arXiv:2611.32295v1 [cs.DC] 24 Nov 2026

NORN：面向上下文长度的权重与 KV 缓存放置优化，用于卸载式大语言模型推理

Naoto Sugiura [1] **Hiroki Matsutani** [1]

摘要

将模型权重和键值（KV）缓存卸载到主机内存后，大语言模型（LLM）即可在单个内存受限的GPU上运行。随着解码过程中 KV 缓存不断增长，GPU 内存需求以及 KV缓存传输和注意力计算的开销会发生变化。然而，现有卸载方法通常采用静态的计算分配以及模型权重和 KV缓存放置策略，无法适应这些变化，从而降低推理性能。我们提出了 Norn，一种具备上下文长度感知能力的卸载式大语言模型推理方法。Norn 基于离线分析，在子层级别联合优化计算分配与放置策略，预先计算多个执行计划及其切换点。在解码过程中，随着上下文长度增加，Norn 会切换执行计划，在无需预先知道输出长度的情况下重叠计算与数据传输。对于预填充，Norn 采用算子优先分块预填充，以避免重复的权重传输，并自动选择分块大小。在两个单GPU系统上使用 Qwen3-14B 和 Qwen3-32B 进行的实验表明，与流水线分片相比，端到端延迟最高可加速 16.8× 倍；与已知输入和输出长度的预知信息变体相比，最高可加速 4.2× 倍。

1 引言

大语言模型的应用已从简短对话扩展到针对复杂问题的复杂推理，以及对海量文档的长上下文处理。随着大语言模型应用不断多样化，人们越来越关注在PC和智能手机等本地设备上运行大语言模型，而不是完全依赖云服务。有关向外部传输个人或机密信息的安全担忧，进一步推动了对本地推理的需求（Bloomberg，2024）。端侧语言模型已经支持了实用的AI功能，例如写作辅助和摘要生成（Li 等人，2025；Pradeep，2024），与此同时，人们也在探索其在延迟敏感型应用中的使用，包括游戏内AI（Ukarande 等人，2026a）。与主要优先考虑高吞吐量的数据中心环境不同，本地环境强调以低延迟处理较小的批大小。

然而，运行参数量较大的大语言模型仍然需要大量图形处理器内存，使得消费级GPU有限的内存容量成为一项重大挑战。在推理过程中，图形处理器内存的消耗不仅来自模型权重，还来自用于保留历史令牌信息的键值缓存。权重的内存占用保持不变，而键值缓存的内存占用则随上下文长度线性增长。例如，在 FP16 精度下，Qwen3-32B（Yang 等人，2025）的键值缓存仅在 128K 个词元时就达到约 32 吉比字节，接近一块消费级GPU的内存容量。此外，每个请求对键值缓存的需求会随输入和输出长度的不同而显著变化。尤其是在生成长输出期间，由于输出长度通常事先未知，最终的键值缓存大小也难以确定。

缓解图形处理器内存限制的一种有前景的方法是卸载式推理，该方法利用主机内存。在卸载式推理中，部分模型权重和键值缓存存储在主机内存中，并根据需要传输至图形处理器，从而能够执行超出可用 GPU 内存容量的大语言模型。然而，中央处理器与图形处理器之间的数据传输会引入额外开销，仅依靠卸载不足以实现高推理性能。为应对这一挑战，研究人员提出了多种方法，通过优化计算在中央处理器或图形处理器上的分配，以及权重和键值缓存的放置，同时将计算与数据传输重叠，以降低推理延迟（Yu 等人，2024；Ukarande 等人，2026a；Sugiura 等人，2026；Park 和 Egger，2024；Zhao 等人，2024）。

然而，现有的卸载方法并未考虑在解码过程中随着上下文长度增加而调整计算分配或放置。例如，流水线分片（Ukarande 等人，2026a）会根据需要处理的新词元数量选择执行计划，但不会

[1]日本神奈川庆应义塾大学。通信作者：Naoto Sugiura <ko1234@keio.jp>。

预印本。

在批大小为一的情况下，根据解码过程中的上下文长度增长调整放置。静态放置在最大支持的上下文长度与推理性能之间引入了权衡。为键值缓存预留 GPU 内存以容纳较长上下文，会减少可用于驻留权重的内存，从而即使对于较短的请求也会降低推理性能。相反，针对较短上下文定制的放置可以让更多权重和键值缓存驻留在图形处理器上，但不断增长的键值缓存可能会在生成过程中耗尽 GPU 内存，导致推理无法在相同放置下继续进行。在我们使用 Qwen3-7×14B 的流水线分片进行的测量中，即使输入和输出长度固定，将最大支持的上下文长度从 2K 增加到 128K 词元，也使每令牌解码延迟增加了 182。为了避免为短请求预留过多的 GPU 内存，用户需要为每个请求配置最大支持的上下文长度，尽管输出长度事先并不确定。

此外，上下文长度增加不仅会影响内存需求，还会影响卸载式推理的效率。随着上下文长度增加，KV缓存传输时间和注意力计算时间会增加，而权重传输时间和前馈网络（FFN）计算时间则保持不变。因此，计算与数据传输的相对成本会发生变化，进而可能改变推理期间的最优计算分配和放置方式。因此，在某一上下文长度下效率较高的放置方式，随着解码推进也可能变得不再最优。

为应对这些挑战，我们提出了 Norn，这是一种会随上下文长度增加而动态切换执行计划的卸载式推理方法。借助离线分析结果，Norn 协同优化注意力子层和 FFN 子层在中央处理器或图形处理器上的执行分配，以及权重和 KV 缓存的放置。它会预先确定多个执行计划及其切换点，同时考虑上下文相关的性能变化和数据迁移成本。解码期间，Norn 会切换计划、调整 GPU 内存分配，并使计算与数据传输重叠。对于预填充，Norn 采用算子优先的分块预填充，以减少重复的权重传输，同时限制激活值的内存使用量；并自动选择分块大小，兼顾预填充延迟和转换到解码阶段的成本。我们在两套桌面系统上使用 Qwen3-14B 和 Qwen3-32B 对 Norn 进行了评估。与 Pipelined Sharding 相比，Norn 的端到端延迟最高可加速 16.8× 倍；与预先已知输入和输出长度时的预知最优配置相比，最高可加速 4.2× 倍。

总而言之，本工作作出了以下贡献：

- 我们表明，静态放置会因预留 GPU 内存以应对未来的KV缓存增长而降低短上下文解码性能。
- 我们提出了一种动态规划方法，该方法预先计算多个执行计划，并优化计算分配和放置，同时确定它们的切换点。
- 我们提出了算子优先分块预填充，并自动选择分块大小，以减少重复的权重传输，同时限制内存使用量以及预填充到解码的转换开销。
- 我们在搭载 Qwen3-14B 和 Qwen3-32B 的双图形处理器系统上评估了 Norn，结果表明，在多种设置下，其推理延迟低于流水线分片。

2 背景与动机

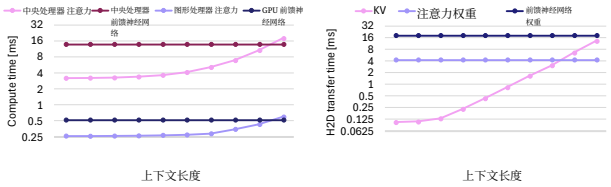
2.1 卸载式大语言模型推理的调度

在卸载式大语言模型推理中，模型权重和键值缓存可以存放在图形处理器内存或主机内存中，而计算则可以分配给中央处理器或图形处理器。GPU执行可以采用驻留执行或流式执行：驻留执行会将权重保留在图形处理器内存中，而流式执行则将权重存储在主机内存中，并在需要时将其传输到图形处理器。同样，注意力访问的键值缓存既可以驻留在图形处理器内存中，也可以在执行期间从主机内存传输过来。

在其他计算进行期间，传输后续 GPU 计算所需的权重和 KV缓存，可降低推理延迟。现有方法在考虑这种重叠的同时，优化计算在中央处理器或图形处理器上的分配，以及权重和 KV缓存的放置。例如，逐层调度（Sugiura 等人，2026）将每一层分配为由中央处理器执行、GPU驻留执行或 GPU 流式执行，并在中央处理器计算期间传输后续层的权重，以最大化重叠。流水线分片（Ukarande 等人，2026a）以注意力和FFN子层为粒度确定计算分配和放置，并优先让注意力权重和 KV缓存驻留于图形处理器。在结合中央处理器计算与 GPU 流式执行的执行计划中，该方法会在中央处理器计算期间预取下一子层的权重，以使计算与数据传输重叠。

