架构具有长上下文的 LLM 加速器与打包预取调度程序和超大容量片上内存

Ming-Yen Lee†, Faaiq Waqar†, Hanchen Yang†, Muhammed Ahosan Ul Karim*, Harsono Simka*, Shimeng Yu†
†*Georgia Institute of Technology, Atlanta, GA, 30332, USA*
**Samsung Semiconductor Inc., San Jose, CA, 95134, USA*

摘要—长上下文大型语言模型（LLM）推理面临着计算瓶颈的增加，主要是由于注意力计算随着上下文长度的增长而扩展，导致高带宽内存（HBM）中的 KV 缓存传输开销饱和。虽然预取技术通过提前获取 KV 数据来减轻缓存未命中问题，但它们的空间和时间效益为利用这些机会提供了新的可能性。本工作提出了一种与单片 3D (M3D) 后端线（BEOL）兼容的嵌入式内存打包预取调度架构，具有超大芯片容量，以加速长上下文 LLM 推理。我们的优化在使用 TPUv6e 类硬件和额外 512MB BEOL 内存的情况下，相对于串行执行，对 Llama3.1-8B 实现了 8.06 倍的解码加速和 1.83 倍的整体延迟减少。在类似 TPU 架构上对多请求工作负载进行评估显示，在 Llama3.1-8B 和 Llama3.1-70B 模型中，与仅打包方法相比，吞吐量提高了 1.7 倍至 2.4 倍，HBM 带宽减少了 1.5 倍至 2.4 倍。通过打包、预取和 BEOL 内存的协同设计，我们的方法缓解了 HBM 约束并实现了高效的长上下文 LLM 推理。

***Index Terms*—大语言模型推理，预取，调度，片上内存**

I. 介绍

L 大型语言模型（LLMs）在多种应用中展示了卓越的表现，包括对话系统、代码合成和摘要生成。大型语言模型的快速采用使得高效推理成为了一个关键挑战，尤其是在减少时间到第一个令牌和令牌间的时间等延迟指标方面，以满足实时应用程序的需求。先前的工作探索了各种算法优化和硬件加速。通过批处理和操作符融合 [1]，LLM 推理的预填充阶段已经变得计算密集型，减少了离片数据传输并提高了数据重用率。这意味着预填充性能主要受到计算吞吐量限制而不是 HBM 带宽。然而，在解码阶段仍存在一个关键瓶颈，自回归令牌生成过程中由于离片 HBM 和片上计算资源之间的高数据传输开销而受到影响。如图 1 所示，即使改进了 HBM 带宽，解码延迟仍然受 KV 缓存传输的限制，特别是在计算单元不断进步的情况下。这一挑战因现代 LLMs 中不断增加的上下文长度而加剧，这增加了 KV 缓存的大小，并在注意力计算期间给 HBM 带宽带来了更大的压力。

解码阶段主要由两种操作组成：线性运算和注意力运算。可以通过打包来自其他请求预填充阶段的计算密集型操作来优化线性层，因为模型权重在所有请求之间是共享的 [2]。不幸的是，注意力操作仍然需要单独处理并且仍然是内存绑定的。已经提出了预取技术作为补充解决方案。例如，先前的工作 [3] 利用了 KV 缓存和权重预取，在 GPU 通信重叠期间进行操作。我们认为可以进一步探索预取的完整时域和空域潜力。首先，可以通过与计算密集型预填充阶段重叠来利用额外的预取时间，而不仅仅是仅在通信间隔期间进行预取。其次，M3D 内存 [4] 已经作为高密度、高速片上缓冲区的有前途解决方案出现，使得可以容纳更多的预取 KV 缓存。

在这项工作中，我们提出了一种具有超大容量片上内存的打包预取调度架构，以缓解 HBM 瓶颈并提高长上下文 LLM 推理效率。时间上，我们在计算密集型操作期间通过交错填充和解码阶段来利用残留的 HBM 带宽，为 KV 缓存数据预取提供了足够的时间。空间上，我们引入了高密度 M3D 嵌入式内存以满足预取数据存储的容量需求。为了系统地评估我们的方法，我们通过定制 Timeloop [5] 开发了一个 LLM 仿真框架，在 AI 加速器上建模端到端推理。评估结果显示，与串行执行相比，我们在具有额外 512MB M3D 内存的类似 TPUv6e 加速器上运行的 Llama3.1-8B 模型中，打包预取调度

Submitted to IEEE MICRO Special Issue "AI for Hardware and Hardware for AI" for review.

