AIOS综述:安全可信的人工智能原生操作系统

吴庆波[1],伍复慧[2],刘海天[1],刘 军[1],李英俊[1],农俊康[1],
杜宇琪[1],吴昊泽[1],陈龙腾[1],王 静[3]
(1. 麒麟软件有限公司2030实验室,湖南长沙 410000;2. 武汉学院信息工程学院,湖北武汉 430212;
3. 国防科技大学计算机学院,湖南长沙 410000)

摘 要: 随着人工智能技术的快速发展,传统操作系统在安全性、智能性与场景适配性方面面临新的挑战。本文系统梳理了安全可信的人工智能原生操作系统(AI-native Operating System,AIOS)的研究进展与关键技术。首先,设计AIOS的分层架构,包括内核组件库、资源管理服务、共性基础服务以及领域基础软件栈。面向资源、运维与安全的系统级智能化服务框架和围绕安全内核架构、软硬件协同的安全与适配,以及虚拟化支撑的纵深防御安全访问框架构成共性基础服务。通过标准化接口与内核/核心服务在调度、内存、I/O与安全域等方面向上提供领域基础软件栈接口(涵盖人工智能(Artificial Intelligence,AI)软件栈与安全软件栈),支撑行业场景落地。其次,提出人工智能原生操作系统的关键技术:端侧智能操作系统聚焦模型压缩、推理优化(如键值(Key-Value,KV)缓存与分页注意力)与端云协同的异构资源调度及硬件抽象与驱动适配;记忆管理系统围绕记忆存储、检索与管理,结合短期、长期记忆以支撑长上下文理解与动态更新;安全可信机制涵盖内存安全与形式化验证、自动化检测、可信启动与远程证明,以及基于可信执行环境(Trusted Execution Environment,TEE)、机密计算与虚拟化隔离的纵深防御。随后,结合应用场景,综述AIOS在具身智能、工业与移动等领域的部署路径、能力边界与工程权衡,分析端侧推理、实时控制、跨设备协同与多模态交互的实践挑战。进一步,围绕新型操作系统基础技术、智能服务框架与关键技术梳理及安全可信基石,总结架构模块化与软硬件协同、智能服务的资源调度与安全鲁棒性、记忆系统的效率与隐私保护、领域基础软件栈的标准化与生态建设,以及跨平台与跨设备的智能协同与资源共享等开放问题与趋势。本文贡献在于提供结构化视角、术语边界与技术地图,并对典型系统与工具进行对比归纳,为AIOS研究与工程落地提供参考。

关键词: AIOS;人工智能原生;安全可信;内核架构;端侧智能;智能体应用
基金项目: 教育部协同育人项目(No.240800006162639,No.241001613173522)
中图分类号: TP18; **文献标识码:** AA **文章编号:** 0372-2112(2026)04-1584-28
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250763

Systematic Survey of AIOS: A Secure and Trustworthy AI-Native Operating System

WU Qingbo[1], WU Fuhui[2], LIU Haitian[1], LIU Jun[1], LI Yingjun[1], NONG Junkang[1],
DU Yuqi[1], WU Haoze[1], CHEN Longteng[1], WANG Jing[3]
(1. 2030 *Laboratory*, *KylinSoft Co.* , *Ltd.* , *Changsha*, *Hunan* 410000, *China*;
2. *School of Information Engineering*, *Wuhan College*, *Wuhan*, *Hubei* 430212, *China*;
3. *College of Computer Science and Technology*, *National University of Defense Technology*, *Changsha*, *Hunan* 410000, *China*)

Abstract: With the rapid development of artificial intelligence technology, traditional operating systems face new challenges in security, intelligence, and scenario adaptability. This survey systematically reviews the research progress and key technologies of secure and trustworthy AI-native operating systems (AIOS). First, it introduces the layered architecture of AIOS, including kernel components, resource management, general foundational services (covering both the intelligent service framework and the secure access framework), and domain-specific software stacks, highlighting the deep integration of artificial intelligence (AI) capabilities with operating systems. The secure access framework focuses on secure kernel architecture, software-hardware collaborative security and adaptation, and virtualization-based in-depth defense. Second, it summarizes the key AIOS technologies: edge-oriented operating systems emphasize model compression, inference optimization (e.g., key-value (KV) cache and PagedAttention), edge-cloud collaboration with heterogeneous resource scheduling, and hardware abstraction and driver adaptation; the memory management system centers on memory storage, retrieval, and

收稿日期:2025-09-04;录用日期:2025-12-10;责任编辑:覃怀银
*通信作者:吴庆波

management, combining short-term and long-term memory to support long context understanding and dynamic updates; the security and trust mechanisms cover memory safety and formal verification, automated analysis, trusted boot and remote attestation, and defense-in-depth via trusted execution environment (TEE)/confidential computing and virtualization isolation. Finally, by examining practical scenarios such as embodied intelligence, industrial internet of things (IoT), and mobile applications, it analyzes deployment paths, capability boundaries, and engineering trade-offs, and summarizes open issues and trends including architectural modularity, co-design of hardware and software, resource scheduling and security robustness in intelligent services, efficiency and privacy of memory systems, standardization and ecosystem building of domain stacks, and cross-platform/device collaboration and resource sharing. This survey aims to provide a structured perspective, clear terminology boundaries, and a technology map, offering references for AIOS research and engineering practice.

Keywords: AIOS; AI-native; security and trustworthiness; kernel architecture; edge intelligence; intelligent agent applications

Foundation Item(s): Collaborative Education Project of the Ministry of Education (No.240800006162639, No.241001613173522)

0 引言

全球计算产业正处于以人工智能(Artificial Intelligence,AI)为核心驱动力的深刻变革阶段,这一变革不仅重构了技术体系,更重塑了产业竞争格局。与此同时,传统操作系统在安全性、智能性和场景适配性等方面面临着前所未有的挑战。如何让操作系统更好地适应人工智能技术的快速发展,满足多样化智能应用场景的需求,成为当前学术界和产业界关注的焦点。

(1) 计算产业迎来智能化变革

人工智能是推动全球数字化发展的重要赋能技术,正在引领新一轮科技革命和产业变革;加快培育和推进AI核心软硬件技术及产业发展,对推动产业优化升级、生产力整体跃升具有重要战略意义[1]。2017年7月,国务院发布《新一代人工智能发展规划》[2],明确提出了面向2030年我国新一代人工智能发展的指导思想、战略目标、重点任务与保障措施,为我国人工智能发展奠定了先发优势;同年12月,工业和信息化部印发《促进新一代人工智能产业发展三年行动计划(2018—2020年)》,明确提出推动神经网络芯片量产并在重点领域实现规模化应用,对加速我国人工智能芯片自主研发与产业推进产生深远影响。

与此同时,全球多个国家也积极布局人工智能发展战略,将其视为新一轮科技竞争与产业变革的核心。自2013年起,包括美国、中国、欧盟、英国、日本、德国、法国、韩国、印度、俄罗斯、加拿大、越南等在内的20余个国家和地区相继颁布国家级人工智能战略或重大计划[3]。欧盟诸国于2018年4月共同签署《人工智能合作宣言》,协力推进人工智能发展;东盟也积极制定《东盟数字融合框架行动计划》,以促进区域人工智能合作。

其中,以美国为代表的发达国家凭借其在人工智能基础理论、技术积累、高端人才和产业生态等方面的先发优势,较早展开系统性布局。为巩固其领先地位,美国于2016年发布《美国国家人工智能研究和发展战略计划》[4];日本、加拿大、阿联酋等国也于2017年将人工智能上升为国家战略。此后,美国持续加强政策引导和支持力度,于2019年签署《维护美国人工智能领导力的行政命令》[5],启动“美国人工智能计划”,将人工智能研发列为国家优先事项。同年6月,美国政府更新并发布《国家人工智能研发战略计划(2019年版)》[6],将原七大战略扩展为八大战略,进一步强化其对前沿人工智能领域的引领作用。

(2) 操作系统的智能化挑战与AI原生趋势

随着软硬件技术的进步,CPU(Central Processing Unit)、GPU(Graphics Processing Unit)、NPU(Neural Processing Unit)等算力持续提升,推动了高性能计算、人工智能和机器学习等新兴领域的快速发展。与此同时,AI大语言模型(Large Language Model,LLM)、深度学习等技术对操作系统提出了全新挑战:不仅需要支持大规模数据中心的智能计算,还要适应边缘设备、物联网等多样化场景。现有操作系统在资源管理、任务调度、设备驱动等方面多为被动响应式,难以满足AI场景下高动态负载、多模态交互、端侧资源约束等需求。

此外,随着机器人、自动驾驶、工业物联网等智能应用的兴起,操作系统需覆盖桌面、移动、服务器信息空间与穿戴、车载、工控等物理空间[7],并具备AI驱动的动态适配能力。传统操作系统的设计理念和技术栈难以跟上硬件和AI技术的快速演进,暴露出性能优化、功耗控制、安全性等多方面的局限。

在此背景下,操作系统正经历从“被动响应”向“主动智能”转型。人工智能原生操作系统成为新趋势,其核心特征是将AI能力深度嵌入内核、服务与应用层,使操作系统具备感知、推理、决策和自适应优化能力,成为支撑全场景AI应用的系统级基座。

图1展现了操作系统从大型机时代的命令行交互与集中式资源管理,经历个人电脑互联网时代的图

形界面与浏览器、移动互联网时代的多点触控与应用生态，最终迈向以AI为核心驱动力、支持自然多模态交互和智能决策的智能时代。操作系统在不断适应计算场景和交互方式变革的过程中，逐步实现了从被动响应到主动智能、从单一终端到多样化硬件和复杂应用的跨越，成为智能社会的关键基础设施。

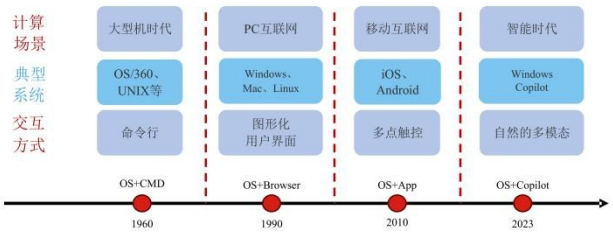

图1 操作系统智能化演进示意图

Figure 1 Schematic diagram of the intelligent evolution of operating systems

1 系统架构

1.1 系统构成

安全可信的人工智能原生操作系统以层次化、组件化理念构建，总体架构如图2所示。主要包括内核组件库、基础资源管理服务、共性基础服务和领域基础软件栈四大模块。引入智能开发工具链，包括基于AI的辅助编程和测试工具，实现智能代码生成与补全、智能代码注释、智能代码理解与重构、智能自动化测试、智能代码质量分析等功能。

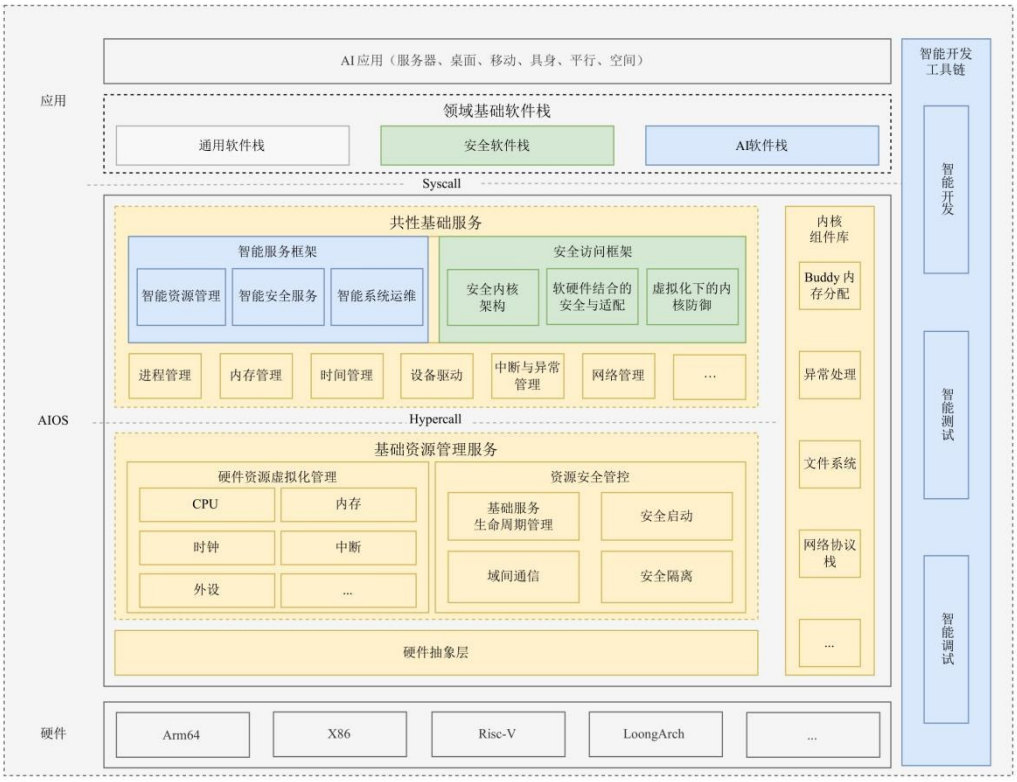

图2 AIOS系统架构

Figure 2 AIOS system architecture

内核组件库负责抽象和封装内存管理、进程调度、文件系统、网络协议等核心功能，为系统各层提供高可复用的底层能力；基础资源管理服务实现对CPU、内存、外设等的虚拟化与安全管控，支持服务生命周期管理、安全启动与隔离、域间通信等；共性基础服务涵盖进程、内存、时间、中断、设备、网络等传统内核管理，并集成安全可信访问框架和智能服务框架，满足多场景下的安全与性能需求；领域基础软件栈则面向不同应用场景，提供通用计算、安全、AI等多类软件支撑，覆盖libc运行时、密码与认证框架、可信计算、机器学习、深度学习框架及硬件加速库等，为上层应用提供丰富的能力基础。

1.2 内核组件库

无论是传统领域还是新兴领域，内核都是操作系统的核心部分，也是衔接硬件资源平台与上层应用的关键环节，内核架构的设计对于安全保障和场景适应能力具有决定性和根本性的影响。面向不同场景存在多种内核，从设计模式和架构来看，最典型

的包括宏内核、微内核和混合内核等3种,分类如图3所示。

表1比较了宏内核、微内核和混合内核3类典型内核架构的特点、优势与局限性。

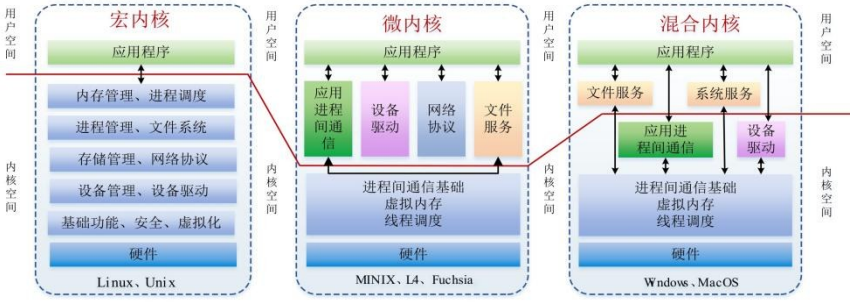

图3　内核架构分类

Figure 3　Classification of kernel architecture

表1　3类典型内核架构

Table 1　3 typical kernel architectures

内核类型	特点	优势	局限性	典型内核
宏内核	所有功能集成在一个内核中,驱动和功能模块均在内核态运行	性能高,功能全	健壮性问题突出,新特性融入困难,难以满足工业级安全认证[8-9]	Linux内核
微内核	功能精简,很多内核功能和系统服务在用户态运行	安全性高,易扩展	IPC通信导致性能损耗,适应场景有限[10]	MINIX[11]、L4[12-14]、鸿蒙内核[15]
混合内核	综合宏内核和微内核的优势	平衡性能和复杂度,适应多场景	实现复杂度较高	Windows NT[16]、Mac OS、XNU[17]

此外,业界提出外内核(Exokernel)和实时内核两种架构在特定场景中展现出独特价值。外内核通过资源保护与管理分离,提供细粒度硬件资源隔离和用户态灵活定制能力,但开发门槛高,生态兼容性差[18-22]。实时内核围绕时间约束和行为可预测性设计,适配动态负载与安全需求,为自动驾驶等复杂场景提供时序保障[23-27]。

1.3　资源管理服务

资源管理服务是人工智能原生操作系统(AI-native Operating System,AIOS)的核心模块之一,负责对计算资源(如CPU、内存、外设等)进行高效管理和安全管控。随着工业物联网、5G、自动驾驶等领域的快速发展,混合关键性系统(Mixed Criticality Systems,MCS)[28]成为核心架构需求。MCS需在单一硬件平台上集成安全关键型(如车辆控制)与非关键型任务(如信息娱乐),同时满足时空隔离、实时性和安全认证要求。虚拟化技术通过硬件资源抽象与分区,成为实现此需求的关键使能器。