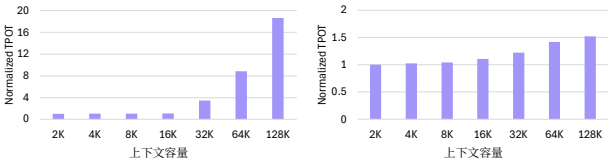
2.2 计算与数据传输成本的变化

随着上下文长度增加，计算与传输成本的变化并不均匀。图 1 展示了 Qwen3-32B 各子层的计算时间和传输时间。当上下文长度从 256 增加到 128K 词元时，注意力计算时间在中央处理器上增加了 5。6× ，在图形处理器上增加了 2。3× 由于注意力会访问前面词元的整个 KV缓存，其计算成本会随上下文长度增加。此外，传输的 KV缓存量随上下文长度线性增长，从而导致



(a) 计算时间。 (b) 传输时间。

图 1：Qwen3-32B 每个子层的计算时间和传输时间。



(a) Qwen3-14B。 (b) Qwen3-32B。

图 2：固定上下文容量对每输出令牌时间的影响。每个模型的每输出令牌时间均以 2K 配置为基准进行归一化。a 123× 传输时间增加。相比之下，中央处理器和图形处理器上的前馈网络计算时间均保持不变，且子层权重传输时间与上下文长度无关。Qwen3-14B 也观察到了类似趋势。计算与数据传输相对成本的这些变化表明，高效的计算分配和放置可能会随上下文长度而变化。

2.3 静态放置的局限性

大多数现有方法在解码过程中不会改变计算任务在中央处理器或图形处理器上的分配，也不会改变权重和键值缓存的放置方式。由于键值缓存会随上下文长度线性增长，静态放置需要预先分配图形处理器内存，以应对未来键值缓存的增长。例如，流水线分片和 llama.cpp（Gerganov，2023）要求指定所支持的最大上下文长度。尽管 Hugging Face 加速（Hugging Face，2025）和逐层调度不要求指定最大上下文长度，但仍必须限制放置在图形处理器上的权重数量，以容纳键值缓存的增长。

这种静态放置会在单个请求的解码过程中，以及在上下文长度不同的请求之间造成性能问题。首先，对于生成较长输出的请求（例如推理任务），解码性能会在生成初期下降。静态放置会为预计在生成末期所需的 KV缓存预留 GPU 内存，从而限制即使在开始阶段 KV缓存很小的情况下仍可驻留在图形处理器上的权重数量。因此，与针对较短上下文进行优化的放置相比，中央处理器计算和权重传输会带来额外的性能开销。此外，如第2.2节所述，计算和传输成本的变化意味着，单一的放置策略可能无法在整个解码过程中始终保持高效。其次，当对不同上下文长度的请求使用相同的静态放置时，必须预留足够的 GPU 内存以容纳较长的请求。这会减少即使对于较短请求也可用于驻留GPU的权重的内存，从而降低推理性能。相反，针对较短上下文定制的放置策略可能会在较长请求中 KV缓存增长时导致 GPU 内存耗尽，使推理无法继续。一种可能的方法是为每个请求指定最大上下文长度。然而，由于实际输出长度无法预先知晓，要求用户为每个请求手动选择适当的设置并不现实。

为量化这一影响，我们在输入长度和输出长度分别固定为 1024 和 32 个词元的情况下，对流水线分片进行了评估，仅改变所支持的最大上下文长度，使其从 2K 变化到 128K 个词元。图 2 展示了结果。对于 Qwen3-14B，每个输出令牌耗时（TPOT）在 16K 设置之前变化不大，但在 32K 和 128K 设置下分别增加到约 3。4× 和 18。7× 倍于 2K 设置的水平。这是因为在 2K 设置下，所有权重和键值缓存都可以驻留在图形处理器上；而增大所支持的最大上下文长度则需要卸载更多权重，以便为键值缓存预留 GPU 内存。对于 Qwen3-32B，与 2K 设置相比，TPOT 在 128K 设置下也增加了 51.8%。这些结果表明，即使实际处理的上下文长度相同，基于未来键值缓存需求的静态放置也会降低解码性能。尤其是，即使能够为每个请求指定合适的最大上下文长度，在整个生成过程中使用相同的放置方式也会阻止系统适应解码期间的性能变化。因此，随着上下文长度增加，必须动态调整权重和键值缓存的放置方式。

2.4 动态放置中的挑战

动态放置要求将执行计划生成与切换点的确定结合起来共同考虑。每个执行计划都必须容纳不断增长的键值缓存，直到下一个切换点，而为此预留的图形处理器内存会限制权重放置和推理性能。因此，放置方案的选择取决于该放置方案将被使用到的上下文长度。在实践中，请求的输入和输出长度各不相同，并且在推理开始时尚未知晓输出长度，这使得这些决策更加困难。

生成执行计划涉及组合式放置搜索问题。对于每个注意力和前馈网络子层，都必须确定计算在中央处理器或图形处理器上的分配，以及权重和键值缓存的放置方式；随着子层数量的增加，候选计划的数量呈组合式增长。TwinPilots (Yu 等人，2024)

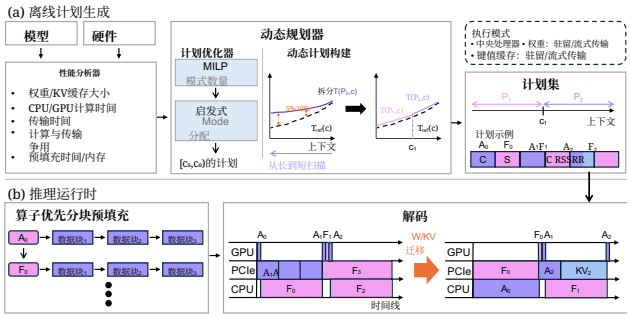


图 3：Norn 概览。

将操作分配到中央处理器和图形处理器的问题表述为一个 NP 完全的负载均衡问题。在解码期间重复执行此类搜索，可能会引入大量计算开销。

确定切换点不仅需要考虑GPU内存约束，还需要考虑随着上下文长度增加，执行计划相对性能的变化。即使某个执行计划满足GPU内存约束，随着上下文长度增加，另一个计划也可能变得更快。然而，切换计划涉及迁移权重和键值缓存，而频繁切换会增加迁移成本。因此，不应在每个上下文长度下独立选择最快的放置方式，而应综合考虑每个计划的执行性能以及计划之间的切换成本，共同确定执行计划集合及其适用的上下文长度范围。

3 提出的方法

3.1 Norn 概述

Norn 是一种卸载式推理方法，会随着上下文长度增加动态切换执行计划。如图 3所示，Norn 由三个组件组成：性能分析器、动态规划器和推理运行时。性能分析器会测量计算时间、数据传输时间以及计算与传输争用。动态规划器基于性能分析结果，针对不同的上下文长度，在线下确定多个执行计划及其切换点。对于预填充，推理运行时执行图形处理器计算，并对较长输入采用算子优先分块预填充。在解码期间，它根据每个计划将计算与数据传输重叠，并在上下文长度达到切换点时迁移必要的数据。这样，无需预测输出长度或在推理期间重新优化执行计划，即可实现动态放置。

3.2 性能分析器

性能分析器在目标硬件上测量计算时间和传输时间，以针对给定的模型和执行环境优化性能。它还会从模型配置中获取每个子层的权重大小和每个词元的 KV缓存大小。对于解码，它会测量每个子层在中央处理器和图形处理器上的执行时间、权重、KV缓存和激活值的传输时间、计算与传输争用，以及启动图形处理器操作的主机端开销。注意力计算时间和 KV缓存传输时间取决于上下文长度，因此会在多个上下文长度下进行测量，并在测量点之间线性插值。对于预填充，它会测量多个输入长度和分块大小下每个子层的图形处理器执行时间以及图形处理器内存使用量峰值。

3.3 动态规划器

动态规划器包括：使用计划优化器优化单个执行计划，以及在考虑性能下降和切换成本的情况下，确定一组执行计划及其切换点。

3.3.1 计划优化器

计划优化器为上下文长度范围 $I = [c_s, c_e$ 生成单个执行计划 P，目标是最小化估计总解码时间。计划 P 由每层注意力和前馈网络的执行模式，以及用于传输权重和键值缓存的缓冲区容量 B 组成。对于前馈网络，有三种执行模式可用，即 CPU执行、GPU驻留执行和GPU流式执行。对于注意力，通过独立选择权重和键值缓存的驻留或流式放置，定义了四种图形处理器执行模式，此外还包括CPU执行。

