面向解耦式大语言模型推理的解析建模与最优 GPU 配置

Akshat Pandey 印度理工学院孟买分校 孟买，印度

Dominic Cheng 英伟达 圣克拉拉，美国

Sidharth Sharma 印度理工学院焦特布尔分校 焦特布尔，印度

Ashwin Gumaste 英伟达 圣克拉拉，美国

arXiv:2611.28760v1 [cs.ET] 21 Nov 2026

摘要

我们提出 OSplit，这是一种精确的分析方法，可高效地将 GPU L1 域划分为预填充和解码工作进程，并支持经典 GPU 和先进 GPU 的分片。OSplit 会考虑模型参数，使任意推理模型都能划分为预填充和解码工作进程，以实现尽可能高的吞吐量。OSplit 基于容量模型，该模型以模型和加载方案（批大小、最大序列长度）为输入，并对推理服务框架（vLLM）进行建模，从而计算每个 GPU 的效率。为此，我们对 H100 和 GB300 GPU 进行建模，并对扩展域以及基于 FlashAttention 的内核进行建模。OSplit 首先计算最优的集体生成策略，并将其作为容量模型的输入，从而实现预填充 GPU 与解码 GPU 的最优划分。我们使用 vLLM 推理服务，在 H100（NVL8）和 GB300（NVL72）上评估了该方案，并展示了吞吐量、预测延迟和实测延迟；针对主流大语言模型（Llama 和通义千问），理论值与实测值几乎完全吻合。

1 引言

解耦推理已成为服务大语言模型（LLM）的一种有效方法 [15, 18, 24]。大语言模型请求会经历两个不同阶段：预填充，用于处理输入提示并构建键值（*KV*）缓存；以及解码，用于生成自回归响应。预填充阶段计算密集，而解码阶段则受内存和带宽（BW）限制 [24]。解耦式服务将这两个阶段部署在独立的 GPU 工作池中，从而能够分别进行优化，在满足延迟约束的同时实现高吞吐量。解耦式部署支持异构部署，使云服务提供商能够在预填充阶段使用最先进的 GPU，在解码阶段使用较旧的 GPU 或 LPU 等专用加速器。这种专门化为改善首个令牌时间（TTFT）、令牌间隔时间（TBT）、GPU 利用率以及每美元每秒生成令牌数提供了机会，同时还能降低资本支出和运营支出。

然而，要实现这些优势，必须回答一些基本的系统设计问题。首先，应如何在预填充和解码之间分配固定的 *GPU* 预算？这种分配必须在机架内或更大规模的扩展域中实现高效利用，并且必须反映两个阶段不同的处理需求。其次，应如何将分配给每个阶段的 *GPU* 组织成模型实例和并行组？张量并行、数据并行，以及对于混合专家（MoE）模型而言的专家并行（EP），都会影响每个池能够支持的计算量、模型权重存储、通信和请求并发度。

分配问题源于每个请求都必须经过这两个阶段。在解耦式服务中，预填充池处理提示，并将生成的 KV 缓存传输到解码池，由后者生成输出词元。因此，部署的稳态请求吞吐量受处理能力较低的池限制。向已拥有过剩容量的池添加 GPU，并不会提升端到端吞吐量。因此，将这些 GPU 分配给瓶颈池，反而能够提升吞吐量，直至两个池的容量达到平衡。最佳分配比例并非固定不变，而是取决于模型架构、参数、GPU特性、服务框架以及输入和输出序列长度（ISL、OSL）。每个池的并行配置还会进一步改变其有效容量。虽然平衡两个池似乎能得出理想分配方案，但在实际中，GPU 必须分配给完整的模型实例，并采用可行的张量并行、数据并行和专家并行配置。因此，最终的优化问题既相互耦合又具有离散性：看似均衡的 GPU 数量可能不可行；在考虑内存需求、通信成本和服务约束后，其性能也可能不如其他分配方案。这对于服务异构智能体工作负载的 AI 工厂尤为重要，因为提示和响应长度可能存在显著差异 [2]。

因此，运营方可以维护多个解耦部署，每个部署针对一定范围的请求长度进行配置，并将传入的提示词路由至合适的部署。历史请求长度直方图可用于确定这些范围，而每个部署内部的资源分配则必须根据其工作负载和系统特性来决定。现有方法通常会在目标硬件上对模型进行性能分析，根据测量结果拟合性能模型，并搜索候选配置 [1, 24, 35, 40]。此类测量的计算成本高昂，并且当模型、图形处理器、服务框架或工作负载发生变化时，必须重新进行。

我们提出了*OSplit*，这是一种用于解耦推理中图形处理器分区和并行方式选择的解析方法。给定 GPU 预算、工作负载特征、模型规格以及硬件和服务框架参数，OSplit 会估算预填充池和解码池的处理能力，并选择使两者吞吐量中较低者最大化的配置。该方法不依赖大量部署测量，

而是结合所需的硬件参数，使用解析容量模型来分析候选配置的计算、内存和通信成本。

OSplit 会针对每个阶段枚举可行的并行配置，并考虑模型兼容性、图形处理器内存容量以及可选的 TBT 约束。借助第 3 节中建立的容量模型，它会估算每次预填充及每个解码输出词元所需的图形处理器时间。这些成本揭示了能够平衡两个池的理想连续分配。由于这种分配可能无法对应整数个完整模型实例，OSplit 随后会考察预填充和解码并行度的可行组合及其可实现的图形处理器分配，并选择使阶段吞吐量最小值最大的组合。其输出包括分配给各个池的图形处理器数量、相应的并行度以及推理吞吐量。

我们的主要部署目标是 L1 扩展域，例如 GPU 机架，不过这一分析框架旨在扩展到更大的域规模。该方法能够在目标部署上逐一执行所有候选配置之前，对资源配置进行推理。它还明确说明了为什么首选拆分会随着工作负载、模型和硬件而变化，而不是将配置视为黑箱搜索的结果。我们使用 H100 GPU 和 GB300 NVL72 上的 vLLM 验证该模型，并研究所选拆分如何随工作负载、模型和 L1 域规模而变化。

本文结构安排如下。§ 2 介绍系统架构、执行模型和资源配置问题。§3 推导两个池的容量，§ 4 提出 OSplit。§ 5 介绍其实现和 vLLM 集成；§6 在 H100 GPU 和 GB300 NVL72 上验证该模型；§ 7 考察工作负载、模型和 L1 域规模对所选拆分方案的影响。§ 8 讨论相关工作；§9总结全文。

2 系统架构与问题表述

本节介绍 GPU 部署环境以及预填充和解码工作进程的组织方式。随后，我们介绍所支持的并行配置，并在内存和延迟约束下，形式化描述如何分配固定的 GPU 预算以最大化推理吞吐量的问题。

2.1 部署环境与通信域

我们考虑一个由 GPU 计算节点组成的 AI 工厂，这些节点通过高带宽扩展网络互连。每个计算节点都包含 CPU 和 GPU，并通过 NVLink 实现 GPU 到 GPU 的通信。机架级系统可以采用胖树拓扑来组织 GPU 和 NVLink 交换机 ASIC。我们将 *GPU 域* 定义为由 OSplit 配置的固定 N GPU 集合。我们的评估考虑两种配置：八 GPU 的 H100 NVL8 节点，以及配备 72 个 GPU 的 GB300 NVL72 机架。OSplit 决定一个域内在预填充工作进程和解码工作进程之间的 GPU 分配，而硬件拓扑和互连带宽则是性能模型的固定输入。

2.2 请求执行与模型实例

外部请求通过数据中心网络进入，并被路由至服务主机；在服务主机上，vLLM [11] 将请求组织成批次并启动 GPU 内核。每个请求都会经历预填充阶段，该阶段处理输入提示并生成 KV 缓存，随后进行自回归解码。工作负载由平均输入序列长度 ISL I和平均输出序列长度 OSL $\bar{O}$表征。

一个 GPU 域可容纳一个或多个模型实例，具体取决于模型大小、精度和可用 GPU 内存。多个实例能够并发处理独立批次。尽管某些实现倾向于采用二次幂并行度，OSplit 的一般形式化描述并未施加这一限制；不过，部署特定的可行性约束仍可能限制可选方案。

2.3 张量、数据和专家并行

在一个模型实例中，张量并行（TP）会将层权重和注意力计算划分到 g_t 个图形处理器上，每个图形处理器负责部分操作 [30]。数据并行（DP）会在 g_d 个组之间复制计算，以便并发处理不同批次。对于稠密模型，我们将每个独立复制的模型视为一个单独的服务实例：一个稠密实例具有 $g_d = 1$，并占用 $G = g_t$ 个图形处理器。因此，部署额外的稠密模型副本会增加实例数量，而不是增加实例内的数据并行度。

对于混合专家（MoE）模型，专家并行（EP）会将一层中的专家分布到多个图形处理器上 [12]。在所建模的配置中，一个 MoE 实例的 $G = g_t g_d$ 个图形处理器组成一个专家并行组。注意力计算会在 g_d 个组上复制，每组通过张量并行划分到 g_t 个图形处理器上，而专家则分布在整个 G 个图形处理器组成的组中。因此，MoE 实例中的 g_d 份注意力副本并不等同于 g_d 个独立的稠密模型实例：它们共享同一专家并行组织。可以将该实例视为一个用于注意力计算的 $g_t \times g_d$ 网格，以及一个用于专家的专家并行组。模型的专家负载均衡器（之后通过 Λ 表示）会影响其性能。OSplit 会为各个阶段选择张量并行/数据并行配置，相应的专家并行组织则由该模型确定。

每个预填充实例使用 $G_{\text{PF}} = g_t^{\text{PF}} g_d^{\text{PF}}$ 个 GPU，每个解码实例使用 $G_{\text{DC}} = g_t^{\text{DC}} g_d^{\text{DC}}$ 个 GPU。在每个池内，我们假设常规实例采用相同的并行配置。部署还可以将一个规模更小、优先级更低的尽力而为实例紧凑地部署到剩余的 GPU 上。

例如，假设 OSplit 将 72 个 GPU 中的 24 个分配给预填充，将 48 个分配给解码。在部署限制为使用 2 的幂次方规模的 TP 组且满足适用的延迟约束的情况下，一种可行的组织方式是部署三个 8-GPU 预填充实例和三个 16-GPU 解码实例，总计使用 72 个 GPU。该示例展示了池级分配如何映射为完整实例。

2.4 资源配置问题

由于每个请求都必须经过预填充和解码阶段，可持续请求速率取两个资源池容量中的较小值。对于给定的工作负载和并行度选择，令 c_{PF} 和 c_{DC} 分别表示相应资源池中每个已分配图形处理器所能支持的可持续请求速率。这些量取决于各阶段特定的 TP/DP 配置，以及第 3 节中建立的解析成本模型。如果有 k_{PF}

表 1：符号说明。下标或上标 PF 或 DC 表示预填充池或解码池。

工作负载		*LLM*	
I	提示中的平均词元数	L	层数
$\bar{O}$	响应中的平均词元数	H_q, H_{kv}	查询头；KV 头
S	解码期间请求的 KV 缓存平均长度（以词元计）	h_{kv}	每个图形处理器的 KV 头
L1 域和并行度		d_h	单个注意力头的大小
N; m、n	配置的 GPU 数量（通常为一个 L1 域）；预填充 GPU 数量、解码 GPU 数量	$E; k$	每层专家数量；每个词元会发送到的专家数量
g_t、g_d	实例的张量并行度和数据并行度	W	每个 GPU 的模型权重字节数
G	每个实例的 GPU 数量 $= g_t g_d$	κ	每个 GPU 每个词元的 KV 缓存字节数
服务框架		*GPU*	
T_b	一次预填充迭代的令牌预算	M_{HBM}	HBM 内存容量
U	框架可使用的 GPU 内存比例	F; N_{SM}	峰值 FP8 张量核心吞吐量；流式多处理器数量
M_{res}	框架为自身预留的内存	β_H, β_{L2}	达到的高带宽内存和 L2 带宽
批次和时间		容量和资源配置	
B	批大小	$c_{\text{PF}}, c_{\text{DC}}$	池中一台图形处理器可持续处理的每秒请求数
b	每份模型副本的批大小 $= B/g_d$	ϕ	L1 域的吞吐量，每秒请求数
R	一层处理的令牌行数	r^*	预填充 GPU 与解码 GPU 的最佳比率
τ_{PF}	预填充迭代时间（对一个提示词批次)	P	预填充阶段每个请求的 GPU 秒数
τ_{DC}	解码迭代时间（一次前向传播的时间，每个请求生成一个词元）	D	解码阶段每个输出令牌的 GPU 秒数
τ	令牌间隔时间的上限	Λ	负载最高的图形处理器的专家负载相对于平均值的比率

个预填充实例和 k_{DC} 个解码实例，其图形处理器分配如下：

$$m = k_{\text{PF}} G_{\text{PF}}, \qquad n = k_{\text{DC}} G_{\text{DC}}.$$

我们旨在寻找能使稳态请求吞吐量最大化的切分与并行配置：

$$\begin{aligned} \max \quad & \phi = \min\left(m\, c_{\text{PF}},\ n\, c_{\text{DC}}\right) \\ \text{subject to} \quad & m + n \le N, \\ & \text{every instance fits in GPU memory}, \\ & \tau_{\text{DC}} \le \tau. \end{aligned} \tag{1}$$

优化变量包括 GPU 分配 m, n、四个阶段特定并行度（g_t^{PF}, g_d^{PF}, g_t^{DC}, g_d^{DC}）以及整数实例数量。这里 τ_{DC} 表示一次解码迭代的时长，τ 表示允许的 TBT。容量是并行配置的非线性函数，因此这是一个离散的非线性资源配置问题。该模型在满足既定约束的前提下最大化吞吐量。模型利用瓦片级 GPU 性能分析，根据计算、内存访问和通信成本推导预填充与解码吞吐量。§ 3 和 §4 介绍了 OSplit 流程，用于计算容量并选择可行分配。表 1 总结了分析中使用的符号说明。

