调研报告

1 项目概述

MEMO 基于 LLM 与 LSTM（长短期记忆网络）等模型，通过提取程序使用数据中的上下文信息来预测用户软件使用行为，实现 PC 端优化 Linux 内存管理调度，提升用户体验。

2 项目背景

2.1 Linux 系统及其内存管理

2.1.1 Linux 系统

Linux 是一个开源、免费的类 Unix 操作系统，在服务器市场占据超过 80% 的份额，几乎所有主流公有云平台（AWS 、Azure 、Google Cloud ）都运行于 Linux ，据 CNCF 2023 报告，96% 的云原生环境运行于 Linux ；TOP 500 超级计算机榜单连续多年显示，100% 的超算系统依赖 Linux；从智能家居设备（如路由器、智能电视）到工业控制器，Linux 凭借轻量级发行版（如 Yocto 、Buildroot ）占据 65% 的嵌入式市场；尽管 Linux 桌面用户约占总市场的 2-3% ，但开发者覆盖率高达 90% 以上；Linux 内核每年接收超 100 万次代码提交，由全球数千名开发者维护。

2.1.2 Linux 系统的文件管理

Linux 操作系统的内存管理作为现代计算机系统中一个重要的组成部分，其主要目的是高效地管理物理内存和虚拟内存，确保系统资源得到合理分配，并最大化系统性能。Linux 内存管理通过以下几个关键组件和机制来实现：

- **虚拟内存**
  Linux 采用虚拟内存管理机制，将每个进程分配到独立的虚拟地址空间。这使得进程能够拥有独立的地址空间，避免相互之间的干扰。虚拟内存不仅支持进程的地址空间隔离，还可以通过分页机制将物理内存扩展到硬盘存储上，形成虚拟内存与物理内存之间的映射。

- **页和页表**
  Linux 内存管理通过页面来划分内存，通常每个页面大小为 4KB 。操作系统通过页表记录虚拟地址与物理地址的映射。当程序访问一个虚拟地址时，内存管理单元会将其转换为物理地址。为了优化内存访问，Linux 还使用了不同级别的页表（如 4 级页表）。

- **内存分配器**
  Linux 内核提供了多种内存分配机制，包括常见的伙伴系统（Buddy System ）、Slab 分配器和 Slub 分配器。伙伴系统用于大块内存的管理，而 Slab/Slub 分配器用于小块内存的管理，这些机制有效减少了内存碎片，提高了内存分配和回收的效率。

- **内存回收与交换（Swap ）**
  当物理内存不足时，Linux 使用交换空间将不活跃的内存页写入磁盘，以腾出物理内存给当前需要的进程。交换机制通过交换空间管理内存的使用，使得系统能够在内存压力大的情况下保持运行。

Linux 系统中频繁的内存换入换出，正适合利用模型预测并优化系统内存管理。

2.2 LLM

2.2.1 LLM 的概念、发展和分类

LLM 的概念

大型语言模型（large language model，LLM），也称大语言模型，是由具有大量参数（通常数十亿个权重或更多）的人工神经网络组成的一类语言模型，使用自监督学习或半监督学习对大量未标记文本进行训练。（摘自 wiki 百科）

与 LLM 不同，**神经网络**是一类受到生物神经结构启发的计算模型，是实现各种 AI 任务的基础。LLM 通常建立在深层神经网络之上，但神经网络的概念远比 LLM 范围更广，涵盖所有由神经元构成的计算结构。

LLM 的发展历程

LLM 能够在海量文本数据上进行训练，能够理解、生成自然语言，并在诸多 NLP 任务上展现出强大能力。其发展经历了多个重要阶段，每个阶段都推动了语言技术的进步，奠定了当前模型的基础。

早期阶段：统计语言模型的初步探索 在 LLM 发展的早期，研究者主要依赖统计语言模型（如 n-gram 模型）来估计词的出现概率。这种方法通过有限的上下文信息来进行语言建模，但由于上下文长度的限制，其在捕捉长距离依赖关系方面存在明显不足。

神经网络的引入：捕捉长距离依赖的能力 随着神经网络方法的兴起，语言建模进入了一个全新的阶段。循环神经网络（RNN）以及其改进版本长短期记忆网络（LSTM）的出现，使得模型能够有效捕捉文本中的长距离依赖关系。这一进展大幅提升了语言模型的表现，并为后续技术奠定了理论基础。

Transformer 的出现：自然语言处理的革命性进步 2017 年，Transformer 模型的提出彻底改变了自然语言处理领域。与传统的序列模型不同，Transformer 通过自注意力机制实现了对全局上下文的高效建模，同时支持大规模并行计算。这一技术突破为后续的大规模语言模型（如 BERT、GPT 系列）的发展提供了核心框架。