计划优化器分别优化各层的模式数量及其分配顺序。对于具有 L 层的模型，最多存在 15^L 种执行模式组合。此外，计算与数据传输之间的重叠取决于时间约束，包括模式分配顺序以及传输的开始和完成时间，因此精确的优化公式计算成本很高。然而，一旦确定分配给各模式的子层数量，就可以估计图形处理器内存使用量以及总计算和传输时间。因此，计划优化器首先使用混合整数线性规划（MILP）确定模式数量，然后通过基于模拟器的启发式搜索确定其分配顺序。这样既减少了MILP整数变量的数量，又考虑了执行顺序相关的计算与传输重叠。

该混合整数线性规划将根据性能分析结果估算的解码时间下界作为最小化目标。逐一评估上下文长度范围内的每个解码步骤在计算上代价高昂。因此，将该范围划分为S 个等间隔区间，并将每个区间的中点 c_i 作为代表点。每个代表点都被赋予一个权重 w_i，该权重对应于其所代表的解码步骤数量

。令 τ_i 表示 c_i处每输出令牌时间的下界。目标函数为:

$$\min \sum_{i=1}^{S} w_i \tau_i \tag{1}$$

对于每种子层类型，模式数量之和必须等于层数 L。令 $\mathcal{Q} = \{\text{Attention}, \text{FFN}\}$ 表示子层类型集合， $\mathcal{M}_q$ 表示类型 $q \in \mathcal{Q}$的可用模式集合。整数变量 $x_{q,m} \in \{0, \ldots, L\}$表示分配至模式$m$的类型为 q 的子层数量。所得约束为:

$$\sum_{m \in \mathcal{M}_q} x_{q,m} = L \tag{2}$$

GPU 内存约束通过使用上下文长度范围末端权重和 KV 缓存的驻留内存需求，纳入解码期间 KV缓存的增长。令 $R_{q,m}(c_e)$ 表示上下文长度 c_e 下，类型为 q 且处于模式 m 的子层所需的 GPU驻留内存， M_{base} 表示固定的 GPU内存使用量， M_{GPU} 表示 GPU 内存容量。所得约束为:

$$M_{\text{base}} + B + \sum_{q \in \mathcal{Q}} \sum_{m \in \mathcal{M}_q} x_{q,m} R_{q,m}(c_e) \le M_{\text{GPU}} \tag{3}$$

传输数据超过 B 的模式将被排除，并在必要时施加主机内存容量约束。

在每个代表点 c_i处，令 $C_i(\mathbf{x})$、 $H_i(\mathbf{x})$ 和 $D_i(\mathbf{x})$ 分别表示基于模式数量 $\mathbf{x}$ 得出的计算、主机到设备（ H2D）传输和设备到主机（D2H）传输的总独立执行时间。以下 TPOT 下界受到约束:

$$\tau_i \ge C_i(\mathbf{x}), \quad \tau_i \ge H_i(\mathbf{x}), \quad \tau_i \ge D_i(\mathbf{x}) \tag{4}$$

为考虑计算与传输争用，对重叠执行对执行时间的影响进行近似。令 $C_{i,a}(\mathbf{x})$ 和 $H_{i,b}(\mathbf{x})$ 分别表示计算类型 a 的总计算时间和传输类型 b的总传输时间。这些量根据模式数量汇总独立执行时间。令$z_{i,a,b} \ge 0$表示重叠时长，并令 $r^C_{i,a,b}$ 和$r^H_{i,a,b}$ 分别表示争用条件下计算和主机到设备传输的相对处理速率。这些速率根据性能分析结果估算，并进行归一化，使其在独立执行时均等于 1。使用$\gamma_{i,a,b} = r^C_{i,a,b} + r^H_{i,a,b} - 1$表示每单位重叠时长对应的顺序执行时间与重叠执行时间之差，加入以下约束:

$$\sum_b r^C_{i,a,b} z_{i,a,b} \le C_{i,a}(\mathbf{x}) \tag{5}$$

$$\sum_a r^H_{i,a,b} z_{i,a,b} \le H_{i,b}(\mathbf{x}) \tag{6}$$

$$\tau_i \ge C_i(\mathbf{x}) + H_i(\mathbf{x}) - \sum_{a,b} \gamma_{i,a,b} z_{i,a,b} \tag{7}$$

在重叠时长$z_{i,a,b}$内，已完成的计算工作量和传输工作量分别表示为$r^C_{i,a,b} z_{i,a,b}$ 和 $r^H_{i,a,b} z_{i,a,b}$。前两个约束通过每种类型所需的总工作量，对重叠期间完成的工作量进行上界限制。最后一个约束通过考虑重叠执行节省的时间，为每输出令牌时间设定下界。

因此，混合整数线性规划确定模式数量，而不直接为各个层分配模式。若直接对每个层的模式选择进行建模，则需要 $8L$ 个二元变量。相比之下，该公式仅使用 8 个表示模式数量的整数变量。由于每种子层类型的模式数量之和为 L，因此可能的组合数量为 $O(L^6)$，从而减少了模式分配的指数级搜索空间。

在混合整数线性规划确定模式数量后，计划优化器通过基于模拟器的启发式搜索，确定这些模式在各层之间的分配顺序，以考虑计算与传输重叠。对于模式分配，计划 P 的总^执行时间近似为 $T(P, I) = \sum_{i=1}^{S} w_i T(P, c_i)$，并将其用作评估函数。此处， $T(P, c_i)$ 表示由模拟器估算的 TPOT，同时考虑子层之间的依赖关系以及计算和数据传输的执行调度。

首先，分别针对注意力和前馈网络，使用平滑加权轮询生成初始分配，并以模式数量作为权重，将各模式分配到不同层。具体而言，选择相对于目标比例而言，累计分配量缺额最大的模式。接着，分别对注意力和前馈网络的模式序列进行循环移位，以调整它们的相对分配顺序。固定一个模式序列后，使用模拟器评估另一个序列的循环移位，并选出使 $T(P, I)$ 最小的^分配。随后交替对注意力和前馈网络重复此过程。

接下来，通过交换同一子层类型中两个不同层所分配的执行模式，进行局部搜索。搜索过程中，使用 $T(P, I)$ 比较候选^分配，并以一定概率接受使结果变差的交换，以避免陷入局部最优。保留并选取搜索过程中找到的最佳计划。由于交换模式不会改变模式数量，因此由混合整数线性规划确定的 GPU 驻留内存需求得以保持不变。

针对传输缓冲区容量 B，也会探索多个候选值。增大 B 可为预取提供更多空间，但会减少可用于驻留权重和 KV缓存的 GPU 内存。因此，根据传输数据的大小以及通过模拟检测到的缓冲区不足所推导出的容量阈值，生成候选容量。对于每个候选值，都会执行混合整数线性规划和

分配顺序优化。最后，选择使 $T(P,\ I)$ 最小的 ^可行方案。

*3.3.2 DynamicPlan*构建

动态规划器的目标是最小化分配给各计划的上下文长度范围内的总解码时间之和，以及方案切换所产生的数据迁移时间。动态规划器递归地划分上下文长度范围，然后调整切换点以计入切换成本，并合并相邻范围以消除不必要的方案切换。范围划分依据各计划相对于参考延迟曲线的性能下降情况进行。随着上下文长度增加，注意力计算时间和键值缓存传输时间都会增加，因此仅凭每输出令牌时间增加，无法判断切换计划是否可能带来收益。因此，将每个上下文长度下可行参考方案中的最低每输出令牌时间用作基线，并将性能显著下降的区间视为划分候选区间。

首先，动态规划器构建一条参考延迟曲线，作为评估计划性能下降情况的依据。它从目标上下文长度范围中选取多个参考点，并使用计划优化器为每个参考点生成一个点方案。点方案会使对应参考点处的每输出令牌时间最小。令 $\mathcal{P}_{\text{ref}}$表示参考方案集合。上下文长度为 c时的参考延迟定义为 $T_{\text{ref}}(c) = \min_{P\in\mathcal{P}_{\text{ref}}} T(P, c)$。计划 P相对于参考曲线的相对性能下降定义为 $\delta(P,\ c) = {}^{T_{(P,\ c)}-T_{\text{ref}}(c)}\,{}_{T_{\text{ref}}(c)}$。