混合关键性系统需在单一硬件平台上实现多种任务的并行运行。虚拟化技术通过硬件资源抽象与分区,需提供以下关键能力:(1)时空隔离,确保高关键任务不受低关键任务的资源抢占(时间隔离)和内存、设备非法访问(空间隔离)。(2)实时性保障,通过优化调度算法(如RT-Xen的分层调度框架)满足实时性需求。(3)安全认证支持,符合DO-178C等行业标准,提供故障隔离证据。为实现此关键能力,业界提出如表2所示的通用虚拟化增强和轻量级虚拟化两种技术方案。

通过通用虚拟化增强和轻量级虚拟化的结合,资源管理服务能够满足多样化场景需求,为AIOS的安全可信性和高效性提供坚实保障。

1.4　共性基础服务

1.4.1　智能服务框架

智能服务框架是AIOS中承上启下的关键组成部分,位于系统架构的核心层次,连接基础资源管理服务与领域基础软件栈。其主要作用是通过深度集成人工智能技术,为操作系统提供动态感知、智能决策和自适应优化能力,从而满足多样化应用场景的需求。在AIOS的系统架构中,智能服务框架不仅承担了资源调度与管理的核心功能,还通过智能化手段提升系统的安全性和可靠性。

如表3所示,智能服务框架的核心功能体现在3个方面:资源管理、安全服务和系统运维。通过引入人工智能技术,传统基于静态规则的资源调度机制逐步进化为具备动态预测能力的智能管理框架;安全领域则利用异常检测算法构建自适应的威胁防护体系;在系统运维层面,智能服务框架展现了故障预测与自修复能力。这些功能的实现,为系统的智能化和高效性提供了重要支撑。

表2 通用虚拟化增强和轻量级虚拟化

Table 2 Universal virtualization enhancement and lightweight virtualization

类型	特点	优势	局限性	典型技术
通用虚拟化增强	基于传统虚拟化技术(Hyper-V[29]、Xen[30]和KVM[31]),通过优化调度机制和中断处理提升实时性	适配性强,支持多种应用场景	性能开销较大,需依赖硬件支持	RT-Xen[32]、PREEMPT_RT[33]
轻量级虚拟化	采用微内核或容器化技术,减少虚拟化层资源占用	资源开销低,适合嵌入式和实时场景	隔离性较弱,适用场景有限	NFV[34]、OKL4[35]、NOVA[36]、CloudVisor[37]、RT-CASE[38]、HermitCore[39]、ClickOS[40]、PikeOS[41]、Xtratum[42]、Bao Hypervisor[43]

表3 智能服务框架

Table 3 Intelligent Service Framework

功能	特点	优势	局限性	典型方法
智能资源管理	利用AI技术动态优化资源调度,替代静态规则策略;覆盖单机、集群、云与物联网场景	提升资源利用率,适应多变负载;实现能效与成本优化	依赖历史数据质量;算法泛化能力与实时性仍需提升	SmartOS[44]、Chronus[45]、分层强化学习框架[46]、AI驱动云计算框架[47]
智能安全服务	利用AI模型进行异常检测、行为分析与多模态特征融合,覆盖恶意软件、APT、勒索软件等威胁	高检测精度,低误报率;实时响应与自动化威胁处置	对抗性攻击规避风险;量子计算等新兴技术应用尚不成熟	MSNdroid[48]、DL-Droid[49]、AE-APT[50]
智能系统运维	基于日志分析与工作负载取证等方式,实现故障诊断、预测及自修复能力	提升运维效率与系统可靠性;减少人为错误,支持实时监控	大语言模型等新技术在实际部署中的稳定性与适配性有待验证	LogEA[51]、Desh[52]

(1)智能资源管理

智能资源管理作为操作系统的核心机制,旨在通过高效调度和分配CPU、内存、I/O等资源,实现公平性、响应速度与系统吞吐量的平衡,对整体系统性能具有决定性作用[53-54]。传统调度器如完全公平调度器(Completely Fair Scheduler,CFS)依赖启发式算法,面对多变应用场景时适应性有限,难以充分应对底层硬件资源竞争[55]。为提升资源利用率,研究者提出结合历史执行信息和机器学习方法进行资源优化,如基于机器学习的资源感知型Linux内核负载均衡方法,以及SmartOS系统通过自动学习用户偏好动态调整资源分配策略[56,44]。

AI技术的引入不仅优化了单机操作系统的进程调度,还扩展到计算机集群、物联网和云资源管理等领域。相关工作如Chronus系统在生产环境中实现了性能与能效的双提升,分层强化学习框架则针对智慧城市物联网场景动态协调能源与设备整合,提升资源共享效率[45-46]。此外,系统性综述与多AI技术融合的智能化框架为云计算资源的全生命周期管理提供了理论与实践基础[47]。

(2)智能安全服务

智能安全服务是AI与操作系统深度融合的重要方向,主要聚焦于恶意软件检测、勒索软件防护、高级持续性威胁(Advanced Persistent Threat,APT)攻击监测等关键场景。针对安卓平台,研究者提出了基于多模态特征融合和AI模型的检测方法,如MSNdroid[48]和DL-Droid[49],分别通过分析应用程序编程接口(Application Programming Interface,API)调用、权限配置、系统特征等,实现了高检测精度和低漏报率。硬件层数据与分类模型的结合进一步提升了检测性能[57]。在勒索软件防护方面,AI模型可实时分析系统调用序列,有效识别潜在威胁[58]。对于APT,AI驱动的AE-APT框架实现了对进行中攻击的实时监测与响应[50]。

此外,AI技术还被用于漏洞检测与修复、网络入侵检测、钓鱼与垃圾邮件识别、威胁情报分析等领域,通过自动化测试、代码分析和实时威胁情报支持,提升了系统的整体安全性与防御能力[59-63]。

(3)智能系统运维

智能系统运维在保障操作系统可靠性和安全性

方面发挥着核心作用。随着IT基础设施规模和复杂度的提升,AI技术在运维中的应用日益突出。LogEA工具[51]利用自然语言处理和机器学习对Linux系统日志进行分析,能够及时发现安全漏洞和异常活动,提升了事后取证能力。Desh框架[52]则侧重于故障的实时诊断与预测,缩短了响应时间。Zhang等人[64]提出的基于硬件的实时工作负载取证方案,通过机器学习模型分析进程行为,实现了独立于操作系统的高效监控。

总体来看,人工智能的引入极大地扩展了IT运维的智能化能力,包括故障检测、预测和调度等环节,从而提升了运维效率、降低了运行成本,并增强了系统的整体可靠性[65-66]。同时,随着大语言模型技术的发展,其在智能运维领域的应用也日益受到关注,相关方法体系不断被提出和验证,为智能运维带来了新的机遇[67]。

1.4.2　安全访问框架

(1)安全内核架构设计

星绽(Asterinas)操作系统提出一种框内核(Framekernel)设计范式[68],旨在突破传统宏内核与微内核在性能与安全之间的权衡,通过功能分解与隔离实现两者的有机统一。该架构将内核职责划分为最小化的特权框架(privileged framework)和功能丰富的去特权服务集(deprivileged OS services)[69]。特权框架(OS Trusted Domain,OSTD)作为系统可信计算基(Trusted Computing Base,TCB),仅包含与安全密切相关的核心功能,如内存管理、CPU状态控制、底层IPC通信和中断处理,并采用形式化验证方法确保关键模块的安全性。相较之下,文件系统、网络协议栈和设备驱动等复杂功能被实现为用户态的去特权服务,彼此隔离,降低了攻击面并提升了故障隔离与交互安全。这一设计有效兼顾了系统的安全性与性能,为高安全需求场景提供了坚实基础。

(2)软硬件结合的安全与适配

现代操作系统通过底层硬件的内存安全原语,实现了从基础隔离到针对性攻击识别的纵深防御体系。主流处理器架构(如x86-64、Arm64)普遍配备内存管理单元(Memory Management Unit,MMU)和页表机制,为进程间内存隔离、数据执行保护(Data Execution Prevention,DEP)和地址空间布局随机化(Address Space Layout Randomization,ASLR)等安全策略提供基础保障,有效防止进程间的内存干扰和部分攻击[70]。然而,单靠MMU的进程级隔离难以防御如缓冲区溢出、释放后使用(use-after-free)、ROP/JOP等高级攻击。为此,现代处理器进一步引入了如内存标记扩展(Memory Tagging Extension,MTE)、指针认证(Pointer Authentication Code,PAC)、控制流强制技术(Control-flow Enforcement Technology,CET)等硬件安全扩展,在极低性能开销下实现对内存错误和恶意篡改的实时检测与阻断,显著提升了系统的整体安全性,表4对比了不同架构内存安全技术。

表4　不同架构高级内存安全技术

Table 4　Advanced memory security technologies for different architectures

架构平台	关键技术	技术特点	防护目标	典型方案/应用
Armv8-A	内存标记扩展(MTE)[71]	通过内存与指针的标签匹配进行硬件访问检查	精细颗粒度的内存安全,保证控制流完整性	Google Android[72]
	指针认证(PAC)	通过密码学签名保护指针不被篡改	精细颗粒度的内存安全,防止缓冲区溢出、释放后使用等问题	Google Android[73]
Intel x86	控制流强制技术(CET)	包含影子堆栈(Shadow Stack)和间接分支跟踪(Indirect Branch Tracking),硬件校验返回地址[74]	控制流劫持攻击(如ROP、JOP)	微软Windows的硬件强制堆栈保护[75]
RISC-V	FIXER、RetTag、CHERI等	包含影子堆栈、指针签名以及软硬件协同的内存安全设计	全面的控制流完整性与内存安全保护	FIXER[76]、RetTag[77]、CHERI[78-79]

针对不同的底层硬件架构异构性(x86、Arm、RISC-V等),操作系统应当屏蔽硬件底层架构的异质性,保障向上层提供服务的一致性。业界普遍使用的一种较为成熟且普适的解决方案是硬件抽象层(Hardware Abstract Layer,HAL)[80]。在跨架构安全适配这一复杂议题中,HAL不仅是实现平台可移植性的关键,更在维护系统安全和提升运行效率方面扮演着至关重要的角色。表5比较了部分操作系统在HAL部分的设计与实现。

(3)虚拟化下的内核纵深防御

现代操作系统逐渐采用硬件虚拟化技术作为一种基础安全原语,通过在操作系统内核之下引入一个更具权限的Hypervisor,来实现超越传统内核/用户态模型的硬件强制隔离。这种架构利用了主流处理器(如Intel VT-x和Arm VHE)提供的硬件支持,使得构建可跨异构硬件移植的统一安全机制成为可能,从而

极大地提升了内核自身及上层服务的安全性,表6总结比较了虚拟化技术在不同操作系统中的应用。

表5 硬件抽象层机制实现

Table 5 Implementation of Hardware Abstraction Layer Mechanism

操作系统/平台	核心抽象层/机制	关键安全实现	实现效果/优势
Android	硬件抽象层(HAL)	通过KeyMint HAL为不同硬件的可信执行环境(TEE)提供统一的密钥管理和密码学操作接口[81]	实现了安全能力的跨架构适配,能有效抵御密钥提取、回滚、旁路攻击等多种威胁
	供应商接口(VINTF)	将硬件厂商的底层实现与核心操作系统框架分离,维护OS内核完整性[82]	实现跨架构、硬件异构的适配与安全性的融合
Windows	硬件抽象层(HAL)	为安全启动(Secure Boot)等依赖固件和硬件特性的功能提供统一调用路径,屏蔽底层UEFI固件差异[83]	保证不同硬件平台下系统启动过程一致性和完整性
星绽(Asterinas)	HyperEnclave抽象层	提供HyperEnclave层管理不同的硬件TEE(如SGX、SEV)	实现跨硬件、架构的可信执行环境统一管理
	组件跨态复用机制	将共性功能组件化,在特权态和用户态间复用[69,84]	减少代码冗余与漏洞风险

表6 虚拟化下的内核纵深防御技术

Table 6 Kernel defense in depth technology under virtualization

操作系统/平台	核心虚拟化技术	关键安全特性与应用	实现目标与优势
Windows	Hyper-V/虚拟化安全(VBS)	运行Hypervisor强制代码完整性(HVCI)和Credential Guard,实现内核隔离[85-86]	利用Hypervisor将内核及核心服务与系统其他部分隔离,在x86-64和Arm64平台提供统一的硬件强制安全保护
Android	Android虚拟化框架(AVF)/pKVM	为敏感代码和数据提供受保护的虚拟机(pVMs)[87]	在多样化的Arm硬件生态中,为第三方代码提供一致且严格的安全隔离保证
macOS(Apple Silicon)	原生虚拟化框架	安全运行Linux虚拟机、隔离iOS应用沙箱[88]	将虚拟化作为通用的安全隔离工具,实现跨平台应用的兼容与安全运行
Linux(KVM)	内核虚拟机(KVM)	将Linux内核转化为一个Type-1 Hypervisor,为上层提供创建和管理虚拟机的核心能力[89]	提供一个集成于内核、高性能且开源的通用虚拟化基础架构,是云基础设施和多种隔离方案的基石
Kata Containers	轻量级虚拟机(Lightweight VMs)	"双重安全层":每个容器都运行在各自独立的轻量级虚拟机中,将容器与宿主机内核完全隔离,防止因容器逃逸而导致宿主机被攻击[89]	融合虚拟机的硬件级安全隔离与容器的速度及性能,可无缝集成到Kubernetes和Docker等主流平台
Qubes OS	Xen Hypervisor	将不同的用户活动域(工作、个人等)划分到独立的轻量级虚拟机中[90]	围绕Hypervisor构建整个操作系统,实现深度的、以隔离为核心的系统安全架构

1.5 领域基础软件栈

1.5.1 AI软件栈

利用自然语言与计算设备进行交互一直是人机交互(Human Computer Interface,HCI)领域的重要研究方向[91]。与传统输入/输出方式相比,语言交互在操作受限或效率低下时为用户提供了更高效的访问渠道。尽管操作指南和沉浸式学习平台等方法有助于用户熟悉界面和功能,但在降低认知负担方面仍有限[92]。随着大规模语言模型(Large Language Models,LLMs)和对话式智能体的发展,用户可以通过自然语言直接操控操作系统和应用,极大提升了交互效率并降低了学习门槛[93]。目前,主流的智能交互方式主要包括基于API和基于图形用户界面(Graphic User Interface,GUI)两类,如表7所示。

(1)基于API的智能体应用

基于API的智能体应用充分发挥了LLMs在推

表7　不同交互模式的AI软件栈

Table 7　AI software stack with different interaction modes

交互模式	核心特点	优势	局限性	典型方法
基于API	通过预定义的API接口与外部工具、函数或服务进行交互;依赖LLMs的自然语言理解与逻辑推理能力	实现自动化与高效化操作;具有明确的接口规范	功能受限于预定义的API范围;扩展功能需要开发额外端点	AIOS[94]、Microsoft Copilot[95]、MCP[96]
基于GUI	利用多模态大语言模型"观察"并操作图形用户界面;通过模拟人类操作实现交互	无需系统后端访问;具备通用性,可跨平台操作;操作过程透明可视,增强用户信任	执行效率低于直接API调用;依赖界面稳定性;多模态理解精度有待提升	MM-Navigator[97]、AppAgent[98]、MobileAgent[99]、UFO[100]、CogAgent[101]、autoMate[102]

理、规划和协作任务中的能力,极大减轻了用户在操作系统和应用交互中的认知负担[103]。这类应用通过定义明确的API,使LLMs能够与外部工具、服务和函数进行程序化交互,实现自动化和高效化操作[104]。典型代表如AIOS[94]和微软Copilot[95],已将基于API与LLMs的交互模式推向主流,并推动相关研究成果快速转化为工业级解决方案。

与传统依赖脚本化流程的自动化系统不同,基于API的智能体能够利用LLMs的自然语言理解、逻辑推理和工具交互能力,根据复杂自然语言输入动态解析任务,在传统自动化难以覆盖的场景中展现出更强的智能化行为[93]。此外,Anthropic提出的模型上下文协议(Model Context Protocol,MCP)[96]为AI与外部工具的标准化交互提供了统一接口,推动了智能体对工具的动态发现、选择与编排,标志着API驱动智能体应用的进一步成熟。

随着集体智能的兴起,AI智能体正从单一体向多智能体协作体系演进。传统模式下,智能体多为独立运行,难以持续协作或跨专业领域协同处理复杂任务[105]。为此,谷歌于2024年推出的AI智能体通信协议[106],为多智能体协作提供了标准化通信机制,使多个专业智能体能够协同应对单一智能体难以独立完成的复杂任务。

(2)基于GUI的智能体应用

基于GUI的智能体应用通过多模态大语言模型(Multimodal Large Language Models,MLLMs),如GPT-4o[107]和Gemini[108],实现了对文本、图像等多模态输入的理解和处理,催生了智能体"观察"并操作图形用户界面的新型交互模式。这类智能体能够模拟人类用户,通过鼠标、键盘或触摸屏等方式直接操作桌面、移动或网页应用,无需系统后端访问,从而提升了安全性、隐私性和通用性[98]。每一步操作均可被用户实时观察,增强了信任感,被视为Agents for Computer Use(ACU)[109]和操作系统级智能体(OS Agents)[110]的典型代表。

