89], relying on hardware FPGA for the I/O communication control instead of relying on OS-level interrupts that can significantly reduce both total I/O latency and its variance and algorithm level. Accessing hardware through the kernel introduces a performance bottleneck. To mitigate this bottleneck, one effective approach is to bypass the kernel altogether, enabling userspace programs to directly access hardware [20].

Bateni et al. [89] presents NeuOS, a comprehensive system solution aimed at providing latency predictability for executing multi-DNN workloads in autonomous systems while simultaneously managing energy optimization and dynamic accuracy adjustments. It coordinates system- and application-level solutions intelligently to ensure that multiple DNN instances operate efficiently and meet their respective deadlines to guarantee latency predictability. It manages energy consumption by dynamically adjusting parameters such as voltage and frequency scaling to minimize energy usage without compromising the latency predictability or accuracy requirements. Based on specific system constraints, NeuOS adjusts the accuracy level of DNN computations in real-time. This allows for trade-offs between computational precision and resource efficiency, ensuring that the system operates within its given constraints while maintaining an acceptable level of performance.

Wang et al. [21] showed what network for GPU AI remoting,a technique where the execution of GPU APIs is managed remotely through a network on a remote proxy instead of running GPU computations locally on the machine. The study takes a GPU-centric perspective to derive minimum latency and bandwidth requirements and aims to ensure that unmodified AI applications can run on remoting setups using commodity networking hardware without performance degradation. The paper introduces a novel theoretical framework that quantifies the minimum network requirements essential for efficient GPU API remoting. By formalizing the relationship between network latency, bandwidth, and remoting efficiency, this model provides foundational insights.

Serizawa et al. [19]proposed an solution focused on I/O bandwidth.The method aims to improve the reading performance of large training datasets by using high-performance I/O storage devices. The authors discusses the problem of copying datasets between local storage and shared storage and proposes a solution to conceal the time spent on copying by overlapping the copying and reading of training data.

5.1. Findings

Addressing Research Question 2, which delves into the ways OSes can elevate the efficiency of AI systems, our analysis underscores the pivotal function of OSes in handling the distinctive needs of AI tasks.

To optimize AI efficiency through the OS, several strategic approaches emerge as essential. Firstly, there is a need to tailor the OS to cater specifically to the requirements of AI accelerators. This includes facilitating high-speed I/O operations, minimizing latency in communications, and enabling autonomous operation of AI tasks. Secondly, the development of specialized runtime systems and schedulers becomes crucial, ensuring optimal allocation of resources and efficient execution of AI processes. Thirdly,optimizing I/O bandwidth further enhances the performance of AI applications. Collectively, these strategies form a comprehensive framework for improving the efficiency and effectiveness of AI systems through optimized OS integration.

6. RQ3: How and which AI techniques are typically used to improve OS?

To answer RQ3, we review all studies that investigate AI techniques applied in OS in this section. Among 38 AI4OS previous work,those did not clear clarify the AI tools excluded,we list the AI tools that are used more than twice. In general, RF, RNN (LSTM), KNN, RL, MLP,etc. are the most widely-used techniques.

6.1. Tools analysis

RF(DT). A decision tree is a type of machine learning model used when the relationship between a set of predictor variables and a response variable is non-linear,while random forest is essentially a collection of decision trees. It is quick to fit to a dataset and easy to interpret. Chowdhury et al. [32] used RF as one if the models to accurately detect the attack from the network traffic. To construct the random forest classifier, the authors employed the Random Forest Regressor , which serves as both a regressor and a meta

estimator. It accomplishes this by fitting multiple decision trees to different subsets of the dataset. For our specific model, we opted for a forest containing 1000 trees.Ahmed et al. [23] used RF to build a device fitness model,based on the Dataset collected during runtime and statically. De Wit et al. [39] trained a statistical classifier able to recognize malware signatures in any log data collected on a smartphone. The classifier was trained, cross-validated, and tested using the dataset described above and RF classifier had a better performance.Metzge el at. [49] used RF to get a optimal kernel runtime switching slice size. The model is a random forest regressor with 50 decision trees with a depth of two for the GPU model. Ongun et al. [36] used RF to get the probability of a command being malicious,based on labels dataset.

RNN(LSTM). RNNs are a class of neural networks designed to handle sequential data. They have feedback connections, allowing them to maintain an internal state or memory. Each step in an RNN processes an input and updates its hidden state based on the current input and the previous hidden state.While LSTM ,a type of RNN architecture with a more complex cell structure,were introduced to address the vanishing gradient problem. Motivated by the problem that exploiting 3D-stacking memory's performance is challenging because bandwidth utilization heavily depends on address mapping in the memory controller, Zhang et al. [17] used a software-defined address mapping, allowing user programs to directly control low-level memory hardware in an intelligent and fine-grained manner.LSTM is used to in a method to get access pattern information to select an address mapping. It identify the major variables that significantly contribute to external memory access and have a substantial impact on memory traffic and data movement.

WordEmbedding. Word embeddings provide a way to achieve this by mapping words to dense vectors in a multi-dimensional space. Fu et al. [34] proposed a AI-based approach to help under-resourced security analysts to find, detect,and localize vulnerabilities. They utilized a word-level Clang tokenizer with a copy mechanism. This tokenizer broke down a C function into a sequence of tokens.Then word embedding was used to generate vector representations for each token in the sequence, capturing the semantic information among the input tokens. And further classifier was , Ongun et al. [36] explore techniques for representing command-line text using word embedding methods.Based on this,they devise ensemble boosting classifiers to differentiate between malicious and non-malicious commands.

KNN. KNN is a supervised machine learning algorithm used for both classification and regression tasks. The fundamental idea behind KNN is simple: neighbors influence each other. If you're surrounded by similar things, you're likely similar too. It is widely applicable in pattern recognition, data mining, and intrusion detection. Yang et al [42] introduce a KNN-based machine learning algorithms can accurately predict the Turnaround-time(TaT) of a process. It can effectively reduce the TaT of the process and reduce the number of process context switches.

MLP. An MLP is a type of feedforward neural network used for supervised learning tasks, such as classification and regression. Chen et al. [3] argues that traditional Linux CFS scheduler maximizes the utilization of processing time but overlooks the contention for lower-level hardware resources and try to solve the above problem using an ML-based resource-aware load balancer.They employed supervised imitation learning to replace a portion of its internal logic with an MLP model. This trained MLP model emulates the kernel's load balancing decisions. MLP is chosen because this current work doesn't require a very complex model and MLP has a relatively simple implementation compared to the other models. Based on this work,Qiu et al. [78] propose the concept of reconfigurable kernel datapaths that enables kernels to self-optimize dynamically to reduce the cost of kernel. The authors also used MLP ML model that can mimic Linux CFS decisions.