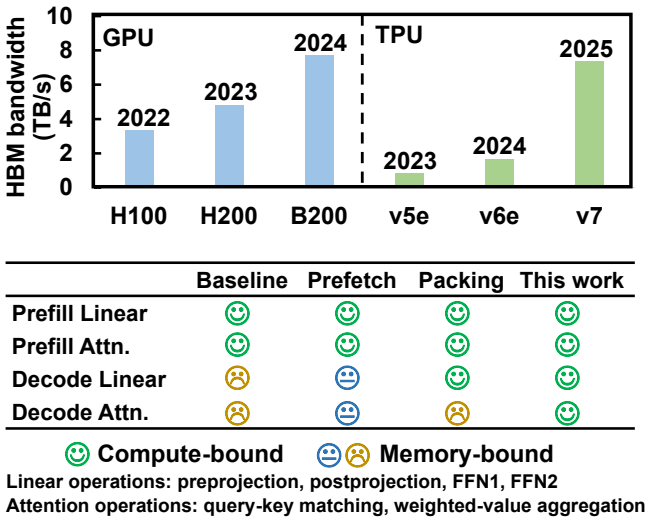

图 1. HBM 带宽在 GPU 和 TPU 中的发展趋势，以及 LLM 推理两个阶段中计算受限操作和内存受限操作的分解。内存受限操作对 HBM 施加了高带宽压力。

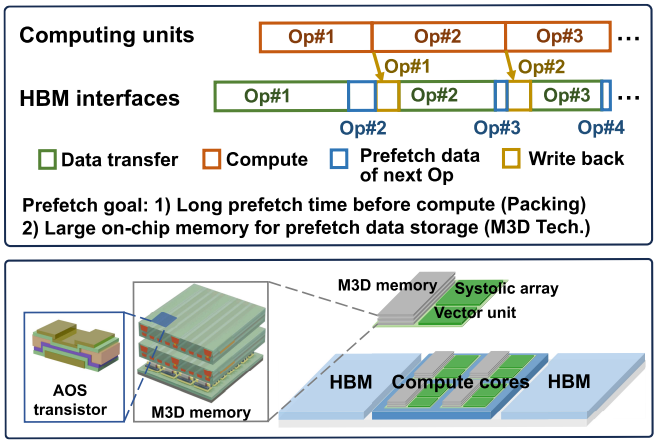

图 2. 计算单元和 HBM 数据传输的时间图在 AI 加速器上。打包和高密度 M3D 技术提供了最大化预取效益的机会。

实现了高达 8.06 倍的令牌到令牌解码速度提升和 1.83 倍的整体延迟减少。我们还在 openchat_sharegpt4 和 arxiv_summarization 数据集上使用多请求工作负载进行了服务级别的评估。在类似 TPU 的架构上，我们的方法与仅打包基线相比，在 Llama3.1-8B 和 Llama3.1-70B 模型中实现了 1.7 倍至 2.4 倍的吞吐量提升以及 1.5 倍至 2.4 倍的 HBM 带宽减少。

II. 背景

图 2 描述了在类似 TPU 架构上执行多操作时计算单元和 HBM 数据传输的时间图。通常，TPU 加速器由高性能计算单元（例如点阵列和向量单元）、片上中间操作数内存以及存储完整模型和 KV 缓存的片外 HBM 组成。在从 HBM 获取操作数分区到片上内存后开始计算。执行过程中，计算单元和 HBM 以流水线方式运行。在进行计算的同时，可以通过片上内存中的双缓冲机制同时进行片内和片外数据传输。对于计算密集型操作，计算时间超过数据传输时间，留下剩余的 HBM 带宽可用于预取即将进行的操作的模型权重或 KV 缓存数据。完成后，输出要么写回到 HBM 要么保留在芯片上以供下一次操作使用，这被称为运算符融合 [1]。因此，后续操作由于之前计算密集型阶段的预取而受益于减少的数据传输延迟。

为了充分利用预取，这可以缓解下一次操作的计算停滞并减轻 HBM 带宽瓶颈，特别是对于解码阶段中内存绑定的注意力操作而言，我们的方法聚焦于两个目标：足够的预取时间和预取空间。首先，在解码阶段从线性操作借用 HBM 带宽通常是无效的，因为这些操作很容易变成内存绑定的。尽管较大的批处理大小可以提高权重重用率，但它也会增加推理延迟和 KV 缓存占用的 HBM 资源。相比之下，填充阶段的操作往往是计算密集型的，并且容易饱和计算资源 [2]。通过在不同请求批次之间交错填充和解码阶段，我们可以利用填充期间剩余的 HBM 带宽来预取后续解码注意力所需的 KV 缓存数据。