当前,基于GUI的计算机使用智能体(Computer Use Agent,CUA)已在商业和学术领域涌现,显著提升了智能体自动理解并执行自然语言指令、与各类应用程序交互的能力。在移动平台上,MM-Navigator[97]、AppAgent[98]和MobileAgent[99]等方法借助多模态大模型(如GPT-4V[111]),通过模拟人类操作实现对智能手机应用的控制,绕开了对系统后端权限或特定API的依赖。更广泛地,UFO[100]、CogAgent[101]和autoMate[102]等项目表明,基于GUI的智能体不仅提升了用户体验和软件可访问性,还为跨平台复杂工作流提供了通用自动化控制能力,成为操作系统级智能体发展的重要方向。

1.5.2　安全软件栈

可信执行环境(Trusted Execution Environment,TEE)是一种硬件级安全技术,旨在为敏感数据和代码提供与主操作系统(Operating System,OS)隔离的独立安全区域,防止因操作系统或虚拟机监控器(hypervisor)被攻击而导致的数据泄露。TEE通过CPU硬件扩展(如Intel SGX、ARM TrustZone、AMD SEV)创建"安全飞地"(Enclave),实现与操作系统、应用及Hypervisor的隔离[112]。TEE整体架构如图4所示。

中国自主研发的TCM芯片对标国际TPM标准,TPCM为其演进版本,TSB则作为内核层可信监控软件,连接硬件安全芯片与上层应用,实现动态可信验证[113]。

机密计算(confidential computing)则专注于保护数据在使用过程中的安全,确保数据在内存中始终加密或硬件隔离,即使操作系统、虚拟机监控器或云服务商也无法访问原始数据,弥补了传统安全模型在"运行中数据"防护上的不足,与静态和传输中数据加密共同构成完整安全链条[114]。

TEE与机密计算的安全性不仅依赖于安全硬件,还需操作系统和TEE多种机制协同保障。操作系统通过可信启动和动态度量建立全局信任链,为TEE运行提供基础;TEE则通过启动证明、远程证明等机制,

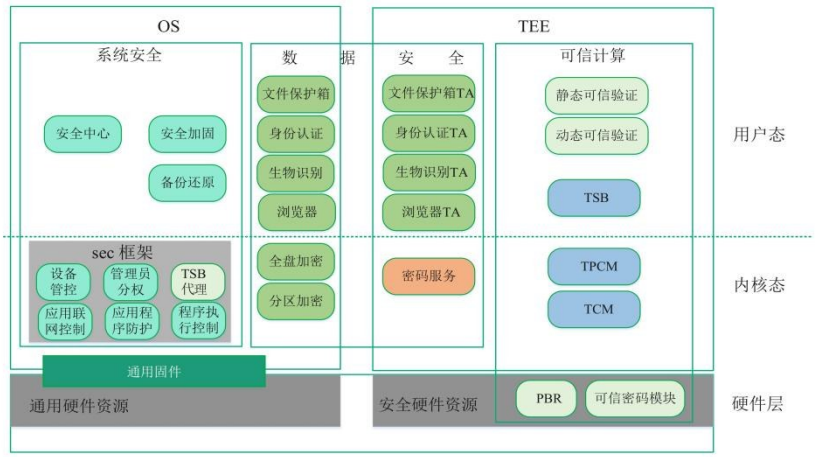

图4 TEE整体架构

Figure 4 TEE overall architecture

确保敏感数据安全。两者协同,构建系统全局的安全防护体系。

(1)可信启动与动态度量

可信启动和动态度量共同构成操作系统安全的基础,分别针对系统启动和运行时的不同威胁场景。可信启动通过硬件级逐级验证固件和引导程序,建立从可信平台模块(Trusted Platform Module,TPM)到操作系统的信任链,防御Bootkit[115](如LoJax)和恶意固件等启动阶段攻击,有效应对系统初始化时的安全威胁[116]。然而,可信启动完成后,系统在运行时仍面临内核漏洞利用(如Dirty Pipe)、恶意驱动加载、Rootkit隐藏进程[117]及无文件内存攻击等威胁,静态度量难以检测运行时内存变异。动态度量机制则通过实时监控内核代码完整性、行为和内存状态,及时发现并应对运行时的安全威胁[118]。可信启动本质上是对传统静态度量的升级,通过信任链传递实现系统全程可信,是保障可信执行环境的重要技术[119]。两者协同配合,为操作系统构建纵深防御体系,显著提升系统整体安全性。

(2)机密计算与机密通信

机密计算是一种基于可信硬件,通过隔离、加密和认证机制保护数据在使用中安全的范式,旨在解决传统方案中数据处理需解密为明文、易被操作系统、虚拟机监控器或云平台管理员窃取的风险[120]。其核心在于利用TEE构建“硬件强制的安全飞地”,确保数据在存储、传输和计算全程均处于加密或受保护状态。

在TEE中,机密计算通过飞地实现与操作系统、虚拟机监控器等特权软件的完全隔离,即使这些软件被攻破也无法访问飞地内的数据。飞地内的代码和数据由CPU硬件加密,仅本地CPU可解密,非法访问会触发硬件中断,保障数据安全。机密计算还依赖远程证明和机密通信机制:飞地启动后生成硬件签名的证明报告,远程服务方验证后才会传输加密数据或密钥[121]。TEE中的机密通信则通过硬件加密和认证机制,在飞地之间或与外部实体间建立受保护的数据通道,确保敏感数据在传输过程中防窃听、防篡[122]。数据仅在飞地内部解密处理,输出前重新加密。

相比传统方案在运行时数据暴露于明文、易受攻击,机密计算与机密通信通过硬件隔离和全程加密,实现了对使用中数据的有效保护,补全了数据安全的最后一环[123]。

(3)安全存储与密钥保护

安全存储指在可信计算系统中,通过特定技术手段安全地存储和管理敏感数据(如密钥、证书、凭证等)。传统加密存储通常在软件层面对静态数据加密,但密钥多驻留于内存或硬盘,易受冷启动、内存转储、恶意引导等攻击,存在泄露风险[124]。相比之下,现代安全存储依赖硬件信任根(如TPM、TEE),密钥在受保护环境中生成和使用,仅在系统状态可信时才释放,且与平台状态绑定。TPM内部生成的密钥只能在TPM内部使用,TEE则实现与操作系统隔离,即使系统被攻破,攻击者也难以窃取密钥。安全存储还结合可信启动和系统状态度量,只有在系统未被篡改时,硬件保护的密钥才能解密数据[125]。因此,安全存储具备极强的防篡改和抗攻击能力,实现了“密钥不出安全区”,有效保障了数据机密性和完整性。

2 关键技术

2.1 端侧智能的操作系统

随着边缘计算的兴起,物联网设备正从数据采集节点转变为具备智能决策和执行能力的终端,对操作系统提出了低延迟和数据隐私的新要求。与此同

时，LLMs的发展为自然语言理解和交互带来了变革，但其庞大的参数量与边缘设备有限的内存、算力和能耗约束形成了突出矛盾。预计2025至2034年，边缘AI市场将以24.8%的年复合增长率快速扩张，推动高效端侧LLMs解决方案的需求持续增长[126]。然而，典型边缘设备的内存仅为4~16 GB[127-128]，难以直接承载如LLaMA-7B[129]等模型所需的十几GB内存，更高参数量模型甚至需要40 GB以上显存[130]。此外，LLMs推理过程对计算能力和能耗的要求极高，例如LLaMA-2-70B模型在高端GPU上生成一个token需百毫秒，生成长序列则需数秒至十秒，且能耗巨大[131-133]。因此，模型规模、计算需求和能耗成为LLMs在边缘端部署的主要瓶颈。为此，业界提出了模型压缩、推理优化、端云协同等多种关键技术与方案(见表8)，以缓解资源受限带来的挑战，推动LLMs在边缘设备上的高效部署与应用[134-135]。

表8　端侧设备LLMs部署关键技术与方案

Table 8　Key technologies and solutions for deploying LLMs on end devices

技术维度	核心方法	优势	局限性	典型技术与方案
模型压缩	参数共享、模型剪枝、知识蒸馏	显著降低模型规模;提升计算效率;保持较小参数下的良好性能	剪枝带来信息损失;大模型全参数微调困难;"涌现能力"可能受损	嵌入权重绑定[136]、Transformer权重共享[137]、模型剪枝[138-144]、知识蒸馏[145-151]
推理优化	投机解码、KV缓存优化、早退机制、内核算子优化、内存卸载	提升解码速度;减少内存占用;优化硬件资源利用率	不同机制实现方式可能存在冲突;内存管理策略需系统级协同	BiLD[152]、LLMCad[153]、FlashAttention[154]、PagedAttention[155]
模型部署	端侧推理框架、端云协同	适应异构硬件;平衡性能与成本;实现低延迟响应;支持在线微调	端侧推理资源受限;端云协同动态负载均衡复杂	Llama.cpp[156]、PowerInfer[157]、Transformer-Lite[158]、Hybrid LLM[159]、Tabi[160]、VLLM[161]、Edge-LLM[162]

2.1.1　模型压缩

模型压缩是缓解LLMs在端侧部署资源瓶颈的关键技术。近年来，业界提出多种小型语言模型(Small Language Models，SLMs)，通过参数共享(如嵌入权重绑定[136]、Transformer权重共享[137])等机制，有效降低模型规模并提升内存与计算效率。另一类方法则对已有LLMs进行压缩，主要包括模型剪枝和知识蒸馏。

模型剪枝旨在去除网络中的冗余结构或权重，分为结构化剪枝和非结构化剪枝。结构化剪枝[138-141]移除具有物理意义的组件，如通道或权重块；非结构化剪枝[142-144]则以神经元为单位进行操作。尽管剪枝后可通过微调恢复部分性能，但信息损失和全参数微调的难度使其在大模型应用中存在局限。

知识蒸馏通过教师模型输出监督信息训练小型学生模型，分为白盒蒸馏(利用内部表示与输出[145-147])和黑盒蒸馏(仅依赖最终输出[148-151])。LLMs的知识蒸馏更注重知识迁移而非单纯压缩[163]。值得注意的是，LLMs参数规模超过阈值后常表现出“涌现能力”，在复杂任务中展现卓越性能[164]。

2.1.2　推理优化

推理优化旨在提升LLMs在端侧部署时的性能与资源利用率，核心在于与AIOS的内核机制、资源管理和硬件适配体系深度协同。常见优化途径包括投机解码、KV缓存优化、早退机制、算子优化和内存卸载等。

投机解码作为非自回归策略，将解码分为候选生成和结果验证两个阶段，由小模型生成候选标记，大模型验证纠错，实现高效解码[165-166]。如BiLD[152]和LLMCad[153]等进一步提升了生成速度和准确性。KV缓存通过存储历史注意力层的键值对减少冗余计算，但随序列增长带来内存瓶颈。为此，研究提出KV缓存稀疏化[167-170]和量化[167,171]，以优化内存占用。早退机制[172-173]允许模型在解码阶段提前结束计算，降低资源消耗，但与KV缓存优化存在冲突，需系统级协同设计[174]。

算子与内核优化则借鉴操作系统的资源管理理念，提升AI硬件加速器(如GPU)的计算与内存调度效率。例如，FlashAttention[154,175-176]高效管理GPU缓存与显存，PagedAttention[155]引入分页机制，Flash-LLM[177]优化稀疏矩阵乘法带宽瓶颈。内存卸载机制则通过按需加载权重突破存储限制，但效率依赖于操作系统的I/O调度和缓存管理，相关研究[178]对此进行了优化。

总体来看，推理优化技术的创新不仅体现在算法本身，更在于将操作系统成熟的资源管理方法(如分页、缓存、调度)应用于AI计算，推动AI系统与底层计算系统的深度协同设计。

2.1.3　模型部署

模型部署策略在设备端LLMs应用中呈现出从云端到边缘端的多样化路径，其核心在于模型算法优化与操作系统资源管理的深度协同。边缘设备资源受限，高效推理框架需依赖操作系统的进程调度、内存

管理和硬件抽象，实现对CPU、GPU等异构计算单元的高效调度与大规模模型数据、动态缓存（如KV缓存）的管理。

本地部署方案如Llama.cpp[156]，通过高效硬件抽象层和进程调度，支持CPU与GPU混合推理及整数量化，提升多平台推理效率。PowerInfer[157]采用离线分析与在线推理结合，利用稀疏矩阵乘法算子优化动态场景，PowerInfer-2[179]进一步细化推理流程，充分挖掘异构资源潜力。Transformer-Lite[158]则通过动态形状推理、低精度算子和KV缓存优化，提升移动设备推理性能。

端云协同部署成为应对复杂任务的重要路径。Hybrid LLM[159]提出混合推理方案，实现端-云环境下的自适应任务调度与负载均衡。Tabi[160]采用分层推理机制，结合小模型与大模型按需处理不同查询。VLLM[161]针对边缘-云协同优化推理过程中的键值存储，创新性地引入PagedAttention机制[155]，借鉴操作系统虚拟内存分页思想。Edge-LLM[162]则通过服务器-节点协作，将主模型部署在服务器端，边缘节点加载适配器，实现实时推理与在线微调，兼顾性能与灵活性。

2.2 AIOS的记忆管理系统

AI记忆管理系统是推动LLMs从无状态交互向具备长期学习与适应能力的智能体演进的关键。LLMs记忆可按多维度分类：按记忆特性分为隐式记忆与显式记忆，按实现方式分为参数化记忆、短暂激活记忆和明文记忆，按时间维度分为感觉记忆、短期记忆和长期记忆。其中，短期记忆通过键值缓存实现对近期token的即时访问，但会话结束后即失效；长期记忆则依赖物理存储缓存历史信息，支持历史记录检索，突破上下文窗口限制，是LLMs智能化的主要挑战[180]。

为实现类人化的记忆管理，LLMs记忆系统通常包括记忆存储、管理与检索等核心环节（见表9）。这种设计不仅提升了模型对长期上下文的理解和利用能力，也反映了向人类认知系统架构靠拢的努力。

表9 LLMs记忆系统构建方法关键技术

Table 9 Key technologies for constructing LLMs memory systems

技术维度	方法类别	核心机制	优势	典型方法
记忆存储	参数化记忆	通过预训练/微调将知识编码至模型参数;支持隐式长期记忆	知识深度内化;推理时无需外部检索	大规模预训练[181-182]、参数调优[183-184]、SLayer[185]、LoRA[186]、内存编辑[187-190]
	外部显式记忆	使用外部存储保存知识;支持结构化组织与动态更新	存储容量大;支持精确编辑与动态管理	树状结构[191]、图结构[192-193]、Zep[194]、A-MEM[195]
	类人化记忆机制	模仿人类记忆层级与笔记习惯;结合多模态与知识图谱	组织性强;支持情境感知与持久化	基础记忆[196]、PGRAG[197]、Memory3[198]
记忆检索	文本检索	压缩冗余信息,提取关键内容	可解释性强;支持语义权重评估	三重评分机制[199]、AdaPlanner[200]、Voyager[201]、Structure-R1[202]
	向量化检索	将记忆编码为潜在向量,基于相似度进行高效检索	检索速度快;支持语义检索	LongMem[203]、情感RAG[204]、Memoryllm[205]
记忆管理	自我优化	支持反思、摘要、遗忘、截断等核心操作;实现记忆动态更新	模拟人类记忆规律;保持信息动态平衡	Reflexion[206]、MemoryBank[207]、AI-town[199]
	系统化治理	借鉴OS设计理念,提供分层存储、API抽象、权限管理等系统服务	支持大规模记忆的系统级管理	MemoryOS[208]、MemOS[209]、AutoGen[210]

2.2.1 记忆存储方式

LLMs的记忆存储方式可按不同维度划分：在表征层面，分为基于token的存储与基于潜在空间（latent space）的存储[198,211]；在组织形式上，包括文本存储、知识结构化存储，以及按任务、时间或语义层次化组织等方式[212-214]。

隐式长期记忆主要源于模型训练过程,如大规模预训练[181-182]和参数调优[183-184],通过参数更新内化知识。部分方法在训练中显式引入记忆机制,如SLayer[185]对相关层局部微调,LoRA[186]通过低秩适配器实现轻量化参数更新,“内存编辑”则支持定向知识注入与可控更新[187-190]。

显式长期记忆则依赖外部存储和高效检索,研究逐步转向树状[191]、图结构[192-193]等层次化组织,并引入时间建模(如Zep[194])和动态语义更新(如A-MEM[195])。此外,类人化记忆机制受到人类认知启发,如Memory3[198]将KV-cache显式化,PGRAG[197]自动生成思维导图,A-MEM[195]构建跨会话笔记网络,“基础记忆”方法[196]结合视觉-语言模型与知识图谱,提升上下文推理与感知能力。