6.2. Findings

In addressing Research Question 3, which explores the methods by which AI techniques are utilized to enhance OS, our analysis reveals a variety of approaches tailored to meet distinct OS requirements. These AI techniques are strategically chosen to address a spectrum of OS challenges, encompassing security enhancements, performance forecasting, resource management, and process scheduling. The widespread adoption of these methods attests to their versatility and underscores the profound influence of AI in augmenting OS functionality and performance. This integration of AI into core OS components not only boosts operational efficiency but also paves the way for more intelligent and adaptable systems, capable of meeting the evolving demands of modern computing environments.

7. RQ4: How and what novel OS architectures create opportunities for AI4OS?

Comprehensive research has revealed that utilizing the standard kernel pathway for hardware interaction results in notable efficiency losses, rendering conventional approaches less than ideal for the current AI-driven environment. In response, kernel bypass tactics are gaining prominence, aiming to optimize hardware utilization and enhance real-time capabilities specifically for AI tasks. Nonetheless, the absence of an OS means these strategies fall short in providing the necessary tailoring to exploit AI applications' full potential.

In essence, the interplay between modern application design and OS innovation fosters a fertile ground for the conception of systems that not only meet but also anticipate the evolving needs of AI-driven environments. This confluence of advancements signals a pivotal moment in the trajectory of computing, where the synergy between AI and OS architectures could redefine the boundaries of what is achievable in high-performance computing.

7.1. Kernel-bypass OS Structure for AI

Prior research has successfully demonstrated the potential of leveraging hardware acceleration for machine learning within the kernel space, showcasing the feasibility of such an approach [4]. This study introducES an API remoting system, which facilitates access to specialized accelerator interfaces for kernel space applications. Moreover, it simplifies the integration by offering high-level APIs directly to the kernel space, eliminating the necessity for kernel-specific adaptations of complex libraries. The API remoting mechanism transmits commands between kernel and user space. This innovative design not only enhances the performance of AI applications within the kernel but also provides a compelling perspective on the capabilities of modern operating systems.

Raza et al. [53] propose integrating unikernel optimizations into Linux, known for creating secure, compact OS images for single applications. Unikernel Linux (UKL) reduces the number of executed instructions and improves instructions-per-cycle efficiency. Tail latency tests on Memcached—a multi-threaded key-value store—show that UKL achieves a significant performance improvement. It introduces a configuration option that allows a single, optimized process to link directly with the kernel, bypassing the traditional system call overhead, and significantly cuts latency for system call payloads, showcasing the benefits of unikernel-inspired enhancements in refining Linux's performance.

Cadden et al. [59] introduces Serverless Execution via Unikernel SnapShots (SEUSS), a method that leverages unikernel snapshots for rapid deployment and high-density caching of serverless functions. The authors describe the use of unikernel contexts, which consist of a high-level language interpreter configured to import and execute function code, providing isolation and security. This minimalistic approach leads to a reduced memory footprint and faster startup times compared to traditional operating systems like Linux, which is beneficial for serverless environments where rapid function instantiation is crucial. By using unikernels, the SEUSS system can cache a large number of function instances in memory due to the reduced memory footprint.

7.2. Library OS

One significant obstacle that OSs face in effectively leveraging the potential of AI lies in the absence of a comprehensive, universally applicable strategy for adapting AI technologies to the wide array of heterogeneous devices in use.

The Demikernel project [52] unveils an OS architecture optimized for datacenter systems with microsecond-scale requirements, emphasizing low-latency I/O. Adopting a LibOS strategy, it side-steps the traditional kernel in I/O paths, significantly enhancing performance. This design supports kernel-bypass devices, allowing seamless application operation with negligible overhead at the nanosecond scale. For I/O efficiency, Demikernel applies zero-copy techniques for large buffers, optimizes resource use, and maintains system stability through periodic LibOS interaction. Security is bolstered with controlled data placement and potential advanced memory integrity measures. LibOS components employ hardware acceleration to offload critical tasks, minimizing latency and maximizing throughput. Kernel bypass in I/O paths reduces overhead from context switching, system calls, and memory duplication.

7.3. Other Stuctures

Skiadopoulos et al. [60] present DBOS as a superior alternative to conventional cluster OS components, offering comparable functionality but enhanced analytics and reduced code complexity. DBOS matches current systems in scheduling, file management, and inter-process communication, yet excels in analytics and simplifies code through database query-based OS services. It efficiently implements low-latency transactions and ensures high availability. DBOS's integrated DBMS approach is especially advantageous for ML, delivering a cohesive platform for efficient resource management and analytics in large-scale distributed environments, adeptly managing parallel computation and workload dynamics across various hardware.

Shan et al. [61] developed LegoOS—a splitkernel architecture for hardware disaggregation, decentralizing OS functions for scalable, distributed management. It fundamentally breaks down traditional OS functionalities into loosely-coupled monitors that each run on and manage a distinct hardware component. The splitkernel model distributes responsibilities such as scheduling, memory management, and I/O operations across these monitors, effectively creating a distributed set of hardware components that can be independently managed and scaled. The splitkernel model in LegoOS allows for independent scaling of compute and memory resources, essential for ML scenarios where large datasets and complex models necessitate specialized hardware accelerators.

7.4. Findings

Addressing Research Question 4, our analysis highlights the transformative role of novel OS architectures in advancing AI4OS. Pioneering designs like LibOS and unikernels have sparked renewed excitement among researchers, demonstrating their potential to outshine traditional OSes. These cutting-edge architectures enhance the symbiosis between software and hardware, reducing latency to boost AI application responsiveness and real-time processing capabilities. Their streamlined construction optimizes resource allocation, ensuring efficient management of computational assets. Beyond performance, these architectures establish a secure foundation for AI deployment. By simplifying the environment, they minimize security risks, vital for AI applications managing sensitive data. DBOS and LegoOS exemplify this by simplifying distributed AI setup and maintenance, freeing experts to focus on algorithmic innovation. They also adapt nimbly to the variable demands and

hardware diversity of modern ML, enhancing deployment efficiency.