其次，为了利用这个扩展的预取窗口，关键是要有足够的片上内存容量来存储预取的数据。嵌入式 M3D 内存已经作为解决这种存储需求的一种有前景的解决方案出现。借助领先代工厂 [6], [7] 正在追求的设备技术方面的最新进展，非晶氧化物半导体（AOS）晶体管能够实现低漏电和高速嵌入式类似 DRAM 的片上缓冲区。如图 2 所示，密集型存储单元（例如，双晶体管增益单元）堆叠在逻辑层之上，外围电路则通过前端工艺（FEOL）制造，多层存储阵列利用后端互连技术集成。这种 CMOS+X 方法有可能实现数百兆字节的片上内存容量 [4]，为存储预取 KV 缓存数据提供空间。作为参考，最先进的 V-cache 通过将 SRAM 堆叠在微处理器核心之上只能达到 96MB 的容量 [8]。通过结合预取调度与打包和高密度 M3D 内存，我们提出的打包-预取调度架构优化了解码阶段注意力计算中的预取机会。前馈和解码阶段都变得以计算为瓶颈，这缓解了 HBM 带宽瓶颈并提高了长上下文 LLM 推理的效率。

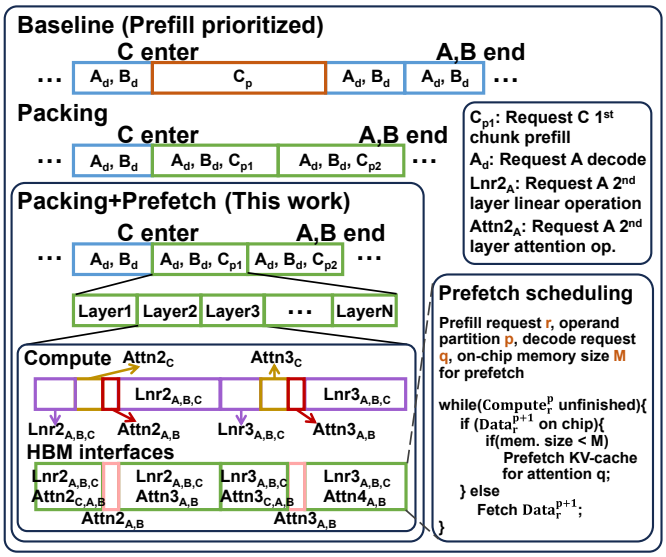

图 3. 时间图和提出的调度算法。

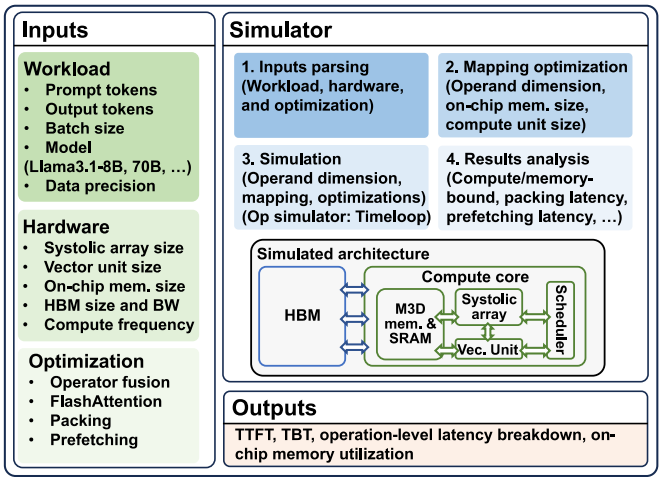

图 4. 提出的仿真框架概述。

III. 建议的打包-预取调度方案

图 3 说明了所提出的打包预取调度的时间图和操作原理，该方法结合了请求间打包和 KV 缓存预取。在基线方法中，当新请求 C 到达时，请求 A 和 B 处于解码阶段。调度器启动请求 C 的预填充阶段，并将所有三个请求排队以进行下一个解码周期。通过打包，请求 C 被分割成两个块，每个块与请求 A 和 B 的解码操作交错 [9]。这种预填和解码阶段的交错重叠了请求 C 的预填阶段中的计算受限线性操作与请求 A 和 B 的解码阶段中的内存受限线性操作。因此，权重复用得到了改进，将解码线性操作从内存受限转换为计算受限。然而，即使有了打包，解码阶段中的注意力操作仍然受内存限制，主要由 HBM 中大量 KV 缓存数据传输主导。为了应对这一瓶颈，我们在计算受限阶段引入预取，通过交错当前打包线性操作的数据获取与即将到来的注意力计算的 KV 缓存数据预取，利用剩余的 HBM 带宽。

如时间图所示，操作按照分层顺序进行调度。计算单元执行打包的线性运算时，调度器根据两个条件动态分配 HBM 带宽以获取数据：（1）操作数取回优先级：如果当前线性运算的操作数分区未加载到片上内存中，则 HBM 优先传输它们。（2）KV 缓存预取机会：如果这些操作数已经在片上并且有可用的片上内存，调度器将使用闲置的 HBM 带宽为下一个注意力操作预取 KV 缓存数据。例如，考虑两个连续的层（如第 2 层和第 3 层）。在第 2 层中，在注意力之前的线性运算期间，计算单元处理请求 A、B 和 C 的打包预填/解码操作，同时 HBM 同时为第 2 层即将到来的注意力预取请求 A 和 B 的 KV 缓存。在执行 A 和 B 的注意力时，所需的 KV 缓存已经在片上，消除了 HBM 传输延迟。同样，在第 2 层中注意力之后和第 3 层之前线性运算期间，调度器根据上述两个条件中的 HBM 带宽可用性和内存空间预取第 3 层的 KV 缓存。通过将计算绑定操作与 KV 缓存预取重叠，我们的打包-预取调度消除了注意力过程中由 HBM 引起的延迟停滞，将操作阶段从内存绑定转变为计算绑定。结果，所有推理阶段都变成计算绑定状态，硬件利用率高，提高了推理吞吐量，并减轻了对 HBM 带宽的压力。