2.2.2　记忆检索方式

LLMs的记忆检索方式主要分为文本检索和向量化检索两类。对于结构化文本记忆,常用方法包括基于“三重评分机制”的检索框架,如Generative Agents[199]提出通过时效性、重要性和相关性对记忆片段进行筛选,优先回忆近期、重要且相关的信息。AdaPlanner[200]和Voyager[201]等则将记忆抽象为技能库,支持按需调用和组合,实现经验迁移与任务规划。Structure-R1[202]采用强化学习方法构建结构化记忆。

对于向量化记忆,信息被编码为潜在向量,通过相似度检索高效回溯历史上下文。LongMem[203]将长文本分片并缓存中间层键值对,推理时检索最相关历史片段并融合当前状态。情感RAG[204]在语义相似度基础上引入情感因素,动态调整记忆调用。Memoryllm[205]则在Transformer各层引入可训练记忆令牌,通过自更新机制实现记忆池的动态平衡。上述方法提升了LLMs对长期上下文的检索效率与适应性。

2.2.3　记忆管理方式

随着对LLM记忆机制理解的深入,研究逐步从隐式表示转向具备工具化接口的显式管理。高效的记忆管理不仅要求支持动态增删改,还需实现编码、反思、摘要、遗忘、截断和优先级判断等核心操作,以灵活调控模型的语义行为[215]。

具体实践中,Reflexion[206]通过存储自我反思反馈,实现历史错误的实时纠正;MemoryBank[207]引入艾宾浩斯遗忘曲线,动态调整记忆强度,模拟人类长期记忆规律;AI-town[199]则以自然语言保存记忆,并通过反思循环筛选关键信息。

在系统化设计方面,MemoryOS[208]提出三级分层存储架构,整合存储、更新、检索与生成模块,支持用户偏好的动态演进。MemOS[209]将LLM记忆视为一等资源,借鉴操作系统理念引入调度、分层、API抽象和权限管理,实现系统级治理。AutoGen[210]则通过多智能体协作与对话共享记忆,提升复杂任务下的协作效率。

2.3　构建安全可信的AIOS

2.3.1　内存安全与形式化验证

内存安全长期是系统安全的核心挑战,相关漏洞占泛在设备安全漏洞的70%以上[216]。传统“补丁修复”难以应对动态威胁。Rust语言通过所有权、借用和生命周期机制,在编译阶段实现严格的静态分析,消除空指针、悬垂指针、缓冲区溢出和数据竞争等常见内存安全隐患,且不牺牲运行时性能[217]。因此,Rust已成为高安全、高性能系统级软件的首选,广泛应用于操作系统内核和虚拟化平台的重构,如表10所示。

然而,Rust并非“银弹”。操作系统开发中仍需使用unsafe代码进行底层操作,这部分代码绕过了编译器的安全保障,正确性需依赖开发者和额外工具验证。例如,Rust for Linux[224]需与C代码交互,星绽操作系统用KernMiri验证unsafe代码仍发现未定义行为[69],Tock[225-226]和Redox[227]也面临类似挑战。此外,逻辑错误、并发缺陷和硬件问题超出Rust语言本身的保障范围。

为实现更高可靠性,Rust常与形式化验证协同应用。Rust的中间表示(Mid-level IR,MIR)[228]保留了关键语义,成为形式化验证的理想入口。主流技术路径包括基于MIR的类型与借用信息提取、自动化模型检查和SMT求解等,进一步提升系统的内存安全和逻辑正确性。表11梳理了基于MIR等中间表示的几种主流形式化验证技术路径。

2.3.2　自动化检测与形式化验证

为构建高可信的底层系统,形式化验证与自动化检测技术提供了超越传统测试的严格保障。这些技术路线各异,从需要深度人工交互的交互式定理证明,到高度自动化的模型检查与推导,共同构成了保障系统安全的工具箱。表12分别从“已验证的系统”与“验证工具”两个维度,梳理对比了该领域的典型实践与技术方法。

2.3.3　启动证明与远程证明

启动证明和远程证明是保障TEE环境可信的核心机制,通过密码学手段验证TEE的完整性与合法性。启动证明用于确认飞地初始化时的安全性和完整性,如Intel SGX在创建飞地时自动计算哈希并由引用飞地(Quoting Enclave,QE)签名,确保代码未被篡改且运行于真实硬件上[241]。

需要注意,可信启动与启动证明虽共同服务于信任链构建,但分属不同层级:可信启动保障系统启动

表 10 Rust在系统编程领域的应用实践

Table 10 The Application Practice of Rust in the Field of System Programming

平台/项目	Rust应用实践	核心安全目标
Windows内核	启动项目以Rust重写部分内核组件,逐步替换C/C++代码[218]	减少内核的攻击面,提升内存安全[219]
AWS Firecracker,Nitro System,Bottlerocket	VMM、Hypervisor以及操作系统等核心组件均大量采用Rust编写[220-222]	利用编译时安全检查,为多租户云环境提供极致的硬件强制隔离,确保虚拟机边界的安全性
Android虚拟化框架	采用Rust开发的crosvm作为虚拟化框架的核心组件[223]	保障受保护虚拟机(pVMs)的安全运行,防止内存损坏风险
星绽操作系统	超过95%的系统代码由Rust编写,将不安全代码封装在最小化的安全API中[69]	遵循"最小权限原则",严格控制不安全代码的使用,避免将风险暴露给上层服务

表 11 基于Rust中间表示的形式化验证技术路径

Table 11 Formal verification technology path based on Rust intermediate representation

项目	核心技术路线	验证目标
Astrauskas等[229]	提取Rust MIR中的类型与借用信息,转化为形式化验证所需的Place Capability Sets与分离逻辑断言,自动翻译至Viper中间验证语言进行验证。	程序的内存安全
AWS实验室[230]	将rustc生成的MIR/LLVM IR翻译为CBMC支持的Goto-C表示,利用SAT/SMT求解器进行有界模型检查,穷尽搜索给定范围内的所有状态。	在设定边界内数学上严格的内存安全与程序属性
Lattuada等[231-232]	(顺序程序)将资源所有权建模为线性幽灵权限,编码为SMT约束交由Z3求解器推理。 (并发程序)将共享状态抽象为状态转移系统,并通过归纳不变式验证线程间协议。 (分布式协议)将关键数据结构映射为EPR逻辑片段,实现全自动证明。	覆盖从内存安全到系统级正确性的完整验证链条

表 12 高可信系统形式化验证实践与方法对比

Table 12 Comparison of Formal Verification Practices and Methods for Highly Trusted Systems

验证对象/工具	类型/范畴	核心机制与特点	验证目标
seL4[233]	交互式定理证明 (Isabelle/HOL[234])	采用多层次细化框架,实现从抽象规范到二进制代码的全程验证,形成"开发即验证"模式。	微内核的功能正确性、完整性与隔离性。
CertiKOS[219,235]	交互式定理证明 (Coq[236])	提出基于环境上下文的组合式验证架构,将并发内核验证简化为顺序任务。	支持细粒度锁的通用并发OS内核(如mC2)。
Hyperkernel[237]	自动化验证(SMT求解)	通过有限接口将内核行为抽象为有限状态机,并借助Z3[238]等SMT求解器实现自动化验证。	内核设计的正确性。
SeKVM[239-240]	交互式定理证明(Coq)	通过架构重构实现功能与安全分离,并对hypervisor核心层(KCore)进行形式化验证。	商品化hypervisor(KVM)在真实硬件下的内存安全。
Prusti[229]	半自动化验证(Rust)	通过代码注解集成Rust所有权系统,依赖SMT求解器自动验证内存安全与复杂逻辑,需人工编写规约。	适用于验证safe Rust代码的规约符合性。
Kani[230]	自动化模型检查(Rust)	基于CBMC引擎,无需手动规约即可自动验证内存安全与断言,特别适合unsafe代码和并发逻辑,但为有界验证。	unsafe Rust代码的内存安全性与用户自定义断言。
Verus[231-232]	半自动化验证(Rust)	基于Rust语法扩展,需人工编写规约但自动化推理,支持复杂数学证明与并发协议,能验证unsafe代码。	系统关键代码的高阶逻辑属性(如星绽页表系统的映射完整性)。

过程的完整性,启动证明则专注于TEE内部飞地的纯净性。可信启动完成后,用户创建飞地时才触发启动证明,前者为后者提供基础保障。

远程证明则允许远程方(如云服务商)验证飞地内应用代码和数据的可信状态,确保敏感计算在未被篡改的环境中执行,广泛应用于云端机密计算和区块链等场景[242]。

3 典型应用

在新型计算环境下,基础软件栈的开发和部署需

要具备更强的应用共性抽象能力。领域基础软件栈是基于内核的统一运行支撑环境,支撑特定领域应用的层级化软件集合,面向具身智能(Embodied Intelligence,EI)、工业物联网(Industrial Internet of Things,IIoT)、移动应用等人机物融合应用场景,为所有上层应用和服务提供了运行环境、资源管理、数据存储、通信能力和必要的支撑工具。

3.1　具身智能领域

具身智能强调智能体通过与环境的物理交互来发展和适应[243],是操作系统在物理世界的核心载体。机器人软件的历史,可以追溯到50多年前的Shakey机器人[244]。此后研究了关于如何构建经典规划器、并发行为和三层架构的关键技术[245-247],并逐步提出了许多软件平台和中间件,使构建机器人系统变得更加容易。YARP[248]开源机器人平台简化了机器人软件开发,提高了研究和开发的效率,它强调灵活性和长期稳定性,且同时支持多种操作系统(如Windows、Linux、QNX)。由斯坦福大学提出的机器人操作系统(Robot Operating System,ROS)[249]被广泛用于构建和开发机器人软件,在智能机器领域产生了深远影响。而ROS2[250]在ROS的基础上进行了重新设计,具备了安全性、嵌入式和实时支持、多机器人通信能力,以及在非理想网络环境中的运行能力。

3.2　工业应用领域

IIoT旨在通过连接设备、传感器和网络,实现工业过程的自动化、优化和智能化[251],其组成架构可以分为传感器、执行器与数据中心等重要部分[252]。一个典型的IIoT由如图5所示的通用技术栈模型[253-255]构成。物联网的设备硬件层与设备软件层可以进一步被划分为工业物联网架构里的物理实体与对应的软件支撑[256],可以细分为工业场景里的实体对象,负责采集数据的传感器以及负责传感器功能操作控制及本地数据处理的边缘层。边缘层和通信层通常集成于工业物联网网关(IIoT Gateway)设备。云平台和应用层对采集的数据进行分析和可视化展示。

不同的IIoT应用场景需构建不同的IIoT技术栈乃至不同的操作系统实例。在实际应用中缺乏统一的标准框架且数据孤岛现象频发,研究者提出了物联网操作系统平台,以作为面向工业物联网的软件基础设施[257]。XiUOS[258]实现了一种典型工业物联应用场景的泛在操作系统,该操作系统支持IIoT应用,帮助解决车间智能化生产面临的"全面感知、泛在互联、实时认知、精准调控"等问题。

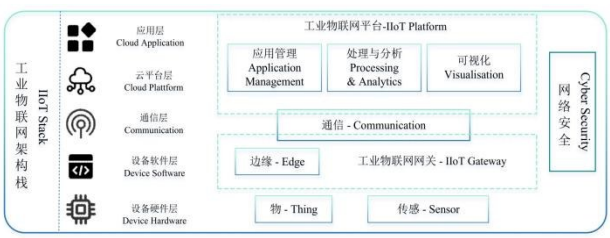

图5　物联网技术栈

Figure 5　Internet of Things Technology Stack

3.3　移动应用领域

移动应用是一个更为广泛的领域,其基础软件栈为移动应用提供了底层支持,屏蔽硬件差异。移动应用软件栈是一个分层的架构,提供完整的开发、运行和管理移动应用程序的能力[259-260]。

当前,最为主流的移动应用分层架构之一是基于Android系统的应用软件栈,它采用Linux内核作为底层基础,如图6所示,整个系统架构自下而上分为多个功能层:最底层是直接与设备硬件交互的Linux内核层,其上是中间件层(包括类库、应用框架等),它们共同构成了Android的核心功能体系[261]。

移动应用基础软件栈的未来发展趋势将朝着更高效的计算架构、更强的AI能力,以及更安全、更灵活的云服务等方向发展[262-263]。同时,移动应用的用户体验和安全性也将成为持续关注的重点。

3.4　其他领域

随着Web3.0技术的出现与发展,群体智能已成为解决社会问题的有力手段。相关研究中不乏面向群体智能应用和平台问题的框架,为群体智能基础软件栈的开发提供了实践基础。Atzori等人[264]在研究中于社交物联网分析平台上构建了移动群智感知(Mobile Crowd Sensing,MCS)管理框架,尝试公平分配社交虚拟对象资源,以避免节点过载;FROG框架[265]通过任务调度器和通知模块,能以高可靠性和低延迟将众包任务分配给合适的工作者;Herbertt等人[266]在为构建智慧城市参与式感知解决方案的MCS平台提出了一种新的方法,采用现成组件并提供直观易懂的参考架构,并在特定领域展示了该架构。然而,这些研究往往针对特定领域和任务,软件设计未考虑可重用性、可扩展性和可移植性,其成果难以应用于实际场景,这极大地限制了群体智能的广泛采用。为了设计系统性方法解决这些问题,CrowdOS[267]提出一种群体智能基础软件栈,不仅能管理系统中的异构硬件资源,还能为不同应用场景提供独特的资源抽象和软件定义服务,并提供丰富的功能组件和软件开发工具包。

4　挑战与展望

4.1　新型操作系统基础技术

面向未来智能时代,AIOS在融合通用性、智能性

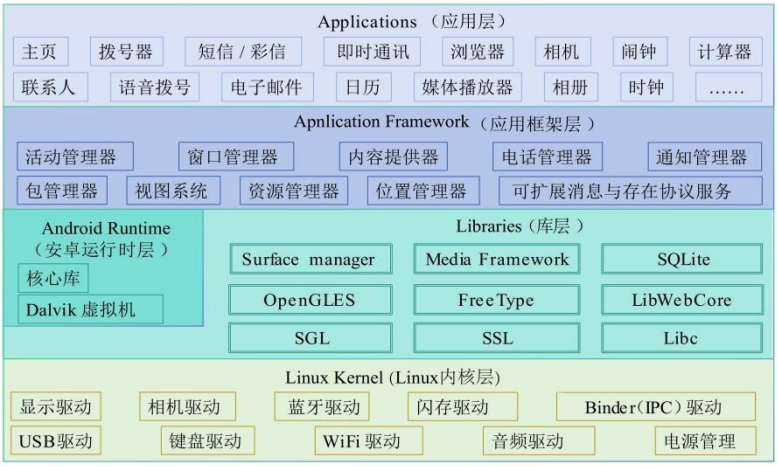

图6 Android应用软件栈

Figure 6 Android application stack

与安全性等多重需求的同时，还需具备对不同规模和类型硬件平台的灵活适配能力。这对操作系统的架构设计提出了更高要求，要求其能够支撑复杂多样的软硬件生态。当前，业界和学界在新型操作系统基础技术方面已展开多维度探索，但仍面临诸多挑战。

首先，如何实现操作系统的高度模块化与可定制性，是AIOS架构设计的核心难题之一。Unikraft[268]作为微库操作系统（micro-library OS）的代表，通过全模块化设计，将操作系统原语拆解为独立微库，开发者可根据具体应用需求精准组合，实现unikernel[269]的定制化开发。Unikraft还通过可组合的高性能API，支持如内存分配器、调度器等多种组件的按需替换，极大提升了系统的灵活性和适应性，能够契合多样化的应用场景。

其次，如何从根本上减少系统组件间的状态耦合、提升系统的可维护性与可靠性，也是AIOS亟需解决的问题。Theseus[270]作为一款基于Rust的实验性操作系统，提出了“最小化状态溢出（state spill）”和“细粒度组件化”的创新设计理念。其通过“细胞（cell）”结构将系统功能模块化，每个细胞作为独立的功能单元（对应Rust crate），拥有明确的运行时边界和依赖关系元数据，支持动态加载、链接与替换。通过“不透明导出（opaque exportation）”机制，Theseus将交互状态的管理权从服务端转移到客户端，避免了单一组件故障对全局的影响，从而提升了系统的健壮性和可维护性。

此外，针对多内核兼容的特殊需求，操作系统还需具备高效的多内核适配能力。相关研究提出了一种操作系统多内核适配装置[271]，通过模块化设计进一步提升系统灵活性。该装置包括设计模块、注册模块和适配模块：设计模块负责解析目标应用的内核配置文件，生成包含多个内核对象的目标内核构件；注册模块为每个内核对象分配唯一标识，并通过双向链表统一管理，形成可复用的目标数据；适配模块则基于目标数据对内核对象进行分类（被动型、主动型、事件型），经可行性验证后装配生成通用服务接口，实现从默认内核构件中动态适配目标内核。这一方案为AIOS在多内核环境下的高效适配和资源调度提供了有力支撑。

