计算机工程与科学
Computer Engineering & Science
ISSN 1007-130X,CN 43-1258/TP

《计算机工程与科学》网络首发论文

题目：　基于机器学习方法的操作系统优化研究综述
作者：　黄卓懿，彭龙，徐浩，李琢，刘晓东，刘敏，余杰
网络首发日期：　2025-09-12
引用格式：　黄卓懿，彭龙，徐浩，李琢，刘晓东，刘敏，余杰．基于机器学习方法的操作系统优化研究综述[J/OL]．计算机工程与科学.
https://link.cnki.net/urlid/43.1258.tp.20250911.1637.005

网络首发时间：2025-09-12 13:15:21　网络首发地址：https://link.cnki.net/urlid/43.1258.tp.20250911.1637.005

基于机器学习方法的操作系统优化研究综述

黄卓懿[1]，彭 龙[1]，徐 浩[1]，李 琢[1]，刘晓东[1]，刘 敏[2]，余杰[1]

（1.国防科技大学计算机学院，湖南省 长沙市 410000；

2.麒麟软件有限公司，湖南省 长沙市 410000）

摘　要：随着计算机硬件与应用的快速发展，现代操作系统在硬件资源管理方面面临巨大挑战。机器学习为操作系统优化提供了创新解决方案。综述基于机器学习的操作系统资源管理方法，总结机器学习模型在操作系统中的具体应用案例，分析其优势与局限性，并探讨当前研究面临的挑战。最后对未来研究方向及应用前景进行展望.

关键词：操作系统；机器学习；进程调度；内存管理；

中图分类号：TP316

A Review of Operating System Optimization Research Based on Machine Learning Methods

HUANG Zhuoyi[1], PENG Long[1], XU Hao[1], LI Zhuo[1], LIU Xiaodong[1], LIU Min[2], YU Jie[1]

(1.College of Computer Science and Technology, National University of Defense Technology, Changsha 410000;

2.KylinSoft Corporation, Changsha 410000, China)

Abstract: The rapid advancement of computer hardware and software applications poses significant challenges to modern operating systems (OS) in terms of efficient resource management. Increasing structural complexity and frequent updates in both software and hardware necessitate more intelligent and adaptive management strategies. Machine learning (ML) provides innovative solutions to address these issues. Existing research is examined to offer a systematic overview of ML-based methods for optimizing operating systems. Surveyed studies include ML-driven OS optimization techniques, concrete application cases of ML models across various OS contexts, and discussions of their strengths and limitations. Furthermore, ongoing challenges are identified, and promising future research directions are suggested, highlighting potential applications of machine learning in operating systems for enhanced resource management.

Key words: Operating System; Machine Learning; Process Scheduling; Memory Management;

1　引言

随着计算机硬件和应用程序的快速发展，现代操作系统在有效管理硬件资源方面遭遇了前所未有的挑战[1]。操作系统的发展呈现出两个主要趋势：一方面，操作系统结构日益复杂化，随着硬件种类的增加，需要支持多核处理器、GPU、FPGA、NVMe 存储等异构硬件资源的管理和调度，增加了设计和实现的复杂性；另一方面，硬件和应用软件迭代速度的加快对操作系统提出了更高的适应性要求。这种趋势推动了操作系统设计的模块化、可扩展性和动态更新能力的发展，同时也为智能化技术的应用提供了契机。通过机器学习等智能化技术，操作系统能够实现资源的自适应管理、性能的动态优化，从而更好地应对

基金项目：国家重点研发计划（2024YFB4506200）

通信作者：彭龙（penglong@nudt.edu.cn），余杰（yj@nudt.edu.cn）

通信地址：410073 湖南省长沙市国防科技大学计算机学院

Address: College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, Hunan, P.R.China

复杂硬件环境和快速迭代的挑战。

操作系统的智能化是未来发展的关键趋势之一，它是人工智能技术与操作系统领域交汇融合的产物。目前，一些研究已经成功地运用机器学习方法对操作系统的部分功能进行了优化和改进。例如，Maas 和 Andersen 等人[6]开发的 Llama 新型内存管理器，采用长短期记忆循环神经网络(LSTM)预测内存对象的寿命，通过预测到的信息合理分配内存，进而减少了高达 78%的内存碎片，提高了 C++服务器内存分配和管理的效率。Hao 等人[7]通过在操作系统中内置轻量级神经网络，预测 I/O 队列中 I/O 操作的快慢，并根据预测结果对 I/O 操作进行提交或撤销。与现有的启发式方法相比，减少了 9.6%~79.6%的平均 I/O 延迟，推理准确率达到 87%~97%，并且每个 I/O 的推理开销仅为 4~6 微秒。这些研究结果表明，将机器学习技术引入操作系统能够有效提升系统性能。

然而，目前基于机器学习优化操作系统的方法也存在一些问题：

1.尽管已有相当一部分学者使用机器学习方法对操作系统进行了优化，但总体的研究数量仍然较少，且比较分散。大部分研究仅对某个单一的内核子系统进行了优化，而并没有结合多个子系统或从操作系统整体进行优化，当然这么做的难度也非常大，还需要很长一段时间的研究和探索。

2.在利用机器学习方法优化操作系统时，机器学习模型在用户态运行和在内核态运行各有各的优缺点，但大部分研究人员认为从内核层面结合机器学习才是未来智能化操作系统的主要趋势。而要在内核层面运用机器学习技术，面临的一大难题就是机器学习框架。目前广泛应用的机器学习框架，如 Tensorflow、Pytorch 等，占用空间太大，无法在空间有限的内核态下运行。因此要在内核空间运用机器学习技术需要一个可以直接在内核中使用的轻量级机器学习框架，而目前相关的研究成果还比较少。

3.机器学习模型具有决策不确定性和不可解释性，在操作系统中引入机器学习技术，在给系统带来性能提升的同时也带来了安全风险。解决模型的安全性验证问题，是智能化操作系统必不可少的一个环节。

综上所述，机器学习技术在操作系统中的应用前景广阔，但同时也面临着诸多挑战，需要进一步的研究和探索。

本文的贡献主要体现在以下几个方面：

1.系统化总结。本文系统地梳理了近年来将机器学习技术应用于操作系统的研究成果，并将这些成果按照操作系统的不同内核子系统进行了分类与归纳。

2. 识别主要挑战。本文分析了现有研究中面临的主要挑战，包括决策时效性与模型深度、模型开销与优化收益以及安全缺陷与加固机制。

3. 展望未来研究方向。本文展望了未来的研究方向，包括开发轻量级智能架构、面向全局的智能协同技术以及从被动防御到主动免疫的安全验证架构。

如图 1 所示，本文首先介绍机器学习技术在操作系统中的重要性和潜力。接着，第 2 节按操作系统的子系统，将基于机器学习方法的操作系统性能优化技术划分为智能化进程调度、智能化内存管理技术、基于机器学习的存储系统性能优化技术、通用的内核机器学习框架四个主要研究方向。第 3 节对当前研究中存在的关键技术挑战进行总结。最后，第 4 节对未来的研究方向进行展望。

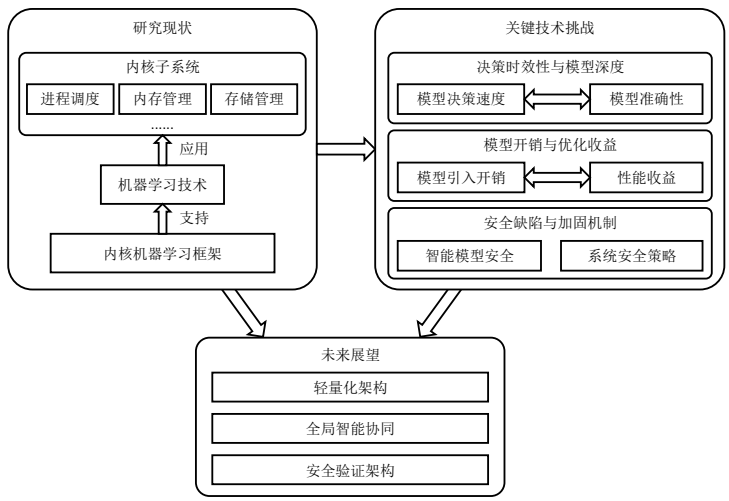

Figure 1 The architecture of this article

图 1 本文框架脉络

2 国内外研究进展

目前，操作系统中使用的大部分决策算法都是基于固定规则的启发式算法，例如最近最少使用、先进先出等。启发式适用于预先定义的情况，无法自动适应复杂的、未见过的数据和场景，此外调整启发式算法需要专业的知识和长时间的手动调整。启发式方法虽然简单、快速，并且在某些已知且稳定的环境下表现良好，但在动态适应性、泛化能力以及复杂问题处理方面的能力较弱。而采用机器学习方法可以使操作系统动态适应不断变化的工作负载和环境条件，减少人工干预的需求。因此，在内核中使用机器学习技术，可以弥补启发式方法的不足，并提供一种更先进、更灵活的方法来解决操作系统中的复杂问题[2,3,4]。

