doi:10.11887/j.issn.1001-2486.25050035 http://journal.nudt.edu.cn

资源受限场景下基于算子感知的大模型推理张量卸载方法

张建锋,谢 栋,蹇松雷*,李 宝,王晓川,郭 勇,余 杰
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:在一些资源受限场景下,大语言模型的高效推理部署面临严峻挑战。当前主流的模型推理优化技术,虽然在一定程度上提高了模型推理效率,但是仍然存在部署粒度较为粗糙、推理精度较差等问题。根据不同算子对GPU亲和度不同的发现,提出算子感知张量卸载(operator-aware tensor offloading,OATO)方法。OATO能够提取算子的语义知识,并基于此设计了智能算子调度算法,可以生成全局最优模型部署方案。同时,将OATO方法集成进最新的大模型推理框架Llama.cpp中,实现了算子感知的张量卸载增强推理引擎OALlama.cpp。实验结果表明,相比于业内最先进的推理引擎Llama.cpp和FlexGen,OALlama.cpp在3种大模型上均取得最好的推理性能,尤其是在LlaMA3-8B模型GPU加载75%权重的场景下,OALlama.cpp的首词生成速度相比FlexGen和Llama.cpp提升近1倍。

关键词:大语言模型;资源受限;模型推理;算子GPU亲和度;算子感知张量卸载方法
中图分类号:TP181 **文献标志码**:A **文章编号**:1001-2486(2025)06-060-11

论文拓展

Operator-aware tensor offloading approach for large language model inference in resource-constrained scenarios

ZHANG Jianfeng, XIE Dong, JIAN Songlei, LI Bao, WANG Xiaochuan, GUO Yong, YU Jie*
(College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

Abstract: Efficient inference deployment of large language models faces severe challenges in resource-constrained scenarios. Although current mainstream inference optimization techniques have improved model inference efficiency to some extent, they still suffer from issues like coarse-grained deployment and poor inference accuracy. Based on the discovery that different operators exhibit varying degrees of GPU affinity, an OATO (operator-aware tensor offloading) approach was proposed. OATO could extract operators′semantic knowledge and used it to design an intelligent scheduling algorithm, which further yielded a globally optimal model-deployment plan. Meanwhile, the OATO approach was integrated into the latest large model inference framework Llama.cpp to implement an operator-aware tensor offloading enhanced inference engine, referred to as OALlama.cpp. Experimental results show that compared with the state-of-the-art inference engines Llama.cpp and FlexGen, OALlama.cpp achieves the best inference performance on three large models. Notably, in the scenario where 75% of the LlaMA3-8B model weights are loaded on the GPU, the first-token generation speed of OALlama.cpp is nearly doubled compared with FlexGen and Llama.cpp.

Keywords: large language models; resource constraints; model inference; GPU affinities of operators; operator-aware tensor offloading approach

近年来,生成式大语言模型(large language models,LLMs)已成为推动人工智能发展的核心力量。凭借其卓越的语言理解和生成能力,LLMs在语言相关领域表现出色,并在自动化编程、图像生成和个性化助理等多领域广泛应用[1-3]。基于Transformer架构[4]的LLMs,如生成式预训练模型(generative pre-trained transformer, GPT)系列[5-7]、LlaMA系列[8-9]、MiniCPM系列[10-11]以及DeepSeek系列[12-14]等,进一步重塑了机器学习和自然语言处理的格局[15-17]。

然而,这些模型通常规模庞大,只能部署在配备大量昂贵服务器级图形处理器(graphics

收稿日期:2025-05-24
基金项目:国家自然科学基金创新群体资助项目(62421002);国防科技大学自主科研基金资助项目(24-ZZCX-JDZ-07)
第一作者:张建锋(1984—),男,陕西宝鸡人,副研究员,博士,E-mail:jfzhang@nudt.edu.cn
***通信作者**:蹇松雷(1991—),女,贵州遵义人,副研究员,博士,硕士生导师,E-mail:jiansonglei@nudt.edu.cn
引用格式:张建锋,谢栋,蹇松雷,等. 资源受限场景下基于算子感知的大模型推理张量卸载方法[J]. 国防科技大学学报,2025,47(6):60-70.
Citation:ZHANG J F, XIE D, JIAN S L, et al. Operator-aware tensor offloading approach for large language model inference in resource-constrained scenarios[J]. Journal of National University of Defense Technology, 2025, 47(6): 60-70.

processing unit, GPU)的数据中心[18]。与此同时,随着数据隐私保护、模型定制化和降低推理成本的需求增加,本地部署 LLMs(如在配备消费级 GPU 的个人计算机上)成为一种新的趋势[18]。与数据中心部署所需高吞吐量不同,本地部署更侧重于处理小批量数据时的低延迟需求[19-20]。

与此同时,由于 LLMs 的规模、复杂性以及对内存和计算资源的高需求,消费级 GPU 上的部署仍面临较大挑战。特别是,LLMs 采用自回归迭代方式生成文本,每次生成都需要访问包含数千亿参数的整个模型,这使得推理过程受到 GPU 内存的严格限制,尤其是在资源受限的消费级 GPU 场景中。目前,常用的解决方案主要包括模型压缩和模型卸载。

模型压缩是降低模型内存占用、加速大语言模型推理的一种重要思路,当前模型压缩的方法主要有量化、剪枝或稀疏化等。量化技术[21-24]主要通过降低模型中的数据精度,以减少内存占用和计算量,如英伟达的 TensorRT[24]推理优化器在图像识别大语言模型的推理过程中,能够将原本 32 位的浮点数精度降低至 16 位甚至 8 位,从而将高精度的复杂计算转化为低精度运算。剪枝或稀疏化[19,25-28]则是从模型结构入手,去除对模型输出影响较小的部分,如 Dejavu 证明了大模型存在上下文稀疏性,并使用小的预测模型对模型的稀疏性进行提取预测,大大减少模型推理的计算量,以提高推理速度[25]。虽然模型压缩技术可以减小模型规模,但仍然存在两大问题:一是即使经过深度压缩,模型对消费级 GPU 而言仍过于庞大;二是压缩不可避免地会降低模型推理精度。

模型卸载[29-36]也是大语言模型推理优化领域的另一种重要思路,尤其是在一些内存受限的场景下,发挥尤为出色。这种方法不改变模型本身,在 GPU 内存不充足的情况下,通过借用中央处理器(central processing unit, CPU)内存完成推理,例如,Llama. cpp[29]采取了分层卸载的方法,将一部分层卸载到 GPU,一部分层卸载到 CPU,从而实现资源受限环境中 CPU 和 GPU 混合推理。斯坦福大学的研究者在 FlexGen[31]中提供了一种方法,在考虑 GPU、CPU 和磁盘可用硬件资源限制的情况下,实现了不同的工作负载卸载方式。类似地,苹果公司的研究者将模型参数存储在更大容量的闪存中,通过需要时再将其传输至速度更快的内存的方式[20],提升了推理执行的效率。上述这些混合推理方法都给资源受限环境中运行大语言模型提供了很好的思路和解决方案,然而,这些方法受限于外围组件互联高速总线(peripheral component interconnect express, PCIe)互联速度和 CPU 计算能力,导致推理效率低下,延迟较高。此外,由于 LLMs 中每层的算子具有不同的特点:有些算子内存占用小但计算量大,而有些则相反,简单地按层分配模型无法充分利用 GPU 的强大计算能力。