In essence, these advanced OS architectures elevate performance and security, fostering an innovative landscape for AI4OS. By catering to AI application requirements, they promise to rewrite the rules of computing, blending AI and OS synergies for smarter, more efficient, and secure computing futures.

8. RQ5: How LLMs create opportunities for OS?

The integration of LLMs into OSes presents a significant opportunity to enhance the user experience and overall system functionality [67] [68]. LLMs, with their advanced natural language processing capabilities, [62] [63] [64] can transform the way users interact with and manage their computing environments. [76] We simply divide these works into two big categories: LLM AS OS and LLM4OS.

8.1. LLM AS OS

"LLM AS OS" refers to the integration of LLMs into the core of an OS, effectively serving as the "brain" of the system,make OS capable of understanding and responding to natural language commands, thereby enabling more intuitive and flexible human-computer interaction [70] [69].

Kamath et al.'s LLaMaS [54] addresses the complexities of diverse computing environments by leveraging LLMs to ease OS challenges in hardware adaptation and resource management. It adapts to new devices by interpreting plain text specs, recommending optimized OS tactics. The LLM frontend deciphers system characteristics, converting them into actionable embeddings. The backend uses these for on-the-fly OS decisions, like memory tier data movement. An experiment with ChatGPT demonstrated its skill in adjusting memory allocation for CPU and GPU tasks based on usage. Aimed at reducing administrative and research costs, LLaMaS autonomously aligns with hardware changes via LLMs' innate zero-shot learning, eliminating the need for manual adjustments or detailed device-specific coding.

AIOS-Agent ecosystem [55] [56], where LLMs act essentially as an "operating system with soul".LLMs are embedded in the OS kernel for intelligent decision-making and resource allocation. Its context window acts as memory, managing relevant data, while external storage serves as a file system with enhanced retrieval capabilities. Hardware and software are treated as peripherals and libraries, enabling agent-environment interaction. Natural language becomes the primary programming interface, democratizing software development. This ecosystem supports single and multi-agent applications for executing a broad range of tasks.

MemGPT [57] presents a solution to overcome the fixed-length context window limitation in LLMs, a hurdle for tasks demanding deep analysis of lengthy dialogues or extensive texts. Drawing parallels with traditional OS hierarchical memory systems, MemGPT introduces virtual context management, echoing virtual memory principles. It enables LLMs to access information beyond immediate capacity, mirroring OS memory management through strategic 'paging'. This system reacts to events like user messages, system alerts, and timers, appending pertinent data to the primary context buffer before processing. Additionally, it supports function chaining for uninterrupted, sequential task execution, accommodating intricate operations and long-term planning within the LLM's context limitations.

8.2. LLM4OS

SecRepair [58], designed to address the challenge of identifying and repairing code vulnerabilities in software development. This system is powered by a LLM and incorporates reinforcement learning and semantic rewards to enhance its capabilities.To support the training process and prepare a robust model for vulnerability analysis, the authors have compiled and released a comprehensive instruction-based vulnerability dataset.They also propose a reinforcement learning technique with a semantic reward mechanism to generate concise and appropriate code commit comments. This technique is inspired by how humans fix code issues and provides developers with a clear understanding of the vulnerability and its resolution.

Rahman et al. [71] introduce ChronoCTI, an automated pipeline for mining temporal dynamics between attacker actions from text-based accounts of cyber incidents. The focus is on pinpointing repetitive action sequences, termed temporal attack patterns, crucial for preemptive defense strategies by security professionals against impending cyber threats. ChronoCTI is anchored by a curated dataset linking sentences to adversary maneuvers, temporal linkages across 94 attack narratives, and large language models refined on cybersecurity topics. The research delineates a structured approach, harnessing cutting-edge language models, NLP methods, and ML techniques to decipher the temporal structure within attack stories.

8.3. Findings

Addressing Research Question 5, our analysis highlights the incorporation of LLMs within operating systems heralds a transformative era, reshaping system dynamics and enhancing user engagement. They can enrich user interfaces by facilitating natural language processing for voice commands and text inputs, leading to more intuitive and personalized user experiences. In the realm of development, LLMs can automate code generation and optimization, streamlining coding processes and reducing errors. Additionally, they contribute to dynamic system management by analyzing usage patterns to optimize resource allocation, boosting system responsiveness. LLMs also play a pivotal role in intelligent troubleshooting, diagnosing system issues more accurately and recommending preventive measures to minimize downtime. Furthermore, they bolster OS security by detecting anomalies indicative of malicious activities and

assisting in secure configuration management. Lastly, LLMs act as intelligent aids for developers, offering insights into function queries, suggesting best practices, and even generating documentation, thus accelerating the development process and enhancing code quality. Through continuous learning, LLMs ensure that OS performance and efficiency improve over time, adapting to user needs and operational contexts.

9. Opportunities for Future Work

9.1. Optimized OS for AI

The escalating sophistication of AI workloads, characterized by high-velocity I/O, minimal latency, and bandwidth-intensive requirements, is catalyzing an urgent need for AI-specialized OS optimization. As AI permeates diverse sectors, the imperative for a finely calibrated infrastructure escalates, poised to support the burgeoning demands of next-generation applications. The emergence of AI-dedicated hardware has dramatically reshaped AI computation landscapes. These technological marvels, while delivering unrivaled performance, necessitate software ecosystems meticulously attuned to their singular attributes.

In this multifaceted hardware ecosystem [20] [23], the need for an OS that can seamlessly manage and orchestrate AI tasks across a spectrum of devices is more pressing than ever. AI has transcended beyond the realm of traditional neural networks, with innovative models leading the change in natural language processing and graph neural networks adeptly handling intricate relational data. These advanced applications are calling for specialized optimizations that a one-size-fits-all approach cannot provide. Juggling performance, energy conservation, and scalability constitutes a multifaceted challenge, demanding an OS equipped for dynamic adjustment to cater to the singular requisites of every workload.

Portatble OS for AI. In industrial settings, IoT devices generate massive amounts of data. Deploying deep learning models directly on edge devices (such as sensors, gateways, or edge servers) in equally important as relying solely on cloud-based processing. In a certain sense, deploying on edge devices can be a more challenging scenario because the computational power and bandwidth, of edge devices are more constrained [77] [12] [1]. Due to the diversity of edge devices,current AI platforms' lack of portability across different edge platforms hinders widespread adoption ,and bypassing the kernel and directly accessing hardware while adapting to various hardware configurations can be extremely challenging. Therefore, an operating system that can accommodate multiple hardware types and is specifically tailored for AI applications is a potential solution [7] [8].