IV. 提出的大型语言模型推理框架和架构

为了系统地建模和评估不同优化策略下 LLM 推理的性能，我们开发了一个支持可配置的 LLM 模型、硬件架构和优化技术的端到端仿真框架。图 4 展示了所提框架的概述，它接受三个主要输入：1) LLM 工作负载，包括输入/输出标记长度、批量大小、LLM 模型（例如 Llama3.1-8B, Llama3.1-70B）和数据精度等参数；2) 硬件架构，指定计算资源的数量（例如点阵列、向量单元）、运行频率以及包括片上存储器和片外 HBM 在内的内存层次结构；3) 优化技术，如操作符融合、FlashAttention [10]、打包 [9] 和预取 [3]。

接收到配置后，模拟器解析所有参数，并根据操作数维度、可用内存和计算资源执行操作级映射。使用迭代搜索进行的映射空间探索然后确定一种优化的映射策略。注意，在线性和注意力操作中的操作数参数比典型的卷积层要小，导致搜索空间更窄且整体映射优化的

表 I
大语言模型和类似 TPU 的架构配置。

大语言模型	**llama3.1-8B [11]**	**llama3.1-70B [11]**
TPU-like architecture	TPUv6e-like	TPUv7-like
FP16 performance	918 TFLOPS	4614 TFLOPS
Systolic array	128×128×16	256×256×16
Vector unit	128×16×16	256×32×16
On-chip memory	80MB(Compute) +512MB(Prefetch)	160MB(Compute) +1GB(Prefetch)
HBM capacity	32 GB	220 GB
HBM bandwidth	1.64 TB/s	7.4 TB/s

表 II
评估数据集 [9]。

数据集	**提示标记**			**输出令牌**		
	Median	P90	Std.	Median	P90	Std.
openchat_ sharegpt4	1730	5696	2088	415	834	101
arxiv_ summarization	7059	12985	3638	208	371	265

运行时间少于一分钟。每种优化技术要么在映射阶段实现，要么被集成到外部模拟器的配置文件中。当操作数维度、片上内存容量和映射约束允许时，自动应用操作符融合。FlashAttention 通过头部级别的分块实现，并且与查询-键匹配和值加权聚合之间的操作符融合兼容，以减少离芯片的数据传输。打包和预取直接通过指定操作维度并评估它们对执行重叠和数据移动的影响来进行建模。

所有配置设定后，我们调用 Timeloop 模拟器 [5] 执行操作级模拟。对于每个操作，框架记录计算延迟、数据传输时间和片上内存利用率。最终报告提供了关键推理指标，包括预填充阶段的首次令牌生成时间（TTFT）和解码阶段的令牌间延迟（TBT），这些指标在使用和不使用优化技术（例如打包和预取）的情况下均有提供。我们可以定量比较我们调度方案与其他优化基线之间的这些指标。

V. 评估

我们评估了提议的打包预取调度优化在两个大语言模型及其硬件配置上的效果：Llama3.1-8B 在类似 TPUv6e 的架构上，以及 Llama3.1-70B 在类似 TPUv7 的架构上，如表 I 所示。每个实验都在单个处理节点下使用 FP16 推理进行。每个设置的片上内存包括两个部分：(1) 一个 80MB 缓冲区用于存储输入、权重和中间操作数，以及 (2) 对于 Llama3.1-8B 是最多 512MB 的预取缓冲区，而对于 Llama3.1-70B 则是 1GB。这些额外的缓冲区大小足以预先加载单个层的 KV 缓存数据以处理最大上下文长度为 128K 令牌的一批数据。同样地，在两个系统上配置的 HBM 容量可以存储整个模型中一批 128K 令牌的 KV 缓存。我们进行了四项案例研究：案例研究 1 和 2 关注于阶段级别的延迟分析，评估解码阶段的时间间隔令牌延迟（TBT）以及端到端前填-解码延迟，这与总体吞吐量直接相关。案例研究 3 和 4 使用与先前工作 [9] 相同的真实世界多请求数据集来评估服务级别的性能：openchat_sharegpt4 表示与 ChatGPT-4 [12] 的互动用户对话，arxiv_summarization 特征是使用 arXiv.org 文章进行长形式摘要任务。我们在 99 分位数（P99）TBT 延迟约束下展示了每秒查询次数（QPS）的吞吐量改进，并且在两个模型上利用这两个数据集展示了提议方法节省 HBM 带宽的效果。我们还探讨了 arxiv_summarization 数据集上的两种打包块大小下的吞吐量-延迟权衡。