目前，已有相当一部分学者利用机器学习等智能化技术对操作系统进行了性能改善的研究，有利用机器学习技术解决某个子系统的性能问题的[5,6,7]，也有开发内核通用的机器学习架构的[8,9,10]，这些框架适用于进程调度、内存管理、文件系统、网络系统等各个子系统。

如表 1 所示，按操作系统的子系统将其划分为智能化进程调度、智能化内存管理、智能化外存性能优化、通用的内核机器学习架构四个主要研究方向。

Table 1 Classification of ML-based Operating System Optimization Techniques

表 1 基于机器学习的操作系统性能优化技术分类

智能化进程调度	资源感知的进程调度
	改善任务突发的进程调度
智能化内存管理	内存碎片管理
	多级内存管理
	内存预取
智能化外存性能优化	性能参数预测
	最优配置预测
	智能化存储预取
内核机器学习框架	KML
	LAKE
	RMT

2.1 智能化进程调度

进程是操作系统进行资源分配和调度的基本单元，是应用程序运行的实例，拥有独立的内存空间。线程是进程的执行单元，也被称为轻量级进程。而操作系统的进程调度子系统，则负责管理和调度进程或线程对 CPU 资源的使用。它需要确保多个进程能够高效、公平地共享处理器资源，使进程能够及时响应。

当前进程调度面临多重性能瓶颈：其一，传统调度器将 CPU 大/小核、GPU、DPU、NPU 等异构硬件单元视为同质资源，无法适配硬件特性差异，导致能效与性能双重损失；其二，多核/众核系统中任务争抢末级缓存、内存带宽、PCIe 通道等共享资源，引发严重的性能劣化；其三，在云原生及 Serverless 场景下，任务到达率呈脉冲式爆发，如秒杀活动、微服务调用链雪崩，静态资源配额机制难以快速响应突发负载。面对日益复杂的应用场景，传统时间片轮转与优先级调度等方法已难以突破性能瓶颈。在此背景下，机器学习技术为进程调度的性能优化提供了新范式[11]。

目前进程调度方面的主要挑战集中异构硬件资源管理困难、多任务资源竞争以及突发负载。机器学习技术的识别和预测能力与操作系统进程调度方案相结合后，可以使进程调度提高资源感知和应对任务突发的能力。

2.1.1 资源感知的进程调度

传统的进程调度决策方案包括先来先服务、最短作业优先、最短剩余时间优先、轮转调度、完全公平调度(CFS)等调度策略。目前 Linux 的主流调度策略是 CFS 算法，它执行一个公平的调度策略，旨在让所有任务都能获得公平的处理时间配额，广泛应用于通用计算工作负载。然而 CFS 只关注 CPU 时间的公平分配，没有考虑到硬件资源的竞争和利用情况，以及缺乏对动态上下文信息的利用[5]。

为了解决 CFS 在处理硬件资源竞争时的局限性，[5]提出了一个基于机器学习的资源感知负载均衡器，用于改进服务器端 Linux 内核中的负载均衡问题。该负载均衡器采用多层感知机(MLP)模型，其中输入层有 15 个输入特征，隐藏层包含 10 个节点，以及一个输出层。MLP 模型通过分析 CFS 运行时收集的数据，学习进程的 CPU 利用率和其他硬件资源使用模式，然后基于这些学习到的模式来预测哪些进程应该在不同 CPU 核心之间迁移，以实现更优的负载分配。这个模型模仿 CFS 的调度模式，并在此基础上输入 CPU 使用时间、内存访问模式、缓存命中率等反映硬件资源使用情况的特征对 MLP 模型进行训练，使负载均衡器具有资源感知的能力。实验结果表明，基于机器学习的负载均衡器在做出进程调度决策时的准确率高达 99%，并且仅将延迟增加了 1.9 微秒，对系统延迟的影响很小。此外，通过测试，该方法在 5 分钟内平均 CPU 负载为 1.773，与 CFS 的 1.758 相比有所提高，在服务器端即使是微小的提升也具有重要意义。

尽管在内核中使用机器学习模型会带来一些额外的开销，但这些开销并不会显著影响系统的整体性能。[5]中利用 MLP 机器学习模型模仿 CFS 调度器，使 MLP 能够准确地做出 CFS 的决策结果，并用 MLP 调度器替换无法感知资源利用情况的 CFS。这种方法将“白盒”的决策调度器巧妙地替换为了同等效果的“黑盒”机器学习模型，并赋予了资源感知的新特性，是进程调度决策与机器学习结合的一种可行方法。

[14]结合了静态知识库和自适应的强化学习，使用静态知识库(基于历史数据训练的决策树)来为新进程分配初始时间片，并通过自适应的强化学习模块根据运行经验来调整和优化时间片。文章通过这种方式改善操作系统调度器的性能，使得多个进程能够更高效地共享 CPU 资源，减少了因频繁切换进程而产生的开销，提高了系统整体的吞吐量。

2.1.2 改善任务突发的进程调度

随着服务器功能复杂化，任务数量秒级爆发，传统启发式调度算法面临复杂性和不确定性的问题，应用创新技术来改善任务调度变得尤为重要。在[12]中，为了应对上述问题，作者使用强化学习(RL)技术开发了一个新的调度模型，该模型对任务调度和资源利用进行了优化。RL 调度器包含两个部分：一部分使用深度神经网络(DNN)预测任务的突发时间的类别，另一部分根据预测结果，使用深度 Q 网络模型(DQN)来生成任务执行的时间表，最大化累积奖励，最小化平均作业延迟。实验结果表明，DNN 在预测突发时间方面表现良好，大多数类别的 F1 分数接近或等于 1，显示出较高的准确性，DQN 也能够根据 DNN 的预测来优化作业调度。

上述研究证明了机器学习模型作为调度器的方法是可行的，且具有潜力的。这些成果不仅证明了机器学习在进程调度决策中的有效性，也为未来调度策略的发展提供了新的方向。

除了上述直接使用机器模型作为调度器的方法，在进程调度中一种更为常见的应用机器学习的方法是，基于进程执行的历史信息训练机器学习模型，学习进程的 CPU 利用模式，然后对 CPU 工作负载进行分类，针对不同的工作模式灵活使用不同的调度策略。

[13]使用机器学习技术对进程调度器中的参数时间片进行了优化。[13]通过分析进程的静态和动态属性，使用 C4.5 决策树、K 近邻(KNN)等机器学习算法来预测 CPU 突发时间，并根据预测结果调整进程时间片，从而最小化进程的周转时间(Turn-around-Time, TaT)。实验结果表明，预测性调度可以减少 1.4%到 5.8%的 TaT，这主要是由于减少了完成进程执行所需的上下文切换次数。C4.5 决策树算法在实验中被证明是最有效的方法，用于解决这个问题。

在传统时间片轮转方法中，时间片参数对系统性能具有重要影响，时间片既不能太长也不能太短。时间片过短会导致频繁的任务切换，使系统开销增大；时间片过长则会导致进程并发执行的效率降低。时间片的设置需要考虑开销和效率之间的平衡关系。在操作系统中，当参数会影响性能和开销之间的平衡，且随着工作负载变化，参数的最优值也随之变化时，使用机器学习技术动态调整参数是一个可行的解决办法。