2.5 输入、输出与假设

输入。 OSplit 的输入包括大语言模型架构和精度；GPU 的计算、内存和互连特性；服务框架内存预留和预填充词元预算；工作负载输入/输出长度分布；以及可用的 GPU 预算 N 和域拓扑。我们对 vLLM 0.26 进行建模。并在 § 5 中介绍 vLLM 0.32。

输出。OSplit 返回预填充与解码的图形处理器比率、每个池的整数图形处理器分配和并行配置，以及预测的每图形处理器吞吐量和聚合吞吐量。

假设与范围。我们考虑在稳态且负载足够高的情况下，将单个大语言模型部署在同构 GPU 上，并规定每个池内采用统一的并行配置。硬件拓扑结构和互连特性是固定输入，而非优化变量。工作负载由其输入序列长度/输出序列长度分布进行描述。因此，OSplit 无需在目标硬件上对每个候选项进行穷举分析，即可评估部署配置。图 1 总结了 OSplit 工作流：从估算候选并行配置的预填充和解码容量，到选择 GPU分配，并在 L1 域内组织由此产生的模型实例。

3 GPU 容量模型

本节推导公式（1）c_{PF} 和 c_{DC}中的两个未知容量，这些容量对应一组候选并行度；§ 4 将评估每个有效候选项并选择最优并行度。我们以 NVIDIA H100 GPU 作为参考架构，介绍分析公式。

3.1 容量

在稳态下，两个池处理相同的请求流，因此我们都以每秒请求数来衡量它们的容量，这是两者吞吐量必须匹配的单位。一个请求会为两个池带来不同数量的工作量：I 预填充阶段的提示词词元和 $\bar{O}$ 解码阶段生成的词元，因此两者的每秒词元数不同，无法直接比较。一个预填充实例以一个批次处理 B_{PF} 个提示词，在时长为 τ_{PF}的一次迭代中处理它们，然后开始下一个批次，因此每 B_{PF} 完成 τ_{PF}个请求。一个解码实例持有 B_{DC} 个请求，每个请求持续 $\bar{O}$ 次时长为 τ_{DC}的迭代，因此该实例每 B_{DC} 完成 $\bar{O}\tau_{\text{DC}}$ 个请求（公式（7）和（8））。这两个结论都源于利特尔法则 [14]：完成率等于正在处理的请求数除以每个请求所耗费的时间，

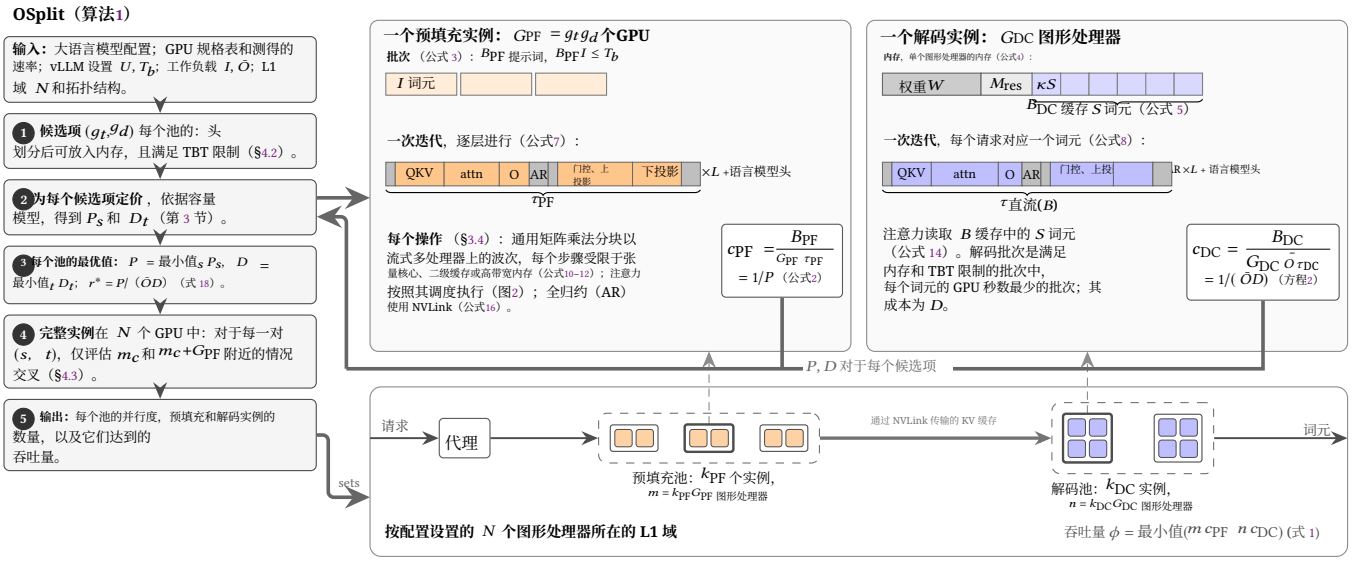

图1：L1 域上的 OSplit。左侧：OSplit算法（算法 1，§ 4）。右上：图形处理器容量模型（§ 3），它为每个候选项提供成本 P 和 D，此处展示的是两个框选实例的成本。下方：生成的 L1 域，其中包含 2 图形处理器预填充实例和 4 图形处理器解码实例。

因此，对于每个图形处理器，

$$c_{\mathrm{PF}} = \frac{B_{\mathrm{PF}}}{G_{\mathrm{PF}}\,\tau_{\mathrm{PF}}}, \qquad c_{\mathrm{DC}} = \frac{B_{\mathrm{DC}}}{G_{\mathrm{DC}}\,\bar{O}\,\tau_{\mathrm{DC}}}. \tag{2}$$

要评估方程（2），我们需要了解每个池的批大小（§ 3.2）和迭代时间（§ 3.3）。

3.2 批次大小

预填充批次受单次迭代处理的词元数量限制；解码批次则受内存容量限制，因为解码实例会将整个提示的 KV 缓存保留在 GPU 内存中，直到生成最后一个词元。

在 vLLM 中，预填充令牌预算 T_b 限制了单次迭代中处理的词元数量。每个模型副本会将完整提示词批处理到不超过此词元上限的大小，同时受其 KV 缓存可用 GPU 内存的限制。由于每个提示包含 I 个词元，且一个实例拥有 g_d^{PF} 个模型副本，因此预填充批大小为：

$$B_{\mathrm{PF}} = g_d^{\mathrm{PF}} \min\left(\left\lfloor \frac{T_b}{I} \right\rfloor, \left\lfloor \frac{UM_{\mathrm{HBM}} - W - M_{\mathrm{res}}}{\kappa\, I} \right\rfloor\right). \tag{3}$$

第二项表示在分配模型权重和框架预留的内存后，剩余内存能够容纳其 KV 缓存的提示词数量；W、M_{res} 和 κ 的定义如下。我们设定 $T_b \geq I$，从而确保每个提示都能容纳在令牌预算内，并且每个模型副本在每次迭代中至少可以处理一个提示词。

解码批次大小取决于可用于 KV 缓存的 GPU 内存量。参数 gpu_memory_utilization 用于设置可使用的 GPU 内存比例 U M_{HBM}。模型权重在每个 GPU 上占用 W 字节，而框架会为自身缓冲区预留 M_{res}（峰值激活内存、CUDA 图内存、上下文和通信缓冲区）。由于每个请求平均需要 κS 字节，剩余内存决定每个模型副本可容纳多少个请求。对于包含 g_d^{DC} 个副本的实例，KV 缓存能够容纳的最大批次为

$$B_{\mathrm{DC}} = g_d^{\mathrm{DC}} \left\lfloor \frac{UM_{\mathrm{HBM}} - W - M_{\mathrm{res}}}{\kappa\, S} \right\rfloor. \tag{4}$$

W 以及 κ 取决于候选并行度。并行度是一个输出：模型会针对每个候选项计算公式（4），而第 § 4 节会选出最优方案。

输出长度 $O(I)$ 取决于输入提示和模型特征，因此难以在执行前预测。因此，我们将 $O(I)$ 建模为一个随机变量，其分布为 $\hat{}O$，平均值为 $\bar{O} = \mathrm{E}[O(I)]$，，根据工作负载进行估计。在解码过程中，请求的 KV 缓存长度会从 I增长到 $I + O(I)$ 个词元。在满负载情况下，解码批次槽位会被后续请求持续占用。因此，随机观察到的槽位更有可能包含较长的响应，其位置在该响应的 [6]范围内均匀分布。因而，平均缓存长度 S 为（c_O 表示 $O(I)$ 的变异系数）：

$$S = I + \frac{\mathrm{E}\left[O(I)^2\right]}{2\,\mathrm{E}\left[O(I)\right]} = I + \frac{\bar{O}}{2}\left(1 + c_O^2\right), \tag{5a}$$

$$S \approx I + \frac{\bar{O}}{2}. \tag{5b}$$

对于每层中的每个键值头，缓存都会存储一个键和一个大小为 d_h 的值向量。每个 GPU 上的键值头数量取决于候选张量并行度 g_t。在 vLLM 中，g_t 必须整除查询头数量 H_q，并且 g_t 必须整除 H_{kv}，或者 H_{kv} 必须整除 g_t。键值头会均匀分布在 g_t 个 GPU 上，或者每个 GPU 持有一个键值头，并且每个头会在 g_t/H_{kv} 个 GPU 上复制。每元素字节数为 e_{kv}（FP8 时为 1），则一个词元在单个 GPU 上占用 κ

字节：

$$\kappa = 2\,L\,h_{kv}\,d_h\,e_{kv}, \qquad h_{kv} = \max\left(1, \frac{H_{kv}}{g_t}\right) \tag{6}$$

公式（4）表明，最优 GPU 划分取决于服务框架、模型架构和硬件特性。更高的张量并行度会增加每个 GPU 可容纳的请求数，因为每个 GPU 存储的权重份额更小，每个请求的缓存也更小；但这一趋势会持续到每个 GPU 仅容纳一个键值头为止。

3.3 预填充和解码时间

预填充和解码的迭代时间 τ_{PF} 和 τ_{DC} 是公式（2）的输入。这两个时间与批次大小共同决定每个池的容量，进而决定应如何在预填充和解码之间划分 GPU。在预填充阶段，模型一次处理整个批次；而在解码阶段，模型会为批次中的每个请求生成一个新词元（一次迭代）。这两个阶段都按层执行相同的操作序列，但每项操作的工作量各不相同。我们首先用这些操作表示各阶段的迭代时间，然后在下一小节推导每项操作所需的时间。

一次前向传播会重复执行层计算 L 次，然后执行最终归一化和语言模型头，以获取下一个词元的分数。当图形处理器上有 b 个请求时：

$$\tau_{\text{PF}} = L\left(T_{\text{layer}}(bI) + T_{\text{kv}}\right) + T_{\text{head}}, \tag{7}$$

$$\tau_{\text{DC}} = L\,T_{\text{layer}}(b) + T_{\text{head}} + T_{\text{samp}} + T_{\text{host}}. \tag{8}$$

这里 T_{kv} 表示每层将提示中的键和值写入缓存所需的时间； T_{head} 涵盖最终归一化、语言模型头计算，以及跨 g_t 个 GPU 的集体通信。 T_{samp} 是采样下一个词元所需的时间， T_{host} 是 vLLM 每个解码步骤固定的 CPU 时间，每个图形处理器测量一次。两个阶段的语言模型头都针对每个请求处理一行，因为只有提示的最后一个位置会生成词元。

g_d 个数据并行模型副本并行处理 b 个请求组成的独立批次。这些副本同时对大小相同的批次执行相同的操作，只有在混合专家层进行词元交换时才会交换数据，因此我们对单个副本所需的时间进行建模。在每个副本内部， g_t 个张量并行 GPU 处理模型的不同分片，同时执行相同的一系列操作。在 vLLM 中，每层的内核会在每个 GPU 上依次运行，因此它们的耗时会累加（在 GB300 上进行解码时，当 GPU 上的词元数不超过 256 个，DeepSeek 的共享专家会与路由专家重叠执行）；单个 GPU 处理 b 个请求时，一层所需的时间为：

$$T_{\text{layer}}(R) = 2\,T_{\text{norm}}(R) + T_{\text{ATT}}(R) + T_{\text{FFN}}(R), \tag{9}$$

其中，

$$\begin{aligned} T_{\text{ATT}} &= T^{QKV}_{\text{GEMM}} + T_{\text{attn}} + T^{O}_{\text{GEMM}} + T_{\text{AR}}(g_t), \\ T^{\text{dense}}_{\text{FFN}} &= T^{\uparrow}_{\text{GEMM}} + T^{\downarrow}_{\text{GEMM}} + T_{\text{AR}}(g_t), \\ T^{\text{MoE}}_{\text{FFN}} &= T_{\text{AG}}(G) + T_{\text{RS}}(G) + \Lambda\left[T^{\uparrow}_{\text{exp}} + T^{\downarrow}_{\text{exp}}\right]. \end{aligned}$$

其中， R 表示一层处理的令牌行数。层时间 T_{layer} 是注意力模块、前馈模块以及两次归一化的总和，每个模块之前各进行一次归一化。注意力时间 T_{ATT} 包括 QKV 投影、注意力计算、输出投影，以及一次用于合并各 g_t结果的全归约张量并行 GPU。前馈时间取决于模型架构。对于稠密模型，$T^{\text{dense}}_{\text{FFN}}$包括融合门控与上投影、下投影以及一次全归约。对于混合专家模型， $T^{\text{MoE}}_{\text{FFN}}$包括将词元分发给选定的专家、计算专家输出并返回结果。与 vLLM 一样，分发是在 G GPU 上执行的全收集，返回则是归约散射。 Λ 表示负载最高的 GPU 相对于平均值的预期负载（§ 3.4）。

各阶段的主要区别在于每个操作接收的工作量不同。在预填充阶段，每个提示贡献 I 个词元行，因此大小为 b 的批次会为层操作提供 $R = bI$ 行。在解码阶段，每个请求仅贡献一个新词元，因此提供 $R = b$ 行。预填充注意力在每个提示内部关联词元，因此其工作量会随 I呈二次增长。而解码注意力则读取每个请求的键值缓存，其平均长度为 S。因此，预填充时间取决于批大小和提示长度，而解码时间取决于批大小和缓存长度。