案例研究 1: 我们首先使用 Llama3.1-8B 模型评估解码操作和组合的预填充-解码打包阶段在不同预填入标记长度和 KV 缓存标记大小下的延迟改进情况。图 5 展示了结果，其中延迟被归一化为顺序执行（即先进行预填充再进行不带打包或预取的解码）的基线，并以加速比的形式报告。仅打包的基线使用 80MB 的片上内存，而我们的优化利用额外的 512MB M3D 内存来存储预取的 KV 缓存数据。对于解码阶段，我们的打包-预取调度在使用 2048 个预填充标记和 128K KV 缓存标记时实现了最高 8.06 倍的加速比，与相同条件下仅打包基线相比提高了 1.41 倍的速度。预取的好处与预填充期间可用的剩余 HBM 带宽相关联。当预填充标记减少到 512 个时，剩余带宽降低。因此，随着 KV 缓存大小增加，加速比下降，因为预取不能完全隐藏解码注意力的记忆延迟。在考虑整体打包阶段的延迟时，由于未优化预填充延迟，并将其计入总运行时间中，因此加

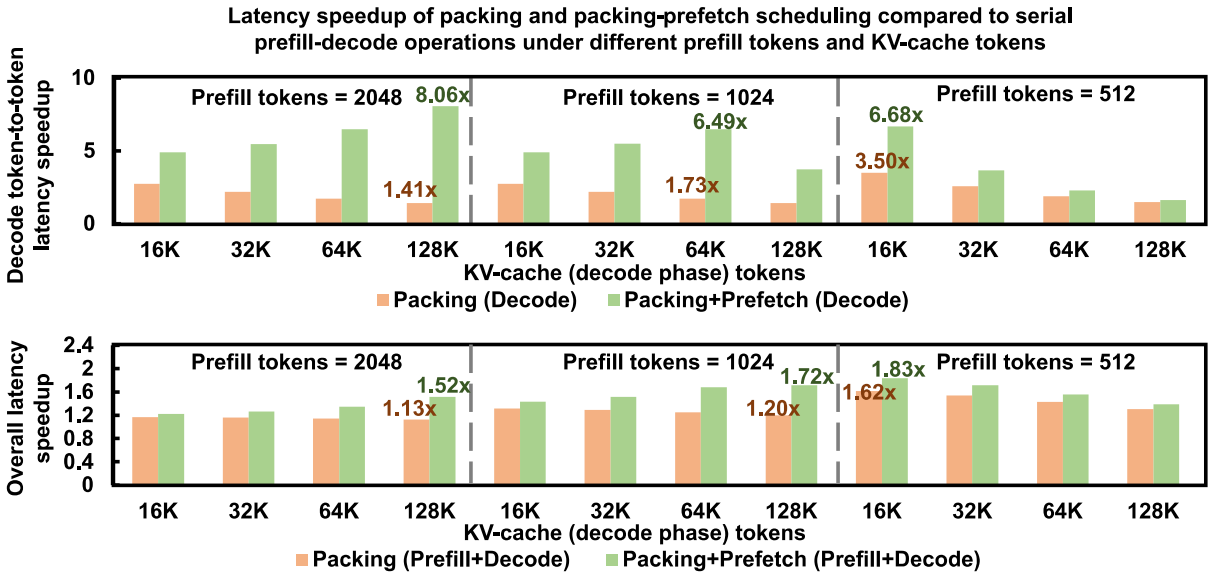

图 5. 打包和打包预取调度在不同预填标记和 KV 缓存标记下的延迟加速情况，针对 Llama3.1-8B。