综上所述，AIOS在新型操作系统基础技术方面的挑战主要体现在架构的高度模块化、组件间状态解耦以及多内核环境下的高效适配等方面。未来，需持续推动架构创新与软硬件协同设计，进一步提升AIOS的灵活性、可扩展性与适应性，为智能时代的多样化应用场景提供坚实的系统基础。

4.2 AIOS的关键技术挑战

智能服务框架作为AIOS的重要组成部分，已在智能资源管理、智能安全服务和智能系统运维等方面展现出显著成效。然而，未来仍面临诸多挑战。例如，如何进一步提升资源调度的实时性与适应性，如何在复杂威胁场景下构建更具鲁棒性的安全防护体系，以及如何实现系统运维的全面自动化与智能化，都是亟待解决的问题。此外，领域基础软件栈的标准化与模块化设计虽已初见成效，但在跨平台、跨设备的智能协同与资源共享方面仍需进一步优化，以满足多样化场景的需求。

端侧智能操作系统在资源受限环境中的高效运行能力是AIOS发展的重要方向。尽管模型压缩、推理优化等技术已显著提升了端侧设备的AI推理能力，但在模型性能与资源消耗之间的平衡仍需进一步探索。例如，如何在保证推理效率的同时降低能耗，如何优化端云协同机制以实现动态负载均衡，都是未

来研究的重点。此外,随着边缘设备的多样化,操作系统需进一步适配异构硬件环境,提升其通用性与扩展性。

AIOS的记忆管理系统通过短期记忆与长期记忆的结合,为复杂任务中的知识动态更新与迁移提供了支持。然而,未来仍需解决以下挑战:如何优化记忆存储与检索机制以提升效率,如何在多模态交互中实现记忆的动态关联与更新,以及如何构建更接近人类认知的类人化记忆机制。此外,记忆管理系统的安全性与隐私保护也是重要议题,需在保障数据安全的同时提升系统的可用性。

展望未来,智能服务框架需进一步突破现有技术瓶颈,推动AI能力与操作系统的深度融合;端侧智能操作系统需在资源优化与硬件适配方面取得更大进展;记忆管理系统需在效率、动态性与安全性方面实现全面提升。通过持续的技术创新与协同优化,AIOS有望在多样化应用场景中展现更强的智能化与适应性,为智能时代的操作系统发展提供坚实支撑。

4.3　构建AIOS的安全可信基石

随着人工智能在各领域的深度渗透,AI系统的安全可信已从单纯的技术需求上升为战略层面的刚需。当前,人工智能技术,尤其是LLMs,由于其复杂的参数结构和神经网络决策机制,呈现出“黑盒”特性,导致模型决策过程难以解释。这不仅可能带来虚假或偏见性输出,还在数据隐私、伦理合规等方面引发潜在风险。以GPT-4、DeepSeek等前沿模型为例,虽然其千亿级参数和复杂训练机制带来了强大的语言生成能力,但也显著放大了数据泄露、模型被攻击(如对抗样本攻击)等安全隐患,进一步凸显了传统计算系统在AI安全保障方面的局限性。

面对上述挑战,可信计算技术不断演进,目前已发展到主动可信的“主动免疫可信计算3.0”理论模型[272]。如图7所示,该模型通过“计算+可信”双部件平台架构,在计算部件进行运算的同时,可信部件并行实施安全监控,实现对网络信息系统的主动免疫和积极防护。可信计算作为保障信息系统可预期性的关键技术,为AIOS的安全架构提供了坚实的理论基石。与传统外挂式安全机制不同,可信计算强调在计算过程中同步实施安全防护,通过构建从硬件根信任到软件执行环境的全链路信任体系,确保计算结果与预期一致、全程可测可控。

在AIOS的设计框架中,可信计算技术的融入需要在硬件、内核、应用3个层面实现深度协同。硬件层面,AIOS通过芯片内置的可信根(如可信计算模块TCM)构建硬件信任基石,确保系统启动和运行过程中关键代码与数据的完整性与保密性。内核层面,

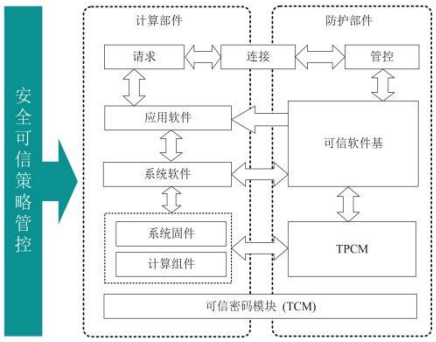

图7　主动免疫可信计算3.0架构

Figure 7　Active immune trusted computing 3.0 architecture

AIOS需将可信度量机制嵌入内核加载、进程调度、内存管理等核心模块,通过对系统组件的实时度量与动态认证,阻断恶意代码的执行路径,确保内核执行环境的纯净性。同时,利用可信计算的保密存储功能,对AI训练数据、模型参数等敏感信息进行加密存储与访问控制,防止数据泄露与篡改。应用层面,AIOS应提供可信应用开发框架,支持开发者TEE构建安全AI应用,通过隔离执行空间保障应用代码与数据在运行时的安全性。此外,借助区块链等分布式信任技术,实现AI模型的可信发布、溯源与验证,提升模型在跨机构、跨平台应用中的可信度。

5　结束语

本文系统梳理了安全可信的AIOS领域的最新研究进展与关键技术路径。围绕AIOS的分层架构、智能服务框架、安全访问框架、端侧智能、记忆管理、多模态智能体、安全可信机制等方面,归纳了当前主流技术方案与发展趋势,并结合具身智能、工业物联网、移动应用等典型场景,分析了AIOS的实际应用现状与面临的主要挑战。

总体来看,AIOS作为新一代操作系统的代表,正处于从理论探索到工程实践的快速演进阶段。未来发展面临以下5个重要方向。

(1)架构创新与软硬件协同。随着AI应用场景的不断扩展,AIOS需进一步推动系统架构的高度模块化与可定制性,强化异构硬件的统一抽象与高效调度,提升系统的灵活性与可扩展性。

(2)智能能力与操作系统深度融合。智能服务框架将持续拓展AI驱动的资源管理、安全服务和系统运维能力,推动AI能力从外围插件向内核深度嵌入,实现操作系统的主动感知、智能决策和自适应优化。

(3)安全可信机制的纵深演进。面对AI模型“黑盒”特性和数据安全新挑战,AIOS需加强内存安全、形式化验证、可信执行环境、机密计算等纵深防御体

系建设,推动安全机制的自动化、智能化和全链路覆盖。

(4)多场景适配与生态建设。AIOS需适应具身智能、工业物联网、移动应用等多样化场景,推动领域基础软件栈的标准化和生态完善,促进跨平台、跨设备的智能协同与资源共享。

(5)未来趋势展望。展望未来,AIOS将在隐身化、泛在化、跨域协同、智能体交互等方向持续创新,推动操作系统从传统支撑平台向智能化、可信化、泛在化的基础设施转型。

综上,AIOS作为支撑智能时代多样化应用的关键基础设施,其发展有赖于体系结构创新、软硬件协同、安全可信机制与AI能力的深度融合。未来,AIOS将在理论研究和产业实践中持续突破,为人工智能应用的安全、智能和可信发展提供坚实支撑。

参考文献

[1] 高蕾, 符永铨, 李东升, 等. 我国人工智能核心软硬件发展战略研究[J]. 中国工程科学, 2021, 23(3): 90-97.
Gao L, Fu Y Q, Li D S, et al. Development strategy for the core software and hardware of artificial intelligence in China [J]. Strategic Study of CAE, 2021, 23(3): 90-97. (in Chinese)

[2] 中华人民共和国国务院. 国务院关于印发新一代人工智能发展规划的通知[EB/OL]. (2017-07-20)[2021-4-22]. https://www.gov.cn/zhengce/content/2017-07/20/content_5211996.htm.
The State Council of the People´s Republic of China. Circular of the state council on issuing the development plan for a new generation of artificial intelligence[EB/OL]. (2017-07-20) [2021-04-22]. https://www. gov. cn/zhengce/content/2017-07/20/content_5211996.htm. (in Chinese)

[3] 中国信息通信研究院. 全球人工智能战略与政策观察(2020)[EB/OL]. (2020-12)[2025-10-10]. https://www.cai-ct.ac.cn/kxyj/qwfb/ztbg/202012/P020201229520426700957.pdf.
China Academy of Information and Communications Technology. Global AI strategy and policy review (2020)[EB/OL]. (2020-12)[2025-10-10]. https://www.caict.ac.cn/kxyj/qwfb/ztbg/202012/P020201229520426700957.pdf. (in Chinese)

[4] 美国国家科学技术委员会(中国信息通信研究院政策与经济研究所编译组整理). 美国国家人工智能研究和发展战略计划[EB/OL]. (2016-10)[2025-10-10]. http://www.caict.ac.cn/kxyj/qwfb/ztbg/201804/P020161102381492504518.pdf.
National Science and Technology Council of the United States (Compiled by the Compilation Group of the Institute of Policy and Economics, China Academy of Information and Communications Technology). National artificial intelligence research and development strategic plan[EB/OL]. (2016-10) [2025-10-10]. http://www. caict. ac. cn/kxyj/qwfb/ztbg/201804/P020161102381492504518.pdf. (in Chinese)

[5] The White House. Executive order on maintaining American leadership in artificial intelligence[EB/OL]. (2019-02-11) [2025-10-10]. https://trumpwhitehouse. archives. gov/presidential-actions/executive-order-maintaining-american-leadership-artificial-intelligence/.

[6] The White House. The national artificial intelligence research and development strategic plan: 2019 Update[EB/OL]. (2019-06) [2025-10-10]. https://www. nitrd. gov/pubs/National-AI-RD-Strategy-2019.pdf.

[7] 梅宏, 曹东刚, 谢涛. 泛在操作系统: 面向人机物融合泛在计算的新蓝海[J]. 中国科学院院刊, 2022, 37(1): 30-37.
Mei H, Cao D G, Xie T. Ubiquitous operating system: Toward the blue ocean of human-cyber-physical ternary ubiquitous computing[J]. Bulletin of the Chinese Academy of Sciences, 2022, 37(1): 30-37. (in Chinese)

[8] Choi Y, Park S, Cha H. Graphics-aware power governing for mobile devices[C]//Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services. New York: ACM, 2019: 469-481.

[9] Choi Y, Park S, Jeon S, et al. Optimizing energy consumption of mobile games[J]. IEEE Transactions on Mobile Computing, 2022, 21(10): 3744-3756.

[10] Du D, Hua Z C, Xia Y B, et al. XPC: Architectural support for secure and efficient cross process call[C]//Proceedings of the 46th International Symposium on Computer Architecture. New York: ACM, 2019: 671-684.

[11] Herder J N, Bos H, Gras B, et al. MINIX 3: A highly reliable, self-repairing operating system[J]. ACM SIGOPS Operating Systems Review, 2006, 40(3): 80-89.

[12] Liedtke J. On micro-kernel construction[J]. ACM SIGOPS Operating Systems Review, 1995, 29(5): 237-250.

[13] Liedtke J. Toward real microkernels[J]. Communications of the ACM, 1996, 39(9): 70-77.

[14] Elphinstone K, Heiser G. From L3 to seL4 what have we learnt in 20 years of L4 microkernels? [C]//Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. New York: ACM, 2013: 133-150.

[15] Chen H B, Miao X, Jia N, et al. Microkernel goes general: Performance and compatibility in the HongMeng production microkernel[C]//18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). Berkeley: USENIX Association, 2024: 465-485.

[16] Microsoft. MS Windows NT Kernel-mode user and GDI white paper[EB/OL]. (2014-01-01) [2025-08-10]. https://learn.microsoft.com/en-us/previous-versions/cc750820(v=technet.10).

[17] APple. XNU project[EB/OL]. (2025-07-01)[2025-08-10]. https://github.com/apple-oss-distributions/xnu.

[18] Engler D R. The Exokernel operating system architecture[D]. Cambridge: Massachusetts Institute of Technology, 1998.

[19] 余琪. 基于外内核操作系统的调度研究[D]. 兰州: 兰州大学, 2014.
Yu Q. Research on Exokernel system scheduling[D]. Lanzhou: Lanzhou University, 2014. (in Chinese)

[20] 李肇中. 基于外内核操作系统的文件系统研究[D]. 兰州: 兰州大学, 2016.
Li Z Z. A study of file system based on Exokernel operating system[D]. Lanzhou: Lanzhou University, 2016. (in Chinese)

[21] 刘群. 面向多核处理器的外内核操作系统研究[D]. 兰州: 兰州大学, 2014.
Liu Q. A study of Exokernel operating system on[D]. Lanzhou: Lanzhou University, 2014. (in Chinese)

[22] 胡俊. 外内核时间子系统的初步研究[D]. 兰州: 兰州大学, 2014.
Hu J. Preliminary research on time subsystem in Exokernel[D]. Lanzhou: Lanzhou University, 2014. (in Chinese)

[23] Hambarde P, Varma R, Jha S. The survey of real time operating system: RTOS[C]//2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies. Piscataway: IEEE, 2014: 34-39.

[24] Stankovic J A, Ramamritham K. The Spring kernel: A new paradigm for real-time operating systems[J]. ACM SIGOPS Operating Systems Review, 1989, 23(3): 54-71.

[25] Mantegazza P, Dozio E L, Papacharalambous S. RTAI: Real time application interface[C]//Linux Journal. New York: ACM, 2000: 10-es.

[26] Hildebrand D. An architectural overview of QNX[C]// Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures. New York: ACM, 1992: 113-126.

[27] Grattan N, Brain M. Windows CE 3.0[M]. Upper Saddle River, NJ: Prentice Hall PTR, 2001.

[28] Burns A, Davis R I. A survey of research into mixed criticality systems[J]. ACM Computing Surveys, 2018, 50(6): 1-37.

[29] Kappel J A, Velte A T, et al. Microsoft virtualization with Hyper-V[M]. New York: McGraw Hill, 2009.

[30] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 164-177.

[31] Kivity A, Kamay Y, Laor D, et al. KVM: The Linux virtual machine monitor[C]//Proceedings of the Linux Symposium. Ottawa: OLS, 2007: 225-230.

[32] Xi S S, Wilson J, Lu C Y, et al. RT-Xen: Towards real-time hypervisor scheduling in xen[C]//Proceedings of the Ninth ACM International Conference on Embedded Software. New York: ACM, 2011: 39-48.

[33] Abeni L, Faggioli D. An experimental analysis of the xen and KVM latencies[C]//2019 IEEE 22nd International Symposium on Real-Time Distributed Computing. Piscataway: IEEE, 2019: 18-26.

[34] Cziva R, Pezaros D P. Container network functions: Bringing NFV to the network edge[J]. IEEE Communications Magazine, 2017, 55(6): 24-31.

[35] Heiser G, Leslie B. The OKL4 microvisor: Convergence point of microkernels and hypervisors[C]//Proceedings of the First ACM Asia-Pacific Workshop on Workshop on Systems. New York: ACM, 2010: 19-24.

[36] Steinberg U, Kauer B. NOVA: A microhypervisor-based secure virtualization architecture[C]//Proceedings of the 5th European Conference on Computer Systems. New York: ACM, 2010: 209-222.

[37] Zhang F Z, Chen J, Chen H B, et al. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization[C]//Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. New York: ACM, 2011: 203-216.

[38] Cinque M, Della Corte R, Eliso A, et al. Rt-cases: Container-based virtualization for temporally separated mixed-criticality task sets[C]//31st Euromicro Conference on Real-Time Systems (ECRTS 2019). Wadern: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019: 5:1-5:22.

[39] Lankes S, Pickartz S, Breitbart J. HermitCore: A unikernel for extreme scale computing[C]//Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers. New York: ACM, 2016: 1-8.

[40] Martins J, Ahmed M, Raiciu C, et al. ClickOS and the art of network function virtualization[C]//Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation. New York: ACM, 2014: 459-473.

[41] Kaiser R, Wagner S. Evolution of the PikeOS microkernel[C]// First International Workshop on Microkernels for Embedded Systems. Kensington: National ICT Australia, 2007: 50.

[42] Masmano M, Ripoll I, Crespo A, et al. Xtratum: A hypervisor for safety critical embedded systems[C]//11th Real-Time Linux Workshop. Dresden: IEEE, 2009: 1-9.

[43] martins j, tavares a, solieri m, et al. Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems[C]//Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020). Dagstuhl: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2020:

3:1-3:14.

[44] Goodarzy S, Nazari M, Han R, et al. SmartOS: Towards automated learning and user-adaptive resource allocation in operating systems[C]//Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems. New York: ACM, 2021: 48-55.

[45] Aaen Springborg A, Albano M, Xavier-de-Souza S. Automatic energy-efficient job scheduling in HPC: A novel SLURM plugin approach[C]//Proceedings of the SC ′23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis. New York: ACM, 2023: 1831-1838.

[46] Othmen S, Mansouri W, Khdhir R. Applying artificial intelligence techniques for resource management in the Internet of Things (IoT)[J]. International Journal of Electrical and Computer Engineering Systems, 2025, 16(2): 183-194.