目前，机器学习与进程调度的结合在服务器和云计算领域较为常见，而在桌面操作系统和移动设备上的应用则相对较少。这种现象的出现，部分原因在于服务器和云环境面临着更繁重的调度任务，并且配备了大量的 CPU 核心、存储设备和加速器等硬件资源。这些环境的工作场景也更为复杂多变，对负载均衡和 CPU 利用率等关键性能指标有着更为严格的要求。在进程调度领域，研究主要集中在两个方面：一是基于机器学习模型设计和开发全新的进程调度器；二是在保持现有调度器架构不变的情况下，利用机器学习技术动态调整进程参数。展望未来，预计会有更多机器学习技术被应用于调度器的构建中，同时，机器学习的应用可能会扩展至时间片之外的其他进程参数，这将有助于进一步提升进程调度的智能化水平和系统的整体性能。

2.2 智能化内存管理

内存管理是操作系统的一个核心组件，它负责有效地分配、监控和回收内存资源。优化内存管理是操作系统领域的关键研究方向之一，随着摩尔定律的放缓，内存访问速度的瓶颈问题变得更加突出。当前，内存优化技术主要依赖于启发式方法，广泛应用于服务器和移动设备。服务器端工作负载复杂多样，用户需求千差万别，且常常采用异构内存架构；而移动设备则受限于有限的内存空间，用户日常使用中对系统性能有着较高的期望。因此，两者都需要高效的内存管理策略。

在服务器端内存管理优化领域，研究涵盖了多种方向：包括[15]对异构内存系统内存管理改进方案的综述；[16]提出的支持 unikernel 多粒度内存分配的异构内存管理机制 UCat；[17]构建的由三种针对延迟、带宽和功耗分别优化的内存模块组成的异构内存系统；以及[29]通过存内处理技术执行数据移动密集型函数以减少数据传输、从而降低能耗并提升效率的实践。与此同时，面向移动设备的内存管理优化研究也提出了多种

方案，例如[18]的前后台应用感知页面回收方案 Acclaim、[19]的冻结后台进程方案 ICE、[20]的低内存杀死策略 LMK，以及[21]和[22]关注的内存交换技术。

然而，尽管这些方法在各自领域都取得成效，它们本质上主要依赖预先设定、静态的启发式规则。随着多级异构内存架构的普及和用户需求的日益多样化与动态化，这种静态的启发式管理方法在面对高度复杂且持续变化的工作负载模式时，其适应性和优化潜力显得愈发不足。具体表现为：预设的规则难以动态适应突发或演变的访问模式，导致内存碎片加剧，有效空间利用率下降；在管理包含不同性能特性的多级异构内存时，静态分配策略无法根据实时负载敏感度进行最优的数据放置和迁移，使得整体系统性能潜力无法充分释放，甚至可能引入不必要的开销；固定的预取策略在面对不可预测的数据流时准确率显著降低，引发大量无效的数据移动和缓存污染。内存碎片累积、多级内存管理失配、预取效率低下等由静态规则僵化性引发的具体问题，共同凸显了当前内存管理面临的关键挑战。因此，需要引入更智能、更动态的技术，以突破静态规则的局限，实现对复杂多变负载环境的自主感知、学习与优化决策。

目前，一些研究已经开始探索机器学习技术在减少内存碎片、提升异构内存管理效率和提高内存预取准确性等方面的应用，并取得了积极的效果。

2.2.1 内存碎片优化

[6]指出现代 C++服务器内存管理中的两个关键问题：内存占用随时间变化，和长时间运行的对象分配导致堆碎片问题。传统的 C++内存管理器无法移动对象，导致使用大页面时碎片问题加剧。文章提出了一种新的方法来解决大页面碎片问题，该方法结合了现代机器学习技术，开发了一个名为 Llama 的新型内存管理器。在不移动对象的情况下，Llama 实现了堆碎片的自动减少。Llama 内存管理器基于对象生命周期和大页面(被划分为块和行)来管理堆，并使用基于神经网络的语言模型预测生命周期类别。该模型学习以前运行的上下文敏感的每个分配站点的生命周期，并能泛化到不同的二进制版本，以及从样本外推到未观察到的调用上下文。Llama 使用 LSTM 模型，来学习常见和罕见的上下文，通过符号化调用上下文进行训练，以预测对象的生命周期类别。Llama 的堆不是按对象大小类别组织，而是按生命周期类别动态调整。通过预测和观察内存行为，在块粒度上动态调整生命周期类别，减少了内存碎片。实验表明，Llama 在多个生产服务器上仅使用大页面时减少了高达 78%的内存碎片。与现有的内存分配器(如 TCMalloc)相比，Llama 在不破坏大页面的同时，显著减少了碎片。

2.2.2 多级内存管理

随着硬件技术的不断进步，多级异构内存系统应运而生[23]，它们集成了动态随机存取存储器(DRAM)、静态随机存取存储器(SRAM)以及非易失性内存(NVM)等多种类型的内存模块。在多核系统中，不同应用对内存需求各异，为了满足不同内存需求的内存系统，[17]提出了一种包含三种不同内存模块的异构内存系统，每个模块对应优化延迟、带宽、功耗中的一个参数。三种异构内存模块的设计如下：

(1)延迟优化内存模块：主要优化延迟，使用 DRAM 技术。为了降低延迟，它采用了较小的行缓冲区和较小的预取缓冲区深度。此外，它还采用了 SRAM 寻址模式，这通常用于减少内存访问延迟。

(2)带宽优化内存模块：负责优化带宽，同样使用的是 DRAM 技术。为了提高带宽，它采用了较多的存储体和较大的行缓冲区大小。此外，还使用了 DRAM 多路复用寻址模式来提高有效引脚带宽，并将预取缓冲区深度设置为最大值。

(3)功耗优化内存模块：针对功耗敏感的应用，使用的是 Low-power DRAM(LPDRAM)技术。LPDRAM 通过部分阵列自刷新和温度补偿自刷新技术来降低刷新功耗。此外，它还采用了较小的存储体数量、较小的行大小、最小的预取缓冲区大小，并且选择了最低的操作电压。

[17]使用离线分析算法，根据应用的 L2 缓存未命中率和内存级并行性对应用进行分类，并按分类结果选择不同的内存模块进行存储。实验结果表明，该异构内存系统平均提升了 13.5%的系统性能，并降低了 20%的内存功耗。异构内存系统在面对不同应用需求时，可以更灵活地对页面进行放置。随着硬件技术的不断进步，异构内存系统的应用正变得越来越普遍。

由[17]的例子可以看出，多级异构内存系统的内存模块各自拥有独特的性能特性。但这也引出了性能不匹配的问题，导致传统的操作系统在面对应用程序对内存性能的多样化需求时，往往难以实现整体性能的均衡。为了应对这一挑战，文献[24]提出了利用机器学习技术，使系统能够动态预测应用程序的内存访问模式，并据此动态调整内存资源的分配策略，以解决性能不匹配问题，提高异构内存系统的性能。在异构内存系统中，混合内存页面指 DRAM、SRAM、NVM 等不同类型的内存技术组合而成的内存页面。[24]设计了一种混合内存页面调度器 Kleio，它通过

机器学习技术来优化数据中心和超算环境中大数据应用的性能。Kleio 通过训练循环神经网络(RNN)来预测页面在未来的访问情况，并根据 RNN 的预测结果，对所有页面进行访问频率的排序，优先将热页(即访问频率高的页面)分配到最快的内存技术组件中，以提高多级异构内存系统的性能。与 Oracle 页面调度器相比，Kleio 在平均性能上能够减少 80%的性能差距，显著提高了应用程序性能，同时限制了系统资源开销。这项研究表明了，机器学习技术能够有效解决异构系统的性能不均衡问题。

2.2.3 内存预取技术

内存预取技术在现代计算机系统中非常重要，尤其是在处理器速度远快于内存访问速度的情况下，预取成为了缓解内存墙问题的主要手段之一。传统的预取器基于历史数据预测未来的内存访问，并提前将数据从内存加载到较快的缓存或处理器寄存器中。从而减少处理器等待数据从内存中读取的时间，提高系统整体的计算效率和吞吐量。但随着工作负载复杂性的增加和摩尔定律的放缓，传统的预取器预测的准确性下降，需要新的方法来提高计算效率。在内存预取技术中，通常依赖于历史访问数据序列来预测未来可能访问的数据。循环神经网络，尤其是长短期记忆神经网络(LSTM)，在处理此类序列预测问题上表现出色。文献[25]、[26]和[27]中的方法便是利用 LSTM 对数据访问模式进行预测，并将预测结果用于指导数据从内存预取至缓存的决策。这些研究展示了 LSTM 在预测未来数据访问方面的有效性，为内存预取策略提供了强有力的支持。