理想状态下,模型卸载需依据 LLMs 中每层算子的内存需求与计算特性,将其按需部署至 CPU 或 GPU,从而充分发挥硬件效能。为此需攻克三项关键技术挑战:①算子特性表征与分析,需要深入理解 LLMs 中不同算子的内存需求以及在不同设备上的性能表现;②智能调度算法设计,需要开发自适应算法来决定如何根据算子特性进行放置;③透明化部署机制,需要构建对开发者透明的算子放置机制,无须开发人员手动干预。

为解决上述挑战,针对资源受限的大模型推理场景,在研究分析了大语言模型算子特性的基础上,提出了算子感知张量卸载(operator-aware tensor offloading, OATO)方法,并将其集成于 Llama. cpp 中,构建了算子感知增强的大模型推理引擎OALlama. cpp(OATO-enhanced Llama. cpp)。通过在不同内存限制和提示长度场景下的测试表明,相比于 Llama. cpp 和 FlexGen[31]等典型推理引擎,OALlama. cpp 均显著提升了 LLMs 在资源受限环境中的推理效率。

1　预备知识

在大语言模型推理过程中,每层的算子操作基于不同的计算和存储需求,主要分为需要加载权重、需要中间张量以及需要缓存的算子这三种类型,以下是对其具体情况的详细阐述。

需要加载权重的算子:权重是大语言模型的核心参数,它记录了模型在训练过程中学习到的知识和模式。对于需要加载权重的算子,其在推理开始前,必须从存储设备(如硬盘、显存等)中读取预训练好的权重参数。以 Decoder-only 架构中的注意力机制和前馈神经网络层为例,在计算注意力分数、进行矩阵乘法运算等操作时,都需要加载相应的权重矩阵。这些权重参数加载过程可能会受到存储设备读写速度的影响,从而对推理速度产生一定限制。

需要中间张量的算子:中间张量是算子在计算过程中产生的临时数据,用于记录中间计算结果,以便后续操作使用。这类算子在执行计算任务时,首先会生成中间张量。然而,中间张量具有

临时性,在完成其对应的计算任务,即不再被后续计算步骤所需要时,就会被释放。这是因为中间张量通常占用大量的内存空间,如果不被及时释放,会导致内存资源的过度消耗,甚至引发内存溢出等问题,影响模型推理的正常进行。

需要缓存的算子:需要缓存的算子主要是为了优化推理性能,减少重复计算。在推理过程中,某些算子的计算结果在后续步骤中可能会被多次使用,为避免重复计算,将计算结果进行缓存。在自注意力机制的计算中,键值缓存(key-value cache, KV Cache)机制可以保存已经计算好的键值矩阵,当处理后续输入序列的不同位置时,如果需要用到相同的键值对计算注意力分数,可直接从缓存中读取,而无须重新计算。

2 动机分析

2.1 模型内存占用瓶颈分析

为了了解大语言模型推理过程中内存占用的相关情况,对 MiniCPM3 – 4B、LlaMA3 – 8B 和 QWQ – 32B 模型在推理过程中的内存占用情况进行了分析,由于面向的资源受限场景以短文本交互为主要形式,因此将文本长度的上限统一设定为 1 024。模型内存占用情况如图 1 所示,在 MiniCPM3 – 4B 模型的推理过程中,模型权重、键值缓存和其他(中间张量等)的内存占用分别为 8 129.66 MB、775 MB、155.74 MB。对于 LlaMA3 – 8B 和 QWQ – 32B 而言,模型权重、键值缓存和其他的内存占用分别为 15 137.02 MB、128 MB、258 MB和 62 494.27 MB、256 MB、1 792 MB。通过上述对比可以看出,在模型推理进程中,尽管键值缓存与其他内存部分在内存占比上存在差异,但模型权重的内存始终都占据了几乎所有的份额,其中 MiniCPM3 – 4B 中的模型权重内存占比 89.7%,LlaMA3 – 8B 中的模型权重内存占比 97.5%,QWQ – 32B 中的模型权重内存占比 96.8%。

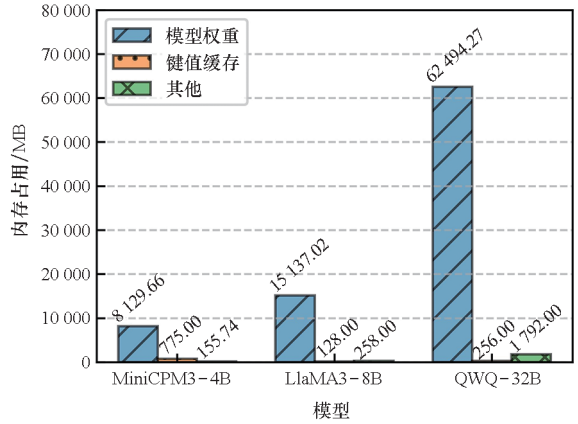


图 1 模型内存占用情况

Fig. 1 Model memory usage

发现 1:在资源受限的大模型推理场景下,当提示长度不超过 1 024 时,模型权重占据内存空间的主体部分,因此,优化权重张量的放置策略具有关键意义。

2.2 算子在不同设备运行对模型推理的影响

为了深入了解模型推理过程中 CPU 算子和 GPU 算子的性能表现情况,提取了不同模型在推理过程中算子级别的语义知识,表 1 展示了 MiniCPM3 – 4B、LlaMA3 – 8B 和 QWQ – 32B 模型在 GPU 和 CPU 上单独运行时的不同算子的性能表现,其中 C 指的是 CPU,G 指的是 GPU。可以明显观察到,无论是哪个模型,相较于 GPU 上的算子计算延迟,同样的算子在 CPU 上的计算延迟会有上百甚至上千倍的增加,因此,将算子卸载到 CPU 上会严重影响推理延迟。

表 1 不同模型推理场景下算子延迟

Tab. 1 Operator latency of different models in inference scenarios

单位:μs

算子	MiniCPM3 – 4B		LlaMA3 – 8B		QWQ – 32B	
	G	C	G	C	G	C
mul-mat	11	13 902	8	55 922	13	110 469
add	4	246	3	1 020	5	964
Mul	4	354	3	517	5	817
Norm	4	289	3	459	5	557
Softmax	4	273	3	274	6	322

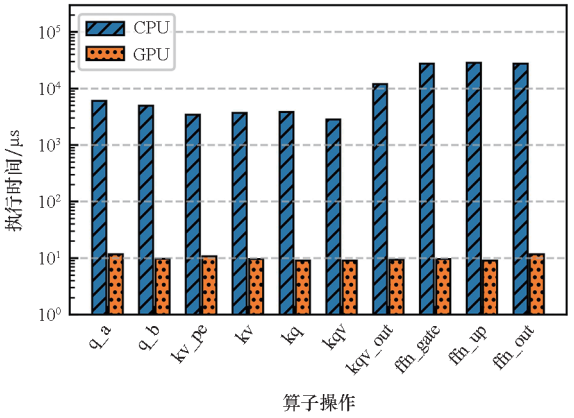
此外,在三个模型中,矩阵乘法(mul-mat)算子 CPU 执行时间都相对较长,如 QWQ – 32B 模型下 CPU 执行矩阵乘法算子耗时约 110.5 ms,而 GPU 执行时间则短很多。这是因为矩阵乘法计算量极大,CPU 受限于其架构,串行处理能力在应对大规模矩阵运算时效率低下;而 GPU 拥有大量计算核心,能并行处理矩阵元素计算,大幅缩短执行时间。同时,可推断出在 CPU 设备上,mul-mat 算子因计算量极大,成为 CPU 计算的主要瓶颈,是提升整体运算效率的亟待突破的关键。