动态规划器将上下文长度范围从较长上下文划分到较短上下文。这是因为 KV缓存的大小会随上下文长度单调增加。如果某个计划在一个范围的末端满足 GPU内存约束，那么它在更短的上下文长度下仍然可行。相反，从较短上下文到较长上下文确定计划时，需要根据每个计划将持续使用的时间，为 KV缓存预留 GPU 内存。如果预留容量不足，就必须进行未规划的方案切换。由于预留容量也会影响计划性能，因此很难确定合适的容量。通过从较长上下文到较短上下文确定范围，规划器可以根据每个范围已知的末端分配内存，从而避免因 KV缓存增长而导致的未规划切换。

如果某个计划在其适用范围内相对于参考曲线的性能下降超过 10%，动态规划器就会递归地划分该计划的适用范围。进行范围划分时，它会按上下文长度从长到短扫描每个计划的适用范围，并找出相对于参考曲线的性能下降首次超过 5% 的点。随后，它继续向更短的上下文长度扫描，以找出性能下降首次超过 10% 的点。10% 的阈值用于判断是否需要划分，而 5% 的阈值用于确定划分点。根据此划分规则，动态规划器首先生成一个覆盖整个目标上下文长度范围 $I = [1, c_{\max}）$的初始计划。为防止 $c_{\max}$ 成为搜索划分点的起点， $c_{\max}$ 处的性能下降被限制在 5% 以内。如果不满足此条件，优化范围就会从整个范围 I 改为后半部分$I' = [\ （1+c_{\max}）/2,\ c_{\max}）$，并重新生成计划。由于内存使用量随上下文长度单调增加，重新生成的计划在原始范围 I内仍然可行。接下来，将划分规则应用于初始计划。如果需要划分，就在一个临时切换点处分割范围，并由计划优化器针对得到的范围重新生成计划。重复这一过程，直到不再存在需要通过划分来处理的性能下降。

通过递归范围划分获得的切换点，是基于性能下降确定的临时边界，并不一定能使总执行时间最小化。因此，动态规划器从较短的上下文长度开始，向较长的上下文长度推进，比较相邻计划的解码时间，并将切换点调整到其每输出令牌时间曲线的交点，或由内存容量决定的可行范围边界处。当边界调整后，系统会针对新的适用范围重新生成计划，并且仅当该调整能够降低总执行时间（包括方案切换所产生的数据迁移时间）时，才接受此次调整。

此外，范围划分所带来的解码时间缩短，未必足以抵消方案切换产生的数据迁移成本。因此，动态规划器会针对两个相邻范围合并后的范围生成新计划，并比较合并前后的总执行时间。比较时会同时考虑各范围内的解码时间，以及方案切换时权重和 KV缓存的传输时间。如果合并能缩短总执行时间，就会移除这两个范围之间的切换点。最终的执行计划集合包括每个计划适用的上下文长度范围、各子层的执行模式、传输缓冲区容量，以及计划切换所需的数据迁移信息。

3.4 推理运行时

推理运行时针对预填充和解码采用不同的执行策略。在预填充期间，Norn 将所有计算分配给图形处理器，因为中央处理器上的计算通常比 PCIe 传输更慢。通过将图形处理器计算与后续子层的权重传输重叠，延迟得以降低——

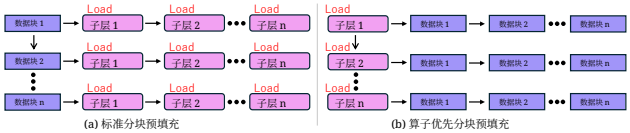


图 4：标准分块预填充与算子优先分块预填充的比较。

将图形处理器计算与后续子层的权重传输重叠。Norn 通过自动调整传输缓冲区大小和采用算子优先分块预填充（OMCP），优化权重传输和激活内存使用量。

Norn 根据性能分析结果确定将图形处理器计算与权重传输重叠所需的传输缓冲区容量。令 C_A 和 C_F 分别表示注意力和前馈网络的图形处理器计算时间，令 H_A 和 H_F 表示它们的权重传输时间。在提前预取一个子层的情况下，每层的稳态执行时间为 $T_1 = \max(C_A, H_F) + \max(C_F, H_A)$，而在计算与传输完全重叠的情况下，理论下界为 $T_{\text{lb}} = \max(C_A + C_F, H_A + H_F)$。如果 $T_1 = T_{\text{lb}}$，提前预取一个子层即可达到该下界。因此，当 $T_1 \leq T_{\text{lb}}$ 时，Norn 会提前预取一个子层，否则会提前预取两个子层。传输缓冲区容量根据预取距离和权重大小计算得出。这确保了预取所需的最小容量，同时兼顾计算与传输重叠效率。

对于较长的输入，会将输入序列划分为多个数据块，以减少激活内存使用量。如图4所示，标准分块预填充（Agrawal 等人，2024）会处理每个数据块的所有层，导致重复传输卸载权重。为避免这些传输，Norn 采用 OMCP，先处理每个子层的所有数据块，再继续处理下一个子层。分层预填充（Lee 等人，2026）通过在混合专家模型服务中按层组处理，减少分块预填充期间从片外 GPU 内存重复加载专家权重的情况。然而，该论文报告称，将其应用于稠密模型时，其性能低于标准分块预填充。相比之下，本文证明，即使在单 GPU 卸载环境中，按子层处理数据块也能有效减少权重传输。

标准分块预填充仅保留每个数据块的隐藏状态，而 OMCP 必须保留整个输入序列的隐藏状态，以便将其传递给下一个子层。令 N 表示输入长度，M 表示分块大小，d 表示隐藏层大小，b 表示每个元素的字节数。额外的内存使用量约为 $(N - M)db$字节。例如，使用 FP16 精度的 Qwen3-32B，在 $N = 128K$ 和 $M = 1K$的情况下，激活内存使用量会增加 1.24 吉比字节。因此，尽管 OMCP 比标准分块预填充需要更多激活内存，但它避免了重复传输权重。尽管如此，它仍大幅降低了峰值与不进行分块的预填充相比，内存使用量

分块大小会影响预填充执行时间和激活内存使用量。较小的分块大小可降低激活内存峰值，并使预填充后更容易在图形处理器上保留初始解码计划所需的权重，但可能会降低计算与传输重叠程度。因此，Norn 使用性能分析结果和估算的内存使用量，评估预填充执行时间与切换到初始解码计划所需的权重传输时间之和。它会在满足图形处理器内存约束的候选分块大小中，选择估算执行时间最低的方案。预填充期间生成的 KV缓存会根据初始解码计划指定的设备放置进行放置；在 GPU 内存容量允许的范围内，初始计划所需的权重会保留在图形处理器上。这样可以减少从预填充切换到解码时的数据传输。

解码期间，每个子层都会在执行计划指定的中央处理器或图形处理器上执行。中央处理器执行使用存储在主机内存中的数据，而图形处理器执行则根据权重和键值缓存的放置情况，使用驻留GPU的数据或从主机内存传输的数据。对于未驻留在图形处理器上的数据，会在后续子层需要之前进行预取，以便让传输与计算重叠。当上下文长度达到切换点时，会在令牌边界切换执行计划。运行时根据新旧计划之间的放置差异，释放不再需要的驻留GPU的权重，传输新需的权重，重新放置键值缓存，并调整传输缓冲区容量；在可能的情况下，复用现有的驻留GPU的数据和内存区域。

权重和 KV缓存的主机到设备传输共用一个 CUDA流，并按照子层执行顺序发起传输。如果可用缓冲区容量不足以容纳较早发起的传输，则不会提前发起后续传输，执行会等待，直到有足够的缓冲区空间可用。当执行设备在中央处理器和图形处理器之间切换时，也会传输激活值。这些传输使用单独的 CUDA流，并进行同步，以满足子层之间的数据依赖关系。在流式执行中，权重和先前生成的 KV缓存保留在主机内存中，因此只有新生成的 KV缓存会从图形处理器传输到主机内存。此传输在注意力计算完成后，通过专用 CUDA流异步执行，并与后续计算重叠。

3.5 实现细节

Norn 的推理运行时基于 PyTorch（Paszke 等，2019）和 Hugging Face Transformers（Wolf 等，2020）实现。Norn 不依赖于任何特定的稠密模型。Norn 还是一种无损方法，不