[25]提出了两种基于 LSTM 的预取模型：Embedding LSTM 和 Clustering + LSTM，旨在通过提高预取的精度和召回率，从而提升预取性能和系统的整体能效。Embedding LSTM 模型使用输入和输出词汇表来预测内存访问模式。Clustering + LSTM 模型通过聚类地址空间来减少模型的词汇表大小和内存足迹。在 SPEC CPU2006 和 Google 的网络搜索工作负载上，LSTM 模型在预测缓存未命中方面表现出色。与标准流预取器和 GHB PC/DC 预取器相比，LSTM 模型在精确度和召回率方面表现更好。

[26]在离线状态下使用 LSTM 长短期记忆网络对缓存进行预测，并得到了比当前硬件预测器更好的准确性。接着，通过对这个 LSTM 模型进行分析，得出结论：缓存决策主要依赖于程序控制流历史中少数几个元素，并且对输入序列的顺序不敏感。基于以上结论，作者设计一个了更简单的在线模型，这个在线模型在显著降低成本的同时，能够匹配离线模型的准确性。

[27]也基于 LSTM 提出了一个名为 DeepCache 的自适应缓存机制，它能够自动学习请求流量模式的变化，并预测内容的未来流行度，从而做出缓存和驱逐的决策以最大化缓存命中率。DeepCache 由两个主要部分组成：对象特征预测器和缓存策略组件。对象特征预测器使用 LSTM 编码器-解码器模型来预测对象的流行度。缓存策略组件则利用这些预测信息来做出智能的缓存决策。

综合上述案例，可以看到内存访问模式预测是机器学习技术与内存管理结合的核心。这一点至关重要，因为随着工作负载的复杂性增加和摩尔定律的放缓，传统的基于表的预测器（例如硬件中的预取器）在处理大规模、不规则的数据中心工作负载时遇到了挑战。这些预测器的准确性往往会随着工作集的增大而显著下降。上述相关研究表明，利用机器学习技术来优化内存管理是有效的，将机器学习应用于系统层面的内存管理问题是一个有前景的新研究方向。

2.3 智能化外存性能优化

机器学习技术在存储管理领域的应用日益广泛，尤其在解决性能参数预测、最优配置预测和智能化存储预取等方面。以下是一些关键研究进展的概述。

2.3.1 性能参数预测

这里以延迟参数为例，[7]针对现代 SSD 内部复杂性导致的延迟不可预测性问题，提出了 LinnOS 操作系统。LinnOS 通过集成轻量级神经网络，实现了对每个 I/O 操作速度的预测，以提高存储性能的可预测性。主要工作包括设计一个二元推断模型并利用实际工作负载数据训练神经网络，将延迟推断问题简化为“快”或“慢”，快的数据会被直接提交，而慢的数据会进入下一个 SSD 的 I/O 队列继续等待。实验表明，LinnOS 在不同工作负载和存储设备上，相比传统方法平均减少了 9.6%~79.6%的 I/O 延迟，准确率达 87%~97%，且每 I/O 推断开销仅 4~6 微秒，验证了机器学习在操作系统中优化存储系统的有效性。

2.3.2 最优配置预测

[28]提出了一个名为 FSbrain 的智能 I/O 性能调优系统，旨在解决物联网环境中不同工作负载下文件系统性能调优的挑战。FSbrain 通过机器学习技术自动推荐合理的文件系统配置，无需手动干预或修改内核代码。文献的主要工作包括模型训练和配置调优两个阶段，首先通过岭回归识别关键参数，并基于 LSTM 构建性能预测模型；然后在配置调优阶段，利用遗传算法和探索策略在配置空间中寻找最优配置。实验结果表明，

FSbrain 推荐的配置平均能将 I/O 性能提升 1.28 倍，相较于手动调优减少了 90%的调优时间，验证了该系统的有效性和实用性。[29]提出了一种基于机器学习的方法，通过选择最合适的 RAID 配置来优化 I/O 性能，旨在解决云计算环境中 RAID(独立磁盘冗余阵列)系统 I/O 性能优化问题。研究开发了一个 RAID 预测模型，使用随机森林、决策树和 KNN 等机器学习算法进行训练和测试。实验结果表明，所提出的模型能够通过选择适合特定工作负载的最优 RAID 配置，提高整体系统性能。其中，随机森林模型在预测 RAID 级别方面表现最佳，准确率、召回率、精确率和 F1 分数等评价指标上均优于其他模型。

2.3.3 智能化存储预取技术

数据移动是现代计算机系统中造成延迟的主要原因之一[34]。数据预取技术是解决这一问题的有效方法，它通过预测未来的数据访问需求，提前将数据从较慢的存储设备加载到较快的存储层次中，从而减少访问延迟和提高系统性能。预取技术主要分为硬件预取和软件预取两大类，其中软件预取技术根据存储层次的不同，可以进一步细分为内存预取、文件页缓存预取和闪存预取等，如图 2 所示。这些预取技术旨在提升数据访问效率，通过机器学习技术的应用，可以进一步提高预取的准确性和效率，从而更有效地预测和满足未来的数据访问需求。以下是一些将机器学习技术与页缓存预取、闪存预取相结合的研究进展。

文件页缓存预取。[30]为解决固态硬盘(SSD)的 I/O 访问延迟问题，通过文件系统的页缓存预取技术预测未来的块访问，并提前将数据从 SSD 加载到主内存中，从而减少访问延迟。然而，由于 SSD 的大规模稀疏地址空间和多线程访问导致的复杂性，使得准确预测未来的 I/O 访问变得困难。作者提出了一种基于轻量级神经网络的预取方法，利用长短期记忆网络从块级 I/O 跟踪中学习应用的空间 I/O 访问模式。实验结果表明，该方法与现有基于步长的预取器相比，性能提升了高达 800 倍，与基于马尔可夫链的预取器[31]相比，性能提升了高达 8 倍。此外，通过地址映射学习技术，展示了该方法对未见过的 SSD 工作负载的适用性，并进行了超参数敏感性研究。[32]针对 Linux 内核文件子系统的页缓存预取问题，设计了基于动态工作负载的预取算法，并在微观负载和真实场景下验证了其优化效果，如在 Mail-Server 场景下吞吐量提升近 3.1 倍，File-Server 提升 1.2 倍。实验结果表明，该方法在多数场景下能取得优化效果，尤其在随机读和更新负载下性能提升显著。

闪存预取。[33]提出了一种基于机器学习技术的 DeepPrefetcher 闪存数据预取器，通过捕获块访问模式的上下文，使用 LSTM 模型进行上下文感知的预取数据预测，并将未来可能被访问的数据从 NAND 闪存芯片提前加载到 SSD 的 DRAM 缓冲区中。该框架实验结果显示，DeepPrefetcher 在平均预取准确率、覆盖率和加速比上分别比基线预取策略提高了 21.5%、19.5%和 17.2%，表现优于其他预取方案。

机器学习技术在存储管理中发挥着重要作用，通过预测性能参数、优化配置和智能化预取技术，显著提升了存储系统的性能和效率。这些研究不仅减少了 I/O 延迟，还提高了数据访问的准确性和速度，展示了机器学习在存储领域的广泛应用前景。