Fast deployment and standard API interface. Simplified deployment means that even non-experts can utilize AI applications effectively. Developers, data scientists, and industrial engineers can quickly integrate the framework into their existing edge devices without extensive configuration or manual setup. And a standard API interface ensures that software components can communicate seamlessly with each other.

Minimized OS consumption. Resource-constrained environments, such as edge devices or embedded systems, often operate with tight constraints on memory, processing power, and energy while in scenarios like cloud computing or data centers, minimizing resource consumption directly impacts operational expenses. As OS and applications share the resources, the OS consumption determines the actual operational performance of the algorithmic model. So there is a need to design an efficient OS that can achieve small memory and fast training and inference. There are several promising approaches to achieve this. Apart from bypassing the kernel, unikernel OSes are designed to be fast and lightweight [11] [32].

Privacy and security. The problem of privacy preservation in the context of machine learning is quite different from that in traditional data privacy protection, as machine learning can act as both friend and foe. Increasingly, edge devices are equipped with AI applications. This trend comes with privacy risks as models can leak information about their training data [24]. OS need to find ways to provide safeguards for ensuring the confidentiality and integrity of its data and programs.

9.2. Intelligent OS

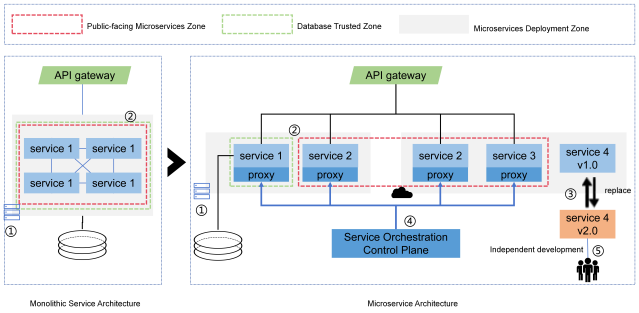

Figure 3. Monolithic application to Microservice: ① Microservices enable independent, scalable deployments ② Microservices provide fine-grained security controls ③ Microservice enables fine-grained flexible upgrading and replacement of individual microservice ④ Microservice enables service intelligent orchestration ⑤ Microservice allows agile and independent development for each service

9.2.1. Potential for Intelligent LibOS. A recently emerging trend of Internet-based software systems is "resource adaptive," i.e., software systems should be robust and intelligent enough to the changes of heterogeneous resources [9], both physical and logical, provided by their running environment [26]. The key principle of such an OS is based on resource disaggregation, resource provisioning as a service, and learning-based resource scheduling and allocation. Meanwhile, the OS should leverage advanced machine/deep

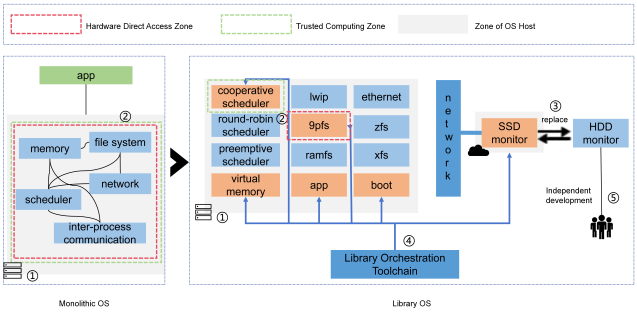

Figure 4. Monolithic OS to LibOS:①LibOS spans across multiple hosts.②LibOS offers flexible, granular security and kernel-bypass capabilities.③Only LibOS supports granular updates to individual components.④LibOS enables orchestrating components intelligently for adaptable system contexts.⑤LibOS facilitates agile, independent development of single components.

learning techniques to derive configurations and policies and automatically learn to tune itself and schedule resources. One general and efficient Spark tuning framework is proposed and applied it to Tencent's data platform [5].

LibOS have risen to prominence as highly attractive options in scenarios that prioritize agility and robust security, thanks to their inherent minimalist and modular architectures. Figure 3 illustrates the progression from monolithic applications to microservices, and figure 4 the contrast between a monolithic OS and LibOS.

Microservices excels notably in granular security, isolated updates, and performance optimization through dynamic service management. AI technologies have proven adept at governing microservices, enabling intelligent service orchestration and customization, as highlighted in research [65] and [66], setting the stage for advanced system control and optimization [81].LibOS similarly exhibit gains in agility, composability, enhanced isolation, and facilitated deployment across heterogeneous hardware. LibOS presents a more auspicious ground for intelligent OS. There are at least 5 ways where LibOS will benefit to enable a Intelligent OS:

Isolation and security. FlexOS [82] presents a modular LibOS framework, comprised of discrete elements that can be segregated using a spectrum of hardware safeguards, complemented by adaptable data sharing protocols and software fortification methods. FlexOS adopts micro-libraries as fundamental units for segmentation. Integration or isolation of these components is dynamically managed, guided by safety and performance requirements.

Drawing from the microservices paradigm's strengths in isolation and precise access governance, we posit that LibOS offers a strategic avenue to uphold data confidentiality during processing and safeguard overall system integrity. In the AI domain, where security and privacy are paramount [37]. LibOS's robust isolation features make them ideally suited for secure operation in environments with concurrent tasks of varying privacy sensitivity. The innate simplicity and modularity of LibOS create an ecosystem that effectively quarantines critical workloads, bolstering confidence and aligning with rigorous privacy statutes.

Heterogeneous deployment. In the IoT epoch, a surge in varied endpoint devices underscores the imperative for AI integration. Yet, conventional monolithic OS kernels struggle, weighed down by excessive resource demands and challenges in hardware acclimatization, hindering their agility in today's rapidly transforming milieu. In alignment, LibOS surface as a beacon of hope for versatile, cross-platform deployment amid the AI revolution. The Demikernel project [52] substantiates LibOS's flexibility, enabling applications to operate fluidly across disparate devices sans customization. In LibOS's schema, hardware interface components are architected as pluggable modules, amplifying interoperability and malleability.The intrinsic modularity and adaptability simplify AI assimilation into a spectrum of devices, nurturing a more dynamic and reactive IoT framework.