上述方程用各个操作的操作时间表示了预填充和解码迭代时间。接下来，我们将根据 GPU 的计算、内存和通信开销推导这些时间。

3.4 操作时间

我们根据各个操作所使用的资源来估算其时间：张量核心、二级缓存、高带宽内存以及 GPU 之间的链路。该模型使用 GPU 的规格，以及每种 GPU 类型测量一次得到的一些值，例如实际达到的 HBM 带宽和 vLLM 每步固定的主机开销，而不是对单个操作进行性能分析。对于通用矩阵乘法和注意力，我们基于 TileSight [19], 的分块级公式，并将其扩展到 FP8算术 [17], 、不同的分块尺寸和分割K因子，以及用于混合专家的分组通用矩阵乘法。本小节概述了每个操作的建模方式。

通用矩阵乘法。 通用矩阵乘法将一个由 $M \times K$ 激活值组成的矩阵与一个 $K \times N$权重矩阵相乘。图形处理器以包含 $t_m \times t_n$ 个元素的数据块执行这一计算。每个线程块计算一个数据块，并将 K 维度分成大小为 t_k的块进行处理。使用分割K时， K维度会在 n_s 个块之间进行划分，并在最后合并各块的部分结果，如 [23]所述。因此，该内核会启动 n_{cta} 个块，每个块执行 n_k个步骤，

$$n_{\text{cta}} = n_s\left\lceil\frac{M}{t_m}\right\rceil\left\lceil\frac{N}{t_n}\right\rceil, \qquad n_k = \left\lceil\frac{K}{n_s\,t_k}\right\rceil. \tag{10}$$

图形处理器每次最多在每个 N_{SM} 流式多处理器上调度一个块，因此会有 $p = \min(n_{\text{cta}}, N_{\text{SM}})$个块同时运行，这些块分为$n_w = \lceil n_{\text{cta}}/N_{\text{SM}}\rceil$ 个波次运行，最后一个波次可能未满。用 CUDA 术语来说，内核以包含 n_{cta} 个线程块的网格启动；块的索引 (blockIdx.x, blockIdx.y) 决定它处理的输出数据块，而 blockIdx.z 则决定它的分割K切片。在每个步进中，每个块都会加载 $v_a = t_m t_k e_w$ 字节的激活值和 $v_b = t_k t_n e_w$ 字节的权重，其中 e_w是每元素字节数。这些数据中有些可能已存在于 L2 中，其余部分则必须从高带宽内存中读取。从高带宽内存读取的字节数为$v_{\text{miss}} = v_a(1 - h_A) + v_b(1 - h_B)$，其中 h_A和 h_B分别是激活值数据和权重数据在 L2 中的命中比例。当有 p个块并发运行时，每个步进的时间取决于以下三种资源（张量核心、L2缓存或高带宽内存）中哪一种构成瓶颈：

$$\tau_{\text{step}} = \max\left(\frac{2\,t_m t_n t_k}{F/N_{\text{SM}}},\ \frac{p\,(v_a + v_b - v_{\text{miss}})}{\beta_{L2}},\ \frac{p\,v_{\text{miss}}}{\beta_H}\right), \tag{11}$$

其中 F/N_{SM} 是单个流式多处理器的计算速率。

当内核流水线填充并写入输出时，每个波次也会产生开销。在前 δ 步中，其中 δ 为内核流水线深度，流水线仍在加载数据，因此受内存带宽限制。其余步骤以 τ_{step} 运行，波次最后通过写入输出块结束，这需要 τ_e。因此，一个波次的时间 T_{wave} 为：

$$T_{\mathrm{wave}} = \delta\,\tau_{\mathrm{mem}} + \max(n_k - \delta, 0)\,\tau_{\mathrm{step}} + \tau_e, \tag{12}$$

与 τ_{mem} 方程（11）中的两个内存项中较大的那个。总通用矩阵乘法时间通过累加 $n_w - 1$ 个完整波次和最后一个波次的时间得到；使用分割K时，还需加上一次归约过程。模型会评估适合共享内存的分块尺寸以及可用的分割K因子，并选择估计执行时间最低的配置 T_{GEMM}。

我们根据内核将输出块分配给块的顺序来估算 h_A 和 h_B。数据块在首次访问时会未命中；后续访问只有在该数据块仍位于二级缓存中时才会命中，这取决于期间加载了多少其他数据块。我们使用 TileSight 的栈距离模型 [19]，计算这一概率；该模型将二级缓存建模为组相联LRU缓存，并假设数据块随机映射到缓存组。

该模型根据每个通用矩阵乘法所处理的工作量，刻画预填充与解码之间的差异，而不是假设前者受计算限制、后者受内存限制。解码处理的行数较少，因此读取模型权重占通用矩阵乘法操作移动的大部分字节数，HBM 带宽可能成为该操作的瓶颈。预填充处理的行数较多，因此张量核心吞吐量可能成为瓶颈。传统的屋顶线 [33] 估算可以识别这些总体趋势，但无法体现有多少数据由 L2缓存提供，也无法体现最后一个部分填充的波次所造成的额外时间。

专家通用矩阵乘法。混合专家层会将一个大型前馈网络通用矩阵乘法替换为许多较小的通用矩阵乘法：每个图形处理器持有 E/G 个专家，并为每个专家分别执行一次矩阵乘积。我们使用与稠密通用矩阵乘法相同的逐步模型来估算这些矩阵乘积的时间，即公式（11）所示的模型，并采用 vLLM 专家内核中的分块尺寸。

在预填充阶段，一个批次包含 $B_{\mathrm{PF}}I$ 个词元激活值，每个激活值会被路由到 E 个专家中的k 个。因此，每个专家平均会接收约 $B_{\mathrm{PF}}Ik/E$ 个词元。由于路由具有随机性，且某些专家比其他专家更常被选中，实际分布会因批次而异。由于图形处理器必须完成各自的工作后，该层才能继续执行，因此接收词元最多的图形处理器决定了执行时间。我们使用 Λ来衡量这种负载不均衡；该值表示负载最高的 GPU 的预期负载相对于平均值的比率，取决于词元数量和分布的不均匀程度。

在解码阶段，每个请求每次迭代只贡献一个词元，因此一个批次只会用到部分专家。对于包含 B_{DC}个请求的批次，每个图形处理器上活跃专家数量的期望值为：

$$a = \frac{E}{G}\left(1 - \left(1 - \frac{k}{E}\right)^{B_{\mathrm{DC}}}\right). \tag{13}$$

因此，解码迭代只需读取该批次激活的专家的权重。由于这些权重读取占了解码期间传输的大部分数据，活跃专家的数量对决定层的执行时间起着关键作用。

动态特性：预填充和解码对实例大小的需求方向相反。在预填充阶段，将专家分散到更多 GPU 上会使每个 GPU 分到的专家更少，并可能加剧负载不均衡，从而提高 Λ。因此，较小的实例对于预填充可能更高效。但在解码阶段，较大的实例会减少每个 GPU 在每一步需要读取的活跃专家权重数量，并为 KV 缓存留出更多内存。因此，模型在预填充时可能更适合较小的实例，而在解码时更适合较大的实例。

注意力。在解码阶段，每个请求都会读取其包含 S 个词元的 KV 缓存。当批次较小时，这项工作可能不足以让图形处理器的所有流式多处理器都保持繁忙。内核可以将每个缓存拆分为σ 个部分，并在不同的流式多处理器上并行处理，但必须合并部分结果。不进行拆分时，内核为每个请求和键值头运行一个块，总计 $b\,h_{kv}$ 个块，可能少于 N_{SM} 个流式多处理器。将每个缓存拆分为 σ 个部分后，可得到$\sigma b\,h_{kv}$ 个块。模型采用内核自身拆分规则中的 σ（H100 上的 FA-3 [29]），并让这些块在流式多处理器上分波次运行：

$$T_{\mathrm{attn}}^{\mathrm{DC}} = \left\lceil \frac{\sigma\, b\, h_{kv}}{N_{\mathrm{SM}}} \right\rceil \max\left(t_{\mathrm{comp}},\ \frac{v_{\mathrm{blk}}}{\beta_H},\ t_{\mathrm{lat}}\right) + t_{\mathrm{merge}}(\sigma), \tag{14}$$

其中，块以实际达到的带宽读取 v_{blk} 字节的键和值，t_{comp}：张量核心、特殊函数和 CUDA 核心上的计算时间，t_{lat}：其依赖加载的延迟，t_{merge}：合并部分结果的成本。在 GB300 上，MLA 解码内核采用类似的方式建模，每个内核都有自己的拆分规则和合并操作。

在预填充期间，每个词元都会关注自身以及提示中位于它之前的词元。内核以 t_m行作为一个块处理查询；对于每个查询块，它会处理位于其之前的 t_n 键块，每次更新一个键块。平均而言，一个查询块会进行 J_{eff} 次此类更新。每次更新都会加载一个键和值块，在张量核心上将查询与键相乘，在特殊函数核心和 CUDA 核心上计算 Softmax，再将得到的注意力权重与值相乘。由于乘积计算和 Softmax 在不同单元上运行，连续更新可以重叠，因此所需时间取决于这些操作的执行顺序。图 2 展示了连续键块更新的执行过程，以及内存传输、张量核心操作和 Softmax 计算之间的重叠情况。我们找出这些操作在依赖关系允许的情况下所需的最短执行顺序，分别针对一次更新（T_1）和连续两次更新（T_2）。因此，总注意力时间为：

$$T_{\mathrm{attn}}^{\mathrm{PF}} = \min_{t_m, t_n}\left[T_1 + (J_{\mathrm{eff}} - 1)\,\max\left(T_2 - T_1,\, u_{\max}\right) + T_{qo}\right]. \tag{15}$$

首次更新耗时 T_1。连续更新发生重叠后，每次额外更新增加 $T_2 - T_1$，但下限为 $u_{\max}$，即每次更新中最繁忙资源所需的时间。T_{qo} 计入了加载查询和写入输出的时间。

集合通信。集合通信操作会在多个阶段中交换 GPU 之间的数据。每个阶段的耗时取决于最慢的传输，而每次集合通信都会增加固定的延迟 α。我们将总耗时建模为：

$$T_{\mathrm{coll}} = \alpha + \sum_{\text{stages}} \max_{\text{senders, links}} \frac{\text{bytes}}{\beta_{\mathrm{NV}}}. \tag{16}$$

在 L1 域内的 g 个 GPU 上对 X 字节执行全归约时，环形算法需要 $2(g-1)$ 个阶段，每个阶段发送 X/g 字节

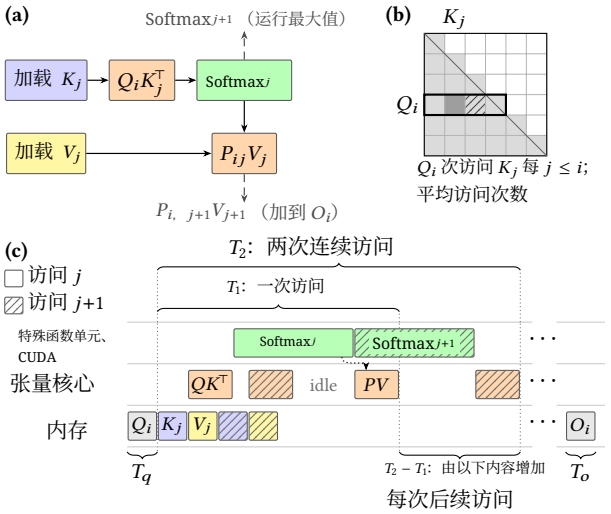

图 2：一个查询块的预填充注意力，展示操作序列。(a) 在使用键块 j 进行更新时，查询块 i 加载 K_j 和 V_j，计算 $Q_iK_j^\top$，应用 Softmax，并将结果乘以 V_j。下一次更新使用更新后的运行最大值，并累加到同一输出 O_i 中。(b) 使用因果掩码时，查询块 i 仅处理键块 $j \le i$。(c) 连续更新之间存在重叠：更新 j 的 Softmax 运行时，张量核心便开始处理更新 $j+1$。因此，两个更新耗时为 T_2，而非 $2T_1$；此后每次更新增加 $T_2 - T_1$，此处该值等于每次更新中最繁忙资源的耗时，即 Softmax 所需时间 ($u_{\max}$)。

阶段。因此， $T_{\mathrm{AR}} = \alpha + 2(g-1)X/(g\,\beta_{NV})$。全收集或归约-散播的阶段数是全归约的一半，而全互换是一个单独的阶段，在该阶段中每个图形处理器发送 $(g-1)X/g$ 字节。这些通信成本源于 TP 和 EP。

其他操作。对于归一化、KV缓存写入、采样和其他逐元素操作，我们根据传输的数据量和图形处理器的 HBM 带宽估算执行时间。

这样便完成了模型。本小节的运算时间相加后，得到迭代时间 τ_{PF} 和 τ_{DC} （§3.3）；再结合 §3.2中的批次大小，即可得到方程（2）中的两个容量，它们是方程（1）所述资源配置问题中仅有的未知数。

4 OSplit 方法

本节求解方程（1）中的资源配置问题。根据 §3中计算出的预填充和解码容量，对于每种候选并行度，我们推导出理想的 GPU 划分和每 GPU 吞吐量，为每个池选择最佳并行度，并确定要在含有 N 个 GPU 的 L1 域中部署的实例数量。

4.1 均衡划分

对于固定的并行度选择，吞吐量受容量较低的资源池限制。向该资源池分配更多 GPU 会提升吞吐量，直到两个资源池的容量相匹配；我们的目标是从吞吐量角度计算出最佳匹配。当 L1 域中的 GPU 数量使得最优分配向上取整后，预填充和解码的 GPU 数量都为整数时，