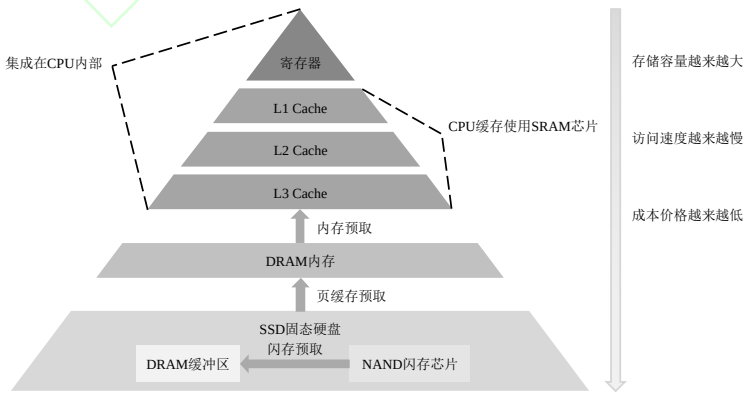

Figure 2 Hierarchy of Prefetching Techniques
图 2 预取技术的层次

2.4 内核机器学习框架

机器学习技术在操作系统内核的多个关键子系统，包括进程管理、内存管理和存储管理等领域的应用，已被证明能够有效提升系统性能。然而，由于 TensorFlow 和 PyTorch 等现有的机器学习框架体积庞大，它们并不适合直接集成到资源受限的内核环境中。目前，缺少一个能够集成到内核中的轻量级、通用的机器学习框架，以标准化各子系统的机器学习实现，从而促进子系统间的协同和交流。因此，开发一个适用于内核的通用机器学习框架变得至关重要，目前已有一些研究开始探索这一领域，为内核通用机器学习框架的发展奠定了基础[8,9]。

2.4.1 KML

[8]设计并实现了 KML 内核机器学习架构，它能够直接嵌入到操作系统中，用于动态优化存储参数。作者将其应用于两个案例研究：优化读预取(readahead)和网络文件系统(NFS)的读大小(rsize)。KML 架构支持在内核空间进行训练和推理，也支持用户空间离线训练和内核空间推理两种使用方式。这种灵活的设计允许开发者在用户空间中轻松开发和调试 ML 模型，然后将相同的模型部署到内核空间。实验效果方面，在 readahead 中，KML 提高了 I/O 吞吐量，最高可达 2.3 倍，在 NFS rsize 优化案例中，性能提升高达 15 倍。此外，KML 的资源消耗非常低，动态内核内存消耗小于 4kB，CPU 开销小于 0.2%，证明了 KML 在实际应用中的可行性和有效性。KML 能够在保持低资源消耗的同时，显著提高存储系统的 I/O 性能，验证了内核级机器学习框架在性能上的优势。

2.4.2 LAKE

[9]提出了一个名为 LAKE 的系统，它支持在内核空间使用 ML，并且可以将内核的机器学习计算映射到硬件加速器 GPU。LAKE 包括用于跨抽象层和模块边界的特征收集和管理的 API，简化了不同内核子系统中的特征收集和管理。作者将 LAKE 框架应用于内存、进程和 I/O 调度等子系统，与仅使用 CPU 的版本相比减少了 ML 模型的推理延迟时间，最高可达 96%，并提供高达 3.1 倍的推理加速。文章通过构建 LAKE 系统，展示了 ML 在操作系统内核中的应用潜力，并提供了实验验证，证明了其在提高性能和降低资源消耗方面的有效性。

2.4.3 RMT

[10]设计了一个内核内可重配置匹配表(RMT)的原型，并在页预取和 CPU 调度两个关键领域进行了实证研究。在页预取案例中（案例一），通过对 OpenCV 视频缩放和 NumPy 矩阵卷积的测试，RMT 支撑的决策树模型将预取准确率提升到最高 92.91%，任务完成时间缩短 30%；在 CPU 调度案例中（案例二），基于 PARSEC 负载的验证表明，RMT 承载的 MLP 模型以 99%的准确率复现了 Linux 调度器规则，且仅需 2 个精简特征就维持了 94%以上的决策精度。这一架构通过内核内的虚拟机实现了 RMT 的抽象，使得 RMT 能够在运行时匹配当前的执行上下文，并对由 ML 计算得到的上下文特定优化进行编码。它支持包括离线和在线学习算法在内的多种学习策略，并且可以适应不同的内核子系统。在执行方式上，该框架允许在字节码层面进行解释执行或者通过即时编译来进一步优化内核数据路径。RMT 为内核重配置提供了一个轻量级且硬件友好的解决方案，案例一和案例二共同验证了该框架在具体场景中的有效性，强调了将 ML 技术整合到操作系统内核中的巨大潜力，并展示了通过 RMT 实现这一目标所面临的挑战。

2.4.4 智能框架的性能对比

为了进一步评估内核机器学习框架的实际性能，本文从对应参考文献中提取相关的模型性能数据。具体数据如表 2，三种内核智能化框架的主要功能各不相同。KML 着重于为智能化模型提供轻量化的库支持，在保证高达 95.5%和 98.6%的准确率的情况下，全连接神经网络的初始化大小仅占 3.9kB。LAKE 主要功能是使用 GPU 加速智能化内核模块的模型推理，GPU 推理仅消耗 7.25 微秒，而 CPU 上推理则达到了 100 微秒，推理速度上提升接近 14 倍。而 RMT 更关注内核机器学习模型的硬件兼容性，KML 和 LAKE 都不支持 INT8，只有 RMT 支持。

Table 2 Performance Comparison of Intelligent Frameworks
表 2 智能化框架性能对比

框架	推理延迟(μs)	内存占用(kB)	模型大小(kB)	INT8 支持	准确率(%)	数据来源
KML	21（神经网络） 8（决策树）	(1)框架：5,500 kB (2)Readahead 模块：432 kB	(1)初始化：3.9 kB (2)推理临时：0.7 kB	否，使用 FPU 浮点运算，KML §2.4	(1)Readahead：95.5% (2)NFS rsize：98.6%	KML §4.3
LAKE	100（CPU） 7.25（GPU）	未提及	未提及	未提及	未提及	LAKE §7.1
RMT	未提及	未提及	未提及	是	92.91%（案例 1） 99%（案例 2）	RMT §4& §5.2

尽管当前内核机器学习架构的研究较少，但现有的成果证明了这一领域的研究价值和潜力，同时也揭示了它所面临的重大挑战。KML 和 LAKE 框架通过优化存储参数和利用 GPU 加速内核 ML 计算，分别在 I/O 吞吐量和推理速度上取得了显著提升。这些探索为将来开发轻量级、通用的内核 ML 框架奠定了基础。

3 关键技术挑战

随着计算范式向异构化、泛在化和智能化演进，传统基于静态规则的操作系统管理机制在应对动态负载、复杂硬件环境及多样化服务质量需求时面临根本性局限。因此，在操作系统中引入智能化技术已成为提升系统适应性的必然要求。在操作系统智能化演进历程中，研究者们面临着一系列结构性矛盾，这些挑战共同定义了当前操作系统智能化进程的关键技术挑战。本文从决策时效性、模型开销与系统安全三大维度，基于现有研究进展和成果作了进一步分析。

3.1 决策时效性与模型深度

现代操作系统对实时响应的要求与深度学习模型固有的计算深度形成矛盾。在进程调度领域，Linux 负载均衡器集成 MLP 模型[5]使调度延迟增加 13%，虽采用定点运算优化，但在数据中心级多核扩展场景仍显不足。强化学习调度器[12]需 120 轮离线训练迭代，无法满足实时任务动态迁移需求，暴露出时序建模与响应速度的冲突。存储预取场景同样如此，LSTM 预取模型[30]推理延迟高达 734μs，远超 Optane SSD 的 10μs 硬件延迟，导致预测完成时数据已过时。DeepPrefetcher 框架[33]虽通过逻辑块差值特征压缩提升效率，但 LSTM 的序列依赖处理仍引入不可忽视的延迟。在系统加速领域，LAKE 框架[9]中 GPU 加速的页面分类比纯 CPU 方案快 3.1 倍，但动态切换阈值需人工设定，无法自适应平衡批量任务与实时性需求。此外，数据收集与模型执行产生复合延迟，LTTng[35]和 BCC[5]等工具虽能追踪内核事件，但其数据采集过程本身增加额外开销。时序模型的固有特性加剧了矛盾，如内存访问模式预测研究[25]所揭示，LSTM 需积累百步历史 delta 序列才能建立有效上下文。