Intelligent OS orchestration. A significant advancement brought about by the microservices paradigm is its capacity for orchestration for components,exemplified by tools like ISTIO. This capability serves as an inspiration for the realization that similar traffic control principles can be harnessed within the realm of LibOS to achieve intelligent resource management. In the context of demanding tasks such as machine learning training that consume vast amounts of resources, as well as in resource-constrained edge devices, meticulous oversight over resource allocation is paramount. This empowers dynamic, autonomous resource allocation, optimizing performance and efficiency across varied workloads and settings, perfectly suiting complex, evolving AI applications and infrastructures.

Intelligent OS library replacement. The modular LibOS architecture promotes autonomous component operation through precise interfaces, ensuring loose coupling and stable system evolution akin to microservices. Components can be independently upgraded, patched, or optimized, minimizing disruption and extensive testing. Upon detecting issues like performance drops, an orchestration layer automates replacements, preserving system robustness, security, and adaptability without compromising user experience or structural coherence.

Intelligent OS development. Microservices architecture heralds a seismic shift in software development paradigms, dismantling centralized structures in favor of agile, decentralized teams, each specializing in a unique microservice [83]. This transition boosts efficiency and reliability, as focused teams oversee discrete services, revolutionizing software engineering approaches. Contrastingly, monolithic systems' vast scale and tangled dependencies escalate code generation complexities. Automating code in large, interconnected systems demands meticulous integration within a dense module network [84]. LibOS, with their compact, independent nature, simplify code generation, offering fertile ground for efficient automation and AI-driven development.

9.2.2. Multi-agent LLM for OS Development. Previous research has demonstrated the efficacy of employing AI in optimizing, safeguarding, and evaluating system software including operating systems [10] [22] [28]. Our study reveals a notable dearth of efforts dedicated to leveraging LLMs in the context of operating systems. Nonetheless, we identify at least one distinct domains where LLMs can potentially revolutionize the functioning and capabilities of OSes as Multi-agent LLM for OS Development.

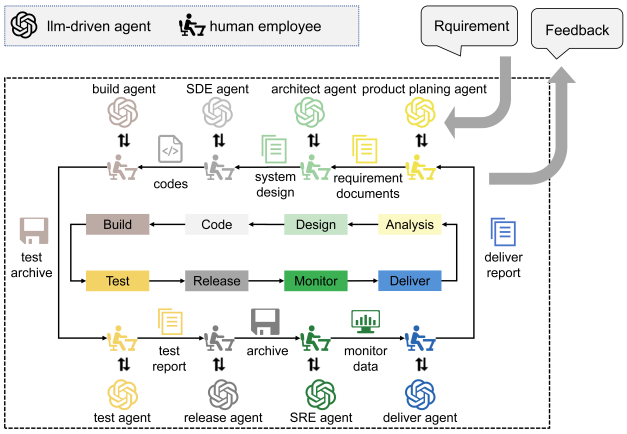

Figure 5. One example for Multi-agent LLM for OS Development

Recent breakthroughs in LLM-based multi-agent systems, building on the success of single LLMs as unified decision-makers, have unlocked new horizons in complex problem-solving and environment simulation. This progress signals the potential for multi-agent LLMs to revolutionize OS development. Studies like [79] and [80] showcase multi-agent LLM models' efficacy in software development, from analysis to debugging.

In the realm of OS development, the integration of multi-agent systems fosters a specialized, collaborative, and flexible environment. A constellation of agents, each dedicated to niches such as kernel architecture, driver engineering, or cybersecurity, collectively enhances project productivity [72] [73] [74] [75]. these agents facilitate knowledge exchange and consensus-building akin to human collaboration. They concurrently tackle OS components, recalibrate in response to evolving project dynamics, and oversee integration, testing, and debugging phases, proactively addressing potential issues.Inherently adaptable, the system reassigns agent responsibilities or integrates fresh entities to align with technological advancements, market dynamics, or emerging security vulnerabilities, ensuring continuous OS refinement. Certain agents conduct simulations to assess OS performance under varied scenarios, supplying empirical insights to inform judicious decision-making, preserving the OS's technological relevance and competitiveness.

9.2.3. LLM AS OS. Future research in the domain of LLMs integrated as OS will focus on following areas.

Platform-Agnostic Adaptation. LLMs' inherent understanding of natural language empowers them to interpret a spectrum of system artifacts, from logs to hardware specs, irrespective of software platforms or hardware architectures. This capability facilitates dynamic OS parameter adjustment and context-aware optimization strategies, ensuring the system stays in sync with hardware advancements and performs optimally across diverse ecosystems.

Intelligent Resource Allocation. Leveraging pattern recognition and predictive analytics, LLMs can make informed real-time resource management decisions, preemptively addressing resource demands and workload fluctuations. This proactive approach optimizes system performance, reduces latency, and maximizes resource utilization.

Natural Language Interface. Integrating LLMs into the OS interface enables a conversational interaction model, replacing traditional interfaces with intuitive text-based requests. This simplifies user engagement and broadens accessibility, enhancing user experience and lowering the learning curve.

Customized User Experience. LLMs learn from user behavior and preferences, dynamically adjusting the OS interface and functionality to cater to individual needs. This personalization fosters a more engaging and productive user interface, tailored to unique usage patterns.

Enhanced Security Measures. LLMs contribute to OS security by analyzing data for threats, enforcing policies, and managing access. They can predict security incidents, provide real-time code analysis, and offer personalized security guidance, fortifying the system against vulnerabilities.

10. Conclusion

The synthesis of AI and OS is reshaping the landscape of contemporary computing, as evidenced by our meticulous review of 108 primary studies. AI's integration into OS frameworks has catalyzed significant improvements in key areas like memory oversight, task coordination, and security vigilance. By leveraging intelligent decision-making beyond traditional rule-bound paradigms, AI bolsters resource efficiency and system agility.

In tandem, advanced OS designs are morphing to better serve AI applications, especially as data-intensive workloads surge across various computing platforms. The OS's evolving role in adeptly handling specialized hardware and optimizing data flow underscores its importance in sustaining AI software performance.

The integration of LLMs into OS interfaces, as highlighted in our analysis of 17 studies, marks a pivotal advancement in user experience. Conversational interfaces enable seamless human-computer interaction, simplifying device management and personalizing user assistance through natural language processing.

Despite the optimistic outlook, integrating AI into OS presents critical challenges. Safeguarding against security breaches and privacy infringements while maintaining efficient AI operations necessitates rigorous design considerations. Moreover, the demand for streamlined AI algorithms that can operate within constrained environments remains a high-priority research area.