大规模预训练与微调：LLM 的成熟阶段 近年来，诸如 GPT-3 和 BERT 等预训练模型通过在海量数据上进行预训练，并结合微调技术，在多任务、多场景下展现出了卓越的性能。这一阶段标志着大规模语言模型进入了“规模越大、能力越强”的时代，推动了人工智能在语言领域的广泛应用。

LLM 的分类

LLM 从不同角度可以进行不同的划分，常见的划分如下图1所示：

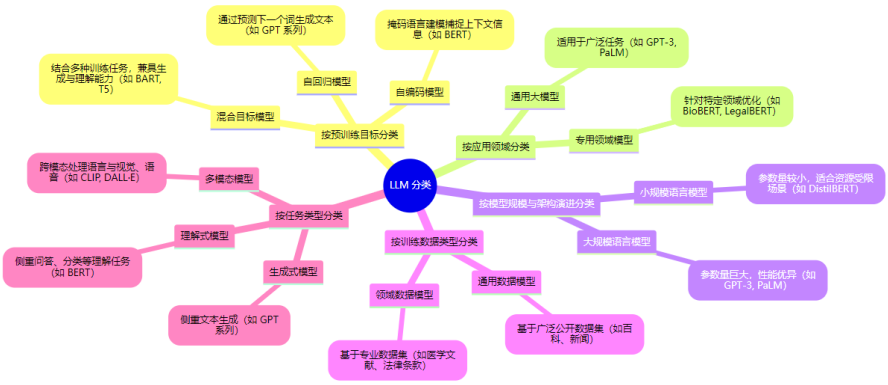

图 1: LLM 分类图

LLM 的特点

LLM（大规模语言模型）具有一些显著特点。

首先，它依托于海量参数与数据，通常拥有数十亿到上万亿的参数，并在大规模数据集上进行预训练，这使得模型具备了强大的表征能力和语言生成能力。**其次**，大多数 LLM 采用预训练-微调范式，通过无监督或自监督的方式在未标注数据上进行大规模学习，然后结合少量标注数据进行微调，从而展现出跨任务处理的灵活性。**此外**，LLM 在文本生成与推理方面表现出色，不仅能够生成连贯流畅的自然语言，还能完成推理、翻译、问答等复杂任务，尤其是在零样本或少样本的情况下展现出惊人的能力。然而，与

这些优势并存的，是模型在可解释性和伦理问题上的**挑战**。由于 LLM 的黑箱特性，其决策过程通常难以解释，同时可能引发偏见、滥用等问题，这些都已成为当前研究的重要方向。

2.2.2 KVCache

KVCache（Key-Value Cache）是 Transformer 架构实现自回归生成时常用的一种缓存机制，它主要保存了在前向传播过程中计算出的各个 Transformer 层中注意力机制所用到的键（Key）和值（Value）张量。下面我们详细介绍 KVCache 的概念、其与 LLM 的关系以及相关的技术资料。

1. KVCache 的定义与作用 参考 Wikipedia 上对 Transformer 模型的描述（Wiki 上关于 Transformer 的介绍:），**KVCache** 指的是在 Transformer 模型的自注意力机制中，为了加快自回归（autoregressive）生成过程而缓存之前计算过的 Key（键）和值（Value）向量的机制。在模型推理时，每个新生成的 token 只需计算与该 token 相关的 Query（查询）向量，而之前得到的 Key 和 Value 矩阵可以直接复用，这样避免了对整个序列重复计算自注意力，从而大幅度提高了生成效率。

2. KVCache 与 LLM 的关系 在大规模语言模型（LLM）中，如 GPT 系列、PaLM 等，模型通常采用自回归生成文本。KVCache 正是这类模型在推理过程中提高速度和降低计算资源消耗的重要手段。具体表现为:

(a) **加速推理**: 使用 KVCache，可以避免对已经计算的注意力信息进行重复计算，使得在生成长序列时计算量不随序列长度呈平方级别增长（[11]）。

(b) **降低计算资源需求**: 在面对实时生成任务或模型微调、在线应用时，KVCache 的利用减少了冗余运算，降低了延时和资源消耗（[10]）。

(c) **实现长上下文处理**: 部分扩展性改进的 Transformer 架构（例如 Transformer-XL）也利用缓存机制来实现跨越固定长度片段的长距离依赖，这与 KVCache 思路类似（[4]）。

总结来说，KVCache 是大规模语言模型（LLM）在利用 Transformer 进行自回归生成任务中不可或缺的一个部分。它通过**缓存前面各层的注意力键和值**，使得在生成新 token 时能够高效利用历史信息，从而加速推理过程并降低计算资源消耗。对 KVCache 的理解不仅有助于深入理解 Transformer 模型的内部运作，也为优化大规模语言模型的部署和应用提供了技术支持。