发现 2:在 CPU 参与模型推理的场景中,mul-mat 算子已成为影响模型推理延迟的核心瓶颈,其单次推理耗时显著高于其他类型算子。

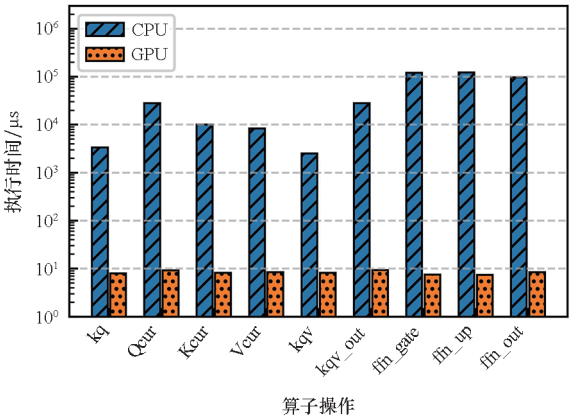
2.3 矩阵乘法算子操作性能分析

为深入剖析 mul-mat 算子的性能特性,对其

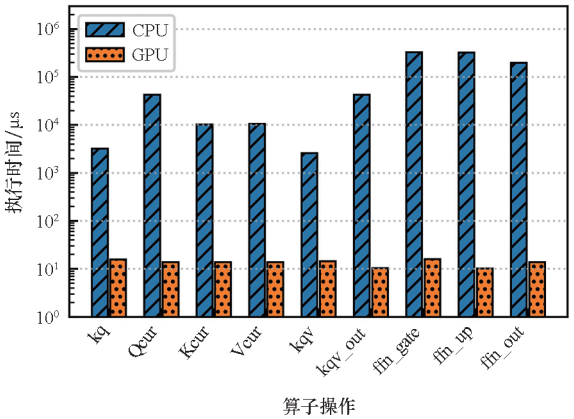
内部不同矩阵乘法算子操作进行了细致拆解与分析。如图 2 所示,该图清晰呈现了 CPU 与 GPU 设备上不同 mul-mat 算子操作的延迟表现差异。数据显示,MiniCPM3 – 4B、LlaMA3 – 8B 和 QWQ – 32B 模型的 mul-mat 算子在 GPU 平台的执行延迟分别约为 11 μs、8 μs 和 13 μs,且各算子间波动范围极小,体现出 GPU 计算资源的高效与稳定;反观 CPU 平台,不同 mul-mat 算子的延迟表现呈现显著分化,在 MiniCPM3 – 4B、LlaMA3 – 8B 和 QWQ – 32B 模型中,其数值区间跨度分别为 2 822.23 μs 至 28 676.96 μs、2 510.11 μs 至 122 479.01 μs和 2 574.95 μs 至 327 259.25 μs。这一结果充分表明,将不同 mul-mat 算子从 CPU 迁移至 GPU 时,所带来的性能增益存在显著差异。



(a) MiniCPM3 – 4B



(b) LlaMA3 – 8B



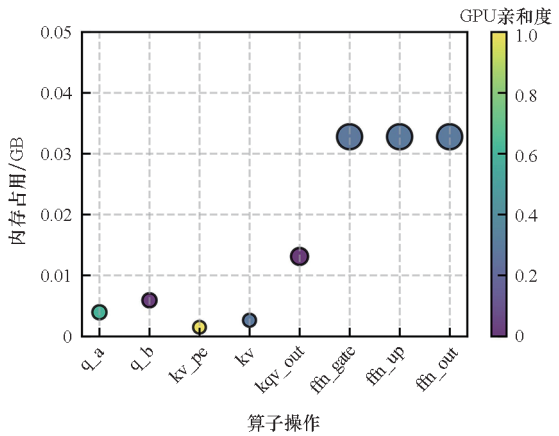
(c) QWQ – 32B

图 2　不同矩阵乘法算子操作的延迟

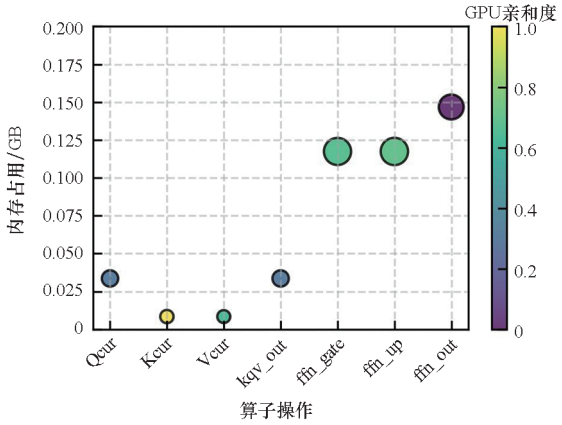
Fig. 2　Latency of varying mul-mat operations

为深入探究 GPU 加速与权重内存之间的内在关联,对需要加载权重张量的 mul-mat 算子进行更进一步分析,以 mul-mat 算子在 CPU 和 GPU 平台的执行时间差值作为算子加速度量指标,继而将该差值除以算子运行所需权重张量的内存容量,并通过归一化处理得到单位内存加速比。这一量化指标精准刻画了单位内存对推理速度的优化效能,即数值越高,意味着将算子卸载至 GPU 执行后,推理速度的提升幅度越显著。以下将该指标正式定义为算子的“GPU 亲和度”,用以系统性评估算子与 GPU 计算资源的适配程度。

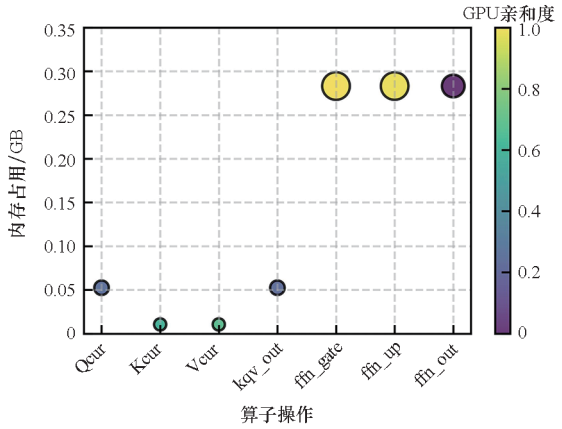
图 3 分别展示了在 MiniCPM3 – 4B、LlaMA3 – 8B 和 QWQ – 32B 三个模型架构下,不同 mul-mat 算子操作的内存占用、时间加速、GPU 亲和度等特征,其中圆圈大小表示算子在 CPU 和 GPU 设备上的运行延迟的差值,圆圈越大说明算子延迟差距越大;圆圈的颜色表示 GPU 亲和度,颜色越浅说明算子 GPU 亲和度越高。从图中可以明显看出,无论哪个模型,算子的 GPU 亲和度都呈现显著差异。这一发现为算子卸载策略的制定提供了重要依据。以图 3(a)为例,将 q_a 算子操作和 q_b 算子操作进行比较,前者的 GPU 亲和度为 0.4,而后者仅为 0.1。这表明在内存资源受限的场景中,优先将 q_a 算子操作卸载至 GPU 执行,能够相较于将 q_b 算子操作卸载至 GPU 执行实现更为显著的推理加速效果。该结论对于优化计算资源分配、提升模型推理效率具有重要的实践指导意义。