涉及量化，尽管它可以与量化结合使用。对于图形处理器注意力计算，Norn 使用 PyTorch 的缩放点积注意力。对于中央处理器注意力计算，Norn 使用以 C++、ISPC 和 OpenMP 实现的自定义内核，该内核基于 NEO（Jiang 等，2025）的并行化方法。该内核沿上下文维度对键值缓存进行分区来实现注意力并行化，同时不会为分组查询注意力复制键值头（Ainslie 等，2023）。权重和存储在主机内存中的键值缓存使用页锁定内存。每个子层的权重存储在连续的内存区域中，并在子层级别一同传输。驻留图形处理器的键值缓存不会预先按照最大上下文长度进行分配。相反，其容量根据计划适用范围的末端和请求的生成上限进行分配，并在切换计划时根据需要扩展。

4 评估

4.1 实验设置

测试平台。 我们使用两台台式机进行实验。桌面 1 配备 Intel Core Ultra 7 265K 中央处理器、128 吉比字节主存，以及一块拥有 32 吉比字节显存的 NVIDIA RTX 5090 图形处理器。中央处理器和图形处理器通过 PCIe 5.0 ×16 连接。桌面 2 配备 Intel Core i9-10980XE 中央处理器、128 吉比字节主存，以及一块拥有 24 吉比字节显存的 NVIDIA RTX 3090 Ti 图形处理器。中央处理器和图形处理器通过 PCIe 3.0 ×16 连接。所有实验均使用 8 个线程进行中央处理器计算。

模型。 我们使用 FP16 格式的 Qwen3-14B 和 Qwen3-32B(Yang 等人，2025)，以便在权重和 KV 缓存的不同内存需求下进行评估。

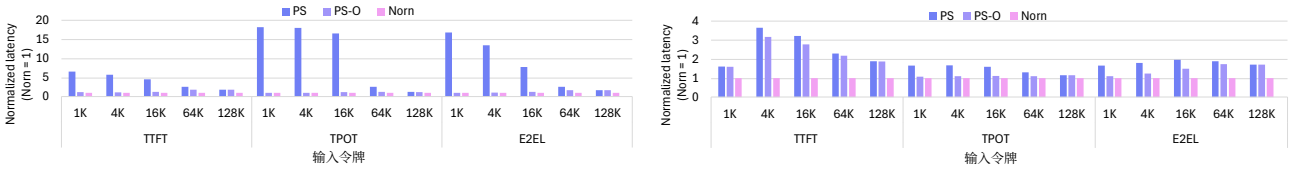
工作负载。 继针对交互式推理开展的先前研究（Chen 等人，2026；Ukarande 等人，2026a；Chen 等人，2025）之后，我们使用批大小为 1 的设置进行了主要评估。为确保可复现性和公平比较，我们采用贪心解码，并固定输出长度。由于我们的方法保持模型精度不变，且不改变所执行的操作，因此我们仅将性能作为输入和输出长度的函数进行评估。我们使用了具有指定长度的虚拟输入序列。评估指标包括首令牌时间（TTFT）、每输出令牌时间（TPOT）和端到端延迟（E2EL）。

基线。 我们将 Norn 与流水线分片（PS）（Ukarande 等人，2026a；b）这一最先进的卸载方法进行了比较。PS 基于 llama.cpp 构建，并具有两个关键参数：`ctx-size` 和 `ubatch-size`。前者指定支持的最大上下文长度，后者指定在预填充或解码期间单次前向传播所处理的最大词元数。对于批大小为 1 的情况，`ubatch-size` 对应于分块预填充的分块大小。为支持多样化的工作负载，我们将 `ctx-size` 设置为 128K，即 Qwen3 支持的最大上下文长度。我们还评估了 PS-O，这是一种预知信息变体假设输入和输出长度预先已知，并据此为每个请求设置 ctx-size。我们将分块大小设置为 1024，这是《流水线分片》论文中常用的设置。

4.2 不同输入长度下的性能

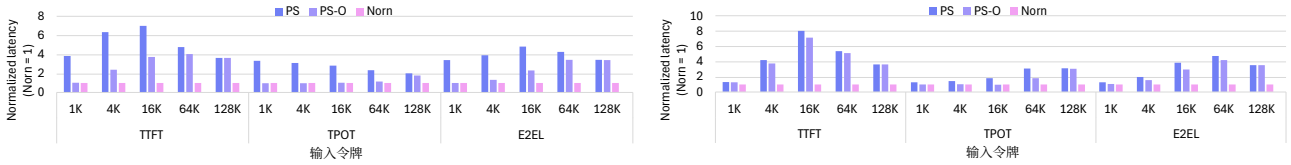
图5 和 6 展示了输出长度固定为 32 个词元、输入长度从 1K 变化至 128K 个词元时的结果。在所有评估条件下，Norn 的端到端延迟均低于 PS，最高可实现 16.8× 倍加速。与假设输入和输出长度预先已知的 PS-O 相比，Norn 在短输入下实现了相当的性能，最高可实现 4.2× 倍加速。在首令牌时间方面，Norn 在所有评估条件下均优于 PS 和 PS-O。尤其是在桌面 2 上使用 16K 词元输入运行 Qwen3-32B 时，Norn 相对于 PS 和 PS-O 的加速比分别达到 8.0× 倍和 7.1×倍。PS 和 PS-O 在分块预填充过程中，会让每个数据块经过所有层处理，并为每个数据块反复传输卸载权重。尽管它们会将紧随其后的子层权重预取至图形处理器，以实现计算与数据传输的重叠，但传输开销并不总能被完全隐藏。相比之下，Norn 使用 OMCP 连续处理每个子层的所有数据块，从而避免重复的权重传输。此外，Norn 会根据计算和传输时间，提前一层或两层预取权重，以提高计算与数据传输之间的重叠程度。这些技术降低了权重传输造成的首令牌时间开销。

在每输出令牌时间方面，提示共享（PS）的最大支持上下文长度固定为 128K，因此性能明显下降，尤其是在输入较短时。对于桌面 1 上的 Qwen3-14B，在输入为 1K 词元时，PS 的每输出令牌时间是 Norn 的 18.2× 倍。相比之下，PS-O 会根据输入和输出长度调整分配的 KV缓存容量，因此在短输入场景下，其性能接近 Norn。然而，长输入时 PS-O 的性能也会下降：对于桌面 2 上的 Qwen3-32B，在输入为 128K 词元时，其每输出令牌时间是 Norn 的 3.0× 倍。放置和预取方面的两项差异可能是 Norn 性能优于 PS-O 的原因。首先，PS-O 根据固定规则决定放置，优先确保注意力权重和 KV缓存驻留于图形处理器，这导致在上下文长度较长时，更多子层被分配到中央处理器执行。对于桌面 1 上的 Qwen3-14B，在输入为 128K 词元时，PS-O 在中央处理器上执行了 32 个 FFN 子层，而 Norn 执行了 19 个。同样，对于桌面 2 上的 Qwen3-32B，PS-O 在中央处理器上执行了 32 个注意力子层，而 Norn 执行了 20 个。这些结果表明，Norn 通过协同优化权重与 KV缓存的放置，以及将计算分配给中央处理器或图形处理器，减少了中央处理器上的计算量。其次，PS-O 的预取仅限于紧接着的下一个子层，而 Norn 可以为后续多个子层预取数据。这可能使 Norn 能够更高效地将中央处理器和图形处理器上的计算与数据传输重叠，并更好地利用权重和 KV缓存的流式执行。



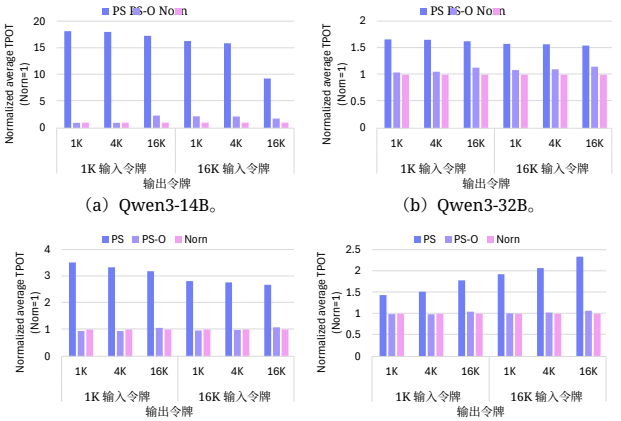
(a) Qwen3-14B。 (b) Qwen3-32B。

图 5：归一化首令牌时间 、Desktop 1 上不同输入长度下的每输出令牌时间和端到端延迟。输出长度固定为 32 个词元 .