Future research agendas should concentrate on the domain of intelligent operating systems, capitalizing on the transformative potential they offer. Central to this endeavor is the exploration of LibOS. With its modular structure, LibOS facilitates the seamless incorporation of AI enhancements, supports granular control, fosters innovation, and upholds stringent security standards. LibOS stands as a robust foundation for the evolution of intelligent computing. Thus, advancing LibOS technologies should be a focal point for researchers aiming to realize the full scope of intelligent OS capabilities and drive the computing industry toward greater adaptability, efficiency, and resilience.

Acknowledgments

This work was supported in part by the Major Program of National Natural Science Foundation of Zhejiang(LD24F020014), and in part by the Zhejiang Pioneer (Jianbing) Project (2024C01032), and in part by the Key R&D Program of Ningbo(2023Z235), and in part by the Ningbo Yongjiang Talent Programme(2023A-198-G).

References

[1] A. Imteaj, U. Thakker, S. Wang, J. Li and M. H. Amini, "A Survey on Federated Learning for Resource-Constrained IoT Devices," in IEEE Internet of Things Journal, vol. 9, no. 1, pp. 1-24, 1 Jan.1, 2022, doi: 10.1109/JIOT.2021.3095077.

[2] T. Li, S. Ying, Y. Zhao and J. Shang, "Batch Jobs Load Balancing Scheduling in Cloud Computing Using Distributional Reinforcement Learning," in IEEE Transactions on Parallel and Distributed Systems, vol. 35, no. 1, pp. 169-185, Jan. 2024, doi: 10.1109/TPDS.2023.3334519.

[3] Chen, J., Banerjee, S. S., Kalbarczyk, Z. T., et al. (2020, August). Machine learning for load balancing in the linux kernel. In Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems (pp. 67-74).

[4] Fingler, H., Tarte, I., Yu, H.,et al. (2023, January). Towards a Machine Learning-Assisted Kernel with LAKE. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (pp. 846-861).

[5] Li, Y., Jiang, H., Shen, Y., et al. (2023). Towards General and Efficient Online Tuning for Spark. arXiv preprint arXiv:2309.01901.

[6] Hao, M., Toksoz, L., Li, N.,et al. (2020). LinnOS: Predictability on unpredictable flash storage with a light neural network. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20) (pp. 173-190).

[7] Mi H B, Xu K L, Feng D W, et al. Collaborative deep learning across multiple data centers. Sci China Inf Sci, 2020, 63(8): 182102, doi: 10.1007/s11432-019-2705-2

[8] Ma N, Li D Y, He W, et al. Future vehicles: interactive wheeled robots. Sci China Inf Sci, 2021, 64(5): 156101, doi: 10.1007/s11432-020-3171-4

[9] Zhang X D. Software system research in post-Moore's Law era: a historical perspective for the future. Sci China Inf Sci, 2019, 62(9): 196101, doi: 10.1007/s11432-019-9860-1

[10] Roychoudhury A, Xiong Y F. Automated program repair: a step towards software automation. Sci China Inf Sci, 2019, 62(10): 200103, doi: 10.1007/s11432-019-9947-6

[11] Kuenzer, S., Bădoiu, V. A., Lefeuvre, H.,et al. (2021, April). Unikraft: fast, specialized unikernels the easy way. In Proceedings of the Sixteenth European Conference on Computer Systems (pp. 376-394).

[12] Hao-Rui Chen, Lei Yang, Xinglin Zhang, Jiaxing Shen, Jiannong Cao Distributed Semi-Supervised Learning With Consensus Consistency on Edge Devices. IEEE Trans. Parallel Distributed Syst. 35(2): 310-323 (2024)

[13] Q. Sun, Y. Liu, H. Yang, Z. Jiang, Z. Luan and D. Qian, "Adaptive Auto-Tuning Framework for Global Exploration of Stencil Optimization on GPUs," in IEEE Transactions on Parallel and Distributed Systems, vol. 35, no. 1, pp. 20-33, Jan. 2024, doi: 10.1109/TPDS.2023.3325630.

[14] Goodarzy, S., Nazari, M., Han, R., et al. (2021, August). SmartOS: towards automated learning and user-adaptive resource allocation in operating systems. In Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems (pp. 48-55).

[15] de A. Rocha, H. M. G., Schwarzrock, J., Lorenzon, A. F.,et al. (2022, July). Using machine learning to optimize graph execution on numa machines. In Proceedings of the 59th ACM/IEEE Design Automation Conference (pp. 1027-1032).

[16] J. Liu et al., "Multi-Job Intelligent Scheduling With Cross-Device Federated Learning," in IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 2, pp. 535-551, 1 Feb. 2023, doi: 10.1109/TPDS.2022.3224941.

[17] Zhang, J., Swift, M., & Li, J. (2022, February). Software-defined address mapping: a case on 3d memory. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 70-83).

[18] Carvalho, P., Clua, E., Paes, A., et al. (2020). Using machine learning techniques to analyze the performance of concurrent kernel execution on GPUs. Future Generation Computer Systems, 113, 528-540.

[19] Serizawa, K., & Tatebe, O. (2019, December). Accelerating machine learning i/o by overlapping data staging and mini-batch generations. In Proceedings of the 6th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (pp. 31-34).

[20] Chen, R., & Sun, G. (2018, December). A survey of kernel-bypass techniques in network stack. In Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence (pp. 474-477).

[21] Wang, T., Chen, Z., Wei, X.,et al. (2024). Characterizing Network Requirements for GPU API Remoting in AI Applications. arXiv preprint arXiv:2401.13354.

[22] Xiong Y F, Tin Y Q, Liu Y P, et al. Toward actionable testing of deep learning models. Sci China Inf Sci, 2023, 66(7): 176101, doi: 10.1007/s11432-022-3580-5

[23] Ahmed, U., Lin, J. C. W., & Srivastava, G. (2022). Heterogeneous energy-aware load balancing for industry 4.0 and IoT environments. ACM Transactions on Management Information Systems (TMIS), 13(4), 1-23.

[24] Mo, F., Shamsabadi, A. S., Katevas, K., et al. (2020, June). Darknetz: towards model privacy at the edge using trusted execution environments. In Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (pp. 161-174).

[25] Serizawa, K., & Tatebe, O. (2019, December). Accelerating machine learning i/o by overlapping data staging and mini-batch generations. In Proceedings of the 6th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (pp. 31-34).