我们将这种分配称为可行。例如，若预填充与解码的比率为 1.5，且 L1 域有 72 个 GPU，则该分配不可行（预填充为 43.2 个 GPU，解码为 28.8 个 GPU）；若比率为 1.4，则该分配可行（预填充为 42 个 GPU，解码为 30 个 GPU）。可行情形满足 $m+n=N$，且余量最小（两组中最多各有 1/2 个 GPU 未使用于 SM+高带宽内存），并且满足 $m\,c_{\mathrm{PF}} = n\,c_{\mathrm{DC}}$。我们可以用各阶段处理一个请求所需的 GPU 成本来表示这种平衡。令：

$$P = \frac{G_{\mathrm{PF}}\,\tau_{\mathrm{PF}}}{B_{\mathrm{PF}}}, \qquad D = \frac{G_{\mathrm{DC}}\,\tau_{\mathrm{DC}}}{B_{\mathrm{DC}}} \tag{17}$$

表示一个请求在预填充阶段所需的 GPU秒数，以及解码阶段生成一个输出令牌所需的 GPU秒数，其中 §4.3 选择解码批次 B_{DC} 以最小化 D。根据方程（2），$c_{\mathrm{PF}} = 1/P$ 以及1 $\bar{O}D$ / ()，因此，使吞吐量最大化的预填充 GPU 与解码 GPU 之比，以及由此得到的每 GPU 吞吐量为：

$$r^* = \frac{m}{n} = \frac{P}{\bar{O}\,D}, \qquad \frac{\phi^*}{N} = \frac{1}{P + \bar{O}\,D}. \tag{18}$$

服务一个请求的 GPU 总成本为 $P + OD$，因此每个 GPU 的理想吞吐量为 $1/(P + \bar{O}D)$。

4.2 选择并行度

根据公式（18），每个 GPU 的吞吐量仅通过并行度之和 $P + \bar{O}D$ 受到影响。 P 仅取决于预填充并行度 $(g_t^{\mathrm{PF}},\ g_d^{\mathrm{PF}})$ 而 D 仅取决于解码并行度 $(g_t^{\mathrm{DC}},\ g_d^{\mathrm{DC}})$。使 P 最小的预填充并行度是最佳选择，使 D 最小的解码并行度是最佳选择。 r^* 由这两个最小值确定。由于 P 和 D 取决于不同的并行度选择，我们可以分别对它们进行最小化，从而求得连续最优值。整数实例可能改变最佳组合，因此 § 4.3 会搜索所有满足每个池各放入一个实例后仍能装入 N 个 GPU 的组合。

公式（1）的其余约束决定哪些并行度属于候选项。组合 $(g_t,\ g_d)$ 属于候选项，当且仅当：

（1） H_q 可被 g_t 整除；同理， H_{kv} 可被 g_t 整除，或者 H_{kv} 可整除 g_t。（2）实例能够装入内存：对于预填充，每个模型副本至少容纳一个提示， $B_{\mathrm{PF}} \ge g_d$ 见方程（3）；对于解码，每个模型副本至少容纳一个请求，$B_{\mathrm{DC}} \ge g_d$ 见方程（4）。（3）对于解码，如果设定了令牌间隔时间上限 τ，则当每个副本的请求批次至少为 1 时，该上限能够得到满足。

令牌间时间约束仅适用于解码。随着解码批次增大，每次迭代耗时会更长，因为通用矩阵乘法要处理更多行，而且注意力机制必须读取更多 KV缓存。因此，该约束会将批次限制为：

$$B_\tau = \max\{B \le B_{\mathrm{DC}} : g_d \mid B,\ \tau_{\mathrm{DC}}(B/g_d) \le \tau\}. \tag{19}$$

解码池随后会采用不超过 B_τ、且能使 D最低的批次。

4.3 资源配置流程

算法1 结合上述容量模型与优化结果，确定应如何为 L1 域配置资源。它会考虑每个池的候选并行配置，估算其成本，并选择能够最大化吞吐量的配置和图形处理器分配。

算法 1在包含 N 个 GPU 的 L1 域中执行 OSplit。

输入：模型、GPU、框架和工作负载参数；L1 域大小 N；可选的 TBT 限制 τ
输出：连续切分 r^* 和吞吐量 ϕ^*/N；对于 N 个 GPU，最佳并行度组合、池大小和吞吐量 ϕ_{int}

C_{PF}，$C_{\text{DC}} \leftarrow$ 候选并行方式（§4.2）
对于每个 $s \in C_{\text{PF}}$ **执行**
  计算$B_{\text{PF},s}$；丢弃 s 当且仅当$B_{\text{PF},s} < g_{d,s}$
  计算 τ_{PF} ,s
  $P_s \leftarrow G_s \tau_{\text{PF},s}/B_{\text{PF},s}$
结束循环
对于每个 $t \in C_{\text{DC}}$ **执行**
  $\bar{B}_t \leftarrow B_{\text{DC},\ t}$ 内存上限
  如果 τ 已指定**则**
    $\bar{B}_t \leftarrow \max\{B \le \bar{B}_t : \tau_{\text{DC},t}(B) \le \tau\}$
  结束如果
  丢弃 t 如果 $\bar{B}_t < g_{d,t}$
  对于每个 $B \le \bar{B}_t$ 满足$g_{d,\ t} \mid B$ **执行**
    $\delta_t(B) \leftarrow G_t\, \tau_{\text{DC},\ t}/B$
  结束循环
  $D_t \leftarrow$ 最小化$_B\, \delta_t(B)$
结束循环
$s^* \leftarrow$ 参数最小值$_s\, P_s$；　$t^* \leftarrow$ 参数最小值$_t\, D_t$
$r^* \leftarrow P_{s^*}/(\bar{O}D_{t^*})$；　$\phi^*/N \leftarrow 1/(P_{s^*} + \bar{O}D_{t^*})$
$\phi_{\text{int}} \leftarrow 0$
对于每一对 $(s,\ t) \in C_{\text{PF}} \times C_{\text{DC}}$ **执行**
  $c_{\text{PF}} \leftarrow 1/P_s$；　$c_{\text{DC}} \leftarrow 1/(\bar{O}D_t)$
  $m^* \leftarrow Nc_{\text{DC}}/(c_{\text{PF}} + c_{\text{DC}})$；　$m_c \leftarrow G_s\lfloor m^*/G_s \rfloor$
  当 $m_c c_{\text{PF}} > n(m_c)\ c_{\text{DC}}$ **时**
    $m_c \leftarrow m_c - G_s$
  结束当
  对于每个 $m \in \{m_c,\ m_c + G_s\}$，其 $n(m) \ge G_t$ **执行**
    $\phi \leftarrow$ 最小$(mc_{\text{PF}},\ n(m)\, c_{\text{DC}})$
    如果 $\phi > \phi_{\text{int}}$，**则**
      $\phi_{\text{int}} \leftarrow \phi$；记录 $(s,t,m,n(m))$
    结束条件判断
  结束循环
结束循环
返回 $r^*, \phi^*/N$，记录的 (s,t,m,n) 和 ϕ_{int}

候选项是一对$(g_t,\ g_d)$，用于指定某个实例的张量并行度和数据并行度。我们只考虑满足第 § 4.2 节约束的配置。对于稠密模型，我们固定 $g_d = 1$，因为数据并行副本可以改为视作独立实例。对于混合专家模型，我们还会搜索 g_d，因为它会影响专家的分布；我们要求 G 能够整除专家数量。预填充和解码的条件有所不同，因此我们分别维护候选集C_{PF} 和C_{DC}。我们用 s 和 t 分别表示预填充候选项和解码候选项。

该流程分为三个阶段。首先，它计算每个候选项的成本。对于预填充候选项 s，它计算批大小$B_{\text{PF},\ s}$ 和迭代时间$\tau_{\text{PF},\ s}$；两者的比率乘以实例大小后，即为每个请求的 GPU 秒数，$P_s = G_s\tau_{\text{PF},\ s}/B_{\text{PF},\ s}$。如果权重和完整令牌预算对应的激活值无法装入内存，或者一个模型副本无法容纳一个提示，则丢弃该预填充候选项。对于解码候选项 t，内存决定最大批大小；如果该批次少于每个模型副本一个请求，则丢弃该候选项。在剩余批次中，流程选取每个输出令牌的 GPU 秒数 D_t 最低的批次。其次，分别使 P_s 和 D_t 最小的候选项给出公式（18）中的连续最优值。最后，流程为包含 N 个 GPU 的 L1 域求出最佳分配。由于每个池必须包含整数个实例，将 m 个 GPU 分配给预填充后，解码池将获得剩余 GPU 中可组成完整实例的部分，即 $n(m) = G_{\text{DC}}\lfloor (N-m)/G_{\text{DC}} \rfloor$。随着 m 增加，预填充容量上升，而解码容量下降。由于吞吐量取两者中较小值，因此对于给定的一对候选项，其最大值要么出现在 m_c，要么出现在 $m_c + G_{\text{PF}}$；其中 m_c 是使预填充容量不超过解码容量的最大 G_{PF} 倍数。对于每一对候选项，流程从连续平衡点 $m^* = Nc_{\text{DC}}/(c_{\text{PF}} + c_{\text{DC}})$开始，以 G_{PF} 为增量逐步递减，直至达到 m_c，评估两种分配方案，并返回吞吐量最高的配置，以及其并行度选择和实例数量。

针对每个候选项，以及解码侧的每个允许批次，都会评估容量模型，整数搜索会重复利用这些结果。因此，该过程根据解析模型和 GPU 特定参数推导出部署配置，而无需对每种部署进行性能分析。

4.4 容量误差敏感性

我们量化了选择与理想比率 r^*不同的 GPU 划分所造成的吞吐量损失。对于连续的池大小，比率为 r 的划分可达到以下比例的最优吞吐量：

$$\frac{\phi(r)}{\phi^*} = \begin{cases} \dfrac{r}{r^*} \cdot \dfrac{1+r^*}{1+r}, & r \le r^* \ \text{(prefill limits)}, \\ \dfrac{1+r^*}{1+r}, & r \ge r^* \ \text{(decode limits)} \end{cases} \tag{20}$$

吞吐量损失取决于误差的方向。如果比率相对偏低 a，吞吐量约下降 $a/(1+r^*)$；而比率偏高 a 时，吞吐量约下降 $r^*a/(1+r^*)$。这些损失并不对称：当$r^* < 1$时，为预填充分配不足的 GPU 比分配过多的代价更高。

这决定了容量模型的哪些误差会产生影响。如果模型预测 P和 D的相对误差分别为 ϵ_P和 ϵ_D，则根据公式（18）可得预测比率：

$$\hat{r} = r^* \frac{1+\epsilon_P}{1+\epsilon_D}. \tag{21}$$

两个阶段共有的误差会相互抵消，它可能改变预测的吞吐量，但不会改变建议的拆分方式。只有预填充和解码之间的误差差异才会改变拆分方式，并可能根据公式（20）降低吞吐量。

5 使用 vLLM 实现和部署

5.1 模型

解析模型使用 Python 实现，并在一台基于 CPU 的系统上运行，该系统配备 AMD Ryzen 7 7840HS 处理器和 16 GB 内存。对于每种候选配置，模型都会预测批次大小和迭代时间，将每次迭代拆解为各项操作，并识别限制每项操作的资源

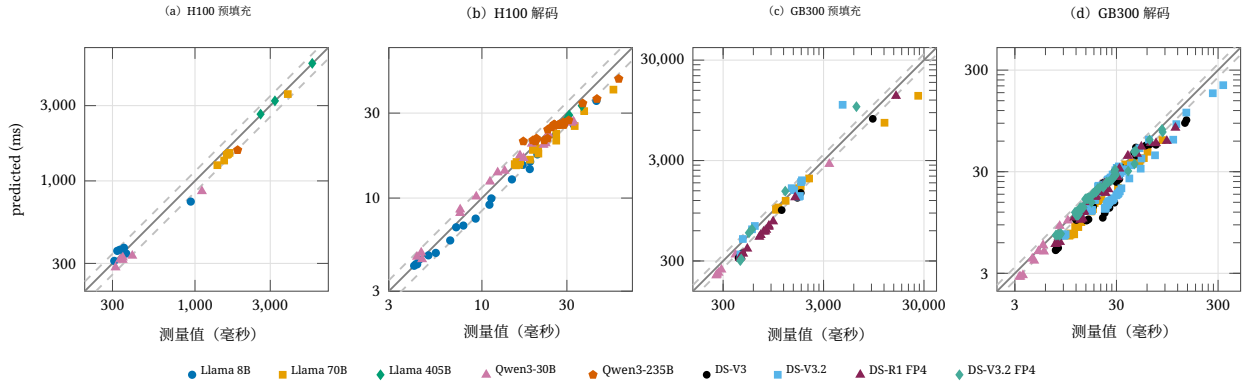

图 3：所有评分点的步进时间预测值与实测值对比：(a、b) H100；(c、d) GB300。实线：相等；虚线：±15%。DS：DeepSeek。

表 2：主要硬件常数。†稠密，浮点运算/时钟周期/流式多处理器。

常数	H100	GB300
SMs	132	152
流式多处理器时钟频率，MHz	1,830	1,957；1,335–1,447
FP8；NVFP4 速率†	5,327；–	15,971；29,061
HBM 带宽	3.13 TB/s	7.4 TB/s
L2 带宽；大小	7.18 TB/s；50 MB	10.2 TB/s；128 兆二进制字节
L1.5 带宽；大小	–	22.5 TB/s；2 MB‡
TMEM 读取；每个 SM 的写入	–	2.08；1.47 TB/s
NVLink 带宽 (β_{NV})	396 GB/s	587–784 GB/s
集体通信延迟 (α)	6.6–19.9； 5 μs	3.1–49.8 μs
主机每步时间	6.0–52 μs，增加 1.5–10.9 ms，最低值	
GPU 内存	81,081 兆二进制字节	276.5 吉二进制字节

。这样，我们就能追溯预测结果，了解决定性能的操作和硬件瓶颈。模型使用一次性测得的 GPU 常数，详见表 2。

5.2 使用 vLLM 部署

算法 1 100, 0 0 26 0 32 0 300 可为 vLLM 生成配置（在 H 上为 . .，在 GB 上为 . .）。对于稠密模型，张量并行度 g_t 通过 `--tensor-parallel-size`设置。对于混合专家模型，我们还将`--data-parallel-size` 设为 g_d 并启用 `--enable-expert-parallel`，从而得到 g_d 个注意力副本，每个副本使用 g_t 个 GPU，专家并行组则覆盖全部 G 个 GPU。令牌预算 T_b、内存占比 U以及每个模型副本的解码批大小分别对应 `--max-num-batched-tokens`、`--gpu-memory-utilization`和 `--max-num-seqs`。