(a) MiniCPM3 – 4B



（b）LlaMA3－8B



（c）QWQ－32B

图 3　算子的 GPU 亲和度

Fig. 3　GPU affinities of operators

发现 3：对于不同的 mul-mat 算子操作，它们的 GPU 亲和度是不同的，优化 GPU 亲和度更高的算子对模型推理的性能提升会更大。

3　系统设计

面向资源受限场景设计了一个基于算子感知的模型张量卸载方法，并将该方法集成于 Llama. cpp之上，从而实现了基于算子混合卸载的模型推理引擎，即算子感知增强的大模型推理引擎 OALlama. cpp。OALlama. cpp 充分利用 2. 3 节中的发现 3，即 mul-mat 算子操作具有 GPU 亲和度不同的特点，通过对模型进行算子级的重新部署，即使是超出内存容量的大语言模型，也能实现更加高效的推理。简单来说，OALlama. cpp 对 mul-mat 算子操作的 GPU 亲和度进行排序，将 GPU 亲和度高的算子部署在 GPU 中，其余部署在 CPU 中。对于部署在 CPU 上的算子，直接在 CPU 上计算，无须将算子移动到 GPU 上。

3. 1　系统概览

图 4 展示了 OALlama. cpp 的系统概览。它主要由三个模块组成：算子解析器、算子调度器和算子放置器。其中：算子解析器用于提取大语言模型推理过程中算子的语义知识；算子调度器基于算子语义知识，对算子进行 GPU 亲和度排序；算子放置器用于简化算子管理并实现自动化的算子放置。

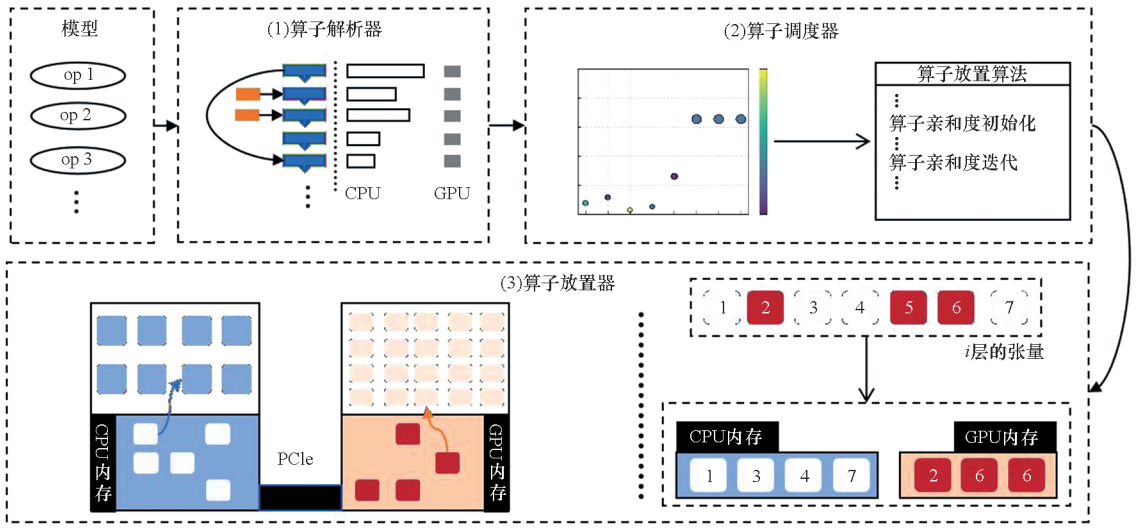


图 4　OALlama. cpp 系统概览

Fig. 4　System overview of OALlama. cpp

给定一个大语言模型，算子解析器会提取模型中每一层算子的语义知识，即量化算子操作的计算量、所需张量的大小（内存占用）及其相关性等。利用这些语义知识，算子调度器将会生成每个算子的 GPU 亲和度，并对其进行排序，从而规划算子的优化部署方案。之后，算子放置器将通过维护一张表来区分算子的后端，完成算子在 CPU 和 GPU 后端的自动化卸载。下述为具体的组件分析。

3.2　算子解析器

算子解析器负责采集算子在 CPU/GPU 上执行的计算时间、张量内存占用及跨设备通信延迟,为资源受限场景下的异构部署提供数据支撑。图 5展示了算子流示例,其中,橙色表示算子被卸载到 GPU,蓝色表示算子被卸载到 CPU。$\boldsymbol{A}$ 和 $\boldsymbol{I}$ 为中间张量,$\boldsymbol{W}$ 为模型权重张量。系统中存在三种算子连接方式(CPU-CPU、GPU-GPU、CPU-GPU),需考虑以上连接方式并量化中间结果传输产生的额外延迟(如 GPU→CPU 的数据迁移)。此外,像 mul-mat2 算子这种需要加载模型权重 $\boldsymbol{W}_1$ 进行张量计算的算子称为目标算子。仅加载中间张量 $\boldsymbol{I}_1$ 进行计算的 mul-mat1 算子,以及不需要其余张量就能执行算子操作的算子(如 scale 算子)或 add 算子这种只需要加载中间张量 $\boldsymbol{I}_2$ 执行算子操作的都归为非目标算子。

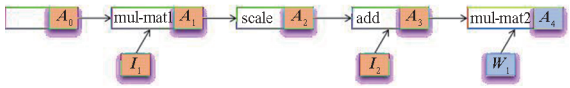


图 5　算子流示例

Fig. 5　An example of operator flow

基于 2.1 ~2.2 节的发现 1 和发现 2,本系统聚焦于需要加载模型权重张量的 mul-mat 算子(目标算子),因其主导大语言模型推理的内存消耗(权重张量)和计算延迟。因此,算子解析器在推理预加载阶段,首先根据算子在大语言模型推理迭代(一轮前向传播)中是否为目标算子进行分类。然后,采集目标算子的相关语义知识,包括在所有支持的后端设备执行计算所需的时间(指加载算子所需张量到指定后端寄存器的时间以及张量计算的时间之和),以及对应权重张量所需的内存大小等。此外,算子解析器仅在模型首次部署的预加载阶段完成相关语义知识的采集工作,后续所有推理过程均无须重复执行知识获取操作。因此,该环节的耗时特性不会对后续的模型推理产生影响。

这些参数构成算子调度器的优化依据,通过量化目标算子的异构执行成本,生成全局最优的算子部署方案。该机制有效平衡了权重密集型计算的设备资源分配与通信开销的矛盾。

3.3　算子调度器

在资源受限场景中,内存资源的有限性成为制约大语言模型高效推理的关键瓶颈。为实现内存资源的全局优化配置与算子的合理部署,设计一种高效的智能算子调度算法势在必行。基于 2.3 节发现 3 中揭示的 mul-mat 算子操作 GPU 亲和度差异现象,传统随机放置或按层划分的算子部署策略存在显著弊端。该类策略可能导致高 GPU 亲和度算子被错误分配至 CPU,致使 GPU 计算资源利用率不足,进而增加系统推理延迟。而简单地将高 GPU 亲和度算子部署至 GPU,又会因 CPU 与 GPU 间频繁的数据传输产生高额开销,同样会对系统性能造成负面影响。因此,提出的智能算子调度算法聚焦两大核心目标:其一,最大限度利用有限的 GPU 内存资源,优先存储具有最高 GPU 亲和度的算子;其二,充分考量数据传输对算子 GPU 亲和度的影响,有效平衡计算与数据传输开销。