这个矛盾主要在于，精度提升依赖于模型的层次与参数规模，这与操作系统的实时性约束直接冲突。为解决这一问题，研究者可以沿着软硬件协同优化的路径展开探索。提高模型时效性可以通过轻量化模型架构来实现，如[5]提出的 8 位量化 MLP 有效降低计算强度，文献[26]的 Glider 框架将复杂 LSTM 优化到仅占用 62kB，极大缓解存储压力。另一个突破方向在于分层决策机制，例如[30]设计的双级预取框架，由轻量级前端处理即时请求，后端复杂模型则聚焦长周期模式，形成时间维度上的任务分工。此外，更具有革命性突破的方法是软硬件协同设计，如[30]与[33]所倡导的将 LSTM 嵌入 SSD 控制器，通过近数据处理消除数据传输瓶颈；LinnOS[7]则提出用 TPU 加速推断至微秒内，使模型计算不再拖累主机资源。

3.2 模型开销与优化收益

ML 模型的计算和存储开销远超传统启发式算法，在高并发、动态负载场景下收益边界模糊。机器学习在操作系统中的资源消耗往往与优化收益形成倒挂，尤其在资源受限环境中这一问题尤为尖锐。在内存管理领域，LLama 分配器[6]全局锁争用导致高并发场景性能下降 2.84 倍，原子操作成为快速路径瓶颈，虽然提高预测性但却牺牲了并发。Kleio 页面调度器[24]揭示了仅 0.1%的页面贡献 95%性能收益，但为训练全页面 RNN 模型需 2 小时/模型，资源投入与收益严重失衡。在存储优化领域，Fio[28]测试显示 LSTM 预取器[30]将 I/O 大小近似为 2 的幂次，导致 50%概率的冗余预取；blktrace[7]追踪证实冗余预取额外消耗带宽与 NAND 寿命。LinnOS 的 68kB 轻量网络推理开销达 4~6 μs，在超低延迟 SSD 中占比超 40%，逼近收益临界点。在微架构领域，

Glider 缓存策略[26]以 62kB 存储开销实现 IPC 提升 14.7%，但其底层 LSTM 模型因存储需求超标量级而无法硬件部署。

效益失衡的本质在于选择困境，轻量化路径如 k-sparse 特征[26]需丢弃 PC 顺序信息牺牲泛化性；高精度路径如地址嵌入矩阵[25]则需百万级参数，导致所需的存储空间爆发式增加。KMLib[36]等框架虽支持内核机器学习计算，但特征收集加剧资源消耗。尤其在突发负载场景[17]，Telemetry[37]监测显示静态模型收益随负载偏移快速衰减。

要解决模型开销与优化收益的矛盾，首先要以收益为导向进行计算，[7]的 LinnOS 框架设计的 Pareto 拐点分析算法，可以动态评估开启 ML 预测的实际价值；[9]则开发 eBPF 策略回调机制，实现 CPU/GPU 模式的实时切换。第二个解决思路是资源感知型模型压缩，[26]的 k-sparse 特征工程将模型内存占用降至 62kB，[5]探索的二值神经网络进一步降低精度需求。

3.3 安全缺陷与加固机制

操作系统内核的智能组件部署面临多重安全挑战。ML 模型存在安全风险，例如文献[5]指出，若采用未经形式化验证的 ML 模型，可能因对抗样本的诱导执行恶意任务迁移，导致模型缺乏鲁棒性保障；调度机制设计也有缺陷，文献[13]中的自适应时间片方案因允许用户设置 STS 值为 INT_MAX 而引发漏洞，攻击者可借此发起 DoS 攻击，核心问题在于未对边界值进行校验；内核框架层面隐患更隐蔽，如 KML 框架[8]因允许加载未签名模型，为恶意代码注入开辟新路径，造成信任链断裂；数据传递架构风险同样显著，LAKE 框架[9]通过用户态传输内核数据的设计，容易因内存泄露等侧信道攻击破坏内核-用户态隔离机制。

硬件安全维度存在类似问题，NVM 的持久性特性[23]增加数据残留攻击风险，而 Lynx[31]的内核集成案例显示，模型驱动的预取机制在缺乏隔离时可能破坏系统稳定性。RAID 调度器研究[29]证实，未整合纠删码的 ML 方案面对对抗性 I/O 请求时可能破坏数据一致性。更根本的矛盾在于可靠性保障，LinnOS[7]不得不采用混合 Hedging 机制补偿 ML 预测误差，反映出机器学习无法独立满足系统级的可靠性要求。

现有方案的共同弱点在于安全设计外挂化，模型无法通过沙盒隔离来阻止故障传播、未建立决策路径形式化验证、忽视硬件可信根集成等。

安全加固需从架构层进行根本性改进。文献[5]与[8]提出内核 ML 沙盒隔离故障域，文献[9]建议 TEE 技术加密传输通道；针对存储安全，[23]主张 NVM 增量加密技术阻断残留攻击；用于操作系统的强化学习[40][41]调度也需要融入可验证的安全指标。

当前操作系统的发展正处在智能化转型的关键时期。模型深度与实时响应、计算开销与效益产出、概率决策与安全保障这三组矛盾，共同构成机器学习赋能系统的关键技术挑战。

4 未来展望

在深入分析基于机器学习的操作系统优化技术面临的时效性、优化效益与系统安全等方面的挑战后，可以发现要解决这些问题并非简单改良操作系统，而是需要架构层面的协同进化。微内核是智能化操作系统架构演进的关键范式。微内核的主要思想是，内核本身只提供最基础的、最核心的服务，如进程/线程调度、进程间通信、最底层内存管理，而文件系统、网络栈、设备驱动等其他的系统服务，则作为独立的用户态服务运行。其技术特性与智能化转型需求形成深度契合，它是为解决上述根本矛盾应运而生的架构级技术载体。未来智能化操作系统将沿着下面三个主要发展方向实现螺旋式上升，如图 3 所示：

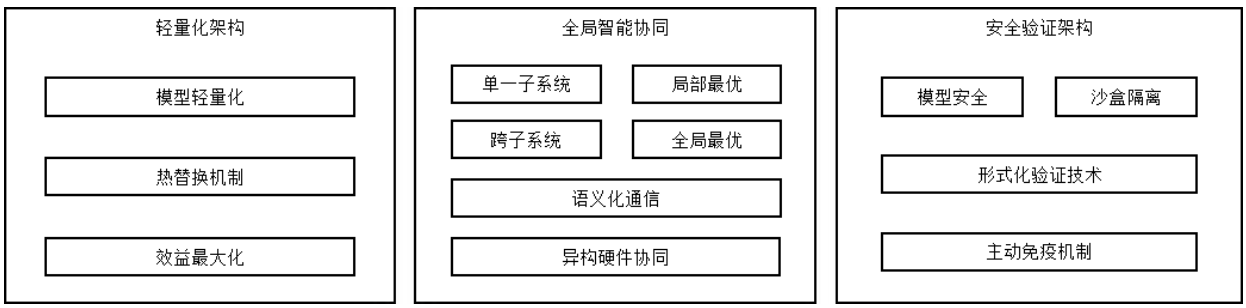

Figure3 The Future Prospects of Intelligent Operating Systems

图 3 智能化操作系统未来展望

1)轻量化架构

在操作系统智能化演进过程中，将机器学

习模型集成到操作系统中面临资源约束挑战。模型应用必须满足两个目标：避免对操作系统造成过量内存与计算负担，同时确保其性能增益超越传统算法范式。这要求构建效益最大化的轻量化架构，在资源消耗与性能提升之间建立精准平衡机制。当前，微内核技术凭借其安全隔离特性和模块化扩展能力，正成为操作系统智能化发展的主流演进方向，其核心价值在于推动系统架构向动态微服务化转型，解决模型资源困境。

微内核架构通过热替换机制实现计算资源的动态供给。热替换机制是一种动态模块化加载技术，其核心在于不中断系统运行的前提下，按需替换或加载操作系统中的机器学习计算单元。该机制将预测功能分解为标准化微服务单元，在系统稳态运行时维持极简预测器的基础运作。当检测到复杂工作模式时，系统快速切换到预测能力更强的模型组件。这种按需供给模式与实时监控功能结合形成智能决策闭环，当监测到特定模型单元的效益成本低于设定阈值时，系统自动触发降级机制，切换至预定义的轻量级替代方案，实现计算资源的动态回收与再分配。