[47] Selvarajan G P. AI-driven cloud resource management and orchestration[J]. International Journal of Innovative Research in Science, Engineering and Technology, 2024, 13(11): 19367-19380.

[48] Qin X X, Zeng F P, Zhang Y. MSNdroid: The Android malware detector based on multi-class features and deep belief network[C]//Proceedings of the ACM Turing Celebration Conference - China. New York: ACM, 2019: 1-5.

[49] Alzaylaee M K, Yerima S Y, Sezer S. DL-Droid: Deep learning based Android malware detection using real devices[J]. Computers & Security, 2020, 89: 101663.

[50] Benabderrahmane S, Hoang N, Valtchev P, et al. Hack me if you can: Aggregating autoencoders for countering persistent access threats within highly imbalanced data[J]. Future Generation Computer Systems, 2024, 160: 926-941.

[51] Dusane P, Sujatha G. LogEA: Log extraction and analysis tool to support forensic investigation of linux-based system[C]//2021 5th International Conference on Trends in Electronics and Informatics. Piscataway: IEEE, 2021: 909-916.

[52] Das A, Mueller F, Siegel C, et al. Desh: Deep learning for system health prediction of lead times to failure in HPC[C]// Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing. New York: ACM, 2018: 40-51.

[53] Li T G, Ying S, Zhao Y S, et al. Batch jobs load balancing scheduling in cloud computing using distributional reinforcement learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(1): 169-185.

[54] Liu J, Jia J C, Ma B C, et al. Multi-job intelligent scheduling with cross-device federated learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(2): 535-551.

[55] Schüpbach A L. Tackling OS complexity with declarative techniques[D]. Zurich: ETH Zurich, 2012.

[56] Chen J D, Banerjee S S, Kalbarczyk Z T, et al. Machine learning for load balancing in the Linux kernel[C]//Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems. New York: ACM, 2020: 67-74.

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

[58] Wihar J, Mathur R, Northington K, et al. A novel approach of LSTM-based ransomware detection in the Linux operating system kernel[EB/OL]. (2024-09-19) [2025-08-19]. https://d197for5662m48. cloudfront. net/documents/publicationstatus/224324/preprint_pdf/7408d68c8fd3caba127923e3a8736481.pdf.

[59] Kommrusch S. Artificial intelligence techniques for security vulnerability prevention[PP/OL]. V1. arXiv (2019-12-14) [2025-12-01]. https://doi.org/10.48550/arXiv.1912.06796.

[60] Agarwal T, Tyagi A K. Artificial intelligence and qubit-based operating systems: Current progress and future perspectives[M]//Quantum Computing in Cybersecurity. Hoboken: Wiley, 2023: 121-136.

[61] George A S. Emerging trends in AI-driven cybersecurity: An in-depth analysis[J]. Partners Universal Innovative Research Publication, 2024, 2(4): 15-28.

[62] Temara S. Harnessing the power of artificial intelligence to enhance next-generation cybersecurity[J]. World Journal of Advanced Research and Reviews, 2024, 23(2): 797-811.

[63] Rizvi M. Enhancing cybersecurity: The power of artificial intelligence in threat detection and prevention[J]. International Journal of Advanced Engineering Research and Science, 2023, 10(5): 55-60.

[64] Zhang Y J, Zhou L W, Makris Y. Hardware-based real-time workload forensics via frame-level TLB profiling[C]//2019 IEEE 37th VLSI Test Symposium. Piscataway: IEEE, 2019: 1-6.

[65] Locher M G. Optimizing IT operations with AIOps: An investigation into the opportunities and challenges for enterprise adoption[D]. Switzerland: Zurich University of Applied Sciences, 2023.

[66] Notaro P, Cardoso J, Gerndt M. A survey of AIOps methods for failure management[J]. ACM Transactions on Intelligent Systems and Technology, 2021, 12(6): 1-45.

[67] Zhang L Z, Jia T, Jia M X, et al. A survey of AIOps in the era of large language models[J]. ACM Computing

Surveys, 2026, 58(2): 1-35.

[68] Peng Y K, Tian H L, Xian J Y, et al. Framekernel: A safe and efficient kernel architecture *via* rust-based intra-kernel privilege separation[C]//Proceedings of the 15th ACM SIGOPS Asia-Pacific Workshop on Systems. New York: ACM, 2024: 31-37.

[69] Peng Y K, Tian H L, Zhang J Y, et al. Asterinas: A linux ABI-compatible, rust-based framekernel OS with a small and sound TCB[PP/OL]. (2025-06-04) [2025-11-10]. https://doi.org/10.48550/arXiv.2506.03876.

[70] Hazarika A, Poddar S, Rahaman H. Survey on memory management techniques in heterogeneous computing systems[J]. IET Computers & Digital Techniques, 2020, 14(2): 47-60.

[71] Arm Limited. Introduction to the memory tagging extension[EB/OL]. [2025-08-19]. https://developer.arm.com/documentation/108035/0100/Introduction-to-the-Memory-Tagging-Extension.

[72] Partap A, Boneh D. Memory tagging: A memory efficient design[PP/OL]. V2. arXiv (2022-11-03) [2025-10-11]. https://doi.org/10.48550/arXiv.2209.00307.

[73] Arm Limited. Pointer authentication code[EB/OL]. [2025-08-18]. https://developer.arm.com/documentation/109576/0100/Pointer-Authentication-Code.

[74] Kim H, Lee J, Kim S, et al. How´d security benefit reverse engineers? : The implication of intel CET on function identification[C]//2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Piscataway: IEEE, 2022: 559-566.

[75] Microsoft. Kernel-mode hardware-enforced stack protection[EB/OL]. (2023-01-27)[2025-08-19]. https://learn.microsoft. com/en-us/windows-server/security/kernel-mode-hardware-stack-protection.

[76] De A, Basu A, Ghosh S, et al. FIXER: Flow integrity extensions for embedded RISC-V[C]//2019 Design, Automation & Test in Europe Conference & Exhibition. Piscataway: IEEE, 2019: 348-353.

[77] Wang Y, Wu J T, Yue T, et al. RetTag: Hardware-assisted return address integrity on RISC-V[C]//Proceedings of the 15th European Workshop on Systems Security. New York: ACM, 2022: 50-56.

[78] Watson R N M, Neumann P G, Woodruff J, et al. Capability hardware enhanced RISC instructions: CHERI instruction-set architecture[R]. Cambridge: University of Cambridge, 2014.

[79] Amar S, Chisnall D, Chen T, et al. CHERIoT: Complete memory safety for embedded devices[C]//Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture. New York: ACM, 2023: 641-653.

[80] Yoo S, Jerraya A A. Introduction to hardware abstraction layers for SoC[C]//2003 Design, Automation and Test in Europe Conference and Exhibition. Piscataway: IEEE, 2003: 336-337.

[81] Mayrhofer R, Vander Stoep J, Brubaker C, et al. The Android platform security model (2023)[PP/OL]. V3. arXiv (2024-01-09) [2025-10-10]. https://doi. org/10.48550/arXiv.1904.05572.

[82] Yim K S, Malchev I, Hsieh A, et al. TREBLE: Fast software updates by creating an equilibrium in an active software ecosystem of globally distributed stakeholders[PP/OL]. V1. arXiv (2024-09-19)[2025-10-10]. https://doi.org/10.48550/arXiv.2410.02809.

[83] Microsoft. Secure boot[EB/OL]. (2024-05-28) [2025-08-18]. https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-secure-boot.

[84] Jia Y K, Liu S, Wang W H, et al. HyperEnclave: An open and cross-platform trusted execution environment[PP/OL]. V1. arXiv (2022-12-08)[2025-10-10]. https://doi.org/10.48550/arXiv.2212.04197.

[85] Microsoft. Virtualization-based security (VBS) [EB/OL]. (2024-05-28) [2025-08-18]. https://learn. microsoft. com/en-us/windows-hardware/design/device-experiences/oem-vbs.

[86] Microsoft. Device guard and credential guard[EB/OL]. (2024-05-28) [2025-08-18]. https://learn. microsoft. com/en-us/windows-hardware/drivers/bringup/device-guard-and-cre - dential-guard.

[87] Wei H J, Lin L K, Lin C Y, et al. Measuring and optimizing the performance of the Android virtualization framework[C]//Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing. New York: ACM, 2024: 1533-1535.

[88] Apple Inc. Virtualization: Create and manage virtual machines on Apple silicon and Intel-based Mac computers [EB/OL]. (2024-06-10)[2025-08-18]. https://developer.apple.com/documentation/virtualization.

[89] Randazzo A, Tinnirello I. Kata containers: An emerging architecture for enabling MEC services in fast and secure way[C]//2019 Sixth International Conference on Internet of Things: Systems, Management and Security. Piscataway: IEEE, 2019: 209-214.

[90] Rutkowska J, Wojtczuk R. Qubes OS architecture[J]. Invisible Things Lab Tech Rep, 2010, 54: 65.

[91] Karat C M, Vergo J, Nahamoo D. Conversational interface technologies[M]//The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications. New Jersey: L. Erlbaum Associ-

ates Inc., 2002: 169-186.

[92] Darejeh A, Mashayekh S, Marcus N. Cognitive-based methods to facilitate learning of software applications via E-learning systems[J]. Cogent Education, 2022, 9: 2082085.

[93] Wang L, Ma C, Feng X Y, et al. A survey on large language model based autonomous agents[J]. Frontiers of Computer Science, 2024, 18(6): 186345.

[94] Mei K, Zhu X, Xu W J, et al. AIOS: LLM agent operating system[PP/OL]. V5. arXiv (2025-08-12) [2025-10-10]. https://doi.org/10.48550/arXiv.2403.16971.

[95] Stratton J. An introduction to microsoft copilot[M]//Copilot for Microsoft 365. Berkeley, CAA Press, 2024: 19-35.

[96] Hou X Y, Zhao Y J, Wang S N, et al. Model context protocol (MCP): Landscape, security threats, and future research directions[PP/OL]. V3. arXiv (2025-10-07) [2025-10-10]. https://doi.org/10.48550/arXiv.2503.23278.

[97] Yan A, Yang Z Y, Zhu W R, et al. GPT-4V in wonderland: Large multimodal models for zero-shot smartphone GUI navigation[PP/OL]. V1. arXiv (2023-11-13) [2025-10-10]. https://doi.org/10.48550/arXiv.2311.07562.

[98] Zhang C, Yang Z, Liu J X, et al. AppAgent: Multimodal agents as smartphone users[C]//Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. New York: ACM, 2025: 1-20.

[99] Wang J Y, Xu H Y, Ye J B, et al. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception[PP/OL]. V2. arXiv (2024-04-18) [2025-10-10]. https://doi.org/10.48550/arXiv.2401.16158.

[100] Zhang C Y, Li L Q, He S L, et al. UFO: A UI-focused agent for windows OS interaction[PP/OL]. V5. arXiv (2024-05-23)[2025-10-10]. https://doi.org/10.48550/arXiv.2402.07939.

[101] Hong W Y, Wang W H, Lv Q S, et al. CogAgent: A visual language model for GUI agents[C]//2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition. Piscataway: IEEE, 2024: 14281-14290.

[102] Dongle. AutoMate[EB/OL]. (2025-03-20) [2025-10-10]. https://github.com/yuruotong1/autoMate.

[103] Huang J, Chang K C. Towards reasoning in large language models: A survey[PP/OL]. V2. arXiv (2023-05-26)[2025-10-10]. https://doi.org/10.48550/arXiv.2212.10403.

[104] Du Y, Wei F Y, Zhang H Y. AnyTool: Self-reflective, hierarchical agents for large-scale API calls[PP/OL]. V1. arXiv (2024-02-06)[2025-10-10]. https://doi.org/10.48550/arXiv.2402.04253.

[105] Guo T C, Chen X Y, Wang Y Q, et al. Large language model based multi-agents: A survey of progress and challenges[PP/OL]. V2. arXiv (2024-04-19)[2025-10-10]. https://doi.org/10.48550/arXiv.2402.01680.

[106] Google. A2A protocol[EB/OL]. [2025-08-12]. https://a2a-protocol.org/latest/.

[107] OpenAI. GPT-4o[EB/OL]. (2024-08-28) [2025-08-12]. https://platform.openai.com/docs/models/gpt-4o.

[108] Team G, Anil R, Borgeaud S, et al. Gemini: A family of highly capable multimodal models[PP/OL]. V5. arXiv (2025-05-09)[2025-10-10]. https://doi.org/10.48550/arXiv.2312.11805.

[109] Sager P J, Meyer B, Yan P, et al. A comprehensive survey of agents for computer use: Foundations, challenges, and future directions[PP/OL]. V2. arXiv (2025-06-04)[2025-10-10]. https://doi.org/10.48550/arXiv.2501.16150.

[110] Hu X Y, Xiong T, Yi B, et al. OS agents: A survey on MLLM-based agents for general computing devices use [PP/OL]. V1. arXiv (2025-08-06) [2025-10-10]. https://doi.org/10.48550/arXiv.2508.04482.

[111] OpenAI. GPT-4V(ision) [EB/OL]. (2024-09-22) [2025-08-12]. https://platform. openai. com/docs/models/gpt-4-turbo-and-gpt-4.

[112] Muñoz A, Ríos R, Román R, et al. A survey on the (in) security of trusted execution environments[J]. Computers & Security, 2023, 129: 103180.

[113] 刘国杰, 张建标, 杨萍, 等. 基于TPCM的容器云可信环境研究[J]. 网络与信息安全学报, 2021, 7(4): 164-174.
Liu Guojie, Zhang Jianbiao, Yang Ping, et al. Research on the trusted environment of container cloud based on the TPCM[J]. Chinese Journal of Network and Information Security, 2021, 7(4): 164-174. (in Chinese)

[114] 徐涛, 孔帅迪, 刘才华, 等. 异构机密计算综述[J]. 吉林大学学报(工学版), 2025, 55(3): 755-770.
Xu Tao, Kong Sshuaidi, Liu Caihua, et al. Overview of heterogeneous confidential computing[J]. Journal of Jilin University (Engineering and Technology Edition), 2025, 55(3): 755-770. (in Chinese)

[115] Zhou Y L, Peng G J, Li Z C, et al. A survey on the evolution of bootkits attack and defense techniques[J]. China Communications, 2024, 21(1): 102-130.

[116] Zheng C M, Li J, Yao X X. Design and implementation of trusted boot based on a new trusted computing dual-architecture[J]. Computers & Security, 2023, 127: 103095.

[117] Mohammadzad M, Karimpour J. Using rootkits hiding techniques to conceal honeypot functionality[J]. Journal of Network and Computer Applications, 2023, 214: 103606.

[118] Huang H X, Zhang J B, Zhang L, et al. SABDTM: Security-first architecture-based dynamic trusted measurement scheme for operating system of the virtual computing node[J]. Computers & Security, 2024, 137: 103648.

[119] Chen H. Optimized design of trusted boot architecture based on embedded devices[C]//2025 4th International Symposium on Computer Applications and Information Technology. Piscataway: IEEE, 2025: 1043-1046.

[120] Feng D G, Qin Y, Feng W, et al. Survey of research on confidential computing[J]. IET Communications, 2024, 18(9): 535-556.

[121] Russinovich M. Confidential computing: Elevating cloud security and privacy[J]. Communications of the ACM, 2024, 67(1): 52-53.

[122] Sabanic P, Misono M, Bodea T, et al. Confidential serverless computing[PP/OL]. V3. arXiv (2025-08-13) [2025-10-10]. https://doi.org/10.48550/arXiv.2504.21518.

[123] Popa R A. Confidential computing or cryptographic computing?[J]. Communications of the ACM, 2024, 67(12): 44-51.

[124] Martishin S A, Khrapchenko M V, Shokurov A V. Study of the problem of ensuring security in storage and processing of confidential data[J]. Programming and Computer Software, 2022, 48(7): 424-434.

[125] Benadjila R, Khati L, Vergnaud D. Secure storage: Confidentiality and authentication[J]. Computer Science Review, 2022, 44: 100465.

[126] Global Market Insights. Global market insights: Market research reports[EB/OL]. (2025-03-01)[2025-10-10]. https://www.gminsights.com/industry-analysis/edge-ai-market.

[127] Qin R Y, Liu D C, Xu C H, et al. Empirical guidelines for deploying LLMs onto resource-constrained edge devices[J]. ACM Transactions on Design Automation of Electronic Systems, 2025, 30(5): 1-58.

[128] Liu L Y, Zhu H B, Chen S L, et al. Privacy-aware task assignment for IoT audit applications on collaborative edge devices[J]. Security and Communication Networks, 2022, 2022: 1336094.

[129] Touvron H, Lavril T, Izacard G, et al. LLaMA: Open and efficient foundation language models[PP/OL]. V1. arXiv (2023-02-27)[2025-10-10]. https://doi.org/10.48550/arXiv.2302.13971.

[130] Seedat N, Huynh N, van Breugel B, et al. Curated LLM: Synergy of LLMs and Data Curation for tabular augmentation in low-data regimes[PP/OL]. V3. arXiv (2024-06-30) [2025-10-10]. https://doi.org/10.48550/arXiv.2312.12112.