（a）Qwen3-14B。 （b）Qwen3-32B。

图 6：Desktop 2 上不同输入长度下的归一化首令牌时间、每输出令牌时间和端到端延迟。输出长度固定为 32 个词元。



（a）Qwen3-14B。 （b）Qwen3-32B。

（c）Qwen3-14B。 （d）Qwen3-32B。

图 7：桌面 1（上）和桌面 2（下）上不同输入和输出长度的归一化平均每输出令牌时间。

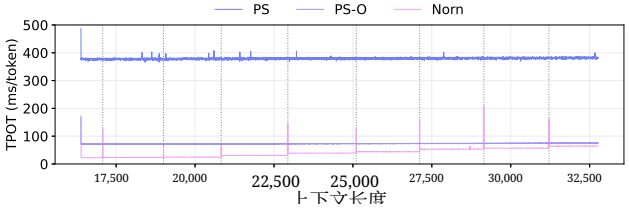


图 8：桌面 1 上，输入为 16K、输出为 16K 词元时，Qwen3-14B 在不同上下文长度下的每输出令牌时间。

以更高效地重叠中央处理器和图形处理器计算与数据传输，并更好地利用权重和KV缓存的流式执行。

4.3 不同输出长度下的解码性能

图7 展示了输入长度固定为 1K 和 16K 词元、输出长度在 1K 、 4K 和 16K 词元之间变化时的平均每输出令牌时间。Norn 在所有评估条件下均优于提示共享，最高实现了 18。1× 倍加速。由于提示共享采用针对最大支持上下文长度 128K 词元配置的静态放置，其性能在较短上下文长度下尤其明显地下降。与

表 1：执行计划数量和规划时间。

配置	最终执行计划	总时间（秒）
桌面 1 / Qwen3-14B	27	1211
桌面 1 / Qwen3-32B	2	245
桌面 2 / Qwen3-14B	21	2081
桌面 2 / Qwen3-32B	12	1643

PS-O 相比，Norn 在 24 种配置中的 16 种配置下实现了更低的平均每输出令牌时间，其中包括全部 8 种输出长度为 16K 词元的配置；在桌面 1 上运行 Qwen3-14B 时，最高实现了 2。33× 倍加速。然而，在某些配置下，PS-O 优于 Norn，尤其是在输出较短时，Norn 的平均每输出令牌时间最高高出 5.8%。PS-O 根据已知的输入和输出长度设置最大支持上下文长度，而 Norn 则在不假设输出长度的情况下，生成覆盖广泛上下文长度的执行计划。因此，对于每个单独的请求，Norn 不一定都能实现最低的平均每输出令牌时间。图 8展示了桌面 1 上 Qwen3-14B 在输入长度为 16K 词元、输出长度为 16K 词元时，每输出令牌时间随上下文长度变化的情况。PS 和 PS-O 的每输出令牌时间几乎保持不变，而 Norn 的每输出令牌时间则因上下文长度增加过程中的执行计划切换而呈阶梯式变化。尽管方案切换时会暂时出现每输出令牌时间峰值，Norn 在生成早期阶段使用更快的计划，即使计入切换开销，平均每输出令牌时间仍分别实现了比 PS 高 9。3× 倍和比 PS-O 高 1。8×倍的加速。这些结果表明，具备上下文长度感知能力的动态放置能够有效提升长输出的解码性能。

4.4 离线规划开销

表1 显示了动态规划器针对最大上下文长度为 128K 个词元时生成的执行计划数量及总规划时间。计划数量和总规划时间分别为 2–27 个和 245–2,081 秒。总规划时间取决于最终计划的数量、针对各个上下文长度范围生成计划的次数，以及每次生成所需的时间

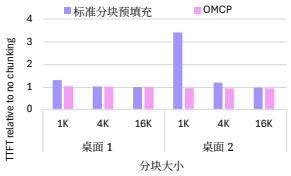


图 9：64K令牌输入下 Qwen3-14B 的归一化首令牌时间。

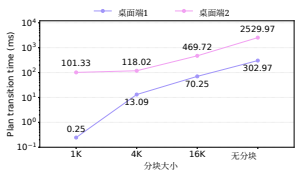


图 10：64K令牌输入下 Qwen3-14B 的计划转换时间。

。在桌面 1 上，针对 Qwen3-14B 和 Qwen3- 32B 各个范围的计划分别生成了 139 次和 8 次，占总规划时间的 64.1% 和 27.7%。对于 Qwen3-14B，桌面 2 所需的计划生成次数较少（83 次），但平均生成时间比桌面 1 更长（13.1 秒，而桌面 1 为 5.6 秒），因此总规划时间更长。我们使用 HiGHS（Huangfu 与 Hall，2018）求解混合整数线性规划问题。在桌面 1 上，针对 Qwen3- 14B 和 Qwen3-32B 分别求解了 3,248 个和 1,152 个混合整数线性规划实例，且全部达到最优解。平均求解时间分别为 84.5 毫秒和 33.5 毫秒，占总规划时间的 22.7% 和 15.7%。

4.5 按算子划分的分块预填充的有效性

我们在桌面 1 和桌面 2 上对 Qwen3-14B 进行了评估，固定输入长度为 64K 词元，分别采用 1K、4K 和 16K 的分块大小，并测量首令牌时间和计划转换时间。图 9展示了相对于不分块配置归一化后的首令牌时间。在桌面 2 上使用 1K 分块大小时，标准分块预填充使首令牌时间达到基线的 3。41× 倍，而 OMCP 仅为 0。93×，相较标准方法提速 3。66× 倍。在桌面 1 上，OMCP 的首令牌时间为 1。02×倍，而标准方法为 1。29× 倍。这是因为 OMCP 会在各数据块之间复用权重，避免重复传输，因此在 PCIe 带宽有限的桌面 2 上尤其有效。随着分块大小增加，两种方法之间的性能差距逐渐缩小，因为更长的计算时间使标准分块预填充能够隐藏权重传输开销。图 10展示了 Qwen3-14B 从预填充切换到初始解码计划的转换时间。使用 1K 分块大小时，OMCP 将桌面 1 上的转换时间从不分块时的 303 毫秒降至 0.25 毫秒，并将桌面 2 上的转换时间从 2,530 毫秒降至 101 毫秒。分块大小越小，转换时间越短。这是因为较小的数据块可降低激活内存峰值使用量，使初始解码计划所需的更多权重和 KV缓存能够保留在图形处理器内存中。这些结果表明，即使分块大小较小，OMCP 也能缩短切换到初始解码计划的转换时间，同时限制首令牌时间开销。

5 相关工作

已有多种方法通过联合利用中央处理器和图形处理器，加速资源受限环境中的大语言模型推理。HeteGen（Zhao 等人，2024）和 TwinPilots（Yu 等人，2024）通过重叠中央处理器计算、图形处理器计算和数据传输来降低推理延迟。FlexInfer（Na 等人，2025）和流水线分片（Ukarande 等人，2026a）根据序列长度和硬件特性确定计算分配与放置。逐层调度（Sugiura 等人，2026）将每一层分配给中央处理器执行、图形处理器驻留执行或图形处理器流式执行，并将权重传输与中央处理器计算重叠。然而，由于 KV缓存的放置取决于每一层所使用的执行设备，权重和 KV缓存无法独立放置。这些方法并未着重研究如何随着解码过程中 KV缓存的增长，动态调整权重与 KV缓存的放置。

PowerInfer（Song 等人，2024；Xue 等人，2024）和 LLM in a Flash（Alizadeh 等人，2024）利用激活稀疏性，通过预测哪些神经元将被激活来减少计算和数据传输。然而，这些方法需要预测器识别活跃神经元，因此在激活稀疏性不足的模型上效果有限。此外，预测误差可能会降低推理质量。FlexGen（Sheng 等人，2023）和 TightLLM（Hu 等人，2025）面向离线批量推理，通过在多个请求间分摊权重传输来最大化吞吐量，但代价是增加延迟。NEO（Jiang 等人，2025）和 APEX（Fan 等人，2026）面向在线服务，通过将 KV缓存和部分注意力计算卸载到中央处理器，并并行执行中央处理器与图形处理器上的计算来提升吞吐量，但前提是假设存在多个并发请求。HeadInfer（Luo 等人，2025）和 InfiniGen（Lee 等人，2024）提出了减少 KV缓存内存使用量和传输开销的方法，但并未联合优化权重与 KV缓存之间的 GPU内存分配。