在资源高度受限的边缘操作系统场景，深度模型压缩技术可以提供轻量化的智能部署。通过结构优化与精度压缩技术，使核心智能功能在严格资源边界内维持运行效能。此外，将模型的计算任务向下迁移到硬件中，也是轻量化的一个重要方向，并可能实现操作系统中智能模型的效能提升，在模型体积实现数量级压缩的同时，达成推理延迟的同步优化。

这种架构转型预示着操作系统智能化的发展方向：通过动态微服务机制精准控制模型加载粒度，在系统稳态时维持最小资源消耗，在复杂场景下动态激活增强的智能模块；通过深度压缩技术突破边缘设备的智能部署边界；通过计算向下迁移的策略重构近端数据处理范式。这些技术路径共同指向新型效能平衡点，机器学习不再作为资源消耗体，而是成为系统能力的倍增器，在严格资源约束下实现智能特性与系统性能的协同进化，最终推动操作系统进入智能赋能与资源效率统一的新发展阶段。

2)全局智能协同

操作系统的智能化，难点在于突破传统的优化范式[45]。进程调度器在优化负载均衡时，没有考虑内存带宽争用问题；内存分配器在减少碎片时，忽略了存储预取引发的缓存波动；存储管理器在提升 I/O 性能时，却因为线程调度造成访问模式突变。这些孤立的智能优化技术各自追求局部最优，却牺牲了全局的效能，形成相互制约的优化困局。未来的一个重要发展方向在于构建跨子系统的协同优化框架，使离散的智能节点融合为整体，而微内核架构的语义化通信机制[42][43]为此提供了关键技术支撑。

语义化通信的本质在于将原始数据传递转变为意图传递。当存储子系统检测到连续大块的读取请求时，不再只是传递内存地址序列，而是发出"顺序流预取+大页内存绑定"的语义化指令；当内存管理器发现高频回收小对象时，语义化通信会向进程调度器传递"短生命周期任务聚集调度"的协同请求。这种携带操作意图的消息传递，使接收方能直接理解上下文语义并触发连锁优化：存储层的访问模式特征驱动内存分配策略的动态调整，线程调度轨迹反向指导预取窗口的智能伸缩，NUMA 拓扑信息则成为负载均衡与缓存迁移的共同约束。微内核的通信管道使进程调度器能够感知内存回收压力，使存储管理器能够预判线程计算变化，最终形成全局优化的认知闭环。

在硬件协同维度，微内核架构也为异构资源协同提供了支持。当智能网卡识别出计算密集型负载特征，可以通过语义通道调度 GPU 参与页面热度预测；当持久内存写入延迟波动，DPU 可以自动切换至降级模式并通知内存控制器调整刷新策略。

这种协同架构不仅是技术层面的重构，更是操作系统优化范式的认知跃迁。当离散的智能节点通过语义化通信联结为整体，操作系统将突破局部孤立优化的困境，实现资源感知的全局智能协同。

3)安全验证架构

智能化操作系统的安全范式面临根本性重构。传统宏内核架构下，机器学习模型的不可解释性为系统安全带来了风险，系统缺乏有效机制阻断模型引起的安全故障。此外，当前操作系统的安全防护停留在漏洞响应层面，无法应对智能时代复杂多变的风险形态。未来的安全架构需要实现双重突破：既要建立模型行为的可验证性，又要构筑持续进化的安全体系，而微内核技术为此提供了解决方案。微内核架构的安全隔离能力化解了基础性风险。当智能调度服务运行时，微内核可以将能力空间限制在无硬件访问权的沙盒域内，即使强化学习策略被恶意样本攻破，崩溃的影响也被控制在单个服务域内，系统可自动重启新实例来恢复状态。

形式化验证技术为智能模型提供确定性保障[46][47]。新型验证内核将机器学习决策过程转换为可证明命题：任务迁移指令需经过定理证

明校验，确保符合安全约束；存储迁移模型被重构为可验证结构，当预测值超过阈值时，输出指令必须附带形式化证明。该验证体系同样应用于硬件层，加速单元执行的预取模型输出需携带可信执行环境签名，接收端验证通过后方可执行，形成端到端可信计算链。

主动防护机制的核心在于故障安全学习框架[48]。系统通过持续监控模型行为，当检测到决策偏离安全基线时，自动触发模型冻结与诊断流程；当性能指标异常波动时，即时回滚至经认证的稳定版本。关键演进在于异常数据的知识转化，诊断模块分析安全事件后，建立防护知识库，自动优化决策空间参数。微内核架构支持新模型在沙盒隔离域内完成形式化验证与测试后，再接入生产环境，保证模型防护能力。

微内核架构在传统隔离机制的基础上，为操作系统构建了安全智能的平台[44]。通过形式化验证将黑盒模型转化为可验证组件，借助故障安全学习框架实现防护能力动态增强。

5 结束语

本文综述了近年来机器学习在操作系统中的应用研究进展。首先，文章介绍了机器学习技术在操作系统中的重要性和潜力，强调了其在提升系统性能和可靠性方面的关键作用。接着，文章按操作系统的子系统，将基于机器学习的方法划分为四个主要研究方向：智能化进程调度、智能化内存管理技术、基于机器学习的存储系统性能优化技术，以及通用的内核机器学习架构。此外，文章还讨论了当前研究中存在的挑战，包括决策时效性、模型开销、安全缺陷等。总体而言，机器学习为操作系统的研究和开发提供了新的视角和方法，能够有效提升系统的性能和可靠性。

参考文献：

[1] SAFAZADEH V M, LOGHMANI H G. Artificial intelligence in the low-level realm: a survey[J]. 2021.

[2] NEGI A, RAJESH K. A review of AI and ML applications for computing systems[C]// Proceedings of the 9th International Conference on Emerging Trends in Engineering and Technology - Signal and Information Processing (ICETET-SIP-19), 2019.

[3] ZHANG Y F, ZHAO X K, YIN J W, et al. Operating system and artificial intelligence: a systematic review[J]. 2024.

[4] ZHANG Y, HUANG Y. “Learned” operating systems[J]. ACM SIGOPS Operating Systems Review, 2019, 53.

[5] 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 (APSys’20). New York: Association for Computing Machinery, 2020: 67-74.

[6] MAAS M, ANDERSEN D G, ISARD M, et al. Learning-based memory allocation for C++ server workloads[C]// Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’20), 2020.

[7] HAO M Z, TOKSOZ L, LI N Q, et al. LinnOS: Predictability on unpredictable flash storage with a light neural network[C]// Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020: 173-190.

[8] AKGUN I U, AYDIN A S, BURFORD A, et al. Improving storage systems using machine learning[J]. ACM Transactions on Storage, 2023, 19(1): 1-30.

[9] FINGLER H, TARTE I, YU H C, et al. Toward a machine learning-assisted kernel with LAKE[C]// Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2023), 2023, 2.

[10] QIU Y M, LIU H Y, ANDERSON T, et al. Toward reconfigurable kernel datapaths with learned optimizations[C]// Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS’21). New York: Association for Computing Machinery, 2021: 175-182..

[11] DIAS S, NAIK S, SREEPRANEETH K. A machine learning approach for improving process scheduling: a survey[J]. International Journal of Computer Trends and Technology, 2017, 43(1).

[12] SARKAR D, SINGAL S. Reinforcement learning in process scheduling[C]// Proceedings of the 13th International Conference on Cloud Computing, Data Science & Engineering (Confluence). Noida: IEEE, 2023: 237-242.

[13] NEGI A, KISHORE KUMAR P. Applying machine learning techniques to improve linux process scheduling[C]// Proceedings of the 2005 IEEE Region 10 Conference (TENCON 2005), 2005..

[14] OJHA P, THOTA S R, VAN M, et al. Learning scheduler parameters for adaptive preemption[C]// Proceedings of the 4th International Conference on

Advanced Information Technologies and Applications, 2015.

[15] KIM J H, CHOE W K, AHN J S. Exploring the design space of page management for multi-tiered memory systems[C]// Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21), 2021: 715-728.

[16] TIAN C, LIU H K, LIAO X F, et al. UCat: heterogeneous memory management for unikernels[J]. Frontiers of Computer Science, 2023, 17(1): 56-66.