[131] Dell. Llama 2: Inferencing on a single gpu[EB/OL]. [2025-08-10]. https://infohub. delltechnologies. com/zh-cn/t/llama-2-inferencing-on-a-single-gpu/.

[132] Zhou Z X, Ning X F, Hong K, et al. A survey on efficient inference for large language models[PP/OL]. V3. arXiv (2024-07-19)[2025-10-10]. https://doi.org/10.48550/arXiv.2404.14294.

[133] Chung J W, Liu J, Wu Z, et al. The ML.ENERGY leaderboard[EB/OL]. [2025-08-10]. https://ml.energy/leaderboard.

[134] Xu J J, Li Z Y, Chen W, et al. On-device language models: A comprehensive review[PP/OL]. V2. arXiv (2024-09-14) [2025-10-10]. https://doi.org/10.48550/arXiv.2409.00088.

[135] 李诚. 面向LLM推理服务的体系结构[R/OL]. (2024-07-01)[2025-07-20]. http://syswonder-cdn.oscommunity.cn/20240706-llmarch-3-lc.pdf.
Li Cheng. Architecture for LLM inference services[R/OL]. (2024-07-01)[2025-07-20]. http://syswonder-cdn.oscommunity.cn/20240706-llmarch-3-lc.pdf.

[136] Press O, Wolf L. Using the output embedding to improve language models[PP/OL]. V3. arXiv (2017-02-21)[2025-10-10]. https://doi.org/10.48550/arXiv.1608.05859.

[137] Liu Z C, Zhao C S, Iandola F, et al. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases[PP/OL]. V2. arXiv (2024-06-27) [2025-10-10]. https://doi.org/10.48550/arXiv.2402.14905.

[138] Santacroce M, Wen Z X, Shen Y L, et al. What matters in the structured pruning of generative language models? [PP/OL]. V1. arXiv (2023-02-07) [2025-10-10]. https://doi.org/10.48550/arXiv.2302.03773.

[139] Ma X Y, Fang G F, Wang X C. LLM-pruner: On the structural pruning of large language models[PP/OL]. V3. arXiv (2023-09-28)[2025-10-10]. https://doi.org/10.48550/arXiv.2305.11627.

[140] Chen T Y, Ding T Y, Yadav B, et al. LoRAShear: Efficient large language model structured pruning and knowledge recovery[PP/OL]. V2. arXiv (2023-10-31) [2025-10-10]. https://doi.org/10.48550/arXiv.2310.18356.

[141] Zhang M Y, Chen H, Shen C H, et al. LoRAPrune: Structured pruning meets low-rank parameter-efficient fine-tuning[C]//Findings of the Association for Computational Linguistics ACL 2024. Stroudsburg: ACL, 2024: 3013-3026.

[142] Frantar E, Alistarh D. SparseGPT: Massive language models can be accurately pruned in one-shot[PP/OL]. V3. arXiv (2023-03-22)[2025-10-10]. https://doi.org/10.48550/arXiv.2301.00774.

[143] Sun M J, Liu Z, Bair A, et al. A simple and effective pruning approach for large language models[PP/OL]. V3. arXiv (2024-05-06)[2025-10-10]. https://doi.org/10.48550/arXiv.2306.11695.

[144] Shao H, Liu B, Qian Y M. One-shot sensitivity-aware mixed sparsity pruning for large language models[C]// ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing. Piscataway:

IEEE, 2024: 11296-11300.

[145] Gu Y X, Dong L, Wei F R, et al. MiniLLM: On-policy distillation of large language models[PP/OL]. V6. arXiv (2026-01-31)[2025-12-01]. https://doi.org/10.48550/arXiv.2306.08543.

[146] Zhang C, Li Q C, Song D W, et al. Towards the law of capacity gap in distilling language models[PP/OL]. V4. arXiv (2025-07-30)[2025-10-10]. https://doi.org/10.48550/arXiv.2311.07052.

[147] Agarwal R, Vieillard N, Zhou Y C, et al. On-policy distillation of language models: Learning from self-generated mistakes[PP/OL]. V3. arXiv (2024-01-17) [2025-10-10]. https://doi.org/10.48550/arXiv.2306.13649.

[148] Zhu X K, Qi B Q, Zhang K Y, et al. PaD: Program-aided distillation can teach small models reasoning better than chain-of-thought fine-tuning[PP/OL]. V2. arXiv (2024-03-20)[2025-10-10]. https://doi.org/10.48550/arXiv.2305.13888.

[149] Li L H, Hessel J, Yu Y, et al. Symbolic chain-of-thought distillation: Small models can also "think" step-by-step [PP/OL]. V2. arXiv (2024-04-15) [2025-10-10]. https://doi.org/10.48550/arXiv.2306.14050.

[150] Wang P F, Wang Z Y, Li Z, et al. SCOTT: Self-consistent chain-of-thought distillation[PP/OL]. V4. arXiv (2023-08-30)[2025-10-10]. https://doi.org/10.48550/arXiv.2305.01879.

[151] Chen Z M, Gao Q Y, Bosselut A, et al. DISCO: Distilling counterfactuals with large language models[PP/OL]. V3. arXiv (2023-06-05)[2025-10-10]. https://doi.org/10.48550/arXiv.2212.10534.

[152] Kim S, Mangalam K, Moon S, et al. Speculative decoding with big little decoder[C]//Proceedings of the 37th International Conference on Neural Information Processing Systems. New York: ACM, 2023: 39236-39256.

[153] Xu D L, Yin W S, Jin X, et al. LLMCad: Fast and scalable on-device large language model inference[PP/OL]. V1. arXiv (2023-09-08)[2025-10-10]. https://doi.org/10.48550/arXiv.2309.04255.

[154] Dao T, Fu D Y, Ermon S, et al. FlashAttention: Fast and memory-efficient exact attention with IO-awareness[PP/OL]. V2. arXiv (2022-06-23)[2025-10-10]. https://doi.org/10.48550/arXiv.2205.14135.

[155] Kwon W, Li Z H, Zhuang S Y, et al. Efficient memory management for large language model serving with PagedAttention[C]//Proceedings of the 29th Symposium on Operating Systems Principles. New York: ACM, 2023: 611-626.

[156] Ggerganov, Slaren, Ngxson, et al. llama. cpp: Light-weight library for approximate nearest neighbors and maximum inner product search[EB/OL]. (2025-08-29) [2025-10-10]. https://github.com/ggerganov/llama.cpp.

[157] Song Y X, Mi Z Y, Xie H T, et al. PowerInfer: Fast large language model serving with a consumer-grade GPU[C]//Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles. New York: ACM, 2024: 590-606.

[158] Li L C, Qian S, Lu J, et al. Transformer-lite: High-efficiency deployment of large language models on mobile phone GPUs[PP/OL]. V3. arXiv (2024-07-05) [2025-10-10]. https://doi.org/10.48550/arXiv.2403.20041.

[159] Ding D J, Mallick A, Wang C, et al. Hybrid LLM: Cost-efficient and quality-aware query routing[PP/OL]. V1. arXiv (2024-04-22)[2025-10-10]. https://doi.org/10.48550/arXiv.2404.14618.

[160] Wang Y D, Chen K, Tan H S, et al. Tabi: An efficient multi-level inference system for large language models[C]// Proceedings of the Eighteenth European Conference on Computer Systems. New York: ACM, 2023: 233-248.

[161] Vllm Project Team. VLLM documentation[EB/OL]. (2025-11-15)[2025-12-01]. https://docs.vllm.ai/en/stable/.

[162] Cai F L, Yuan D, Yang Z, et al. Edge-LLM: A collaborative framework for large language model serving in edge computing[C]//2024 IEEE International Conference on Web Services. Piscataway: IEEE, 2024: 799-809.

[163] Xu X H, Li M, Tao C Y, et al. A survey on knowledge distillation of large language models[PP/OL]. V4. arXiv (2024-10-21)[2025-10-10]. https://doi.org/10.48550/arXiv.2402.13116.

[164] Wang R, Gao Z Y, Zhang L Y, et al. Empowering large language models to edge intelligence: A survey of edge efficient LLMs and techniques[J]. Computer Science Review, 2025, 57: 100755.

[165] Chen C, Borgeaud S, Irving G, et al. Accelerating large language model decoding with speculative sampling[PP/OL]. V1. arXiv (2023-02-02)[2025-10-10]. https://doi.org/10.48550/arXiv.2302.01318.

[166] Leviathan Y, Kalman M, Matias Y. Fast inference from transformers via speculative decoding[PP/OL]. V2. arXiv (2023-05-18)[2025-10-10]. https://doi.org/10.48550/arXiv.2211.17192.

[167] Gholami A, Hooper C, Keutzer K, et al. KVQuant: Towards 10 million context length LLM inference with KV cache quantization[C]//Advances in Neural Information Processing Systems 37. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024: 1270-1303.

[168] Xiao G X, Tian Y D, Chen B D, et al. Efficient streaming

language models with attention sinks[PP/OL]. V4. arXiv (2024-04-07)[2025-10-10]. https://doi.org/10.48550/arXiv.2309.17453.

[169] Liu Z C, Desai A, Liao F S, et al. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time[PP/OL]. V2. arXiv (2023-08-28)[2025-10-10]. https://doi.org/10.48550/arXiv.2305.17118.

[170] Zhang Z Y, Sheng Y, Zhou T Y, et al. H$_2$O: Heavy-hitter oracle for efficient generative inference of large language models[PP/OL]. V3. arXiv (2023-12-18)[2025-10-10]. https://doi.org/10.48550/arXiv.2306.14048.

[171] Liu Z R, Yuan J Y, Jin H Y, et al. KIVI: A tuning-free asymmetric 2bit quantization for KV cache[PP/OL]. V2. arXiv (2024-07-25)[2025-10-10]. https://doi.org/10.48550/arXiv.2402.02750.

[172] Schuster T, Fisch A, Gupta J, et al. Confident adaptive language modeling[PP/OL]. V2. arXiv (2022-10-25)[2025-10-10]. https://doi.org/10.48550/arXiv.2207.07061.

[173] Zeng Z Q, Hong Y H, Dai H L, et al. ConsistentEE: A consistent and hardness-guided early exiting method for accelerating language models inference[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2024, 38(17): 19506-19514.

[174] Del Corro L, Del Giorno A, Agarwal S, et al. SkipDecode: Autoregressive skip decoding with batching and caching for efficient LLM inference[PP/OL]. V1. arXiv (2023-07-05)[2025-10-10]. https://doi.org/10.48550/arXiv.2307.02628.

[175] Dao T. FlashAttention-2: Faster attention with better parallelism and work partitioning[PP/OL]. V1. arXiv (2023-07-17)[2025-10-10]. https://doi.org/10.48550/arXiv.2307.08691.

[176] Bikshandi G, Dao T, Ramani P, et al. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision[C]//Advances in Neural Information Processing Systems 37. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024: 68658-68685.

[177] Xia H J, Zheng Z, Li Y C, et al. Flash-LLM: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity[PP/OL]. V1. arXiv (2023-09-19) [2025-10-10]. https://doi. org/10.48550/arXiv.2309.10285.

[178] Sheng Y, Zheng L M, Yuan B H, et al. FlexGen: High-throughput generative inference of large language models with a single GPU[PP/OL]. V2. arXiv (2023-06-12)[2025-10-10]. https://doi.org/10.48550/arXiv.2303.06865.

[179] Xue Z L, Song Y X, Mi Z Y, et al. PowerInfer-2: Fast large language model inference on a smartphone[PP/OL]. V3. arXiv (2024-12-12)[2025-10-10]. https://doi.org/10.48550/arXiv.2406.06282.

[180] Liu L, Yang X Y, Shen Y, et al. Think-in-memory: Recalling and post-thinking enable LLMs with long-term memory[PP/OL]. V1. arXiv (2023-11-15) [2025-10-10]. https://doi.org/10.48550/arXiv.2311.08719.

[181] Brown T B, Mann B, Ryder N, et al. Language models are few-shot learners[PP/OL]. V4. arXiv (2020-07-22)[2025-10-10]. https://doi.org/10.48550/arXiv.2005.14165.

[182] Devlin J, Chang M W, Lee K, et al. BERT: Pre-training of deep bidirectional transformers for language understanding[C]//Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Kerrville: Association for Computational Linguistics, 2019: 4171-4186.

[183] Bai Y T, Jones A, Ndousse K, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback[PP/OL]. V1. arXiv (2022-04-12)[2025-10-10]. https://doi.org/10.48550/arXiv.2204.05862.

[184] Ouyang L, Wu J, Xu J, et al. Training language models to follow instructions with human feedback[PP/OL]. V1. arXiv (2022-03-04)[2025-10-10]. https://doi.org/10.48550/arXiv.2203.02155.

[185] Chen T X, Tan Z T, Gong T, et al. Llama SLayer 8B: Shallow layers hold the key to knowledge injection[PP/OL]. V1. arXiv (2024-10-03)[2025-10-10]. https://doi.org/10.48550/arXiv.2410.02330.

[186] Hu E J, Shen Y L, Wallis P, et al. LoRA: Low-rank adaptation of large language models[PP/OL]. V2. arXiv (2021-10-16)[2025-10-10]. https://doi.org/10.48550/arXiv.2106.09685.

[187] Mitchell E, Lin C, Bosselut A, et al. Memory-based model editing at scale[PP/OL]. V1. arXiv (2022-06-13)[2025-10-10]. https://doi.org/10.48550/arXiv.2206.06520.

[188] Dong Q X, Dai D M, Song Y F, et al. Calibrating factual knowledge in pretrained language models[PP/OL]. V2. arXiv (2022-10-18)[2025-10-10]. https://doi.org/10.48550/arXiv.2210.03329.

[189] Cheng X, Lin Y K, Chen X Y, et al. Decouple knowledge from parameters for plug-and-play language modeling[PP/OL]. V2. arXiv (2023-09-18)[2025-10-10]. https://doi.org/10.48550/arXiv.2305.11564.

[190] Hartvigsen T, Sankaranarayanan S, Palangi H, et al. Aging with GRACE: Lifelong model editing with discrete key-value adaptors[PP/OL]. V5. arXiv (2023-10-18)[2025-10-10]. https://doi.org/10.48550/arXiv.2211.11031.

[191] Chen H, Pasunuru R, Weston J, et al. Walking down the memory maze: Beyond context limit through interactive

reading[PP/OL]. V1. arXiv (2023-10-08) [2025-10-10]. https://doi.org/10.48550/arXiv.2310.05029.

[192] Edge D, Trinh H, Cheng N, et al. From local to global: A graph RAG approach to query-focused summarization [PP/OL]. V2. arXiv (2025-02-19) [2025-10-10]. https://doi.org/10.48550/arXiv.2404.16130.

[193] Guo Z R, Xia L H, Yu Y H, et al. LightRAG: Simple and fast retrieval-augmented generation[PP/OL]. V3. arXiv (2025-04-28)[2025-10-10]. https://doi.org/10.48550/arXiv.2410.05779.

[194] Rasmussen P, Paliychuk P, Beauvais T, et al. Zep: A temporal knowledge graph architecture for agent memory[PP/OL]. V1. arXiv (2025-01-20)[2025-10-10]. https://doi.org/10.48550/arXiv.2501.13956.

[195] Xu W J, Liang Z J, Mei K, et al. A-MEM: Agentic memory for LLM agents[PP/OL]. V11. arXiv (2025-10-08)[2025-10-10]. https://doi.org/10.48550/arXiv.2502.12110.

[196] Ocker F, Deigmöller J, Smirnov P, et al. A grounded memory system for smart personal assistants[PP/OL]. V1. arXiv (2025-05-09)[2025-10-10]. https://doi.org/10.48550/arXiv.2505.06328.

[197] Liang X, Niu S M, li Z Y, et al. Empowering large language models to set up a knowledge retrieval indexer via self-learning[PP/OL]. V1. arXiv (2024-05-27)[2025-10-10]. https://doi.org/10.48550/arXiv.2405.16933.

[198] Yang H K, Lin Z H, Wang W J, et al. Memory^3: Language modeling with explicit memory[PP/OL]. V1. arXiv (2024-07-01) [2015-10-10]. https://doi. org/10.48550/arXiv.2407.01178.

[199] Park J S, O´Brien J, Cai C J, et al. Generative agents: Interactive simulacra of human behavior[C]//Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology. New York: ACM, 2023: 1-22.

[200] Sun H T, Zhuang Y C, Kong L K, et al. AdaPlanner: Adaptive planning from feedback with language models [PP/OL]. V1. arXiv (2023-05-26) [2025-10-10]. https://doi.org/10.48550/arXiv.2305.16653.

[201] Wang G Z, Xie Y Q, Jiang Y F, et al. Voyager: An open-ended embodied agent with large language models[PP/OL]. V2. arXiv (2023-10-19) [2025-10-10]. https://doi. org/10.48550/arXiv.2305.16291.