预填充实例和解码实例通过 vLLM 的 NIXL 连接器通信 [21], , 该连接器会通过 NVLink 传输每个请求的 KV 缓存。以 NVLink 带宽传输 KV 缓存所需的时间为 ~1计算时间的百分比。此外，传输与下一次预填充重叠，因此不会产生额外开销（§ 6）。

5.3 扩展至 GB300

我们将算法1 及其基础数学扩展到采用 vLLM 0.32.0 的 GB300 NVL72 机架。容量模型和资源配置搜索保持不变，但解码批次会填充至 vLLM 捕获 CUDA 图所需的大小。主要扩展是对 §3.4中的操作时间模型进行扩展，以纳入 Blackwell 的张量核心和内存层次结构，以及 vLLM 选择的内核，包括 FA-4 [38]。这些模型涵盖多头潜在注意力 [3], 稀疏 [5]和线性注意力 [10], FP4 权重 [22], 宽域专家并行 [4] , 以及在 CUDA 图之外执行的内核的启动成本。通用矩阵乘法和注意力模型也考虑了 Blackwell 特有的功能，例如张量内存和双层 L2 缓存。与 H100 一样，硬件常数来自数据手册和微基准测试。

6 评估

我们从三个方面测试 OSplit：KV缓存容量、预填充和解码步进时间，以及节点在预填充与解码之间的每种拆分方式的吞吐量。本节中的所有预测均来自我们的解析模型。在 H100 上，我们针对五种模型测量了 11 个 KV 容量和 100 个步进时间（22 个预填充、78 个解码），每个实例使用 1 至 8 个 GPU。输入序列长度（ISL）范围为 1024 至 32,768，解码批次大小为 1 至 1000，并测量了六条拆分曲线。对于 GB300，我们针对六种模型测量了 12 个 KV 容量和 336 个步进时间（60 个预填充、276 个解码）。

6.1 设置

我们评估了五种模型：三种稠密模型——Llama 3.1 8B、70B 和 405B [7, 16, 20, 28], , 以及两种混合专家模型——Qwen3-30B-A3B 和 Qwen3-235B-A22B [26, 27], , 每个词元激活 128 个专家中的 8 个 [37]。

硬件与软件。 所有测量均在一台配备 8 个通过 NVLink 连接的 H100 SXM GPU 的节点和一台 GB300 NVL72 上进行。H100 使用 vLLM 0.26.0，GB300 使用 vLLM 0.32.0。**模型。** Llama 3.1 405B 和 Qwen3-235B-A22B 无法在单个节点上拆分，因为一个 405B 实例需要全部 8 个 GPU，而一个 235B 预填充实例和一个解码实例合计需要 12 个 GPU。我们使用 Llama 3.1 8B 和 Qwen3-30B-A3B 测量拆分，每个实例使用 1 个 GPU。Llama 8B 和 Qwen3-30B 每个使用 1 个 GPU，70B 使用 2 个，405B 使用 8 个。在 GB300 上，我们评估 Llama 3.170B、Qwen3-30B-A3B，以及四种 DeepSeek 模型（V3、V3.2、采用 NVFP4 权重的 R1 和 V3.2），每个实例使用 1 至 32 个 GPU。

工作负载。 我们在三种工作负载下测量拆分性能，其平均输入序列长度/输出序列长度分别为：W1（1,024/2,048）、W2（8,192/700）和 W3 （ 32,768/256 ）。提示词由长度精确的随机词元序列构成；输出长度在均值的 ±15%范围内均匀抽取，生成时忽略序列结束词元。

表 3：OSplit 和屋顶线的平均绝对百分比误差（%），括号内为数据点数量。DS：DeepSeek。

	KV	预填充		解码	
	容量	OSplit	屋顶线	OSplit	屋顶线
H100					
Llama 8B	0.9 (2)	9.3 (6)	55.0 (6)	11.0 (20)	40.7 (15)
Llama 70B	2.6 (1)	9.2 (6)	48.2 (6)	10.7 (14)	35.2 (12)
Llama 405B	6.7 (1)	0.7 (3)	43.3 (3)	7.2 (3)	47.2 (3)
Qwen3-30B	3.3 (1)	11.5 (6)	75.4 (6)	9.1 (17)	40.7 (15)
Qwen3-235B	4.0 (6)	15.8 (1)	–	7.9（24）	–
all	3.5（11）	9.0（22）	57.2（21）	9.4（78）	39.7（45）
GB300					
Llama 70B	0.4 (2)	12.2 (12)	47.5 (12)	24.3 (32)	45.1 (32)
Qwen3-30B	1.2 (1)	15.6 (6)	75.0 (6)	8.1 (17)	45.8 (17)
DS-V3	6.5 (3)	21.5 (10)	63.8 (10)	19.8 (40)	56.6 (40)
DS-V3.2	7.1 (2)	21.7 (12)	65.8 (12)	22.4 (95)	66.2 (95)
DS-R1 FP4	5.5 (2)	23.0 (14)	47.4 (14)	13.2 (33)	46.9 (33)
DS-V3.2 FP4	6.4 (2)	26.0 (6)	58.5 (6)	6.0 (59)	52.4 (59)
all	5.0 (12)	19.9 (60)	57.7 (60)	16.8 (276)	55.8 (276)

测量。 我们记录 4 项指标。 内存： vLLM 在启动时报告的 KV 缓存容量。预填充迭代时间： 我们暂停服务器，将一个完整的提示词批次加入队列，然后恢复执行，使该批次作为一次迭代运行。解码迭代时间： 批次达到稳态后每次迭代所需的时间。 拆分吞吐量： 稳态时间窗口内完成的每秒请求数。**基线与指标。** 屋顶线模型 [33] 对每项操作，取其峰值张量核心速率下的浮点运算次数与标称带宽下的 HBM 流量两者中的较大值。 *AIConfigurator* [35] 根据在每个图形处理器上测量并存入数据库的内核时间和通信时间，预测拆分方案。对于每项量值，我们都测量平均绝对百分比误差和带符号误差中位数。对于拆分方案，我们测量模型是否选出了实测效果最佳的方案，以及其预测吞吐量在所有拆分方案上的误差中位数。

6.2 内存

该模型对 vLLM 报告的 KV缓存容量进行预测，在 11 种 H100 配置上的平均绝对百分比误差为 3.5%（各模型为 0.9–6.7%），在 12 种 GB300 配置上的误差为 5.0%（表 3）。该模型会考虑检查点中指定的各模块精度，以及 vLLM 预留的内存。这些因素很重要，例如 Meta 的 FP8 Llama 405B 检查点中，有几个模块仍采用 BF16 [16]。

6.3 迭代时间

图 3对比了预测与实测的预填充和解码时间。

预填充。 在 H100 上，该模型预测预填充时间的平均绝对百分比误差为 9.0%，而屋顶线的误差为 57%（表 3，图3a）。五种模型的误差范围为 0.7–15.8%。屋顶线的误差范围为 43% 至 75%，因为它假设注意力达到峰值张量核心吞吐量，且线程块波次均被完全占用。对于 Llama 8B，随着输入序列长度从 1K 增至 32K，相对误差保持在 −6.3% 至 −8.0% 之间。当某一阶段内各候选项的误差相近时，它会按比例缩放其估计容量，但不会改变最佳候选项。其对最优划分的影响取决于它与解码误差的差异（公式 21）。在 GB300 上（图 3c），预填充的平均绝对百分比误差为 19.9%（各模型为 12–26%），而屋顶线的误差为 58%。

解码。 在 H100 上，模型预测解码时间的平均绝对百分比误差为

表 4：单个 8 GPU H100 节点上的拆分：测量值和两种方法均采用的最佳预填充 GPU 数量，以及各方法吞吐量曲线的中位误差。

		最佳 m	曲线误差	
		（全部三者）	OSplit	AIConfigurator
Llama 8B	W1	1	16.3%	5.0%
	W2	4	13.1%	14.1%
	W3	6	25.0%	20.9%
Qwen3-30B	W1	1	3.7%	38.1%
	W2	3	8.4%	30.0%
	W3	6	27.0%	19.5%

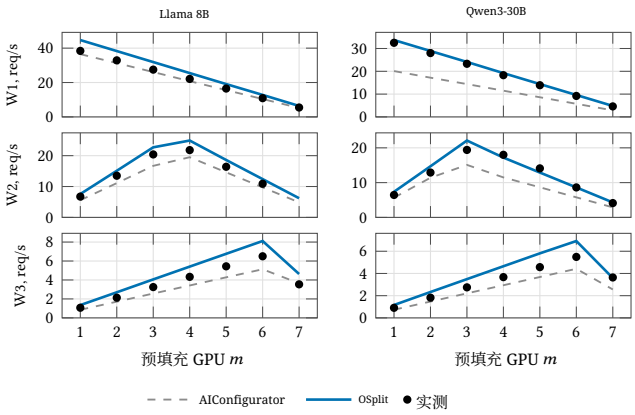

图4：将一个 8 GPU H100 节点拆分为 m 个预填充 GPU 和 $8-m$ 个解码 GPU、每个实例使用一个 GPU 时，各种拆分方案的吞吐量。在全部六种情况下，两种方法都选出了实测表现最佳的拆分方案。

9.4%（各模型为 7.2–11.0%）。相比之下，屋顶线的平均绝对百分比误差为 40%。屋顶线对解码时间的预测低估了 35–47%，因为在小批次下，解码步骤主要受小型内核的固定开销、注意力调用开销以及每个步骤的主机时间支配，而带宽和浮点运算次数都无法捕捉这些因素。在 GB300 上（图 3d），我们的模型对解码时间的平均绝对百分比误差为 16.8%，而屋顶线为 56%。

6.4 OSplit 实战

在全部六种情形中，OSplit 均选出了最优划分（图 4，表 4）。对于 W2，我们的吞吐量中位预测误差为 8.4–13.1%，而 AIConfigurator 为 14.1–30.0%。对于 W3，AIConfigurator 的预测更接近（19.5–20.9%，OSplit 为 25.0–27.0%）。OSplit 的两项特性使这一选择具有稳健性：（1）划分仅取决于比率 $P/(\bar{O}D)$，因此，同时影响预填充和解码的误差会相互抵消（公式（21））；（2）对于 $8\times$ 个 GPU，划分每次只能改变 1 个 GPU，因此，比率中的剩余误差只有大到足以使比率越过相邻两种划分吞吐量相同的临界点时，才会改变选择。对于 W1，解码会限制每一种划分，因此 OSplit 和 AIConfigurator 都会选择最小的预填充池。值得注意的是，OSplit 和 AIConfigurator 在全部六种情形中都找出了实测的最优 GPU 划分（表4），这表明，对于这些工作负载，两种方法都能提供足够准确的容量估计，以选出最佳分配。不过，两者采用的方法不同：AIConfigurator 使用离线分析的算子性能数据库，并枚举候选工作进程分配；而 OSplit

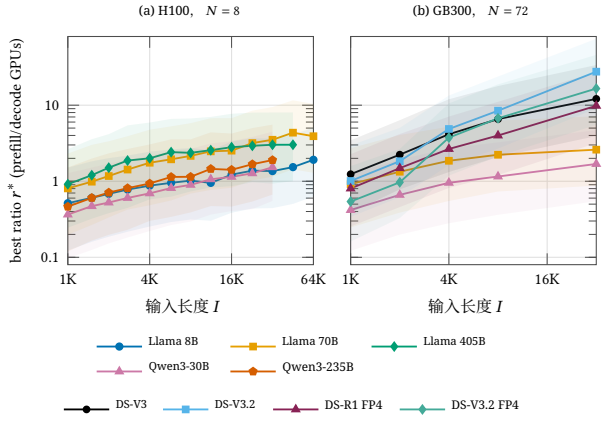

图5：输入长度对应的最佳预填充与解码 GPU 比率 r^*。线条：平均输出长度约为 700 个词元；阴影区域：从 2,048（底部）到 256（顶部）。DS：DeepSeek。

则以解析方式预测各阶段的容量，并根据吞吐量目标函数的结构推导出最优整数分配。因此，OSplit 无需平台专用的算子延迟数据库，也能实现相同的分配决策。

6.5 方法的成本

针对一个切分进行完整的资源配置搜索，在单个 CPU 核心上需要 0.39–0.82 秒。计算第 § 7 节的完整扫描（1,189 个候选项）需要在 8 个 CPU 核心上运行 112 秒。混合专家模型的路由表只需计算一次。

7 模型告诉用户什么

现在，我们将解析模型应用于上述 5× 大语言模型，考察输入序列长度从 1,024 到 65,536、输出序列长度为 256 到 2,048 词元的情况，以及六种配备 $N = 72$ 的 GB300 模型。我们将考察这些设置下最优资源配置如何变化，解释驱动变化的因素，并讨论其部署影响。

切分方式随工作负载而变化。 对于固定的模型和输出长度，预填充与解码 GPU 的最佳比率会随着输入长度从 1,024 个输入词元增加到所考察的最长输入长度而在 H100 上增长 3.3–4 倍（图 9×（图 5），而在输入长度固定时，该比率会随着输出词元从 256 增加到 2,048 而下降 8.0–13 倍。2× 这是由公式（18） $r^* = P/(\bar{O}D)$所致。请求的预填充成本会随输入长度线性增长（来自通用矩阵乘法），并通过注意力二次增长 $P \approx p_1 I + p_2 I^2$。每个词元的解码成本包含一个固定部分 d_0，主要用于读取权重；另一个部分则随缓存注意力读取量增长 $D \approx d_0 + d_1 S$。这个闭式表达式仅用于解释趋势，而非进行预测：将 p_1，p_2，d_0，d_1拟合到模型自身的 P和 D后，它对每个模型和工作负载的预测误差都在 8.2% 以内。因此，这一趋势源自两种成本结构，而非任何单个模型的具体细节。对于所有模型，若为所有工作负载统一选定一个比率，则在最不利的工作负载下，吞吐量会损失 41–43%。图 5b 展示了 OSplit 对 GB300 的预测。