6 结论

本文提出了 Norn，一种卸载式大语言模型推理方法，可根据上下文长度动态切换执行计划。Norn 会预先计算多个执行计划，优化计算任务在中央处理器或图形处理器上的分配，以及权重和 KV缓存的放置，并确定相应的切换点，以适应解码过程中不断增长的上下文长度。它还采用算子优先分块预填充，避免重复传输权重，同时降低内存使用量。在两套桌面系统上使用 Qwen3-14B 和 Qwen3-32B 进行的实验表明，与流水线分片相比，Norn 的加速比最高可达 16.8×；与其预知最优配置（即预先知晓输入和输出长度的配置）相比，加速比最高可达 4.2× 。

参考文献

Agrawal, A.、Kedia, N.、Panwar, A.、Mohan, J.、Kwatra, N.、Gulavani, B.、Tumanov, A. 和 Ramjee, R. 使用 Sarathi-Serve 驾驭大语言模型推理中的吞吐量—延迟权衡。载于第 *18* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 24*），第 117–134 页，2024 年。

Ainslie, J.、Lee-Thorp, J.、De Jong, M.、Zemlyanskiy, Y.、Lebr´on, F. 和 Sanghai, S. GQA：从多头检查点训练广义多查询变换器模型。载于 *2023* 年自然语言处理实证方法会议论文集，第 4895–4901 页，2023 年。

Alizadeh, K.、Mirzadeh, S. I.、Belenko, D.、Khatamifard, S.、Cho, M.、Del Mundo, C. C.、Rastegari, M. 和 Farajtabar, M. 大语言模型闪存：有限内存下的高效大语言模型推理。在 计算语言学协会第 *62* 届年会论文集（第 *1* 卷：长篇论文），第 12562–12584 页，2024 年。Bloomberg。生成式人工智能 2025：评估不断演变的万亿美元以上市场中的机遇与颠覆。https://assets.bbhub.io/professional/sites/41/Generative-AI-Outlook.pdf，2024 年。访问日期：2026-10-23。Chen, H.、Xie, W.、Zhang, B.、Tang, J.、Wang, J.、Dong, J.、Chen, S.、Yuan, Z.、Lin, C.、Qiu, C.、Zhu, Y.、Ou, Q.、Liao, J.、Chen, X.、Ai, Z.、Wu, Y. 和 Zhang, M. KTransformers：释放 CPU/GPU 混合推理在混合专家模型中的全部潜力。在 *ACM SIGOPS* 第 *31* 届操作系统原理研讨会论文集，第 1014–1029 页，2025 年。Chen, H. M.、Mo, Z.、Lu, G.、Liang, S.、Ma, L.、Luk, W. 和 Fan, H. FastTTS：加速边缘大语言模型推理的测试时扩展。在 第 *31* 届 *ACM* 编程语言与操作系统体系结构支持国际会议论文集，第 *2* 卷，第 732–748 页，2026 年。Fan, J.、Zhang, Y.、Li, X. 和 Nikolopoulos, D. S. APEX：在受限 GPU 上针对在线大语言模型推理的异步 CPU-GPU 并行执行。在 *2026* 年电气电子工程师学会并行与分布式处理研讨会（*IPDPS*），第 1190–1203 页。电气电子工程师学会，2026 年。Gerganov, G. llama.cpp。https://github.com/ggml-org/llama.cpp，2023 年。访问日期：2026-10-14。Hu, Y.、Liu, X.、Yang, G.、Li, L.、Zeng, K.、Zhao, Z.、Chen, S.、Zhao, L.、Li, W. 和 Li, K. TightLLM：最大化 throughput for llm inf推理 v ia adaptive offloadi理推 l - icy。《*IEEE*计算机汇刊》，74(7):2195–2209，2025。Huangfu, Q. 和 Hall, J. J. 并行化对偶修正单纯形法。《数学规划计算》，10(1):119–142，2018。Hugging Face。Hugging Face Accelerate。https://github.com/huggingface/accelerate，2025。访问日期：2026-10-26。Jiang, X.、Zhou, Y.、Cao, S.、Stoica, I. 和 Yu, M. NEO：通过 CPU卸载解决在线大语言模型推理中的 GPU内存危机。《机器学习与系统会议论文集》，7，2025。Lee, G.、Kim, J.、Park, J.、Lee, Y. 和 Ahn, J. H. 从词元到层：利用分层预填充重新定义混合专家服务的无停顿调度。《机器学习与系统会议论文集》，8:398–421，2026。Lee, W.、Lee, J.、Seo, J. 和 Sim, J. InfiniGen：采用动态KV缓存管理的大语言模型高效生成式推理。载于第*18*届*USENIX*操作系统设计与实现研讨会（*OSDI 24*），第155–172页，2024。Li, E.、Larsen, A. B. L.、Zhang, C.、Zhou, X.、Qin, J.、Yap, D. A.、Raghavan, N.、Chang, X.、Bowler, M.、Yildiz, E.、Peebles, J.、Coleman, H. G.、Ronchi, M.、Gray, P.、You, K.、Spalvieri-Kruse, A.、Pang, R.、Li, R.、Yang, Y.、Soroush, E.、Lu, Z.、Xiao, C.、Situ, R.、Huffaker, J.、Grif- fiths, D.、Ahmed, Z.、Zhang, P.、Parilla, D.、Liberman, A.、Mallalieu, J.、Mazaheri, P.、Chen, Q.、Bilkhu, M.、Zhang, A.、Wang, E.、Nelson, D.、FitzMaurice, M.、Voice, T.、Liu, J.、Shaffer, J.、Zhao, S.、Yadla, P.、Rasteh, F.、Guo, P.、Farooq, A.、Snow, J.、Murphy, S.、Lei, T.、Cho, M.、Hor- rell, G.、Dodge, S.、Hislop, L.、Singh, S.、Dombrowski, A.、Raghavan, A.、Sirovica, S.、Saebi, M.、Lao, F.、Lam, M.、Lu, T.、Xu, Z.、Singh, K.、Kirchner, M.、Mizrahi, D.、Arora, R.、Zhang, H.、Mason, H.、Zhou, L.、Hua, Y.、Jain, A.、Bai, F.、Astrauskas, J.、Weers, F.、Gardner, J.、Chi- ang, M.、Zhang, Y.、Agrawal, P.、Sun, T.、Keunebroek, Q.、Hopkins, M.、Wu, B.、Jia, T.、Chen, C.、Zhou, X.、Wang, N.、Liu, P.、Hou, R.、Rauch, R.、Gao, Y.、Dehghan, A.、Janke, J.、Wang, Z.、Chen, C.、Ren, X.、Nan, F.、Elman, J.、Yin, D.、Goren, Y.、Lai, J.、Fei, Y.、Evans, S.、Yu, M.、Yin, G.、Qin, Y.、Feldman, E.、Garg, I.、Rajamani, A.、Vega, K.、Cheng, W.、Collins, T.、Han, H.、Menacho, R. R.、Yeung, S.、Lee, S.、Mutyala, P.、Cheng, Y.-C.、Gan, Z.、Chu, S.、Lazarow, J.、Pappalardo, A.、Scozzafava, F.、Lu, J.、Daxberger, E.、Duchesne, L.、Liu, J.、G¨uera, D.、Ligas, S.、Kery, M. B.、Ramerth, B.、Sannino, C.、Eichner, M.、Huang, H.、Qian, R.、Schwarzer-Becker, M.、Riazati, D.、Gao, M.、Wang, B.、Cackler, J.、Lu, Y.、Niu, R.、Dennison,