[202] Wu J L, Zhong X R, Sun J S, et al. Structure-R1: Dynamically leveraging structural knowledge in LLM reasoning through reinforcement learning[PP/OL]. V1. arXiv (2025-10-16) [2025-10-20]. https://doi. org/10.48550/arXiv.2510.15191.

[203] Wang W Z, Dong L, Cheng H, et al. Augmenting language models with long-term memory[PP/OL]. V1. arXiv (2023-06-12) [2025-10-10]. https://doi. org/10.48550/arXiv.2306.07174.

[204] Huang L, Lan H Z, Sun Z J, et al. Emotional RAG: Enhancing role-playing agents through emotional retrieval[C]// 2024 IEEE International Conference on Knowledge Graph. Piscataway: IEEE, 2024: 120-127.

[205] Wang Y, Gao Y F, Chen X S, et al. MEMORYLLM: Towards self-updatable large language models[PP/OL]. V2. arXiv (2024-05-26)[2025-10-10]. https://doi.org/10.48550/arXiv.2402.04624.

[206] Shinn N, Cassano F, Berman E, et al. Reflexion: Language agents with verbal reinforcement learning[PP/OL]. V4. arXiv (2023-10-10)[2025-10-10]. https://doi.org/10.48550/arXiv.2303.11366.

[207] Zhong W J, Guo L H, Gao Q Q, et al. MemoryBank: Enhancing large language models with long-term memory[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2024, 38(17): 19724-19731.

[208] Kang J Z, Ji M M, Zhao Z, et al. Memory OS of AI agent[PP/OL]. V1. arXiv (2025-05-30) [2025-10-10]. https://doi.org/10.48550/arXiv.2506.06326.

[209] Li Z Y, Xi C Y, Li C Y, et al. MemOS: A memory OS for AI system[PP/OL]. V3. arXiv (2025-08-05)[2025-10-10]. https://doi.org/10.48550/arXiv.2507.03724.

[210] Wu Q Y, Bansal G, Zhang J Y, et al. AutoGen: Enabling next-gen LLM applications *via* multi-agent conversation [PP/OL]. V2. arXiv (2023-10-03) [2025-10-10]. https://doi.org/10.48550/arXiv.2308.08155.

[211] Wang Y, Krotov D, Hu Y Z, et al. M+: Extending MemoryLLM with scalable long-term memory[PP/OL]. V2. arXiv (2025-05-30)[2025-10-10]. https://doi.org/10.48550/arXiv.2502.00592.

[212] Zhang Z Y, Dai Q Y, Bo X H, et al. A survey on the memory mechanism of large language model-based agents[J]. ACM Transactions on Information Systems, 2025, 43(6): 1-47.

[213] Zeng R H, Fang J Y, Liu S W, et al. On the structural memory of LLM agents[PP/OL]. V1. arXiv (2024-12-17) [2025-10-10]. https://doi.org/10.48550/arXiv.2412.15266.

[214] Tang X R, Hu T Y, Ye M Y, et al. ChemAgent: Self-updating library in large language models improves chemical reasoning[PP/OL]. V1. arXiv (2025-01-11)[2025-10-10]. https://doi.org/10.48550/arXiv.2501.06590.

[215] Zhang Z Y, Dai Q Y, Chen X, et al. MemEngine: A unified and modular library for developing advanced memory of LLM-based agents[C]//Proceedings of the ACM on Web Conference 2025. New York: ACM, 2025: 821-824.

[216] Miller M. Trends, challenge, and shifts in software vulnerability mitigation[C]//BlueHat IL Conference. Tel

Aviv: Microsoft, 2019: 10.
[217] Klabnik S, Nichols C, Krycho C. The rust programming language[M]. San Francisco: No Starch Press, 2018.
[218] Li S W, Sato H. W-kernel: An OS kernel architecture designed with isolation and customizability[C]//Proceedings of the 2023 5th International Conference on Software Engineering and Development. New York: ACM, 2024: 42-50.
[219] Gu R, Shao Z, Chen H, et al. CertiKOS: An extensible architecture for building certified concurrent OS kernels[C]// 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). Berkeley: USENIX Association, 2016: 653-669.
[220] Agache A, Brooker M, Iordache A, et al. Firecracker: Lightweight virtualization for serverless applications[C]// 17th USENIX symposium on networked systems design and implementation (NSDI 20). Berkeley: USENIX Association, 2020: 419-434.
[221] Liguori A. The nitro project-next generation aws infrastructure [EB/OL]. [2025-10-10]. https://www.old.hotchips.org/hc31/HC31_T1_AWS_Nitro_Hot_Chips_20190818-2.pdf.
[222] Amazon Web Services. Bottlerocket[EB/OL]. [2025-08-18]. https://bottlerocket.dev/en/.
[223] Android Open Source Project. Android virtualization framework[EB/OL]. (2025-03-04) [2025-08-19]. https://source.android.com/docs/core/virtualization.
[224] Li Z F, Narayanan V, Chen X D, et al. Rust for linux: Understanding the security impact of rust in the linux kernel[C]//2024 Annual Computer Security Applications Conference. Piscataway: IEEE, 2024: 548-562.
[225] Levy A, Campbell B, Ghena B, et al. The tock embedded operating system[C]//Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems. New York: ACM, 2017: 1-2.
[226] Levy A, Campbell B, Ghena B, et al. Multiprogramming a 64kB computer safely and efficiently[C]//Proceedings of the 26th Symposium on Operating Systems Principles. New York: ACM, 2017: 234-251.
[227] Redox Devel Opers. Redox OS[EB/OL]. (2024-05-31) [2025-08-26]. https://www.redox-os.org/.
[228] The Rustc Dev Guide Contributors. The MIR (Mid-level IR) [EB/OL]. (2024-08-15) [2025-08-26]. https://rustc-dev-guide.rust-lang.org/mir/index.html.
[229] Astrauskas V, Müller P, Poli F, et al. Leveraging rust types for modular specification and verification[J]. Proceedings of the ACM on Programming Languages, 2019, 3: 1-30.
[230] VanHattum A, Schwartz-Narbonne D, Chong N, et al. Verifying dynamic trait objects in rust[C]//2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice. Piscataway: IEEE, 2022: 321-330.
[231] Lattuada A, Hance T, Cho C, et al. Verus: Verifying rust programs using linear ghost types[J]. Proceedings of the ACM on Programming Languages, 2023, 7(OOPSLA1): 286-315.
[232] Lattuada A, Hance T, Bosamiya J, et al. Verus: A practical foundation for systems verification[C]//Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles. New York: ACM, 2024: 438-454.
[233] Klein G, Elphinstone K, Heiser G, et al. seL4: Formal verification of an OS kernel[C]//Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. New York: ACM, 2009: 207-220.
[234] Nipkow T, Wenzel M, Paulson L C. Isabelle/HOL[M]. Berlin: Springer, 2002.
[235] Gu L, Vaynberg A, Ford B, et al. CertiKOS: A certified kernel for secure cloud computing[C]//Proceedings of the Second Asia-Pacific Workshop on Systems. New York: ACM, 2011: 1-5.
[236] Paulin-Mohring C. Inductive definitions in the system Coq rules and properties[M]//Typed Lambda Calculi and Applications. Berlin/Heidelberg: Springer-Verlag, 2006: 328-345.
[237] Nelson L, Sigurbjarnarson H, Zhang K Y, et al. Hyperkernel: Push-button verification of an OS kernel[C]//Proceedings of the 26th Symposium on Operating Systems Principles. New York: ACM, 2017: 252-269.
[238] de Moura L, Bjørner N. Z3: An efficient SMT solver[M]// Tools and Algorithms for the Construction and Analysis of Systems. Berlin, HeidelbergSpringer2008: 337-340.
[239] Li S W, Li X, Gu R, et al. Formally verified memory protection for a commodity multiprocessor hypervisor[C]// 30th USENIX Security Symposium (USENIX Security 21). Berkeley: USENIX Association, 2021: 3953-3970.
[240] Li S W, Li X P, Gu R H, et al. A secure and formally verified linux KVM hypervisor[C]//2021 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2021: 1782-1799.
[241] Bai Y K, Li P N, Huang Y B, et al. HyperTEE: A decoupled TEE architecture with secure enclave management[C]//2024 57th IEEE/ACM International Symposium on Microarchitecture. Piscataway: IEEE, 2024: 105-120.
[242] Ling Z, Yan H Y, Shao X H, et al. Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes[J]. Journal of Systems Architecture, 2021, 119: 102240.
[243] Jiang J H, Chen C L, Feng S L, et al. Embodied intelligence: The key to unblocking generalized artificial intel-

ligence[PP/OL]. V1. arXiv (2025-05-11) [2025-10-10]. https://doi.org/10.48550/arXiv.2505.06897.

[244] Kuipers B, Feigenbaum E A, Hart P E, et al. Shakey: From conception to history[J]. AI Magazine, 2017, 38(1): 88-103.

[245] Fikes R E, Nilsson N J. Strips: A new approach to the application of theorem proving to problem solving[J]. Artificial Intelligence, 1971, 2(3/4): 189-208.

[246] Brooks R. A robust layered control system for a mobile robot[J]. IEEE Journal on Robotics and Automation, 1986, 2(1): 14-23.

[247] Gat E. Three-layer architectures[C]//Artificial Intelligence and Mobile Robots. New York: ACM, 1998: 195-210.

[248] Metta G, Fitzpatrick P, Natale L. YARP: Yet another robot platform[J]. International Journal of Advanced Robotic Systems, 2006, 3: 8.

[249] Quigley M, Gerkey B, Conley K, et al. ROS: An open-source Robot Operating System[EB/OL]. [2025-10-10]. http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf.

[250] Macenski S, Foote T, Gerkey B, et al. Robot Operating System 2: Design, architecture, and uses in the wild[J]. Science Robotics, 2022, 7(66): eabm6074.

[251] Kim D S, Tran-Dang H. An overview on industrial Internet of Things[M]//Industrial Sensors and Controls in Communication Networks: From Wired Technologies to Cloud Computing and the Internet of Things. Cham: Springer International Publishing, 2019: : 207-216.

[252] Hooda R, Kumar A, Shivani, et al. Industrial Internet of Things: An analysis of emergence, component and challenges[J]. International Journal on Recent and Innovation Trends in Computing and Communication, 2023, 11(10): 2010-2017.

[253] Mishra S, Tyagi A K, Maity I, et al. Design and implementation of internet of everything's business platform ecosystem[EB/OL]. [2025-10-10]. https://d1wqtxts1xzle7. cloudfront. net/104128678/207d8ad0-be85-4864-8fb2-d286176b3074-libre.pdf.

[254] Abunahla H, Gadhafi R, Mohammad B, et al. Integrated graphene oxide resistive element in tunable RF filters[J]. Scientific Reports, 2020, 10: 13128.

[255] Krutwig M C. Suitability of OPC UA for distributed energy monitoring[J]. Proceedings of the International Conference on Business Excellence, 2019, 13(1): 399-410.

[256] Sufian A T, Abdullah B M, Ateeq M, et al. Six-gear roadmap towards the smart factory[J]. Applied Sciences, 2021, 11(8): 3568.

[257] Xing Z H, Lan Y Q, Sun Z J, et al. IoT OS platform: Software infrastructure for next-gen industrial IoT[J]. Applied Sciences, 2024, 14(13): 5370.

[258] Cao D G, Xue D L, Ma Z Y, et al. XiUOS: An open-source ubiquitous operating system for industrial Internet of Things[J]. Science China Information Sciences, 2022, 65: 117101.

[259] Krajci I, Cummings D. History and evolution of the Android OS[M]//Android on x86: An Introduction to Optimizing for Intel Architecture. Berkeley, CA: Apress, 2013: 1-8.

[260] Pan S D, Guo T C, Zhang L H, et al. A large-scale investigation of semantically incompatible APIs behind compatibility issues in Android apps[PP/OL]. V2. arXiv (2024-06-26)[2025-10-10]. https://doi.org/10.48550/arXiv.2406.17431.

[261] Shokouh J. Detecting GNSS attacks on smartphones[EB/OL]. (2013-07-31)[2025-10-10]. https://www.diva-portal.org/smash/record.jsf pid=diva2%3A678010&dswid=-5192.

[262] Yuan J L, Yang C, Cai D Q, et al. Mobile foundation model as firmware[C]//Proceedings of the 30th Annual International Conference on Mobile Computing and Networking. New York: ACM, 2024: 279-295.

[263] Bhimanapati V B R, Goel D P, Aggarwal A. Integrating cloud services with mobile applications for seamless user experience[J]. Darpan International Research Analysis, 2024, 12(3): 252-268.

[264] Atzori L, Girau R, Martis S, et al. A SIoT-aware approach to the resource management issue in mobile crowdsensing[C]//2017 20th Conference on Innovations in Clouds, Internet and Networks. Piscataway: IEEE, 2017: 232-237.

[265] Cheng P, Lian X, Jian X, et al. FROG: A fast and reliable crowdsourcing framework[J]. IEEE Transactions on Knowledge and Data Engineering, 2019, 31(5): 894-908.

[266] Diniz H B M, Silva E C G F, Nogueira T C C, et al. A reference architecture for mobile crowdsensing platforms[C]//Proceedings of the 18th International Conference on Enterprise Information Systems. SCITEPRESS - Science and and Technology Publications, 2016: 600-607.

[267] Liu Y M, Yu Z W, Guo B, et al. CrowdOS: A ubiquitous operating system for crowdsourcing and mobile crowd sensing[J]. IEEE Transactions on Mobile Computing, 2022, 21(3): 878-894.

[268] Kuenzer S, Bădoiu V A, Lefeuvre H, et al. Unikraft: Fast, specialized unikernels the easy way[C]//Proceedings of the Sixteenth European Conference on Computer Systems. New York: ACM, 2021: 376-394.

[269] Madhavapeddy A, Mortier R, Rotsos C, et al. Unikernels: Library operating systems for the cloud[J]. ACM SIGARCH Computer Architecture News, 2013, 41(1): 461-472.

[270] Boos K, Liyanage N, Ijaz R, et al. Theseus: an experiment in operating system structure and state management[C]//14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, 2020: 1-19.

[271] 刘伟超, 王国柱, 黄琛, 等. 操作系统的多内核适配装置、方法及工业物联网操作系统: CN202410248575.9[P]. 2024-07-16.
Liu Weichao, Wang Guozhu, Chen Hhang, et al. Multi-kernel adaptation device, method for operating system and industrial internet of things operating system: CN117827277B[P]. 2024-07-16. (in Chinese)

[272] 沈昌祥. 自主可信计算筑牢人工智能安全底座[J]. 中国科技产业, 2025(1): 5.
Shen Changxiang. Independent trusted computing to build a safe base of artificial intelligence[J]. Science & Technology Industry of China, 2025(1): 5. (in Chinese)

作者简介

吴庆波　男,1969年4月出生于浙江省宁波市。中国电子信息产业集团有限公司、麒麟软件有限公司首席科学家,中国电子信息产业集团春天研究院院长,天津市操作系统重点实验室主任、研究员。主要研究方向为国产操作系统的研制和推广应用。
E-mail: wuqingbo@kylinos.cn

伍复慧　男,1987年2月出生于江西省赣州市。现为武汉学院信息工程学院副教授。主要研究方向为人工智能、操作系统。
E-mail: fuhui.wu@whxy.edu.cn

刘海天　男,1994年12月出生于湖南省涟源市。现为麒麟软件高级研发工程师。主要研究方向为人工智能、操作系统。
E-mail: liuhaitian@kylinos.cn

刘　军　男,1986年2月出生于江西省新干县。现为麒麟软件2030实验室科研规划部总经理、CCF泛在操作系统开放社区技术委员会委员。主要研究方向为人工智能、操作系统。
E-mail: liujun@kylinos.cn

李英俊　男,1988年8月出生于湖南省涟源市。现为麒麟软件有限公司2030实验室x内核研发部经理。主要研究方向为操作系统。
E-mail: liyingjun@kylinos.cn

农俊康　男,1986年11月出生于广西壮族自治区百色市。现为麒麟软件有限公司高级研发工程师。主要研究方向为虚拟化、操作系统。
E-mail: nongjunkang@kylinos.cn

杜宇琪　男,2004年2月出生于江西省萍乡市。现为麒麟软件有限公司2030实验室助理研发工程师。主要研究方向为机密计算。
E-mail: duyuqi@kylinos.cn

吴昊泽　男,2000年5月出生于甘肃省兰州市。现为麒麟软件有限公司2030实验室助理研发工程师。主要研究方向为操作系统。
E-mail: wuhaoze@kylinos.cn

陈龙腾　男,2000年11月出生于湖南省涟源市。毕业于南京航空航天大学。现为麒麟软件有限公司助理工程师。主要研究方向为操作系统。
E-mail: chenlongteng@kylinos.cn

王　静　女,1986年10月出生于湖南省常德市。现为国防科技大学计算机学院助理研究员,国家重点研发计划项目课题负责人,开源发展技术委员会执行委员。主要研究方向为操作系统、开源生态治理。
E-mail: wangjing@nudt.edu.cn