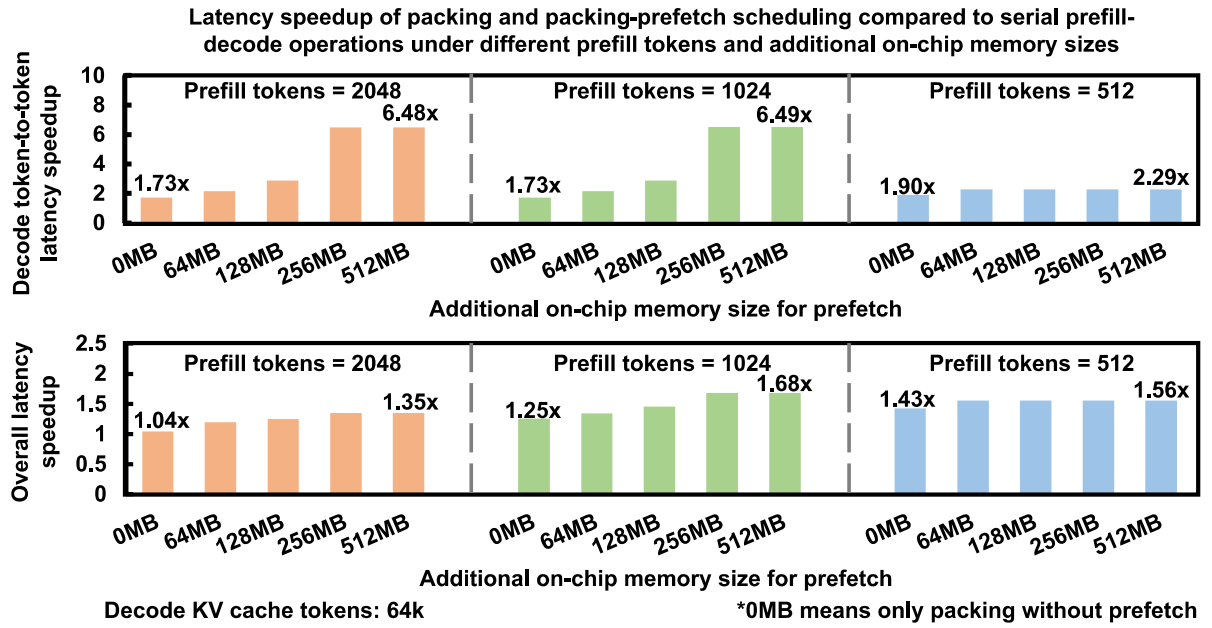

图 6. 与 Llama3.1-8B 中不同预填标记和额外片上缓存大小下的串行预填解码操作相比，打包和打包预取调度的延迟加速。

速比低于解码情况下的数值。使用 512 个预填充标记和 16K KV 缓存标记时，我们的方法实现了总体 1.83 倍的加速。另外，在 1024 个预填充标记的情况下，整体加速达到了 1.72 倍，而仅打包基线为 1.20 倍。虽然更长的预填充序列提供了更多的 KV 缓存预取带宽并提高了解码性能，但也增加了预填充延迟。综合考虑这两个因素，1024 个预填充标记在这个案例研究中是一个有效的点，对解码效率和总运行时间都有显著改进。

案例研究 2: 我们还探讨了不同片上内存容量如何影响解码 TBT 和整体打包延迟。图 6 显示了仅打包和打包预取调度在不同的预填充令牌和片上内存大小下的延迟加速，其中包含 64K 解码 KV-cache 令牌。此处，0MB 表示没有额外的片上内存可用于预取，仅使用基础的 80MB 缓冲区进行计算，即仅打包基线。对于 2048 和 1024 预填充令牌，在相同内存大小下的解码速度提升相似，表明两种情况下预取时间都足够且加速