[26] Liu, X., Wang, S., Ma, Y., Zhang, Y., Mei, Q., Liu, Y., & Huang, G. (2021). Operating systems for resource-adaptive intelligent software: Challenges and opportunities. ACM Transactions on Internet Technology (TOIT), 21(2), 1-19.

[27] Leon M. The dark side of unikernels for machine learning. arXiv preprint arXiv:2004.13081. 2020 Apr 27.

[28] Mei H, Zhang L. Can big data bring a breakthrough for software automation?. Sci China Inf Sci, 2018, 61(5): 056101, doi: 10.1007/s11432-017-9355-3

[29] Cardoso, A. P., Santos, C. P., Collins, E., et al. (2023, November). Evaluation of Automatic Test Case Generation for the Android Operating System using Deep Reinforcement Learning. In Proceedings of the XXII Brazilian Symposium on Software Quality (pp. 228-235).

[30] L. L. Pilla, "Scheduling Algorithms for Federated Learning With Minimal Energy Consumption," in IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 4, pp. 1215-1226, April 2023, doi: 10.1109/TPDS.2023.3240833.

[31] White, T. D., Fraser, G., & Brown, G. J. (2019, July). Improving random GUI testing with image-based widget detection. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 307-317).

[32] Chowdhury, M., Ray, B., Chowdhury, S., et al. (2021). A novel insider attack and machine learning based detection for the internet of things. ACM Transactions on Internet of Things, 2(4), 1-23.

[33] Suzaki, K., Tsukamoto, A., Green, A.,et al. (2020, December). Reboot-oriented IoT: Life cycle management in trusted execution environment for disposable IoT devices. In Proceedings of the 36th Annual Computer Security Applications Conference (pp. 428-441).

[34] Fu, M., Tantithamthavorn, C., Le, T., Nguyen, V.,et al. (2022, November). Vulrepair: a t5-based automated software vulnerability repair. In Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering (pp. 935-947).

[35] Qin, X., Zeng, F., & Zhang, Y. (2019, May). MSNdroid: the Android malware detector based on multi-class features and deep belief network. In Proceedings of the ACM Turing Celebration Conference-China (pp. 1-5).

[36] Ongun, T., Stokes, J. W., Or, J. B., et al. (2021, October). Living-off-the-land command detection using active learning. In Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses (pp. 442-455).

[37] Mo, F., Haddadi, H., Katevas, K., et al. (2022). Ppfl: Enhancing privacy in federated learning with confidential computing. GetMobile: Mobile Computing and Communications, 25(4), 35-38.

[38] Pemberton, N., Schleier-Smith, J., & Gonzalez, J. E. (2021, June). The restless cloud. In Proceedings of the Workshop on Hot Topics in Operating Systems (pp. 49-57).

[39] Panman de Wit, J. S., Bucur, D., & van der Ham, J. (2022). Dynamic detection of mobile malware using smartphone data and machine learning. Digital Threats: Research and Practice (DTRAP), 3(2), 1-24.

[40] Cruz-Carlon, J., Varshosaz, M., Le Goues, C., et al. (2023). Patching locking bugs statically with crayons. ACM Transactions on Software Engineering and Methodology, 32(3), 1-28.

[41] Schäfer, M., Nadi, S., Eghbali, A., et al. (2023). An empirical evaluation of using large language models for automated unit test generation. IEEE Transactions on Software Engineering.

[42] Yang, X., & Bai, Z. (2022, June). Improvement of lottery scheduling algorithm based on machine learning algorithm. In Proceedings of the 2022 2nd International Conference on Control and Intelligent Robotics (pp. 894-897).

[43] Qi, C., Shao, S., Guo, Y.,et al. (2021, February). An Efficient Method for Analyzing Widget Intent of Android System. In Proceedings of the 2021 9th International Conference on Communications and Broadband Networking (pp. 78-85).

[44] He, P., Xia, Y., Zhang, X., et al. (2023, November). Efficient query-based attack against ML-based Android malware detection under zero knowledge setting. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (pp. 90-104).

[45] Chengeta, K. (2021, December). Comparing the performance between Virtual Machines and Containers using deep learning credit models. In Proceedings of the International Conference on Artificial Intelligence and its Applications (pp. 1-8).

[46] Chochliouros, I. P., Pages-Montanera, E., Alcázar-Fernández, A., et al. (2023, October). NEMO: Building the Next Generation Meta Operating System. In Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum (pp. 1-9).

[47] Yan M, Xia X, Zhang X H, et al. Software quality assessment model: a systematic mapping study. Sci China Inf Sci, 2019, 62(9): 191101, doi: 10.1007/s11432-018-9608-3

[48] Zhao, Z., Kou, B., Ibrahim, M. Y.,et al. (2023, November). Knowledge-based version incompatibility detection for deep learning. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 708-719).

[49] Metzger, P., Seeker, V., Fensch, C., et al. (2021). Device Hopping: Transparent Mid-Kernel Runtime Switching for Heterogeneous Systems. ACM Transactions on Architecture and Code Optimization (TACO), 18(4), 1-25.

[50] Yu, S., Fang, C., Ling, Y.,et al. (2023, October). LLM for test script generation and migration: Challenges, capabilities, and opportunities. In 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS) (pp. 206-217). IEEE.

[51] Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., ... & Winwood, S. (2009, October). seL4: Formal verification of an OS kernel. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (pp. 207-220).

[52] Zhang, I., Raybuck, A., Patel, P., et al. (2021, October). The demikernel datapath os architecture for microsecond-scale datacenter systems. In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (pp. 195-211).

[53] Raza, A., Unger, T., Boyd, M.,et al. (2023, May). Unikernel Linux (UKL). In Proceedings of the Eighteenth European Conference on Computer Systems (pp. 590-605).

[54] Kamath, A. K., & Yadalam, S. (2024). Herding LLaMaS: Using LLMs as an OS Module. arXiv preprint arXiv:2401.08908.

[55] Ge, Y., Ren, Y., Hua, W., et al. (2023). LLM as OS, Agents as Apps: Envisioning AIOS, Agents and the AIOS-Agent Ecosystem. arXiv e-prints, arXiv-2312.

[56] Mei, K., Li, Z., Xu, S., et al. (2024). LLM Agent Operating System. arXiv preprint arXiv:2403.16971.