解码读取的内容。对于每种模型，解码步骤中用于注意力的时间占比都会随输入长度增加而上升（图6）：Llama 8B 从 66% 升至 90%，Llama 70B 从 33% 升至 65%，Llama 405B 则从 7% 升至 17%。Llama 405B 每个

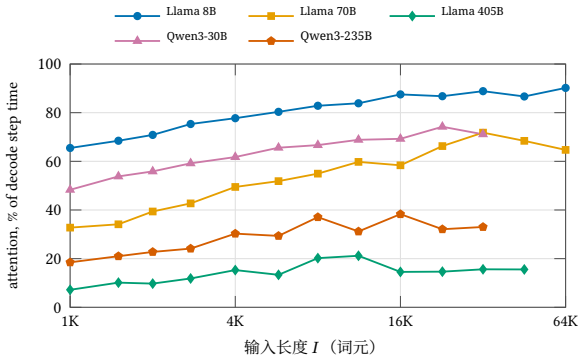

图 6：H100 上注意力所占解码步骤时间的比例与输入长度的关系。

缓存词元读取的权重远多于其他模型，因此即使输入长度最长，其解码步骤的大部分时间仍用于读取权重。当权重读取占主导时，增大批次可将权重读取开销分摊到更多请求上，从而降低 D。当 KV 缓存成为流量的主要来源时，每个请求都会增加自身的缓存读取，从而限制增大批次带来的收益； D 也会随上下文长度增加。

8 相关工作

解耦式大语言模型服务。 DistServe [40], 、Splitwise [24],、Tetri- Infer [8], 和 Mooncake [25] 展示了解耦预填充与解码的优势。后续工作探索了跨这些阶段的资源分配与适应。DOPD [13] 动态调整预填充–解码比率，而 SplitScaling [34] 则采用强化学习进行工作负载感知的扩缩容。Evolving In- fra [9] 针对智能体工作负载调整计算资源分配和 KV缓存管理。

性能建模与配置优化。Vidur [1]结合算子剖析与仿真来预测大语言模型服务性能。LLMCompass [39] 对不同硬件配置下的推理性能进行建模，而 TileSight [19]开发了一个针对图形处理器计算、内存和通信的数据块级解析模型，其中部分模型也在本文中采用。AIConfigura- tor [35] 将算子解析模型与校准后的性能数据库相结合，以优化服务配置。对于解耦推理，SARA [15]对预填充、KV缓存传输和解码排队进行建模，以优化 SLO约束的有效吞吐量；Yan等人 [36] 则从解析角度对服务容量和功耗进行建模，以实现资源配置。OSplit 基于硬件级性能建模来估算预填充和解码容量，同时考虑混合专家的专家执行以及服务框架约束。随后，它利用两阶段吞吐量目标的结构，在解析模型下联合选择并行配置，并推导出精确的整数 GPU 分配。

9 结论

我们提出了 OSplit，这是一种快速高效的技术，用于计算预填充与解码工作进程之间的解耦比率。OSplit 将解析容量模型与搜索算法相结合，以计算解耦拆分，并在 H100 和 GB300 GPU 上进行了测试

涵盖稠密模型和混合专家模型。在所有测量案例中，OSplit 都能在无需对任何部署进行性能分析的情况下找出最佳拆分，表明解耦推理的资源配置可以基于基本原理推导得出。

参考文献

[1] Amey Agrawal、Nitin Kedia、Jayashree Mohan、Ashish Panwar、Nipun Kwatra、Bhargav Gulavani、Ramachandran Ramjee 和 Alexey Tumanov。2024年。Vidur：用于大语言模型推理的大规模仿真框架。发表于 机器学习与系统会议（*MLSys*）。[2] Chaokun Chang、Yukun Zhou、Kaihua Fu、Dakai An、Tianyu Feng、Hanfeng Lu、Sheng Yao、Pu Guo、Yinghao Yu、Yizhou Shan、Bo Li、Binhang Yuan 和 Wei Wang。2026年。从大语言模型推理到智能体工作负载：特征分析及其对服务系统的启示。*arXiv* 预印本 *arXiv:2608.15127*（2026年）。[3] DeepSeek-AI、Aixin Liu、Bei Feng、Bin Wang、Bingxuan Wang、BoLiu、Chenggang Zhao、Chengqi Deng、Chong Ruan、Damai Dai、Daya Guo、Dejian Yang、Deli Chen、Dongjie Ji、Erhang Li、Fangyun Lin、Fuli Luo、Guangbo Hao、Guanting Chen、Guowei Li、H. Zhang、Hanwei Xu、Hao Yang、Haowei Zhang、Honghui Ding、Huajian Xin、Huazuo Gao、Hui Li、Hui Qu、J. L. Cai、Jian Liang、Jianzhong Guo、Jiaqi Ni、Jiashi Li、Jin Chen、Jingyang Yuan、Junjie Qiu、Junxiao Song、Kai Dong、Kaige Gao、Kang Guan、Lean Wang、Lecong Zhang、Lei Xu、Leyi Xia、Liang Zhao、Liyue Zhang、Meng Li、Miaojun Wang、Mingchuan Zhang、Minghua Zhang、Minghui Tang、Mingming Li、Ning Tian、Panpan Huang、Peiyi Wang、Peng Zhang、Qihao Zhu、Qinyu Chen、Qiushi Du、R. J. Chen、R. L. Jin、Ruiqi Ge、Ruizhe Pan、Runxin Xu、Ruyi Chen、S. S. Li、Shanghao Lu、Shangyan Zhou、Shanhuang Chen、Shaoqing Wu、Shengfeng Ye、Shirong Ma、Shiyu Wang、Shuang Zhou、Shuiping Yu、Shunfeng Zhou、Size Zheng、T. Wang、Tian Pei、Tian Yuan、Tianyu Sun、W. L. Xiao、Wangding Zeng、Wei An、Wen Liu、Wenfeng Liang、Wenjun Gao、Wentao Zhang、X. Q. Li、Xiangyue Jin、Xianzu Wang、Xiao Bi、Xiaodong Liu、Xiaohan Wang、Xiaojin Shen、Xiaokang Chen、Xiaosha Chen、Xiaotao Nie、Xiaowen Sun、Xiaoxiang Wang、Xin Liu、Xin Xie、Xingkai Yu、Xinnan Song、Xinyi Zhou、Xinyu Yang、Xuan Lu、Xuecheng Su、Y. Wu、Y. K. Li、Y. X. Wei、Y. X. Zhu、Yanhong Xu、Yanping Huang、Yao Li、Yao Zhao、Yaofeng Sun、Yaohui Li、Yaohui Wang、Yi Zheng、Yichao Zhang、Yiliang Xiong、Yilong Zhao、Ying He、Ying Tang、Yishi Piao、Yixin Dong、Yixuan Tan、Yiyuan Liu、Yongji Wang、Yongqiang Guo、Yuchen Zhu、Yuduan Wang、Yuheng Zou、Yukun Zha、Yunxian Ma、Yuting Yan、Yuxiang You、Yuxuan Liu、Z. Z. Ren、Zehui Ren、Zhangli Sha、Zhe Fu、Zhen Huang、Zhen Zhang、Zhenda Xie、Zhewen Hao、Zhihong Shao、Zhiniu Wen、Zhipeng Xu、Zhongyu Zhang、Zhuoshu Li、Zihan Wang、Zihui Gu、Zilin Li 和 Ziwei Xie。2024年。DeepSeek-V2：一种强大、经济且高效的混合专家语言模型。arXiv：2405。04434 [cs.CL]

[4] DeepSeek-AI、Aixin Liu、Bei Feng、Bing Xue、Bingxuan Wang、Bochao Wu、Chengda Lu、Chenggang Zhao、Chengqi Deng、Chenyu Zhang、Chong Ruan、Damai Dai、Daya Guo、Dejian Yang、Deli Chen、Dongjie Ji、Erhang Li、Fangyun Lin、Fucong Dai、Fuli Luo、Guangbo Hao、Guanting Chen、Guowei Li、H. Zhang、Han Bao、Hanwei Xu、Haocheng Wang、Haowei Zhang、Honghui Ding、Huajian Xin、Huazuo Gao、Hui Li、Hui Qu、J. L. Cai、Jian Liang、Jianzhong Guo、Jiaqi Ni、Jiashi Li、Jiawei Wang、Jin Chen、Jingchang Chen、Jingyang Yuan、Junjie Qiu、Junlong Li、Junxiao Song、Kai Dong、Kai Hu、Kaige Gao、Kang Guan、Kexin Huang、Kuai Yu、Lean Wang、Lecong Zhang、Lei Xu、LeyiXia、Liang Zhao、Litong Wang、Liyue Zhang、Meng Li、Miaojun Wang、Mingchuan Zhang、Minghua Zhang、Minghui Tang、Mingming Li、Ning Tian、Panpan Huang、Peiyi Wang、Peng Zhang、Qiancheng Wang、Qihao Zhu、Qinyu Chen、Qiushi Du、R. J. Chen、R. L. Jin、Ruiqi Ge、Ruisong Zhang、Ruizhe Pan、Runji Wang、Runxin Xu、Ruoyu Zhang、Ruyi Chen、S. S. Li、Shanghao Lu、Shangyan Zhou、Shanhuang Chen、Shaoqing Wu、Shengfeng Ye、Shengfeng Ye、Shirong Ma、Shiyu Wang、Shuang Zhou、Shuiping Yu、Shunfeng Zhou、Shuting Pan、T. Wang、Tao Yun、Tian Pei、Tianyu Sun、W. L. Xiao、Wangding Zeng、Wanjia Zhao、Wei An、Wen Liu、Wen- feng Liang、Wenjun Gao、Wenqin Yu、Wentao Zhang、X. Q. Li、Xiangyue Jin、Xianzu Wang、Xiao Bi、Xiaodong Liu、Xiaohan Wang、Xiaojin Shen、Xiaokang Chen、Xiaokang Zhang、Xiaosha Chen、Xiaotao Nie、Xiaowen Sun、Xiaoxiang Wang、Xin Cheng、Xin Liu、Xin Xie、Xingchao Liu、Xingkai Yu、Xinnan Song、Xinxia Shan、Xinyi Zhou、Xinyu Yang、Xinyuan Li、Xuecheng Su、Xuheng Lin、Y. K. Li、Y. Q. Wang、Y. X. Wei、Y. X. Zhu、Yang Zhang、Yanhong Xu、Yanhong Xu、Yanping Huang、Yao Li、Yao Zhao、Yaofeng Sun、Yaohui Li、Yaohui Wang、Yi Yu、Yi Zheng、Yichao Zhang、Yifan Shi、Yiliang Xiong、Ying He、Ying Tang、Yishi Piao、Yisong Wang、Yixuan Tan、Yiyang Ma、Yiyuan Liu、Yongqiang Guo、Yu Wu、Yuan Ou、Yuchen Zhu、Yuduan Wang、Yue Gong、Yuheng Zou、Yujia He、Yukun Zha、Yunfan Xiong、Yunxian Ma、Yuting Yan、Yuxiang Luo、Yuxiang You、Yuxuan Liu、Yuyang Zhou、Z. F. Wu、Z. Z. Ren、Zehui Ren、Zhangli Sha、Zhe Fu、Zhean Xu、Zhen Huang、Zhen Zhang、Zhenda Xie、Zhengyan Zhang、Zhewen Hao、Zhibin Gou、Zhicheng Ma、Zhigang Yan、Zhihong Shao、Zhipeng Xu、Zhiyu Wu、Zhongyu Zhang、Zhuoshu Li、Zihui Gu、Zijia Zhu、Zijun Liu、Zilin Li、Ziwei Xie、Ziyang Song、Ziyi Gao，以及 Zizheng Pan。2024 年。DeepSeek-V3 技术报告。arXiv:2412.19437 [cs.CL]