提出的智能算子调度算法通过三个核心步骤实现全局最优的算子放置方案。算法 1 展示了算子调度器中使用的智能算子调度算法的核心逻辑。

首先,基于算子解析器获取的计算延迟参数,量化评估各算子单位 GPU 内存资源可获得的延迟优化效益:$B_i = (T_{ci} - T_{gi})/M_i$,其中 T_{ci}、T_{gi} 分别表示算子 i 在 CPU、GPU 的执行延迟,M_i 为算子 i 的权重张量内存占用量。进行最小最大归一化处理:$G_i = (B_i - B_{\min})/(B_{\max} - B_{\min})$,将 B 映射成 0 ~1区间的数值,以该数值来表征算子的 GPU 亲和度,建立内存 - 效率的边际效益评估模型。如算法 1 第 2 ~5 行所示,系统对所有算子按 G 值降序排列,建立初步优先级队列。

其次,为消除传输延迟对计算增益的负向影响,构建迭代式亲和度修正机制。针对每个候选算子,引入跨设备张量加载延迟 T',更新延迟优化效益的计算公式为:$B_i = (T_{ci} - T_{gi} - T'_i)/M_i$,对原始计算结果进行修正。通过循环执行“计算增益评估→传输延迟补偿→优先级重排”的迭代过程(算法 1 第 6 ~ 15 行),动态调整算子部署顺序,确保优先级排序能够真实反映网络传输约束下的综合优化效益。

最后,为了预留空间给框架自身开销、潜在内存溢出(out of memory, OOM)风险或其他系统级需求,以 GPU 显存容量的 90% 为资源约束边界,选择迭代优化后 G' 值最高的算子子集部署于 GPU 设备,将其余算子分配至 CPU。该阈值设置既保证显存安全余量,又能最大化资源利用率(算法 1 第 16 ~31 行)。最终生成的部署方案在计算效率、传输开销、内存占用三个维度实现帕累托最优,有效解决了传统方法在硬件异构环境下容易出现的局部最优问题。

算法总述:为了生成最优的放置方案,它会迭代式地为算子的 GPU 亲和度排序,直到找出算子最佳的 GPU 亲和度序列。该算法在整个搜索过程中主要跟踪算子是否存在 CPU 和 GPU 之间的数据传递。在算法的每次迭代中,它会相应地更新算子的 GPU 亲和度。

算法 1　智能算子调度算法

Alg. 1　Smart operator scheduling algorithm

输入:系统 GPU 内存容量 gpu_cap
　　算子列表 operators
　　可用 GPU 内存容量 available_gpu_memory
输出:算子列表 operators

```
Function Operator Placement(gpu_cap, operators, available_gpu_memory):
//计算初始 GPU 亲和度
for op in operators:
    op.B = (op.Tc-op.Tg)/op.M
end for
operators = normalize_and_sort_operators(operators)
//迭代优化
while true:
    prev_affinities = [op.gpu_affinity for op in operators]
    for op in operators:
        op.B = (op.Tc-op.Tg-op.T′)/op.M
    end for
    operators = normalize_and_sort_operators(operators)
    if [op.gpu_affinity for op in operators] = = prev_affinities:
        break
    end if
end while
//确定算子放置后端
memory_limit = 0.90 * available_gpu_memory
total_mem = 0    gpu_count = 0
for op in enumerate(operators):
    total_mem + = op.m
    if total_mem > memory_limit:
        break
    end if
    gpu_count + = 1
end for
for i in range(gpu_count)://放置在 GPU
    operators[i].backend = "GPU"
end for
for i in range(gpu_count, len(operators))://放置在 CPU
    operators[i].backend = "CPU"
end for
return operators
```

3.4　算子放置器

细粒度的算子放置策略虽然能够提升系统资源利用效率,但不可避免地增加了系统内存管理的复杂度。由于该策略涉及大量精细化操作,使得系统难以精准追踪每个目标算子对应的内存空间。为有效应对这一挑战,算子放置器的核心任务聚焦于解决目标算子的合理放置问题。

目标算子-后端设备映射关系如图 6 所示,对于目标算子,算子放置器着重关注其权重张量、中间张量及操作类型这三个关键信息。其中,权重张量作为模型参数的重要载体,具有后端设备类型和内存大小等属性,并直接映射至实际物理设备。因此,研究目标算子的放置问题本质上是探讨权重张量如何放置。

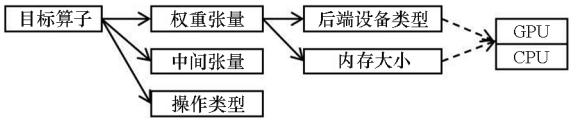


图 6　目标算子-后端设备映射关系

Fig. 6　Mapping between target operator and backend device

在实际的算子部署过程中,算子放置器首先通过算子解析器获取全局最优的算子放置方案,进而构建并维护一张算子-后端设备映射表。该映射表清晰呈现了权重张量与后端设备之间的对应关系,成为系统识别权重张量物理位置的关键依据。通过动态修改这张映射表,系统能够灵活决定权重张量的实际部署设备,并在相应后端设备上准确加载张量,顺利完成计算任务。

4　性能评测

4.1　实验环境及实验设置

性能评测的云服务器配置参数:CPU 为英特尔至强(Intel Xeon)系列的 Silver 4314 处理器,配备了 128 GB 的内存;GPU 为英伟达 GeForce RTX 4090,拥有 24 GB 的 GPU 内存,GPU 通过 PCIe Gen5 x16 接口与主板相连。

将 OATO 方法集成进现有推理引擎 Llama. cpp 中,实现了算子感知增强的大模型推理引擎 OALlama. cpp。

实验中,将 OALlama. cpp 与行业最先进的推理引擎 Llama. cpp 和 FlexGen 进行了比较。其中 Llama. cpp 是资源受限的本地场景中应用最广泛的推理引擎。在模型选择方面,挑选了 MiniCPM-4B、LlaMA3-8B 和 QWQ-32B 三个具有代表性的大语言模型。这三个模型在架构设计上各具特

色,在参数量级上存在显著差异,并且是国内外大语言模型领域的典型产品,能够充分覆盖不同规模和类型的模型场景。需要说明的是,MiniCPM3 – 4B 模型的注意力层采用多头潜在注意力(multi-head latent attention, MLA)架构,与 FlexGen 中的部分加速方式存在兼容性问题,导致无法在 FlexGen 框架下进行有效推理,因此未纳入该模型在 FlexGen 框架下的评估。