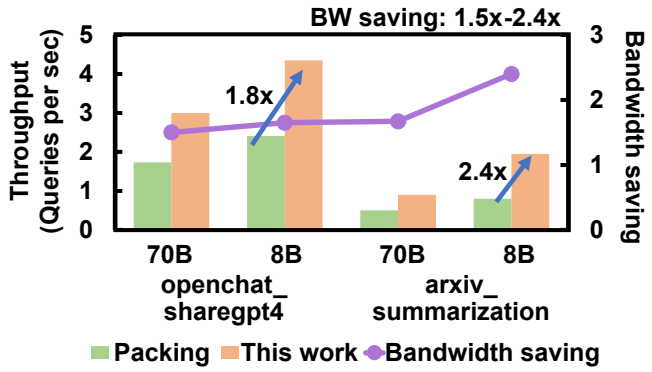

图 7. 两种数据集在 Llama3.1-8B 和 Llama3.1-70B 模型上的服务级别吞吐量比较及仅打包基线的带宽节省（使用打包预取调度，本工作）。

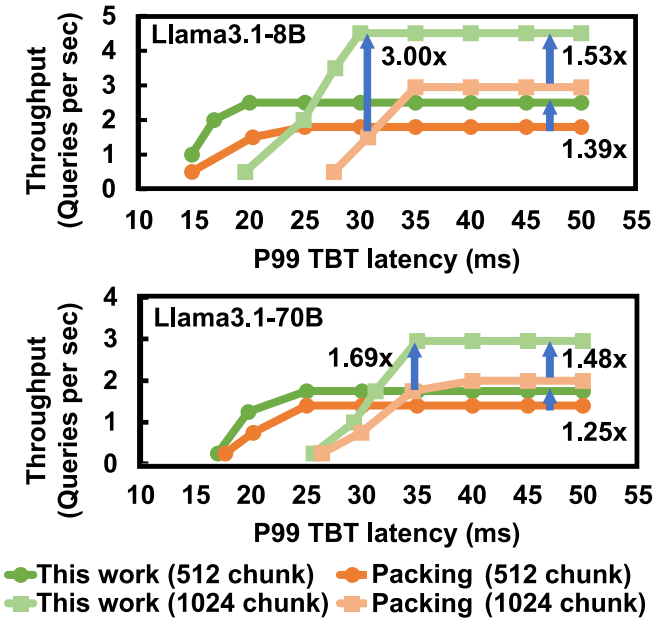

图 8. 吞吐量和延迟权衡：在 arxiv_summerization 数据集上使用不同块大小的打包和预取调度（本工作）。

受到可用片上内存的限制。随着片上内存容量从 0MB 增加到 512MB，解码 TBT 加速改善从 1.73× 提升至 6.49×。对于 512 预填充令牌，即使增加了内存容量，加速改进较小。这与案例研究 1 的发现一致：较短的预填充阶段无论有多少可用内存都无法为预取留出足够时间，导致性能提升有限。整体延迟加速在解码情况下遵循类似的趋势。当我们增加预填充令牌从 512 到 2048，仅打包基线的加速降低到接近 1.0x。通过提出的打包-预取优化和 M3D 内存的整合，对于 2048 预填充令牌的整体加速可提升至 1.35x，而对于 1024 预填充令牌则为 1.68x。

案例研究 3: 除了阶段级分析，我们还使用两个真实世界的数据集评估了所提出的打包预取调度的服务级性能：openchat_sharegpt4 和 arxiv_summarization，并在 Llama3.1-8B 和 Llama3.1-70B 模型上进行了测试。我们采用了与先前工作 [9] 中相同的多请求打包策略，该策略优先处理当前解码请求并将这些请求的预填充部分交错插入到执行队列中。相同请求到达时间遵循泊松分布 [9]。为了避免多个请求在队列中堆积，我们将第 99 百分位（P99）请求调度延迟设置为 1 秒。吞吐量是在由 P99 TBT 延迟定义的服务级目标（SLO）约束下测量的。延迟约束是在 32 个并发解码请求的条件下评估的，每个请求包括 4K KV-cache 令牌。根据所提出的框架进行的评估，Llama3.1-8B 的 SLO 阈值设置为 16.70 毫秒，而 Llama3.1-70B 设置为 19.23 毫秒。为了满足这些约束条件，打包块大小固定为 512 令牌。请注意，在预填充阶段之后，用于解码的初始 KV-cache 令牌数量与输入提示令牌的数量相匹配。由于每个执行周期可以打包多个解码阶段请求，因此总 KV-cache 大小可以从 10K 到 128K 个令牌不等。

结果如图 7 所示。首先，Llama3.1-8B 在两个数据集上都表现出比 Llama3.1-70B 更高的吞吐量。这是因为更大的 70B 模型需要更多的 KV-cache 预取时间，尽管从类似 TPUv6e 到类似 TPUv7 的架构中计算性能和 HBM 带宽有所提升。其次，openchat_sharegpt4 数据集的吞吐量高于 arxiv_summarization，因为摘要工作负载涉及更长的输入和输出序列，需要更多的计算和数据传输时间。第三，尽管绝对吞吐量较低，arxiv_summarization 在提出的优化中展示了更高的相对吞吐量增益。对于 8B 模型，在 arxiv_summarization 和 openchat_sharegpt4 上的吞吐量分别提高了 2.4 倍和 1.8 倍。这归因于摘要任务中的预填标记更长，意味着有更多的预填充块与解码请求一起打包。在这种情况下，随着 KV-cache 预取机会增加，打包效率得到了提升。

我们还将仅打包的基线与增加带宽的 HBM 进行比较，以确定需要多少额外的内存带宽才能达到我们的方法的吞吐量。结果显示，在 Llama3.1-8B 上的摘要数据集中，对于仅打包的基线而言，在 1.95 QPS 下，我们提出的打包预取调度节省了高达 2.4 倍的 HBM 带宽。这个案例研究表明，我们提出的调度优化提高了现实世界中长上下文服务级别的推理吞吐量，并缓解了 HBM 带宽的压力。

案例研究 4: 我们进一步探讨了打包和预取打包策略的吞吐量与延迟权衡，使用 512 和 1024 个标记大小的块在 arxiv_summarization 数据集上进行分析。该分析展示了块粒度如何影响实际条件下的调度效率。类似于案例研究 3 中的观察结果，由于预取时间需求的不同，Llama3.1-8B 的总体吞吐量高于 Llama3.1-70B。在放松 SLO 约束的情况下，吞吐量饱和因为等待队列被更多到达请求填满且系统违反了调度延迟约束。饱和后，我们提出的优化方法分别在 1024 个标记大小和 512 个标记大小的块上实现了 1.53 倍和 1.39 倍的吞吐量提升，在 Llama3.1-8B 模型上。较大的块尺寸能够提供更长的预填窗口，为一次执行中的打包解码请求留下更多时间进行 KV 缓存预取。这导致了 1024 个标记大小相较于 512 个标记大小具有更高的峰值吞吐量。当对 Llama3.1-8B 模型应用严格的 P99 TBT 约束（31 毫秒）时，我们的打包预取调度相比仅打包的基准实现了高达 3.0 倍的吞吐量提升。