[5] DeepSeek-AI、Aixin Liu、Aoxue Mei、Bangcai Lin、Bing Xue、Bingxuan Wang、Bingzheng Xu、Bochao Wu、Bowei Zhang、Chaofan Lin、Chen Dong、Chengda Lu、Chenggang Zhao、Chengqi Deng、Chenhao Xu、Chong Ruan、Damai Dai、Daya Guo、Dejian Yang、Deli Chen、Erhang Li、Fangqi Zhou、Fangyun Lin、Fucong Dai、Guangbo Hao、Guanting Chen、Guowei Li、H. Zhang、Hanwei Xu、Hao Li、Haofen Liang、Haoran Wei、Haowei Zhang、Haowen Luo、Haozhe Ji、Honghui Ding、Hongxuan Tang、Huanqi Cao、Huazuo Gao、Hui Qu、Hui Zeng、Jialiang Huang、Jiashi Li、Jiaxin Xu、Jiewen Hu、Jingchang Chen、Jingting Xiang、Jingyang Yuan、Jingyuan Cheng、Jinhua Zhu、Jun Ran、Junguang Jiang、Junjie Qiu、Junlong Li、Junxiao Song、Kai Dong、Kaige Gao、Kang Guan、Kexin Huang、Kexing Zhou、Kezhao Huang、Kuai Yu、Lean Wang、Lecong Zhang、Lei Wang、Liang Zhao、Liangsheng Yin、Lihua Guo、Lingxiao Luo、Linwang Ma、Litong Wang、Liyue Zhang、M. S. Di、M. Y Xu、Mingchuan Zhang、Minghua Zhang、Minghui Tang、Mingxu Zhou、Panpan Huang、Peixin Cong、Peiyi Wang、Qiancheng Wang、Qihao Zhu、Qingyang Li、Qinyu Chen、Qiushi Du、Ruiling Xu、Ruiqi Ge、Ruisong Zhang、Ruizhe Pan、Runji Wang、Runqiu Yin、Runxin Xu、Ruomeng Shen、Ruoyu Zhang、S. H. Liu、Shanghao Lu、Shangyan Zhou、Shanhuang Chen、Shaofei Cai、Shaoyuan Chen、Shengding Hu、Shengyu Liu、Shiqiang Hu、Shirong Ma、Shiyu Wang、Shuiping Yu、Shunfeng Zhou、Shuting Pan、Songyang Zhou、Tao Ni、Tao Yun、Tian Pei、Tian Ye、Tianyuan Yue、Wangding Zeng、Wen Liu、Wenfeng Liang、Wenjie Pang、Wenjing Luo、Wenjun Gao、Wentao Zhang、Xi Gao、Xiangwen Wang、Xiao Bi、Xiaodong Liu、Xiaohan Wang、Xiaokang Chen、Xiaokang Zhang、Xiaotao Nie、Xin Cheng、Xin Liu、Xin Xie、Xingchao Liu、Xingkai Yu、Xingyou Li、Xinyu Yang、Xinyuan Li、Xu Chen、Xuecheng Su、Xuehai Pan、Xuheng Lin、Xuwei Fu、Y. Q. Wang、Yang Zhang、Yanhong Xu、Yanru Ma、Yao Li、Yao Li、Yao Zhao、Yaofeng Sun、Yaohui Wang、Yi Qian、Yi Yu、Yichao Zhang、Yifan Ding、Yifan Shi、Yiliang Xiong、Ying He、Ying Zhou、Yinmin Zhong、Yishi Piao、Yisong Wang、Yixiao Chen、Yixuan Tan、Yixuan Wei、Yiyang Ma、Yiyuan Liu、Yonglun Yang、Yongqiang Guo、Yongtong Wu、Yu Wu、Yuan Cheng、Yuan Ou、Yuanfan Xu、Yuduan Wang、Yue Gong、Yuhan Wu、Yuheng Zou、Yukun Li、Yunfan Xiong、Yuxiang Luo、Yuxiang You、Yuxuan Liu、Yuyang Zhou、Z. F. Wu、Z. Z. Ren、Zehua Zhao、Zehui Ren、Zhangli Sha、Zhe Fu、Zhean Xu、Zhenda Xie、Zhengyan Zhang、Zhewen Hao、Zhibin Gou、Zhicheng Ma、Zhigang Yan、Zhihong Shao、Zhixian Huang、Zhiyu Wu、Zhuoshu Li、Zhuping Zhang、Zian Xu、Zihao Wang、Zihui Gu、Zijia Zhu、Zilin Li、Zipeng Zhang、Ziwei Xie、Ziyi Gao、Zizheng Pan、Zongqing Yao、Bei Feng、Hui Li、J. L. Cai、Jiaqi Ni、Lei Xu、Meng Li、Ning Tian、R. J. Chen、R. L. Jin、S. S. Li、Shuang Zhou、Tianyu Sun、X. Q. Li、Xiangyue Jin、Xiaojin Shen、Xiaosha Chen、Xinnan Song、Xinyi Zhou、Y. X. Zhu、Yanping Huang、Yaohui Li、Yi Zheng、Yuchen Zhu、Yunxian Ma、Zhen Huang、Zhipeng Xu、Zhongyu Zhang、Dongjie Ji、Jian Liang、Jianzhong Guo、Jin Chen、Leyi Xia、Miaojun Wang、Mingming Li、Peng Zhang、Ruyi Chen、Shangmian Sun、Shaoqing Wu、Shengfeng Ye、T. Wang、W. L. Xiao、Wei An、Xianzu Wang、Xiaowen Sun、Xiaoxiang Wang、Ying Tang、Yukun Zha、Zekai Zhang、Zhe Ju、Zhen Zhang，以及 Zihua Qu。2025年。DeepSeek-V3.2：推动开放式大语言模型的前沿发展。arXiv:2512.02556 [cs.CL]

[6] Robert G . Gallager . 1996. 离散随机过程 . Kluwer Academic 出版社 - ，波士顿，马萨诸塞州。doi:10.1007/978-1-4615-2329-1

[7] Aaron Grattafiori、Abhimanyu Dubey、Abhinav Jauhri、Abhinav Pandey、Abhishek Kadian、Ahmad Al-Dahle、Aiesha Letman、Akhil Mathur、Alan Schelten、Alex Vaughan、Amy Yang、Angela Fan、Anirudh Goyal、Anthony Hartshorn、Aobo Yang、Archi Mitra、Archie Sravankumar、Artem Korenev、Arthur Hinsvark、Arun Rao、Aston Zhang、Aurelien Rodriguez、Austen Gregerson、Ava Spataru、Baptiste Roziere、Bethany Biron、Binh Tang、Bobbie Chern、Charlotte Caucheteux、Chaya Nayak、Chloe Bi、Chris Marra、Chris McConnell、Christian Keller、Christophe Touret、Chunyang Wu、Corinne Wong、Cristian Canton Ferrer、Cyrus Nikolaidis、Damien Allonsius、Daniel Song、Danielle Pintz、Danny Livshits、Danny Wyatt、David Esiobu、Dhruv Choudhary、Dhruv Mahajan、Diego Garcia-Olano、Diego Perino、Dieuwke Hupkes、Egor Lakomkin、Ehab AlBadawy、Elina Lobanova、Emily Dinan、Eric Michael Smith、Filip Radenovic、Francisco Guzmán、Frank Zhang、Gabriel Synnaeve、Gabrielle Lee、Georgia Lewis Anderson、Govind Thattai、Graeme Nail、Gregoire Mialon、Guan Pang、Guillem Cucurell、Hailey Nguyen、Hannah Korevaar、Hu Xu、Hugo Touvron、Iliyan Zarov、Imanol Arrieta Ibarra、Isabel Kloumann、Ishan Misra、Ivan Evtimov、Jack Zhang、Jade Copet、Jaewon Lee、Jan Geffert、Jana Vranes、Jason Park、Jay Mahadeokar、Jeet Shah、Jelmer van der Linde、Jennifer Billock、Jenny Hong、Jenya Lee、Jeremy Fu、Jianfeng Chi、Jianyu Huang、Jiawen Liu、Jie Wang、JiecaoYu、Joanna Bitton、Joe Spisak、Jongsoo Park、Joseph Rocca、Joshua Johnstun、Joshua Saxe、Junteng Jia、Kalyan Vasuden Alwala、Karthik Prasad、Kartikeya Upasani、Kate Plawiak、Ke Li、Kenneth Heafield、Kevin Stone、Khalid El-Arini、Krithika Iyer、Kshitiz Malik、Kuenley Chiu、Kunal Bhalla、Kushal Lakhotia、Lauren Rantala-Yeary、Laurens van der Maaten、Lawrence Chen、Liang Tan、Liz Jenkins、Louis Martin、Lovish Madaan、Lubo Malo、Lukas Blecher、Lukas Landzaat、Luke de Oliveira、Madeline Muzzi、Mahesh Pasupuleti、Mannat Singh、Manohar Paluri、MarcinKardas、Maria Tsimpoukelli、Mathew Oldham、Mathieu Rita、Maya Pavlova、Melanie Kambadur、Mike Lewis、Min Si、Mitesh Kumar Singh、Mona Hassan、Naman Goyal、Narjes Torabi、Nikolay Bashlykov、Nikolay Bogoychev、Niladri Chatterji、Ning Zhang、Olivier Duchenne、Onur Çelebi、Patrick Alrassy、Pengchuan Zhang、Pengwei Li、Petar Vasic、Peter Weng、Prajjwal Bhargava、Pratik Dubal、Praveen Krishnan、Punit Singh -