鉴于本方法聚焦于资源受限环境中的短文本应用场景,在整体实验设计中,输入输出均以短文本为主。具体而言:在端到端的性能评估实验(4.2 节)和不同资源受限情况下的性能评估实验(4.5 节)中统一设置提示长度为 64,生成长度为 32,模拟资源受限环境下的典型短文本交互场景;在探究不同提示长度下的预填充阶段性能评估实验(4.3 节)中设置提示长度为 32、64、128、256 和 512,通过梯度化的参数设置,分析预填充阶段在不同文本长度下的性能表现;在评估不同生成长度下的解码阶段性能评估实验(4.4 节)中设置生成长度为 32、64 和 128,针对性地考察解码阶段在不同文本生成需求下的效率与效果。

4.2　端到端的性能评估

为了评估不同框架在模型推理中的性能表现,针对 FlexGen、Llama. cpp 和 OALlama. cpp 展开了一系列对比实验。实验选取了 LlaMA3 – 8B 和 QWQ – 32B 这两个具有代表性的模型,对它们的端到端推理性能进行了深入分析。

为了更贴近实际应用中的资源受限场景,在对 LlaMA3 – 8B 模型进行推理时,将其权重的 50% 卸载至 GPU 进行计算。实验结果清晰地显示,OALlama. cpp 在这两类模型上均展现出了卓越的性能优势。不同模型端到端推理性能如图 7 所示,从中可以直观地看到,与 Llama. cpp 相比,OALlama. cpp 在 LlaMA3 – 8B 和 QWQ – 32B 模型上的推理速度分别提升了 17.4% 和 6.3%。而与 FlexGen 相比,这一优势更加明显,OALlama. cpp 在 LlaMA3 – 8B 和 QWQ – 32B 模型上的推理速度分别实现了 46.2% 和 36.2% 的显著提升。这是由于 FlexGen 采用将 CPU 端权重加载至 GPU 执行计算的策略,而 Llama. cpp 通过将部分层权重卸载至 CPU 本地计算,有效降低了权重加载延迟。相比之下,OALlama. cpp 因选用了 GPU 亲和度更高的算子,从而实现了更优的推理性能。

4.3　不同提示长度下的预填充阶段性能评估

为了研究预填充阶段 OALlama. cpp 的性能

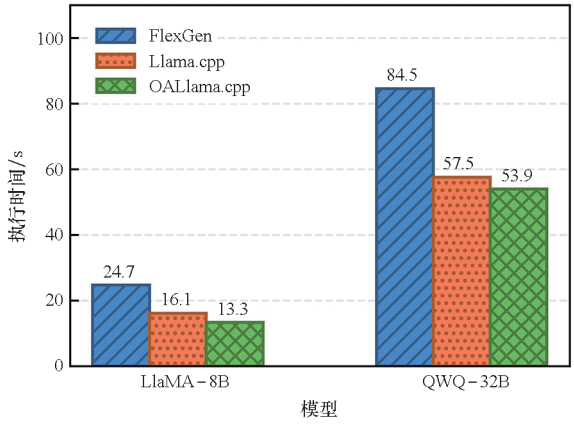


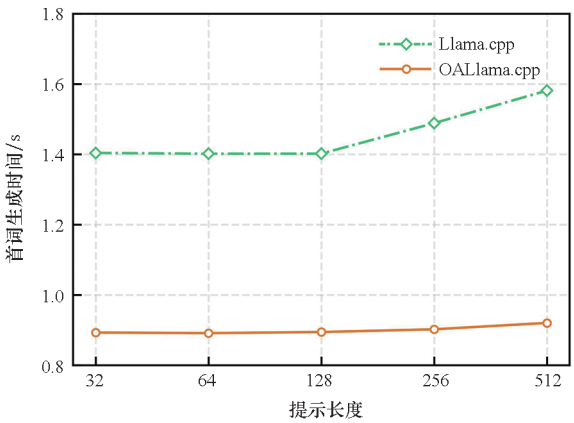
图 7　不同模型端到端推理性能

Fig. 7　End-to-end inference performance of varing models

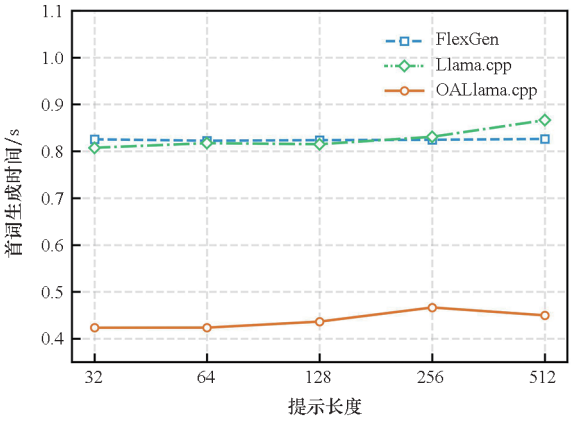
表现,对不同场景中模型推理的首词生成时间(time to first token, TTFT)进行了实验,以 MiniCPM3 – 4B、LlaMA3 – 8B 和 QWQ – 32B 模型为研究对象,收集首词生成时间,对 FlexGen、OALlama. cpp 和 Llama. cpp 的推理效果进行对比分析,此外,由于本实验场景 GPU 内存大于 MiniCPM3 – 4B 和 LlaMA3 – 8B 模型的需求,为了模拟资源受限场景,分别设置加载 25% 的 MiniCPM3 – 4B 模型和 75% 的 LlaMA3 – 8B 模型。

不同提示长度下预填充阶段的推理性能如图 8 所示,在 MiniCPM3 – 4B 模型推理仅能加载 25% 权重的情况下,在较长提示长度范围内,OALlama. cpp 在首词生成的速度上显著优于 Llama. cpp,最多有 41.8% 的速度提升,Llama. cpp 的首词生成时间随提示长度增加而明显增长,而 OALlama. cpp 稳定性好,受提示长度影响小。

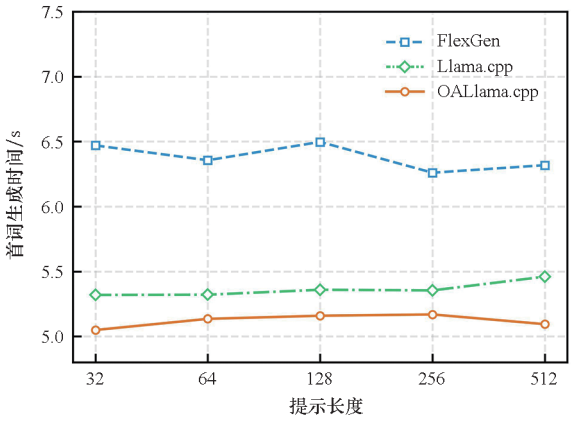
在 LlaMA3 – 8B 模型推理加载 75% 权重的情况下,OALlama. cpp 在首词生成的速度上依旧领先,大约能有接近 1 倍的提升。FlexGen 和 Llama. cpp 相对接近,但 Llama. cpp 随提示长度增加首词生成时间有一定上升,FlexGen 相对稳定。



(a) MiniCPM3 – 4B



（b）LlaMA3 －8B



（c）QWQ －32B

图 8　不同提示长度下预填充阶段的推理性能

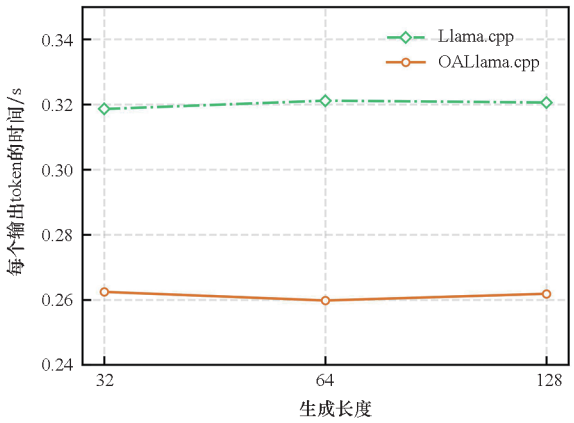
Fig. 8　Inference performance of prefill stage with varying prompt length