VI. 结论

在本研究中，我们提出了一种结合 M3D BEOL 内存的打包-预取调度架构，以解决长上下文 LLM 推理解码阶段中的内存绑定注意力瓶颈问题。我们的方法通过交错填充和解码阶段来提供足够的预取时间，从而最大化预取的好处，同时利用超大容量片上内存确保足够的预取空间。我们使用自主研发的端到端 LLM 推理框架评估了我们的优化方案。在 Llama3.1-8B 和 Llama3.1-70B 上的实验结果表明，在 TPUv6e 和类似 TPUv7 架构上，与串行执行相比，解码延迟最多减少了 8.06 倍，整体延迟提高了 1.83 倍。针对真实世界数据集的服务级基准测试显示，与打包基线相比，吞吐量提升了 1.7 至 2.4 倍，HBM 带宽节省了 1.5 到 2.4 倍。本研究突出了调度、内存子系统和硬件架构的协同设计对于高效长文本 LLM 推理的重要性，为缓解 HBM 瓶颈提供了见解，并缩小了 AI 加速器片上计算与片外内存性能之间的差距。

致谢

此项工作得到了三星半导体的逻辑寻路实验室的支持。

参考文献

[1] S.-C. Kao, S. Subramanian, G. Agrawal, A. Yazdanbakhsh, and T. Krishna, “FLAT: An optimized dataflow for mitigating attention bottlenecks,” in Proc. 28th ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS ’ 23), Vol. 2, Vancouver, BC, Canada, Mar. 25 – 29, 2023, pp. 295 – 310, doi:10.1145/3575693.3575747.

[2] A. Agrawal, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, and R. Ramjee, “Sarathi: Efficient LLM inference by piggybacking decodes with chunked prefills,” arXiv preprint arXiv:2308.16369, Aug. 2023, doi:10.48550/arXiv.2308.16369.

[3] A. C. Yüzügüler, J. Zhuang, and L. Cavigelli, “PRESERVE: Prefetching model weights and KV-cache in distributed LLM serving,” arXiv preprint arXiv:2501.08192, Jan. 2025, doi:10.48550/arXiv.2501.08192.

[4] F. Waqar, M.-Y. Lee, S. Yoon, S. Lim, and S. Yu, ”CMOS+ X: Stacking Persistent Embedded Memories based on Oxide Transistors upon GPGPU Platforms,” arXiv preprint arXiv:2506.23405, Jun. 2025, doi:10.48550/arXiv.2506.23405.

[5] A. Parashar et al., “Timeloop: A systematic approach to DNN accelerator evaluation,” in Proc. IEEE Int. Symp. Performance Anal. Syst. Softw. (ISPASS), Mar. 2019, pp. 304 – 315, doi:10.1109/ISPASS.2019.00042.

[6] J.-E. Yang et al., ”A-IGZO FETs with High Current and Remarkable Stability for Vertical Channel Transistor(VCT) / 3D DRAM Applications,” 2024 IEEE Symposium on VLSI Technology and Circuits, Honolulu, HI, USA, 2024, pp. 1-2, doi:10.1109/VLSITechnologyandCir46783.2024.10631550.

[7] K.H. Chiang et al., ”Integration of 0.75V VDD Oxide-Semiconductor 1T1C Memory with Advanced Logic for An Ultra-Low-Power Low-Latency Cache Solution,” 2025 IEEE Symposium on VLSI Technology and Circuits, Kyoto, Japan, 2025.

[8] R. Bhargava and K. Troester, ”AMD Next-Generation “Zen 4” Core and 4th Gen AMD EPYC Server CPUs,” in IEEE Micro, vol. 44, no. 3, pp. 8-17, May-June 2024, doi:10.1109/MM.2024.3375070.

[9] A. Agrawal et al., “Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve,” in *Proc. 18th USENIX Symp. Operating Syst. Design Implementation (OSDI)*, Santa Clara, CA, USA, July 10 – 12, 2024, pp. 117 – 134, doi:10.5555/3691938.3691945.

[10] T. Dao, “FlashAttention-2: Faster attention with better parallelism and work partitioning,” arXiv preprint arXiv:2307.08691, Jul. 2023, doi:10.48550/arXiv.2307.08691.

[11] A. Dubey et al., “The LLaMA 3 herd of models,” arXiv e-prints, Jul. 2024, arXiv:2407.21783, doi:10.48550/arXiv.2407.21783.

[12] J. Achiam et al., “GPT-4 technical report,” arXiv preprint arXiv:2303.08774, Mar. 2023, doi:10.48550/arXiv.2303.08774.