J.、Klein, G.、Bigham, J.、Gopinath, D.、Shiee, N.、Botten, D.、Tartavel, G.、Garcia, A. G.、Xu, S.、Haladjian, V. M.、Dou, Z.-Y.、Paulik, M.、Mendez, A. L.、Li, Z.、Chen, H.- Y.、Jia, C.、Doshi, D.、Zhang, Z.、Manjani, R.、Franklin, A.、Ren, Z.、Chen, D.、Peshko, A.、Raghuram, N.、Hao, H.、Shan, J.、Nerella, K.、Tantawi, R.、Kumar, V.、Wang, S.、Wershing, B.、Dhingra, B.、Shah, D.、Adaranijo, O.、Zheng, X.、Madsen, T.、Kotek, H.、Liu, C.、Xia, Y.、Li, H.、Jayaram, S.、Sun, Y.、Fakhry, A.、Saveris, V.、Withers, D.、Li, Y.、Aygar, A.、Teran, A. R. M. Y.、Huang, K.、Lee, M.、Li, X.、Li, Y.、Johnson, T.、Tang, J.、Cheng, J. Y.、Peng, F.、Walkingshaw, A.、Guibert, L.、Sharma, A.、Shen, C.、Maj, P.、Tanaka, Y.、Jhang, Y.-C.、Ma, V.、Vehvilainen, T.、Zou, K.、Nichols, J.、Lei, M.、Qiu, D.、Qian, Y.、San- thanam, G.、Wu, W.、Han, Y.、Moritz, D.、Fu, H.、Xu, M.、Rathod, V.、Liu, J.、D'hauwe, L.、Ba, Q.、Sun, H.、Yan, H.、Dufter, P.、Nguyen, A.、Feng, Y.、Wang, E.、He, K.、Nair, R.、Shah, S.、Lu, J.、Sonnenberg, P.、Warner, J.、Li, Y.、Pan, B.、Zhong, Z.、Zhou, J.、Davarnia, S.、Saarikivi, O.、Belousova, I.、Burger, R.、Wu, S.-C.、Feng, D.、Straathof, B.、Chou, J.、Zhang, Y.、Zuliani, M.、Jimenez, E.、Sun- dararajan, A.、Du, X.、Lan, C.、Shahdadpuri, N.、Grasch, P.、Sima, S.、Newnham, J.、Paidi, V.、Wang, J.、Haag, K.、Braunstein, A.、Molinari, D.、Wei, R.、Yang, B.、Lusskin, N.、Arreaza-Taylor, J.、Cao, M.、Seidl, N.、Wang, S.、Hu, J.、Ma, Y.、Li, M.、Liu, K.、Su, H.、Ravi, S.、Wang, C.、Wang, X.、Smith, K.、You, H.、Karimzadeh, B.、Li, R.、Lei, J.、Fang, W.、Doane, A.、Wiseman, S.、Fernandez, I.、Li, J.、Hansen, A.、Movellan, J.、Neubauer, C.、Zhou, H.、Chaney, C.、Kamaldin, N.、Wolf, V.、Berm´udez-Medina, F.、Pelemans, J.、Fu, P.、Xing, H.、Kong, X.、Shan, W.、Jacoby-Cooper, G.、Shen, D.、Gunter, T.、Seguin, G.、Shi, F.、Li, S.、Xu, Y.、Kamal, A.、Masi, D.、Guha, S.、Zhu, Q.、Thibodeau, J.、Zhang, C.、Callahan, R.、Maalouf, C.、Tsao, W.、Li, B.、Cao, Q.、Sabo, N.、Leong, C.、Wang, Y.、Anupama, A. M.、Reed, C.、Jung, K.、Chen, Z.、Moorthy, M. P. S.、He, Y.、Hornberger, E.、Krishna, D.、Tong, S.、Michael, Lee, Haldimann, D.、Zhao, Y.、Zhang, B.、Gao, C.、Bartels, C.、Rao, S.、Tran, N.、Lehnerer, S.、Giang, C.、Dong, P.、Pan, J.、Wang, B.、Li, D.、Farajtabar, M.、Hwang, D.、Duanmu, G.、Verma, E.、Reddy, S.、Shan, Q.、Gao, H.、Du, N.、Sridhar, P.、Huang, F.、Wang, Y.、Bhendawade, N.、Zhu, D.、Aitharaju, S.、Hohman, F.、Gar- diner, L.、Chiu, C.-C.、Yang, Y.、Kokmen, A.、Chu, F.、Ye, K.、Elgin, K.、Levy, O.、Park, J.、Zhang, D.、Schoop, E.、Wenzel, N.、Booker, M.、Kim, H.、Erdenebileg, C.、Dun, N.、Yang, E. L.、Chhatrapati, P.、Mahtani, V.、Gang, H.、Chia, K.、Seshadri, D.、Yu, D.、Meng, Y.、Peterson, K.、Yang, Z.、Wang, Y.、Peng, C.、Kang, D.、Agarwal, A.、Antony, A.、Tebar, J. L.、Jose, A. M.、Poston, R.、Wang, A. D.、Casamayor, G.、Amirloo, E.、Yao, V.、Kryscin- ski, W.、Duan, K.、and L, L. 苹果智能基础语言模型：技术报告2025，2025。URLhttps://arxiv.org/abs/2507.13575。

Luo, C., Cai, Z., Sun, H., Xiao, J., Yuan, B., Xiao, W., Hu, J., Zhao, J., Chen, B., and Anandkumar, A. Headinfer：通过按头卸载实现内存高效的大语言模型推理，2025年。URLhttps://arxiv.org/abs/2502.12574。

Na, S., Jeong, G., Ahn, B. H., Jezghani, A., Young, J., Hughes, C. J., Krishna, T., and Kim, H. Flexinfer：采用中央处理器计算的灵活大语言模型推理。《机器学习与系统会议论文集》，第7卷，2025年。

Park, D. and Egger, B. 通过中央处理器计算提升面向吞吐量的大语言模型推理。在*2024*年国际并行体系结构与编译技术会议论文集，第233–245页，2024年。

Paszke, A.、Gross, S.、Massa, F.、Lerer, A.、Bradbury, J.、Chanan, G.、Killeen, T.、Lin, Z.、Gimelshein, N.、Antiga, L.、Desmaison, A.、Kopf, A.、Yang, E.、DeVito, Z.、Raison, M.、Tejani, A.、Chilamkurthy, S.、Steiner, B.、Fang, L.、Bai, J. 和 Chintala, S. PyTorch：一种命令式风格的高性能深度学习库。载于 Wallach, H.、Larochelle, H.、Beygelzimer, A.、d'Alch´e-Buc, F.、Fox, E. 和 Garnett, R.（编），《神经信息处理系统进展》，第32卷。Curran Associates公司，2019年。网址：https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf。Pradeep, V. Phi Silica：小巧而强大的端侧小语言模型。https://blogs.windows.com/windowsexperience/2024/12/06/phi-silica-small-but-mighty-on-device-slm/，2024年。访问日期：2026-10-23。Sheng, Y.、Zheng, L.、Yuan, B.、Li, Z.、Ryabinin, M.、Chen, B.、Liang, P.、R´e, C.、Stoica, I. 和 Zhang, C. Flexgen：使用单个GPU实现大语言模型的高吞吐生成式推理。载于国际机器学习会议，第31094–31116页。PMLR，2023年。Song, Y.、Mi, Z.、Xie, H. 和 Chen, H. PowerInfer：使用消费级GPU实现快速大语言模型服务。载于*ACM SIGOPS*第三十届操作系统原理研讨会论文集，第590–606页，2024年。Sugiura, N.、Tokusashi, Y. 和 Matsutani, H. 面向内存受限消费级GPU上大语言模型推理的逐层CPU-GPU调度。载于欧洲并行处理会议，第301–315页。Springer，2026年。Ukarande, A.、Shekhar, D.、Blackstein, M. 和 Rangan, R. 面向客户端的高效显存受限XLM推理。《机器学习与系统会议论文集》，8:1884–1900，2026a。

Ukarande, A.、Shekhar, D.、Blackstein, M. 和 Rangan, R. 流水线分片：MLSys 2026工件，2026b。网址：https://github.com/deepshnv/pipeshard-mlsys26-ae。

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Huggingface 的 transformers：最先进的自然语言处理，2020。网址 https://arxiv.org/abs/1910.03771。

Xue, Z., Song, Y., Mi, Z., Zheng, X., Xia, Y., and Chen, H. Powerinfer-2：在智能手机上实现快速大语言模型推理，2024。网址https://arxiv.org/abs/2406.06282。

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 技术报告，2025。网址 https://arxiv.org/abs/2505.09388。

Yu, C., Wang, T., Shao, Z., Zhu, L., Zhou, X., and Jiang, S. Twinpilots：一种用于 GPU-CPU 并行大语言模型推理的新型计算范式。发表于 第 *17* 届美国计算机协会国际系统与存储会议论文集，第 91–103 页，2024。Zhao, X., Jia, B., Zhou, H., Liu, Z., Cheng, S., and You, Y. Hetegen：面向资源受限设备上大语言模型的高效异构并行推理。《机器学习与系统会议论文集》，6：162–172，2024。