在 QWQ －32B 模型推理的不同提示长度下，OALlama. cpp 在首词生成的速度上相对更快，Llama. cpp 次之，FlexGen 最慢。且随着提示长度增加，OALlama. cpp 和 FlexGen 的 TTFT 有上升也有下降，但 FlexGen 波动相对较大，Llama. cpp 和 OALlama. cpp 变化较为平缓。尤其是，相较于 FlexGen，OALlama. cpp 实现了 28% 的 TTFT 提升。

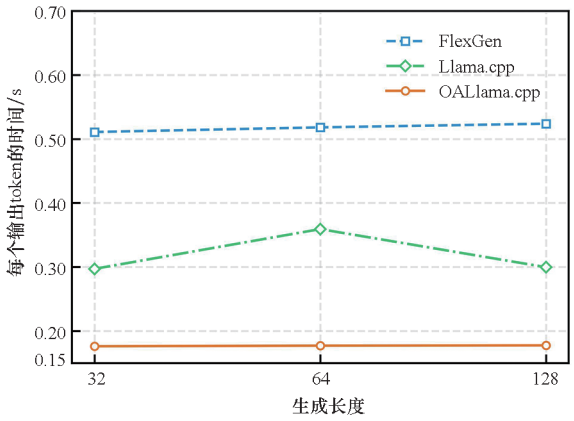
随着提示长度的增加，所有推理引擎的性能整体都是下降的，但 OALlama. cpp 仍优于其他引擎。这是由于 OALlama. cpp 将对 CPU 运算影响较大的算子调度至 GPU 执行计算，尽管保留在 CPU 端的算子仍会受提示长度影响，但该影响已被控制在相对最小范围内。

4.4　不同生成长度下的解码阶段性能评估

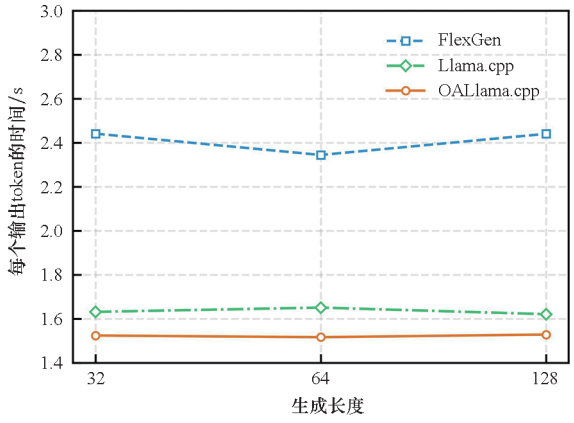
为了研究解码阶段 OALlama. cpp 的性能表现，对不同场景中模型推理的每个输出 token 的时间（time per output token，TPOT）进行了实验。以 MiniCPM3 －4B、LlaMA3 －8B 和 QWQ －32B 模型为研究对象，收集每个输出 token 的时间，对 FlexGen、OALlama. cpp 和 Llama. cpp 的推理效果进行对比分析，实验结果如图 9 所示。



（a）MiniCPM3 －4B



（b）LlaMA3 －8B



（c）QWQ －32B

图 9　不同输出长度下解码阶段的推理性能

Fig. 9　Inference performance of decode stage with varying output length

从三张图呈现的 TPOT 数据来看：在不同生成长度条件下，相比较 FlexGen 和 Llama. cpp，OALlama. cpp 的 TPOT 始终处于低位，且表现最稳定。这些结果充分说明即使在解码阶段，CPU 计算量很小的情况下，OALlama. cpp 的算子放置策略仍然有效降低了每个输出 token 的时间。

4.5　不同资源受限情况下的性能评估

为了研究不同资源受限情况下 OALlama. cpp 的表现,将 LlaMA3 – 8B 模型权重加载至 GPU 的比例分别设定为 25%、50% 和 75%。实验结果如图 10 所示,随着权重比例从 25% 提升到 75%,OALlama. cpp 的推理时间均显著低于 FlexGen 和 Llama. cpp。在 25% 权重比例时,OALlama. cpp 较 FlexGen 快了约 43%,较 Llama. cpp 快了约 15. 5%;当权重比例提升到 50% 时,OALlama. cpp 较 FlexGen 的提效幅度达到约 47. 4%,较 Llama. cpp 快约 20. 7%;到 75% 权重比例时,OALlama. cpp 较 FlexGen 提效高达约 64. 5%,较 Llama. cpp 提效约 40. 8%,优势越发明显。这是由于随着权重比例的增加,OALlama. cpp 能够将更多 GPU 亲和度更高的算子卸载到 GPU 上计算,从而实现更优的效果。

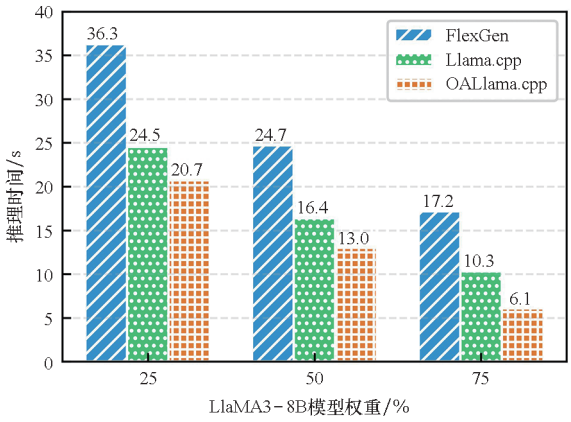


图 10　不同资源受限场景中 LlaMA3 – 8B 的推理性能

Fig. 10　Inference performance of LlaMA3 – 8B under varying resource constraints scenarios

实验结果表明,在模型推理性能的评估中,通过对比不同模型在不同权重比例下使用不同推理框架的表现,能够清晰地看出 OALlama. cpp 的显著优势。

5　结论

在资源受限场景下,研究通过剖析大语言模型的算子特性,发现算子操作间存在差异化的 GPU 亲和度,为算子粒度的模型卸载提供了关键依据。基于此,提出算子感知的模型自动卸载技术 OATO,并将其集成于 Llama. cpp,构建了算子感知的模型推理系统 OALlama. cpp。实验结果表明,OALlama. cpp 在不同场景的大模型推理中,均能在不同程度上提升推理效率,验证了该技术在资源受限环境下的有效性与实用性。

参考文献(References)

[1] ZHAO W X, ZHOU K, LI J Y, et al. A survey of large language models[EB/OL]. (2025 – 03 – 11)[2025 – 05 – 01]. https://arxiv.org/abs/2303.18223.

[2] YUAN Z H, SHANG Y Z, ZHOU Y, et al. LLM inference unveiled: survey and roofline model insights[EB/OL]. (2024 – 05 – 01)[2025 – 05 – 01]. https://arxiv.org/abs/2402.16363.