3. KVCache 本身存在的问题 KVCache 在提升 Transformer 自回归生成效率方面具有重要作用，但同时也存在一些问题和挑战，下面具体分析这些问题及可能的解决方向：

1. **内存消耗与增长问题**

   问题描述：KVCache 的主要任务是保存已计算的 Key 和 Value 向量，当生成文本的序列较长时，缓存的大小将随 token 数量线性增长，导致内存占用越来越高。这在长文本生成或者需要大量并发推理的场景下可能严重影响系统性能。

   可能的解决方向：

   - 滑动窗口或固定长度缓存：通过对上下文进行截断，只保留最近一段信息来避免内存无限扩展，此思路在 Transformer-XL [4] 中有体现。
   - 缓存压缩技术：参考 Compressive Transformers [9] 的思路，对历史缓存进行压缩，降低内存占用，同时尽量保留关键信息。
   - 稀疏化技术：利用稀疏注意力机制，仅保留最重要的缓存项，减少冗余信息存储 [3].

2. **计算与索引效率**

   问题描述：在实际推理过程中，KVCache 需要在每一步生成中高效地检索和整合历史信息。如果缓存数据结构设计不合理，检索效率低下或者数据更新操作繁琐，都会影响整体生成速度。

   可能的解决方向：

   - 优化数据结构：采用专门设计的数据结构（如分段索引、hash-map 等）来加速 Key/Value 检索。
   - 并行计算与内存访存优化：在硬件层面利用并行计算来高速处理缓存数据，同时结合内存层次优化缓存管理策略 [11].

3. **缓存失效与一致性问题**

   问题描述：在某些动态调整策略下，缓存中的信息可能与当前状态不匹配（例如模型参数更新、局部上下文变化等情况下的缓存失效问题）。这种不一致可能导致生成效果降低。

   可能的解决方向：

   - 动态缓存刷新策略：引入缓存失效检测和刷新机制，确保缓存中的数据与当前状态相吻合。
   - 混合模式：在保持缓存优势的同时，适时部分重新计算全部自注意力内容，做一次“整体校验”，以平衡效率和精度 [7].

4. **与 Beam Search 和多样本生成的兼容性**

问题描述：对于 Beam Search 或其它复数候选输出模式，缓存更新和共享可能变得更加复杂，因为不同生成路径之间可能存在差异，如何高效复用缓存成为难题。

可能的解决方向：

- 路径隔离与共享机制：设计一种细粒度的 KV 缓存更新策略，对不同生成路径或候选解分别存储并进行局部共享。
- 融合启发式搜索：在 Beam Search 中引入启发式方法选择最“可信”的缓存片段，使得整体生成更加稳定 [1].

总体来说，KVCache 在提升自回归生成效率方面有显著作用，但在内存管理、计算效率、一致性和多候选生成等场景上仍存在挑战。当前的研究尝试通过固定窗口、压缩技术、优化数据结构和动态刷新等多种方法来解决这些问题，这些方向也为未来如何在大模型部署中高效利用 KVCache 指明了道路。

2.3 vLLM

vLLM 是一种**专为大规模语言模型 (LLM) 推理设计**的高性能引擎，其目标是在保证生成质量的前提下，显著提升推理效率和内存利用率。与传统推理框架相比，vLLM 在内存管理、动态批处理以及 KV cache 的高效复用等方面进行了多项优化。这使得 vLLM 能在面对海量请求和复杂模型时更好地发挥资源优势 [13]。

在原理上，vLLM 的核心优化包括：

1. **动态批处理与调度**：vLLM 通过动态批处理技术，将多个推理请求聚合处理，充分利用 GPU 或其他硬件的并行能力，从而降低单请求的延迟。同时，调度器会根据请求的相似性和时序关系智能安排计算顺序，避免硬件资源的闲置与过度调度 [5]。

2. **高效 KV cache 管理**：在自回归生成中，KV cache 用于保存历史生成过程中的 Key 和 Value 向量，但长序列生成可能导致内存占用激增。vLLM 通过以下方式优化 KV cache：

- 使用内存压缩与分级存储结构，对不活跃的数据进行压缩存储；
- 利用智能缓存更新策略减少冗余存储和重复计算；
- 在动态批处理中复用缓存信息，减少每个请求的内存和计算开销 [13].