[17] PHADKE S, NARAYANASAMY S. MLP aware heterogeneous memory system[C]// Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE 2011), 2011.

[18] LIANG Y, LI J, AUSAVARUNGNIRUN R, et al. Acclaim: Adaptive memory reclaim to improve user experience in android systems[C]// Proceedings of the USENIX Annual Technical Conference (USENIX ATC), 2020: 897-910.

[19] LI C L, LIANG Y, AUSAVARUNGNIRUN R, et al. ICE: Collaborating memory and process management for user experience on resource-limited mobile devices[C]// Proceedings of the 18th European Conference on Computer Systems (EuroSys), 2023.

[20] LEBECK N, KRISHNAMURTHY A, LEVY H M, et al. End the senseless killing: Improving memory management for mobile operating systems[C]// Proceedings of the USENIX Annual Technical Conference (USENIX ATC), 2020: 873-887.

[21] LI C, SHI L, LIANG Y, et al. SEAL: User experience-aware two-level swap for mobile devices[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2020: 4102-4114.

[22] LI C, SHI L, XUE C J. MobileSwap: Cross-device memory swapping for mobile devices[C]// Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC), 2021.

[23] MUTLU O, SUBRAMANIAN L. Research problems and opportunities in memory systems[J]. Supercomputing Frontiers and Innovations: an International Journal, 2014, 1(3): 19-55.

[24] DOUDALI T D, BLAGODUROV S, VISHNU A, et al. Kleio: A hybrid memory page scheduler with machine intelligence[C]// Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC ’19). New York: Association for Computing Machinery, 2019: 37-48.

[25] HASHEMI M, SWERSKY K, SMITH J A, et al. Learning memory access patterns[J]. 2018.

[26] SHI Z, HUANG X R, JAIN A, et al. Applying deep learning to the cache replacement problem[C]// Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’52), 2019.

[27] NARAYANAN A, VERMA S, RAMADAN E, et al. DeepCache: A deep learning based framework for content caching[C]// Proceedings of the 2018 Workshop on Network Meets AI & ML (NetAI’18). Budapest: ACM, 2018.

[28] TANG Y, LIN R H, LI D D, et al. FSbrain: An intelligent I/O performance tuning system[J]. Journal of Systems Architecture, 2022, 129: 102623.

[29] DUTTA P K, SINGH S V, NANDI A. Effective selection of completely fair scheduler algorithm in RAID kernel for improved I/O performance using machine learning[C]// Proceedings of the 7th IET Smart Cities Symposium (SCS 2023), 2023.

[30] CHAKRABORTTI C, LITZ H. Learning I/O access patterns to improve prefetching in SSDs[C]// Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2020). Heidelberg: Springer, 2020: 427-443.

[31] LAGA A, BOUKHOBZA J, KOSKAS M, et al. Lynx: A learning Linux prefetching mechanism for SSD performance model[C]// Proceedings of the 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA), 2016.

[32] 常明喆. 操作系统内核态调优的机器学习方法研究[D]. 成都: 电子科技大学, 2024.
CHANG M Z. Research on machine learning methods for operating system kernel optimization[D]. Chengdu: University of Electronic Science and Technology of China, 2024.

[33] GANFURE G O, WU C F, CHANG Y H, et al. DeepPrefetcher: A deep learning framework for data prefetching in flash storage devices[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(11): 3311-3322.

[34] BOROUMAND A, GHOSE S, KIM Y S, et al. Google workloads for consumer devices: Mitigating data movement bottlenecks[C]// Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18), 2018.

[35] LTTng. LTTng: An open source tracing framework for Linux[EB/OL]. (2019-04) [2024-06-01]. https://lttng.org.

[36] AKGUN I U, AYDIN A S, SHAIKH A, et al. A machine learning framework to improve storage system performance[C]// Proceedings of the 13th ACM Workshop on Hot Topics in Storage (HotStorage’21). New York: ACM, 2021.
[37] Chromium Project. Catapult: Telemetry[EB/OL]. [2024-06-01]. https://chromium.googlesource.com/catapult/+/HEAD/telemetry/README.md.
[38] Google LLC. TensorFlow: Mobile[EB/OL]. [2024-06-01]. https://www.tensorflow.org/mobile/.
[39] GHEMAWAT S, MENAGE P. Tcmalloc: Thread-caching malloc[EB/OL]. (2009) [2024-06-01]. http://goog-perftools.sourceforge.net/doc/tcmalloc.html.
[40] SUN S Q, XU X, et al. Bi-directional influence and interaction for multi-agent reinforcement learning[J]. IEEE Transactions on Artificial Intelligence, 2024, 5(10): 1-12.
[41] XU H, PENG P, et al. KylinTune: DQN-based energy-efficient model for browser in mobile devices[C]// Proceedings of the 2022 IEEE International Performance, Computing, and Communications Conference (IPCCC). 2022.
[42] LIU C, LUO L, LI M, et al. Inter-core communication mechanisms for microkernel operating system based on signal transmission and shared memory[C]// Proceedings of the 7th International Symposium on System and Software Reliability (ISSSR), 2021.
[43] BAMBER S S. Implementation of improved synchronization in inter process communication using threads for microkernel and distributed operating systems[J]. International Journal of Recent Technology and Engineering, 2019, 8(2): 769-771.
[44] PRUTHIVIRAJ B, MADHUSUDHUN G S, VIJAYASARATHY S. A microkernel based secure operating system using Genode framework[J]. Journal of Theoretical and Applied Information Technology, 2012, 38(2): 177-182.
[45] BATENI S, LIU C. NeuOS: A latency-predictable multi-dimensional optimization framework for DNN-driven autonomous systems[C]// Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20), 2020: 371-385.
[46] DAVIDOFF A, COUDER J, OCHOA O. Supporting formal methods for machine learning verification in urban air mobility[C]// Proceedings of the 2024 Conference on AI, Science, Engineering, and Technology (AIxSET), 2024.
[47] GADDE D N, KUMAR A, NALAPAT T, et al. All artificial, less intelligence: GenAI through the lens of formal verification[J]. 2024.
[48] WANG H, YANG R G, XIANG J W. Numerical simulation of gears for fault detection using artificial intelligence models[J]. Measurement, 2022, 203: 111898.

作者简介：

黄卓懿（2001－），女，福建福州人，硕士生，研究方向为操作系统。**E_mail:** huangzhuoyi23@nudt.edu.cn

HUANG Zhuoyi, born in 2001, Graduate student, her research interests include operating systems.

彭龙（1987—），通信作者，男，博士、助理研究员。CCF 专业会员（会员号 I2537M），主研方向为操作系统、系统软件和机器人编程框架。**E_mail**: penglong@nudt.edu.cn

PENG Long, born in 1987, corresponding author, PhD, assistant professor, CCF professional member (membership number I2537M), his research interests include operating systems, system software, and robot programming frameworks.

徐浩（1993—），男，博士生。主研方向为系统软件和人工智能。**E_mail**: xuhao19@nudt.edu.cn

XU Hao, born in 1993, male, PhD candidate, his research interests include system software and artificial intelligence.

李琢（1994—），男，博士后。主研方向为多智能体系统、强化学习。**E_mail**: lizhuo@nudt.edu.cn

LI Zhuo, born in 1994, postdoctoral researcher, his research interests include multi-agent systems and reinforcement learning.

刘晓东（1985—），男，博士、副研究员。主研方向为操作系统、系统软件和软件工程。**E_mail**: liuxiaodong@nudt.edu.cn

LIU Xiaodong, born in 1985, PhD, associate professor, his research interests include operating systems, system software, and software engineering.

governance.

刘敏（1991—），女，麒麟软件有限公司，工程师。主研方向为操作系统、开源软件和开源社区治理。**E_mail**:liumin@kylinos.cn。

LIU Min, born in 1991, Engineer at Kylin Software Co., Ltd. Her research interests include operating systems, open source software, open source community governance.

余杰（1982—），通信作者，男，博士、研究员。主研方向为操作系统、系统软件和人工智能。**E_mail**: yj@nudt.edu.cn

YU Jie, born in 1982, corresponding author, PhD, professor, his research interests include operating systems, system software, and artificial intelligence.