[3] MIAO X P, OLIARO G, ZHANG Z H, et al. Towards efficient generative large language model serving: a survey from algorithms to systems[EB/OL]. (2023 – 12 – 23)[2025 – 05 – 01]. https://arxiv.org/abs/2312.15234.

[4] VASWANI A, SHAZEER N M, PARMAR N, et al. Attention is all you need[C]//Proceedings of 31st Conference on Neural Information Processing Systems, 2017.

[5] BROWN T B, MANN B, RYDER N, et al. Language models are few-shot learners[C]//Proceedings of 34th Conference on Neural Information Processing Systems, 2020.

[6] OUYANG L, WU J, JIANG X, et al. Training language models to follow instructions with human feedback[C]// Proceedings of 36th Conference on Neural Information Processing Systems, 2022.

[7] OpenAI. GPT-4 technical report[EB/OL]. (2024 – 03 – 04)[2025 – 05 – 01]. https://arxiv.org/abs/2303.08774.

[8] TOUVRON H, LAVRIL T, IZACARD G, et al. LLAMA: open and efficient foundation language models[EB/OL]. (2023 – 02 – 27)[2025 – 05 – 01]. https://arxiv.org/abs/2302.13971.

[9] TOUVRON H, MARTIN L, STONE K, et al. LLAMA 2: open foundation and fine-tuned chat models[EB/OL]. (2023 – 07 – 19)[2025 – 05 – 01]. https://arxiv.org/abs/2307.09288.

[10] HU S D, TU Y G, HAN X, et al. MiniCPM: unveiling the potential of small language models with scalable training strategies[EB/OL]. (2024 – 06 – 03)[2025 – 05 – 01]. https://arxiv.org/abs/2404.06395.

[11] YAO Y, YU T Y, ZHANG A, et al. MiniCPM-V: a GPT-4V level mllm on your phone[EB/OL]. (2024 – 08 – 03)[2025 – 05 – 01]. https://arxiv.org/abs/2408.01800.

[12] GUO D Y, ZHU Q H, YANG D J, et al. DeepSeek-Coder: when the large language model meets programming-the rise of code intelligence[EB/OL]. (2024 – 01 – 26)[2025 – 05 – 01]. https://arxiv.org/abs/2401.14196.

[13] LU H Y, LIU W, ZHANG B, et al. DeepSeek-VL: towards real-world vision-language understanding[EB/OL]. (2024 – 03 – 11)[2025 – 05 – 01]. https://arxiv.org/abs/2403.05525.

[14] DeepSeek-AI. DeepSeek-V2: a strong, economical, and efficient mixture-of-experts language model[EB/OL]. (2024 – 06 – 19)[2025 – 05 – 01]. https://arxiv.org/abs/2405.04434.

[15] CHEN M, TWOREK J, JUN H, et al. Evaluating large language models trained on code[EB/OL]. (2021 – 07 – 14)[2025 – 05 – 01]. https://arxiv.org/abs/2107.03374.

[16] LIAN L, LI B Y, YALA A, et al. LLM-grounded diffusion: enhancing prompt understanding of text-to-image diffusion models with large language models[EB/OL]. (2024 – 03 – 04)[2025 – 05 – 01]. https://arxiv.org/abs/2305.13655.

[17] DONG X L, MOON S, XU Y E, et al. Towards next-

generation intelligent assistants leveraging LLM techniques[C]// Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2023: 5792 – 5793.

[18] XU D L, ZHANG H, YANG L M, et al. Empowering 1000 tokens/second on-device LLM prefilling with mllm-NPU[EB/OL]. (2024 – 07 – 08) [2025 – 05 – 01]. https://arxiv.org/html/2407.05858v1.

[19] 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, 2024: 590 – 606.

[20] ALIZADEH K, MIRZADEH S I, BELENKO D, et al. LLM in a flash: efficient large language model inference with limited memory [C]//Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2024: 12562 – 12584.

[21] JACOB B, KLIGYS S, CHEN B, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018: 2704 – 2713.

[22] NAGEL M, BAALEN M V, BLANKEVOORT T, et al. Data-free quantization through weight equalization and bias correction[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019: 1325 – 1334.

[23] ZHAO R, HU Y W, DOTZEL J, et al. Improving neural network quantization without retraining using outlier channel splitting [C]//Proceedings of the 36th International Conference on Machine Learning, 2019.

[24] NVIDIA. NVIDIA TensorRT-LLM: a TensorRT toolbox for optimized large language model inference[EB/OL]. [2025 – 05 – 01]. https://github.com/NVIDIA/TensorRT-LLM.

[25] LIU Z C, WANG J, DAO T, et al. Deja Vu: contextual sparsity for efficient LLMs at inference time [C]// Proceedings of the 40th International Conference on Machine Learning, 2023.

[26] FARINA M, AHMAD U, TAHA A, et al. Sparsity in transformers: a systematic literature review[J]. Neurocomputing, 2024, 582: 127468.

[27] LI L J, DONG P J, TANG Z H, et al. Discovering sparsity allocation for layer-wise pruning of large language models[C]// Proceedings of 38th Conference on Neural Information Processing Systems, 2024.

[28] YI C X, JIAN S L, TAN Y S, et al. HMO: host memory optimization for model inference acceleration on edge devices[C]// Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2024: 2813 – 2819.

[29] GERGANOV G. Ggml-org/llama.cpp: LLM inference in C/C++[EB/OL]. [2025 – 05 – 01]. https://github.com/ggerganov/Llama.cpp.

[30] LEE W, LEE J, SEO J, et al. InfiniGen: efficient generative inference of large language models with dynamic KV cache management[EB/OL]. (2024 – 06 – 28) [2025 – 05 – 01]. https://arxiv.org/abs/2406.19707.

[31] SHENG Y, ZHENG L M, YUAN B H, et al. FlexGen: high-throughput generative inference of large language models with a single GPU[C]// Proceedings of the 40th International Conference on Machine Learning, 2023.

[32] AMINABADI R Y, RAJBHANDARI S, AWAN A A, et al. DeepSpeed-inference: enabling efficient inference of transformer models at unprecedented scale[C]//Proceedings of the SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, 2022.

[33] 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, 2023: 611 – 626.

[34] YI C X, JIAN S L, TAN Y S, et al. MACA: memory-aware convolution accelerating for CNN inference on edge devices[C]// Proceedings of the 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2024: 1250 – 1255.

[35] 葛旭冉, 欧洋, 王博, 等. 大语言模型推理中的存储优化技术综述[J]. 计算机研究与发展, 2025, 62(3): 545 – 562.
GE X R, OU Y, WANG B, et al. Survey of storage optimization techniques in large language model inference[J]. Journal of Computer Research and Development, 2025, 62(3): 545 – 562. (in Chinese)

[36] 梁绪宁, 王思琪, 杨海龙, 等. 基于自适应张量交换和重算的大模型推理优化[J/OL]. 计算机工程, 2025. (2025 – 04 – 14) [2025 – 05 – 01]. https://link.cnki.net/doi/10.19678/j.issn.1000 – 3428.0070644.
LIANG X N, WANG S Q, YANG H L, et al. Adaptive tensor swapping and re-computation for efficient large language model inference[J/OL]. Computer Engineering, 2025. (2025 – 04 – 14) [2025 – 05 – 01]. https://link.cnki.net/doi/10.19678/j.issn.1000 – 3428.0070644. (in Chinese)