3. **内存管理优化**：vLLM 采用精细化的内存管理机制，包括内存池化和分级分配策略，提升了推理过程中的内存利用率。通过实时监控和释放内存，vLLM 能有效降低内存碎片和泄露风险，在大规模推理任务中表现出色 [6]。

vLLM 的这些设计在我们项目中的必要性主要体现在以下几点：

1. - **高效推理能力**：vLLM 的动态批处理机制可显著加速推理过程，满足高并发场景中对低延迟和高吞吐量的需求。

2. - **内存优化优势**：我们项目聚焦于内存优化，vLLM 的精细化 KV cache 管理和内存分配策略能够有效解决长序列生成带来的内存激增问题 [4]。

3. - **灵活扩展**：vLLM 作为开源项目，具备良好的灵活性和可扩展性，支持根据项目需求进行定制化优化。其监控接口与优化方案为动态调整内存分配提供了技术支持 [13]。

总结来说，vLLM 是一款以高性能与内存高效为主要目标的推理引擎，其动态调度、高效 KV cache 管理以及精细化内存管理机制与我们项目的需求高度契合。通过引入 vLLM，我们可以在保证生成质量的同时，有效降低内存占用和延迟，提升系统的性能与扩展性。

3 立项依据

3.1 利用 LLM 与 LSTM 预测用户程序调度

3.1.1 传统操作系统内存管理 vs LLM 赋能新型内存管理

1. **规划内存** 在传统智能调度机制下，操作系统可能无法合理预测应用程序的需求。例如，当一个用户在系统中频繁使用某个应用程序时，操作系统并不提前为其预留足够的内存。这可能导致应用程序启动时资源竞争，从而出现启动延迟或应用卡顿。相比之下，智能调度系统可以根据历史数据和用户行为模式提前预测用户的需求，并提前为可能被打开的应用程序预分配内存资源。这样做能显著减少启动时间和提高响应速度。

2. **合理分配资源** 在多任务处理时，系统可能会低估或高估某些应用的资源需求，造成资源过度分配或资源饥饿，即任务之间无法平衡分配，最终影响整体系统性能。通过智能调度，系统能够根据实时和历史行为调整任务优先级，避免多个任务争夺相同的资源，减少系统负载和资源瓶颈。

3.1.2 优化角度：粗粒度预测与微秒级实时性的平衡

在我们的调研过程中，我们意识到虽然 AI 和机器学习模型在资源调度领域具有巨大的潜力，但它们在进行**细粒度调度**时面临一些挑战。具体来说，操作系统的调度任务通常是在**微秒级别**进行的，这要求操作系统能够实时地进行快速决策和执行。然而，当前的 AI 模型（尤其是基于深度学习和预测的模型）在处理调度任务时，通常需要**秒级别的时间**来进行计算和推断，这与操作系统对实时性要求存在较大差距。因此，使用 AI

Order	prefix	character frequency						T_m	PPM w_m	APPM w_m
		n	t	r	f	s	o			
3	sio	2	1					3	1	0.5
2	io	3	1	1				1	$\frac{1}{4}$	0.5
1	o	3	1	2	6			6	$\frac{1}{4\cdot2}$	0.64
0	ϕ	3	1	2	6	4	12	16	$\frac{1}{4\cdot2\cdot7}$	0.38

图 2: APPM 算法原理图

来直接进行**微观层次的预测**（如 CPU 时间片分配、内存页调度等）可能导致系统响应延迟，从而影响操作系统的稳定性和实时性。

鉴于此，我们选择将 AI 模型的应用范围限制为**粗粒度的调度任务**，具体指针对用户打开软件的需求预测。在这个场景下，系统并不需要在微秒级别上对每个任务进行精细的资源管理，而是可以在秒级别的时间窗口内做出预测和决策。例如，预测用户在某段时间内可能打开哪些应用程序，并根据预测结果提前分配资源、进行预加载等。这种粗粒度的调度不仅能够有效地利用 AI 的预测能力，同时也避免了 AI 模型在实时性要求极高的操作系统任务中可能带来的延迟和风险。

3.1.3 前期工作的研究方法

早期研究

1. **上下文分析 (Contextual Information)**:

   上下文分析指的是通过环境因素例如包括时间、位置、WiFi 信号、电池使用情况等一系列信息对用户的软件使用行为进行预测的方法，其中时间的地理位置对预测有至关重要的决定性因素。在调研过程中，我们还发现针对手机软件预测的研究较多，而针对 PC 进行软件预测的研究则相对较少，这很有可能是由于上下文信息在手机上容易获取，但在电脑上获取较为困难导致的 [12]。

2. **APPM 算法**:

   APPM 算法是一种模仿文本压缩算法 PPM 的预测应用算法。PPM 算法的核心思想为通过字符的前缀来推测后续的字符，从而实现压缩的功能。同样地，APPM 算法将“程序”看做“字符”，将用户使用程序的时间序列看作字符串。通过用户当前使用的程序来预测后续将要使用的程序。与 PPM 算法不同的是，在字符序列中，往往长度最长的字符串与整个字符具有最强的相关性。然而，在用户的应用使用过程中，短的前缀和长的前缀可能是同等重要的。例如，研究者发现在某些情况下，用户倾向于强烈偏爱最近使用的项目；而在其他情况下，用户的使用序列表现出高度连续的软件使用行为 [8]。下图2是 APPM 算法与 PPM 算法工作原理的比较示意图。

3. **PTAN**

PTAN(Parallel Tree Augmented Naive Bayesian Network) 是一种应用程序预测模型，它的任务是根据用户的历史行为（如应用程序的使用时间、位置等）预测用户在下一个时间段内最可能使用的应用程序。提出者将此任务建模为**分类问题**，并通过结合上下文信息和历史数据来实现个性化预测。研究者首先获取用户的应用使用记录,。然后提取这些数据中的上下文信息和会话特征（应用程序之间的时序关系）[2]。

研究者将应用分为短期应用和长期应用。在分类方法上，他们通过拟合应用程序使用数据到 Beta 分布，从而评估应用的时间显著性（即其使用的短期性或长期性）。短期应用指使用频率较高的应用，预测时采用基于用户历史行为的概率。长期应用则通过”集体智慧”来推测预测概率，即基于其他用户的数据进行初步预测。因为对于长期应用来说，使用贝叶斯平均将用户的历史行为与其他用户的平均行为数据结合能得到更准确的预测 [2]。

PTAN 也考虑了**应用冷启动** 和**用户冷启动**问题，并提出了两种策略：应用冷启动主要针对新安装的应用，算法基于其他用户的数据估计其初步的使用概率，并随着用户使用频率增加逐步调整。用户冷启动则主要针对新用户，研究者采用了最相似用户策略和伪用户策略来预测他们的行为模式。最相似用户策略指通过找到与新用户最相似的用户，并借用他们的行为模式进行预测。伪用户策略指通过生成“伪历史”来训练模型，为新用户构建合适的初始数据。

这些早期的方法均存在着同样的缺陷，即它们都忽略了应用程序使用的时间序列依赖性，未能有效捕捉历史信息揭示的长期模式。具体来说，早期的方法主要是一些**概率模型**。这些方法依赖于用户行为的**统计特征**（如使用频率、应用间的关联等），并假设每个时刻的行为是相对独立的，即**条件独立假设**。然而，这种假设无法有效捕捉**应用程序使用的时间序列依赖性**，即一个用户的当前使用的 app，可能与过去一段时间内的行为存在深刻的关联。随着神经网络的迅猛发展，新的预测方法逐渐取代了原生的概率模型，成为更加主流的预测方法。

热门方向　递归神经网络 (RNN)，尤其是长短期记忆网络（LSTM）及其优化版本。

1. **RNN 神经网络**

   RNN（循环神经网络）是一种可以处理序列数据的神经网络，适用于时间序列预测、语音识别、自然语言处理等任务。RNN 的特点是网络中的节点不仅接收来自前一层的输入，还会接收来自上一个时间步的输出，这使得它能够捕捉到序列中的时间依赖关系 [14]。

2. **RNN 神经网络与标准神经网络的区别**

   RNN（循环神经网络）与传统神经网络的主要区别在于其结构和处理数据的方式。传统神经网络（如前馈神经网络）主要处理固定大小的输入数据，每次计算的输

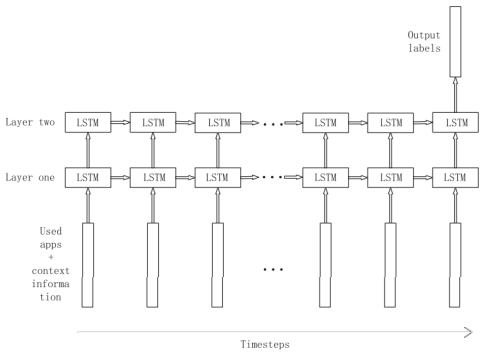

Fig. 5. The proposed LSTM model.

图 3: LSTM 原理图

出仅依赖于当前输入。而 RNN 则能够处理序列数据，其结构设计允许当前的输出不仅依赖于当前的输入，还与之前的输出（或状态）有关。通过这种方式，RNN 能够捕捉数据中的时间依赖性或序列特征 [14]。

相比之下，传统神经网络不能直接处理序列数据，因为它们缺乏记忆能力，无法记住先前的输入状态。而 RNN 通过在网络中建立“反馈”连接，能够存储并使用先前的信息，因此在处理动态时间序列或具有时序特征的数据时，RNN 更为有效。