[57] Packer, C., Fang, V., Patil, S. G., et al. (2023). Memgpt: Towards llms as operating systems. arXiv preprint arXiv:2310.08560.

[58] Islam, N. T., Khoury, J., Seong, A., et al. (2024). LLM-Powered Code Vulnerability Repair with Reinforcement Learning and Semantic Reward. arXiv preprint arXiv:2401.03374.

[59] Cadden, J., Unger, T., Awad, Y.,et al. (2020, April). SEUSS: skip redundant paths to make serverless fast. In Proceedings of the Fifteenth European Conference on Computer Systems (pp. 1-15).

[60] Skiadopoulos, A., Li, Q., Kraft, P.,et al. (2021). DBOS: a DBMS-oriented Operating System.

[61] Shan, Y., Huang, Y., Chen, Y.,et al. (2018). LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18) (pp. 69-87).

[62] Achiam, J., Adler, S., Agarwal, S., et al. (2023). Gpt-4 technical report. arXiv preprint arXiv:2303.08774.

[63] Touvron, H., Lavril, T., Izacard, G., et al. (2023). Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971.

[64] Team, G., Anil, R., Borgeaud, S.,et al. (2023). Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805.

[65] Liu, J., Zhang, S., & Wang, Q. (2023, October). μConAdapter: Reinforcement Learning-based Fast Concurrency Adaptation for Microservices in Cloud. In Proceedings of the 2023 ACM Symposium on Cloud Computing (pp. 427-442).

[66] Hussain,F., Li W., Noye, B., et al, .Intelligent Service Mesh Framework for API Security and Management, 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 2019, pp. 0735-0742, doi: 10.1109/IEMCON.2019.8936216.

[67] Zhang C, Lu W, Ni C,et al. Enhanced User Interaction in Operating Systems through Machine Learning Language Models. arXiv preprint arXiv:2403.00806. 2024 Feb 24.

[68] Wu Z, Han C, Ding Z,et al, Kong L. Os-copilot: Towards generalist computer agents with self-improvement. arXiv preprint arXiv:2402.07456. 2024 Feb 12.

[69] Xing M, Zhang R, Xue H,et al. Understanding the Weakness of Large Language Model Agents within a Complex Android Environment. arXiv preprint arXiv:2402.06596. 2024 Feb 9.

[70] Wu W, Yang W, Li J,et al. Autonomous Crowdsensing: Operating and Organizing Crowdsensing for Sensing Automation. IEEE Transactions on Intelligent Vehicles. 2024 Jan 19.

[71] Rahman, M. R., Wroblewski, B., Matthews, Q.,et al. (2024). Mining Temporal Attack Patterns from Cyberthreat Intelligence Reports. arXiv preprint arXiv:2401.01883.

[72] Yang C, Zhao Z, Zhang L. KernelGPT: Enhanced Kernel Fuzzing via Large Language Models. arXiv preprint arXiv:2401.00563. 2023 Dec 31.

[73] Zheng Y, Yang Y, Chen M, et al. KEN: Kernel Extensions using Natural Language. arXiv preprint arXiv:2312.05531. 2023 Dec 9.

[74] Alrashedy K, Aljasser A. Can LLMs Patch Security Issues?. arXiv preprint arXiv:2312.00024. 2023 Nov 13.

[75] Xing Z, Huang Q, Cheng Y,et al. Prompt sapper: Llm-empowered software engineering infrastructure for ai-native services. arXiv preprint arXiv:2306.02230. 2023 Jun 4.

[76] Hè H. PerOS: Personalized Self-Adapting Operating Systems in the Cloud. arXiv preprint arXiv:2404.00057. 2024 Mar 26.

[77] S. Wang et al., "When Edge Meets Learning: Adaptive Control for Resource-Constrained Distributed Machine Learning," IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Honolulu, HI, USA, 2018, pp. 63-71, doi: 10.1109/INFOCOM.2018.8486403.

[78] A. Imteaj, U. Thakker, S. Wang, J. Li and M. H. Amini, "A Survey on Federated Learning for Resource-Constrained IoT Devices," in IEEE Internet of Things Journal, vol. 9, no. 1, pp. 1-24, 1 Jan.1, 2022, doi: 10.1109/JIOT.2021.3095077.

[79] Qian, C., Cong, X., Yang, C., et al. (2023). Communicative agents for software development. arXiv preprint arXiv:2307.07924.

[80] Hong, S., Zheng, X., Chen, J., et al. (2023). Metagpt: Meta programming for multi-agent collaborative framework. arXiv preprint arXiv:2308.00352.

[81] Alshuqayran N, Ali N, Evans R. A systematic mapping study in microservice architecture. In2016 IEEE 9th international conference on service-oriented computing and applications (SOCA) 2016 Nov 4 (pp. 44-51). IEEE.

[82] Lefeuvre, H., Bădoiu, V. A., Jung, A., et al. (2022, February). FlexOS: towards flexible OS isolation. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 467-482).

[83] Villamizar M, Garcés O, Castro H, et al. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In2015 10th computing colombian conference (10ccc) 2015 Sep 21 (pp. 583-590). IEEE.

[84] Wong, M. F., Guo, S., Hang, C. N., et al. (2023). Natural language generation and understanding of big code for ai-assisted programming: A review. Entropy, 25(6), 888.

[85] Wu, R., Shen, Z., Yang, Z.,et al. (2024, March). Mitigating Write Disturbance in Non-Volatile Memory via Coupling Machine Learning with Out-of-Place Updates. In 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) (pp. 1184-1198). IEEE.

[86] Aaen Springborg, A., Albano, M., & Xavier-de-Souza, S. (2023, November). Automatic Energy-Efficient Job Scheduling in HPC: A Novel SLURM Plugin Approach. In Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (pp. 1831-1838).

[87] Wang, S., Lin, Z., Wu, S.,et al. (2024, March). LearnedFTL: A Learning-Based Page-Level FTL for Reducing Double Reads in Flash-Based SSDs. In 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) (pp. 616-629). IEEE.

[88] Lagar-Cavilla, A., Ahn, J., Souhlal, S., et al. (2019, April). Software-defined far memory in warehouse-scale computers. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 317-330).

[89] Bateni, S., & Liu, C. (2020). NeuOS: A Latency-PredictableMulti-Dimensional Optimization Framework for DNN-driven Autonomous Systems. In 2020 USENIX Annual Technical Conference (USENIX ATC 20) (pp. 371-385).