3. **LSTM**

   LSTM 是 RNN 神经网络的增强版本，为了解决标准的 RNN 在处理长时间依赖关系时会遇到的梯度消失或爆炸的问题。LSTM 通过引入“记忆单元”来保留重要的信息，同时使用门控机制（输入门、遗忘门和输出门）来控制信息的流动 [14]。这使得 LSTM 能够更有效地捕捉长时间跨度的依赖关系，并在处理复杂的时序任务时表现出色。以下3是 LSTM 算法的工作原理图。

该领域自从 2009 年开始便有相关研究出现 [12]，但研究方法主要基于原生的概率模型。直到 2017-2018 年，由于神经网络的快速发展，LSTM 的方法成为了主流趋势。在大模型日趋成熟的当下，我们计划对原有的 LSTM 神经网络进行进一步的优化，从而实现更准确的用户软件预测和更加合理的内存资源调度。

3.1.4 我们的研究方向

1. 在原有通用模型的基础上，实现更加个性化的小模型，能够针对不同的用户，给出不同的调度方案 [15]。

2. 获取 PC 机上充足的上下文信息，从而能够实现 PC 机上更精准的预测 [15]。

3. 尝试联合模型的方法提高预测效果，比如 VLLM 结合 LSTM 的方法。

3.2 AI 与 OS 的交互

我们的原始目标是利用 LLM 预测并优化 OS 的内存管理，其中一种相对可行的粗粒度管理方案是，利用 LLM 的推理能力，预测用户文件系统层次的操作，从而优化内存管理。不论最后如何实现，这一过程都包含了用户、操作系统、LLM 三者之间的交互，在深入分析其中的信息流和控制流之前，我们需要确认 AI 与操作系统交互的可行性。经过一周的文献调研，接下来我们将从现有的工作出发，分析 AI 与 OS 的交互模式，寻找可以借鉴的地方，为我们建立自己的研究范式做准备。

对于电脑端，目前 AI 与操作系统的交互主要有两种模式：

- 智能体应用（AI/LLM-based agent）作为应用程序或服务
- AI 与 OS 融合

而对于移动端，虽然难以直接部署本地智能体，但也出现了许多模型优化使用体验的尝试。

3.2.1 智能体应用与 AIOS

在这种模式中，智能体被视为操作系统中的一个应用程序或服务，通过 API 接口与操作系统交互。按照部署方式，分为”**外挂式**”和**本地 AI 应用**。前者如微软 Windows Copilot，虽然能提供一定的智能服务，但其未能与系统深度融合，无法直接访问系统深层数据，已被降级为渐进式网络应用程序（PWA）；如果开放访问系统深层数据，又涉及安全问题，所以外挂式 AI 虽然落地快，但不利于长期发展，阻碍人工智能技术与操作系统的深度融合。

随着个人电脑的算力提升、模型优化和开源，本地部署并调用用户个人的智能”助理”是智能体应用的趋势。如旅行助理，根据用户输入的大概行程，可以结合用户偏好、天气预测，快速浏览大量的票务、酒店信息，个性化安排旅程。但这催生了新的 OS 问题，如**应用隔离**：LLM 作为独立应用运行，与其他系统组件交互有限；**资源消耗大**：LLM 需要大量计算资源和内存，尤其当 LLM 任务和非 LLM 任务或多个 LLM 任务并行时，给传统操作系统的资源管理带来压力。

这些问题的解决需要对传统 OS 进行革新，Rutgers University 的团队初步设计了服务于智能体应用的 AIOS（2024.3，如下图所示），并逐步付诸实践。

AIOS 系统由两个核心组件构成：AIOS 内核和 AIOS SDK。AIOS 内核作为操作系统内核之上的抽象层，负责管理代理所需的各种资源，模仿传统 OS 进行了模块划分（如下图），每个部分暂时挪用传统 OS 相关算法，且都需要单独验证和优化。AIOS SDK 专为代理用户和开发者设计，通过与 AIOS 内核交互，使他们能够构建和运行代理应用程序。

这个项目完全开源，并欢迎一切完善 AIOS 生态的工作。截至 2025 年 3 月，他们已经开发出了基于 LLM 的文件系统（LLM-based Semantic File System, LSFS），允许用户用自然语言索引文件。内存管理上还没有新动作。

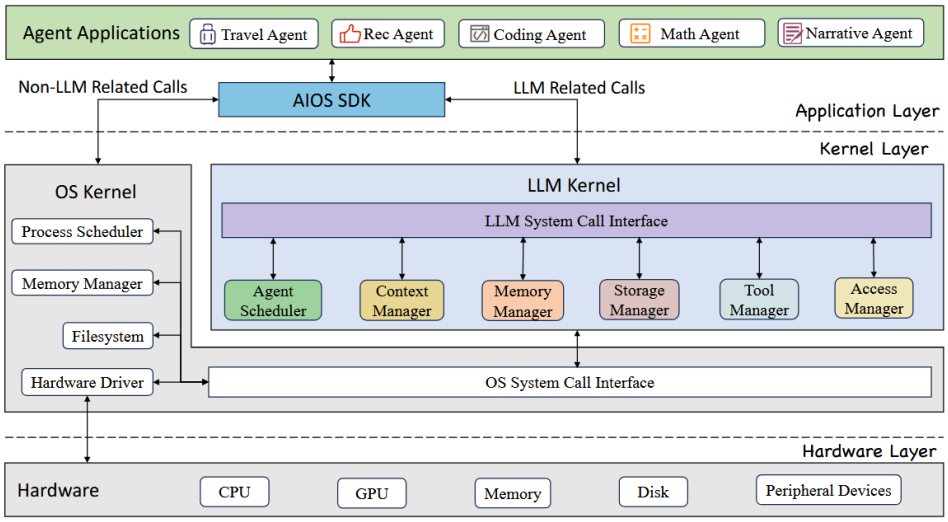

Figure 2: An overview of the AIOS architecture.

图 4: AIOS 的初步设计

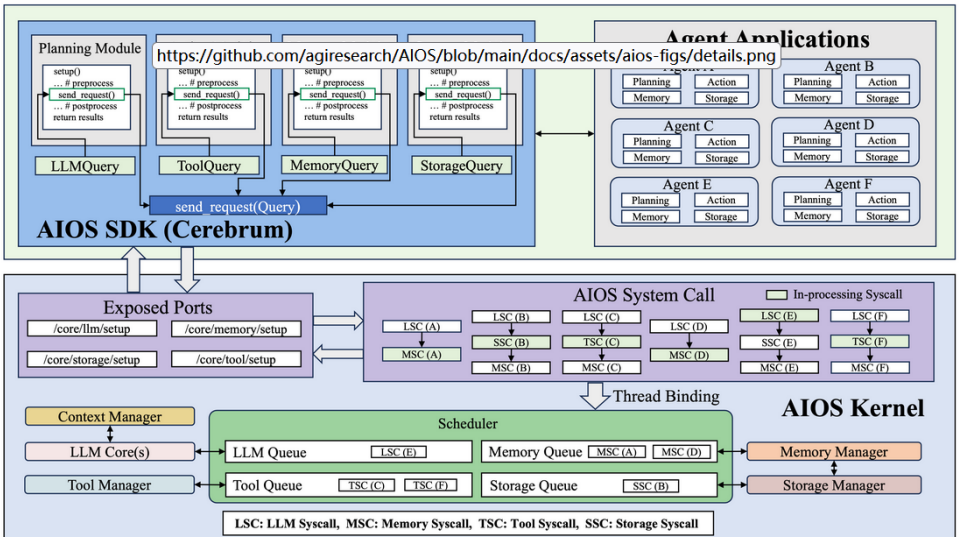

图 5: AIOS 中各模块之间的联系

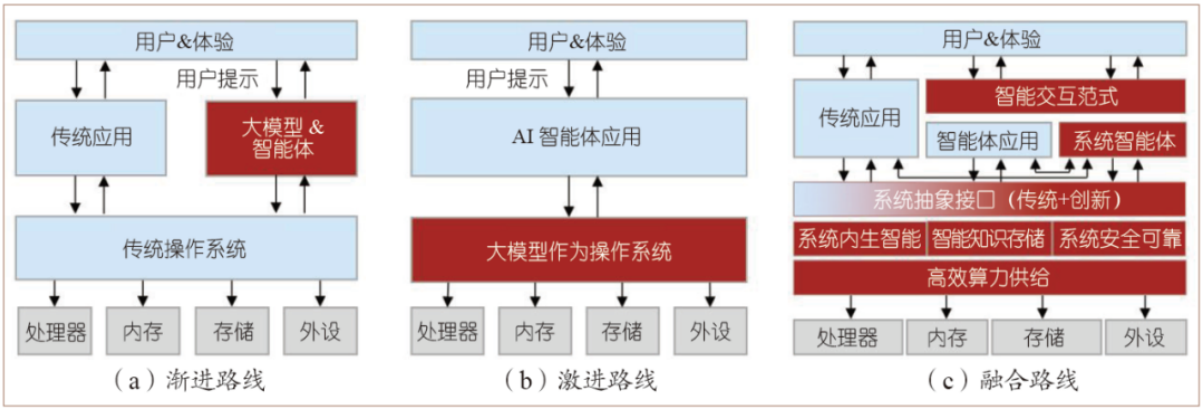

图 6: OS 与模型融合的不同发展路线

图中的路线三是一种更前沿的方法，通过” 模型-系统-芯片” 的全栈协同设计，构建针对 AI 模型优化的操作系统。与 AIOS 相比，更加强调 OS 与模型的融合，其特点包括：

1. **智能交互范式**：重塑用户与系统的交互方式，从直接面向用户转向以智能体为中介的交互模式，并需要解决 UI 理解、交互逻辑动态化和跨应用智能服务等挑战。
2. **创新系统抽象接口**：引入智能化抽象，支持从简单命令到复杂需求的灵活表达，并通过多层次系统服务接口平衡智能应用与系统实时性。
3. **系统内生智能**：构建操作系统级别的通用基础模型，实现跨应用智能协作，提供系统级智能体服务，建立持续学习机制满足多样化智能需求。
4. **智能知识存储**：从面向数据转向面向知识的存储设计，通过软硬协同和跨应用数据互通，提升模型在复杂场景下的智能表现。
5. **高效算力供给**：在模型、系统和硬件多个层面优化，包括模型轻量化、高效推理、计算卸载和异构计算架构等，以平衡算力、内存、功耗与智能水平。
6. **系统安全可靠**：实现数据和模型全生命周期保护，构建可信 AI 软件栈，并通过内生安全审计机制提升模型行为的确定性和可靠性。

3.2.2 模型原生操作系统 (Model-Native OS)

这些特点共同指向一个真正将大型语言模型深度集成到系统架构中的新一代操作系统，它不仅优化了智能应用的执行效率，还提升了整体智能水平，同时确保了安全与隐私保护。

融合路线或许是 AIOS 的最终形态，但还处于 0.5（接近于 0）阶段，考虑到小组作业的局限性，我们按下不表。况且 AIOS 中的部分工作，如 LSFS 的实现也在融合传统 OS 与模型特性，可见两种交互模式并非独立。

3.2.3 移动端的模型优化 OS 案例

4 运行环境

- 硬件要求
- 操作系统要求
- 大模型环境配置
  - vllm

5 前瞻性/重要性分析

6 相关工作

- 科学界相关工作
- 工业界相关工作
- **往年小组相关工作**
  - 主要内容/已经实现/优点/不足

7 参考资料

参考文献

[1] Openai blog / 技术报告中关于 beam search 与缓存管理的讨论, 2021.

[2] Ricardo Baeza-Yates, Fabrizio Silvestri, Di Jiang, and Beverly Harrison. Predicting the next app that you are going to use. pages 285 – 294, Shanghai, China, 2015. Aviate;Cold start problems;Derived features;Large scale experiments;Mobile app;Prediction techniques;Screen sizes;Usage experience;.

[3] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[4] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[5] X. Huang et al. Optimizing inference for large-scale neural networks. *Proceedings of the Machine Learning Optimization Conference*, 2021.

[6] R. Knauerhase et al. Memory pool optimization in high-performance computing. *High Performance Computing Journal*, 2019.

[7] Yafu Li et al. Revisiting cache mechanisms in neural sequence generation. *arXiv preprint arXiv:2106.12616*, 2021.

[8] Abhinav Parate, Matthias Bohmer, David Chu, Deepak Ganesan, and Benjamin M. Marlin. Practical prediction and prefetch for faster access to applications on mobile phones. pages 275 – 284, Zurich, Switzerland, 2013. App predictions;Communication device;Mobile operating systems;Practical systems;Prefetches;Processing capability;Real time content;Training overhead;.

[9] Jack W Rae, Anna Potapenko, Siddhant Jayakumar, Chloe Hillier, and Timothy Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2020.

[10] Mohammad Shoeybi, Mostofa Patwary, Ramesh Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. 2019.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[12] Hannu Verkasalo. Contextual patterns in mobile service usage. *Personal and Ubiquitous Computing*, 13(5):331 – 342, 2009. Business-oriented;Context-based;Context-oriented services;Mobile service;Panel studies;Temporal and spatial;Usage context;User context;.

[13] vLLM Project. vllm: A high-throughput and memory-efficient inference and serving engine for llms. 2025. `https://github.com/vllm-project/vllm`.

[14] Shijian Xu, Wenzhong Li, Xiao Zhang, Songcheng Gao, Tong Zhan, Yongzhu Zhao, Wei-wei Zhu, and Tianzi Sun. Predicting smartphone app usage with recurrent neural networks. In Sriram Chellappan, Wei Cheng, and Wei Li, editors, *Wireless Algorithms, Systems, and Applications*, pages 532–544, Cham, 2018. Springer International Publishing.

[15] Qichuan Yang, Zhiqiang He, Fujiang Ge, and Yang Zhang. Sequence-to-sequence prediction of personal computer software by recurrent neural network. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 934–940, 2017.