Koura、Puxin Xu、Qing He、Qingxiao Dong、Ragavan Srinivasan、Raj Ganapathy、Ramon Calderer、Ricardo Silveira Cabral、Robert Stojnic、Roberta Raileanu、Ro- han Maheswari、Rohit Girdhar、Rohit Patel、Romain Sauvestre、Ronnie Polidoro、Roshan Sumbaly、Ross Taylor、Ruan Silva、Rui Hou、Rui Wang、Saghar Hosseini、Sahana Chennabasappa、Sanjay Singh、Sean Bell、Seohyun Sonia Kim、Sergey Edunov、Shaoliang Nie、Sharan Narang、Sharath Raparthy、Sheng Shen、Shengye Wan、Shruti Bhosale、Shun Zhang、Simon Vandenhende、Soumya Batra、Spencer Whitman、Sten Sootla、Stephane Collot、Suchin Gururangan、Sydney Borodin- sky、Tamar Herman、Tara Fowler、Tarek Sheasha、Thomas Georgiou、Thomas Scialom、Tobias Speckbacher、Todor Mihaylov、Tong Xiao、Ujjwal Karn、Vedanuj Goswami、Vibhor Gupta、Vignesh Ramanathan、Viktor Kerkez、Vincent Gonguet、Virginie Do、Vish Vogeti、Vítor Albiero、Vladan Petrovic、Weiwei Chu、Wenhan Xiong、Wenyin Fu、Whitney Meers、Xavier Martinet、Xiaodong Wang、Xiaofang Wang、Xiaoqing Ellen Tan、Xide Xia、Xinfeng Xie、Xuchao Jia、Xuewei Wang、Yaelle Goldschlag、Yashesh Gaur、Yasmine Babaei、Yi Wen、Yiwen Song、Yuchen Zhang、Yue Li、Yuning Mao、Zacharie Delpierre Coudert、Zheng Yan、Zhengx- ing Chen、Zoe Papakipos、Aaditya Singh、Aayushi Srivastava、Abha Jain、Adam Kelsey、Adam Shajnfeld、Adithya Gangidi、Adolfo Victoria、Ahuva Goldstand、Ajay Menon、Ajay Sharma、Alex Boesenberg、Alexei Baevski、Allie Feinstein、Amanda Kallet、Amit Sangani、Amos Teo、Anam Yunus、Andrei Lupu、Andres Al- varado、Andrew Caples、Andrew Gu、AndrewHo、Andrew Poulton、Andrew Ryan、Ankit Ramchandani、Annie Dong、Annie Franco、Anuj Goyal、Aparajita Saraf、Arkabandhu Chowdhury、Ashley Gabriel、Ashwin Bharambe、Assaf Eisenman、Azadeh Yazdan、Beau James、Ben Maurer、Benjamin Leonhardi、Bernie Huang、Beth Loyd、Beto De Paola、Bhargavi Paranjape、Bing Liu、Bo Wu、Boyu Ni、Braden Hancock、Bram Wasti、Brandon Spence、Brani Stojkovic、Brian Gamido、Britt Montalvo、Carl Parker、Carly Burton、Catalina Mejia、Ce Liu、Changhan Wang、Changkyu Kim、Chao Zhou、Chester Hu、Ching-Hsiang Chu、Chris Cai、Chris Tindal、Christoph Feichtenhofer、Cynthia Gao、Damon Civin、Dana Beaty、Daniel Kreymer、Daniel Li、David Adkins、David Xu、Davide Testuggine、Delia David、Devi Parikh、Diana Liskovich、Didem Foss、Dingkang Wang、Duc Le、Dustin Holland、Edward Dowling、Eissa Jamil、Elaine Montgomery、Eleonora Presani、Emily Hahn、Emily Wood、Eric-Tuan Le、Erik Brinkman、Esteban Arcaute、Evan Dunbar、Evan Smothers、Fei Sun、Felix Kreuk、Feng Tian、Filippos Kokkinos、Firat Ozgenel、Francesco Caggioni、Frank Kanayet、Frank Seide、Gabriela Medina Flo- rez、Gabriella Schwarz、Gada Badeer、Georgia Swee、Gil Halpern、Grant Herman、Grigory Sizov、Guangyi (Jack) Zhang、Guna Lakshminarayanan、Hakan Inan、Hamid Shojanazeri、Han Zou、Hannah Wang、Hanwen Zha、Haroun Habeeb、Harrison Rudolph、Helen Suk、Henry Aspegren、Hunter Goldman、Hongyuan Zhan、Ibrahim Damlaj、Igor Molybog、Igor Tufanov、Ilias Leontiadis、Irina-Elena Veliche、Itai Gat、Jake Weissman、James Geboski、James Kohli、Janice Lam、Japhet Asher、Jean-Baptiste Gaya、Jeff Marcus、J eff Tang、Jennifer Chan、Jenny Zhen、Jeremy Reizenstein、Jeremy Teboul、Jessica Zhong、Jian Jin、Jingyi Yang、Joe Cummings、Jon Carvill、Jon Shepard、Jonathan McPhie、Jonathan Torres、Josh Ginsburg、Junjie Wang、Kai Wu、Kam Hou U、Karan Saxena、Kartikay Khandel- wal、Katayoun Zand、Kathy Matosich、Kaushik Veeraraghavan、Kelly Michelena、Keqian Li、Kiran Jagadeesh、Kun Huang、Kunal Chawla、Kyle Huang、Lailin Chen、Lakshya Garg、Lavender A、Leandro Silva、Lee Bell、Lei Zhang、Liangpeng Guo、Licheng Yu、Liron Moshkovich、Luca Wehrstedt、Madian Khabsa、Manav Avalani、Manish Bhatt、Martynas Mankus、Matan Hasson、Matthew Lennie、Matthias Reso、Maxim Groshev、Maxim Naumov、Maya Lathi、Meghan Keneally、Miao Liu、Michael L. Seltzer、Michal Valko、Michelle Restrepo、Mihir Patel、Mik Vyatskov、Mikayel Samvelyan、Mike Clark、Mike Macey、Mike Wang、Miquel Jubert Her- moso、Mo Metanat、Mohammad Rastegari、Munish Bansal、Nandhini Santhanam、Natascha Parks、Natasha White、Navyata Bawa、Nayan Singhal、Nick Egebo、Nicolas Usunier、Nikhil Mehta、Nikolay Pavlovich Laptev、Ning Dong、Norman Cheng、Oleg Chernoguz、Olivia Hart、Omkar Salpekar、Ozlem Kalinli、Parkin Kent、Parth Parekh、Paul Saab、Pavan Balaji、Pedro Rittner、Philip Bontrager、Pierre Roux、Piotr Dollar、Polina Zvyagina、Prashant Ratanchandani、Pritish Yu- vraj、Qian Liang、Rachad Alao、Rachel Rodriguez、Rafi Ayub、Raghotham Murthy、Raghu Nayani、Rahul Mitra、Rangaprabhu Parthasarathy、Raymond Li、Rebekkah Hogan、Robin Battey、Rocky Wang、Russ Howes、Ruty Rinott、Sachin Mehta、Sachin Siby、Sai Jayesh Bondu、Samyak Datta、Sara Chugh、Sara Hunt、Sargun Dhillon、Sasha Sidorov、Satadru Pan、Saurabh Mahajan、Saurabh Verma、Seiji Yamamoto、Sharadh Ramaswamy、Shaun Lindsay、Shaun Lindsay、Sheng Feng、Shenghao Lin、Shengxin Cindy Zha、Shishir Patil、Shiva Shankar、Shuqiang Zhang、Shuqiang Zhang、Sinong Wang、Sneha Agarwal、Soji Sajuyigbe、Soumith Chintala、Stephanie Max、Stephen Chen、Steve Kehoe、Steve Satterfield、Sudar- shan Govindaprasad、Sumit Gupta、Summer Deng、Sungmin Cho、Sunny Virk、Suraj Subramanian、Sy Choudhury、Sydney Goldman、Tal Remez、Tamar Glaser、Tamara Best、Thilo Koehler、Thomas Robinson、Tianhe Li、Tianjun Zhang、Tim Matthews、Timothy Chou、Tzook Shaked、Varun Vontimitta、Victoria Ajayi、Vic- toria Montanez、Vijai Mohan、Vinay Satish Kumar、Vishal Mangla、Vlad Ionescu、Vlad Poenaru、Vlad Tiberiu Mihailescu、Vladimir Ivanov、Wei Li、Wenchen Wang、Wenwen Jiang、Wes Bouaziz、Will Constable、Xiaocheng Tang、Xiaojian Wu、Xiaolan Wang、Xilun Wu、Xinbo Gao、Yaniv Kleinman、Yanjun Chen、Ye Hu、Ye Jia、Ye Qi、Yenda Li、Yilin Zhang、Ying Zhang、Yossi Adi、Youngjin Nam、Yu (Sid) Wang、Yu Zhao、Yuchen Hao、Yundi Qian、Yunlu Li、Yuzi He、Zach Rait、Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024年. Llama 3 模型家族. arXiv:2407.21783 [cs.AI][8] Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024年. 无干扰推理：面向混合下游工作负载的 LLM 推理解耦. arXiv:2401.11181 [cs.DC][9] Hao Kang, Weili Xu, Ziyang Li, Zheng Du, Hanchen Li, Tian Xia, Song Bian, Tushar Krishna, Huizi Mao, Ion Stoica, and Song Han. 2026年. 演进基础设施：大规模智能体自适应 LLM 服务. arXiv:2611.12304 [cs.DC] doi:10.48550/ arXiv.2611.12304[10] Kimi 团队, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun Du. 2025年. Kimi Linear：一种表现力强且高效的注意力架构. arXiv:2510.26692 [cs.CL][11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023年. 使用 PagedAttention 高效管理大语言模型服务的内存. In Proceedings of the 29th Symposium on Operating Systems Principles (操作系统原理研讨会). 611–626.[12] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. GShard：通过条件计算和自动分片扩展巨型模型. arXiv:2006.16668 [cs.CL] https://arxiv.org/abs/2006.16668[13] Junhan Liao, Minxian Xu, Wanyi Zheng, Yan Wang, Kejiang Ye, Rajkumar Buyya, and Chengzhong Xu. 2026年. DOPD：一种用于最大化 LLM 推理服务有效吞吐量的动态 PD 解耦架构. IEEE 服务计算汇刊 19, 2 (2026年), 1134–1147. arXiv:2511.20982 doi:10.1109/TSC.2026. 3670011[14] J ohn D. C. Little. 1961. 排队公式的一个证明： $L = \lambda W$《运筹学》 9, 3 1961 383 387 10 1287 9 3 383 ( ), – . doi: . /opre. . . Shicong Liu,[15] Xianghao Yu, Zhen Gao, and Jun Zhang. 2026年. SARA：面向解耦式智能体 LLM 服务的感知 SLO 的资源分配. arXiv: 2609 26763 [ ] . 数据结构与算法 Meta. . [16] 2024 Llama-3.1-405B-Instruct-FP8. https://huggingface. co/meta-llama/Llama-3.1-405B-Instruct-FP8. Revision 64a54b704768dfd589a3e4ac05d546052f67f4fd.[17] Paulius Micikevicius, Dusan St osic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. 2022. 面向深度学习的 FP8 格式. arXiv:2209.05433 [cs.LG][18] Tiyasa Mitra, Ritika Borkar, Nidhi Bhatia, Shivam Raj, Hongkuan Zhou, Yan Ru Pei, Vishwanath Venkatesan, Kyle Kranen, Ramon Matas, Dheevatsa Mudigere, Ritchie Zhao, Maximilian Golub, Arpan Dutta, Suresh Nambi, Sailaja Madduri, Dharmesh Jani, Brian Pharris, Itay Neeman, and Bita Darvish Rouhani. 2026年. 不止于炒作：对推理解耦的务实看法. In Proceedings of Machine Learning and Systems (MLSys), A. Chowdhery and Z. Jia (编), Vol. 8. MLSys, 2001–2015. https://proceedings.mlsys.org/paper_files/paper/2026/hash/d49cee5f3a79d97d719df255689d83d7-Abstract-Conference.html[19] Zhiwen Mo, Yu Cheng, Lei Wang, Zhengju Tang, Lei Xu, Guoyu Li, Yuqi Dong, Lingxiao Ma, Yuqing Xia, Jilong Xue, Fan Yang, Luo Mai, Zhi Yang, Wayne Luk, and Hongxiang Fan. 2026年. TileSight：一种从核心到集群、基于第一性原理且以图块为中心的图形处理器性能分析模型 . arXiv: 2607 22432 [ ] . 数据结构与算法 Neural Magic. . Meta-Llama- . - B-Instruct-FP . https: // [20] 2024 3 1 70 8 huggingface.co/RedHatAI/Meta-Llama-3.1-70B-Instruct-FP8. Revision f8d0de9953e442c2fd4da32a1d4da8afb8c9452e.[21] 英伟达. 2026年. NVIDIA 推理传输库 (NIXL)， 1.3.1 版. https: //github.com/ai-dynamo/nixl.[22] 英伟达, Felix Abecassis, Anjulie Agrusa, Dong Ahn, Jonah Alben, Stefania Al- borghetti, Michael Andersch, Sivakumar Arayandi, Alexis Bjorlin, Aaron Blake- man, Evan Briones, Ian Buck, Bryan Catanzaro, Muya Chang, Jinhang Choi, Mike Chrzanowski, Eric Chung, Victor Cui, Steve Dai, Bita Darvish Rouhani, Carlo del Mundo, Deena Donia, Burc Eryilmaz, Henry Estela, Abhinav Goel, Oleg Goncharov, Yugi Guvvala, Robert Hesse, Russell J. Hewett, Herbert Hum, Ujval Kapasi, Brucek Khailany, Mikail Khona, Nick Knight, Alex Kondratenko, Ronny Krashinsky, Ben Lanir, Simon Layton, Michael Lightstone, Daniel Lo, Paulius Micikevicius, Asit Mishra, Tim Moon, Deepak Narayanan, Chao Ni, Abhijit Paithankar, Satish Pasumarthi, Ankit Patel, Mostofa Patwary, Ashwin Poojary, Gargi Prasad, Sweta Priyadarshi, Yigong Qin, Xiaowei Ren, Oleg Ry- bakov, Charbel Sakr, Sanjeev Satheesh, Stas Sergienko, Pasha Shamis, Kirthi Shankar, Nishant Sharma, Mohammad Shoeybi, Michael Siu, Misha Smelyanskiy, Darko Stosic, Dusan Stosic, Bor-Yiing Su, Frank Sun, Nima Tajbakhsh, Shelby

Thomas, Przemek Tredak, Evgeny Tsykunov, Gandhi Vaithilingam, Aditya Vavre, Rangharajan Venkatesan, Roger Waleffe, Qiyu Wan, Hexin Wang, Mengdi Wang, Lizzie Wei, Hao Wu, Evan Wu, Keith Wyss, Ning Xu, Jinze Xue, Charlene Yang, Yujia Zhai, Ruoxi Zhang, Jingyang Zhu, and Zhongbo Zhu. 2025年. 使用 NVFP 4 2509 25149 [ ] 预训练大语言模型. arXiv: . [23] cs.CL Muhammad Osama, Duane Merrill, Cris Cecka, Michael Garland, and John D. Owens. 2023年. Stream-K：一种以工作为中心的并行分解方法，用于在图形处理器上进行稠密矩阵乘法. arXiv:2301.03598 [数据结构与算法][24] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024年. Splitwise：通过阶段拆分实现高效的大语言模型生成式推理. In Proceedings of the 第51届年度国际计算机体系结构研讨会 (ISCA).[25] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025年. Mooncake：以更多存储换取更少计算——一种面向大语言模型聊天机器人的以 KVCache 为中心的架构. In 第23届 USENIX 文件与存储技术会议 (FAST 25). USENIX 协会, 圣克拉拉, CA, 155–170. https://www.usenix.org/ conference/fast25/presentation/qin [26] Qwen 团队. 2025年. Qwen3-235B-A22B-FP8. https://huggingface.co/Qwen/ Qwen3-235B-A22B-FP8. Revision 39eb2b067ea6b8e3e1dd97d3cd0c7ffeaf3e1a35. [27] Qwen 团队. 2025年. Qwen3-30B-A3B-FP8. https://huggingface.co/Qwen/Qwen3- 30B-A3B-FP8. Revision d206ba732169f29bb77fbf80fc2c4b81d4d30782.[28]RedHatAI. 2024年. Meta-Llama-3.1-8B-Instruct-FP8. https:// huggingface.co/RedHatAI/Meta-Llama-3.1-8B-Instruct-FP8. Revision 870a1177ed082c0ba5dbefd62573ab8f91803f6a.[29] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. 2024年. FlashAttention-3：借助异步和低精度实现快速、准确的注意力计算. In Advances in Neural Information Processing Systems (NeurIPS), Vol. 37. 68658–68685. arXiv:2407.08608 doi:10.52202/079017-2193[30] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019年. Megatron-LM：利用模型并行训练数十亿参数语言模型. arXiv: 1909 08053 [ ]. cs.CL vLLM 项目. . [31] 2026 0 26 0 vLLM，. . . 版. https://github.com/vllm-project/vllm/ releases/tag/v0.26.0. [32] vLLM 项目. 2026年. vLLM，0.32.0 版. https://github.com/vllm-project/vllm/ releases/tag/v0.32.0. [33] Samuel Williams, Andrew Waterman, and David Patterson. 2009年. 屋顶线：一种用于多核架构的直观可视化性能模型. Commun. ACM 52, 4 (2009年), 65–76. doi:10.1145/1498765.1498785

[34] Wei Xiao、Xuefeng Huang、Weijia Shi 和 Baokang Zhao。2026年。SplitScaling：通过深度强化学习实现针对流量突发的解耦式大语言模型服务自适应扩展。载于第三十五届国际人工智能联合会议论文集，*IJCAI-26*，Diego Calvanese（编）。国际人工智能联合会议组织，5101–5109。doi:10.24963/ijcai.2026/568 主会场论文。[35] Tianhao Xu、Yiming Liu、Xianglong Lu、Yijia Zhao、Xuting Zhou、Aichen Feng、Yiyi Chen、Yi Shen、Qin Zhou、Xumeng Chen、Ilya Sherstyuk、Haorui Li、Rishi Thakkar、Ben Hamm、Yuanzhe Li、Xue Huang、Wenpeng Wu、An- ish Shanbhag、Harry Kim、Chuan Chen 和 Junjie Lai。2026年。AIConfigurator：面向多框架大语言模型服务的极速配置优化。arXiv: 2601 06288 [ ] 。cs.DC Mingyuan Yan、Haiyu Wang、Linxuan Biao、[36] H. Jonathan Chao、Sai Qian Zhang 和 Wenqi Cui。2026年。面向预填充-解码解耦式人工智能推理的解析式功耗感知资源配置。arXiv：2609。24639 [cs.DC][37] An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv、Chujie Zheng、Dayiheng Liu、Fan Zhou、Fei Huang、Feng Hu、Hao Ge、Haoran Wei、Huan Lin、Jialong Tang、Jian Yang、Jianhong Tu、Jianwei Zhang、Jianxin Yang、Jiaxi Yang、Jing Zhou、Jingren Zhou、Junyang Lin、Kai Dang、Keqin Bao、Kexin Yang、Le Yu、Lianghao Deng、Mei Li、Mingfeng Xue、Mingze Li、Pei Zhang、Peng Wang、Qin Zhu、Rui Men、Ruize Gao、Shixuan Liu、Shuang Luo、Tianhao Li、Tianyi Tang、Wenbiao Yin、Xingzhang Ren、Xinyu Wang、Xinyu Zhang、Xuancheng Ren、Yang Fan、Yang Su、Yichang Zhang、Yinger Zhang、Yu Wan、Yuqiong Liu、Zekun Wang、Zeyu Cui、Zhenru Zhang、Zhipeng Zhou 和 Zihan Qiu。2025年。Qwen3 技术报告。arXiv: 2505 09388 [ ] 。cs.CL Ted Zadouri、Markus Hoehnerbach、Jay Shah、[38] Vijay Thakkar 和 Tri Dao。2026年。FlashAttention-4：面向非对称硬件扩展的算法与内核流水线协同设计。载于机器学习与系统会议论文集（*MLSys*），第8卷。912–926。arXiv：2603.05451[39] Hengrui Zhang、August Ning、Rohan Baskar Prabhakar 和 David Wentzlaff。2024年。LLMCompass：实现高效的大语言模型推理硬件设计。载于*2024*年第*51*届年度国际计算机体系结构研讨会（*ISCA*）。IEEE，1080–1096。[40] Yin min Zhong、Shengyu Liu、Junda Chen、Jianbo Hu、Yibo Zhu、Xuanzhe Liu、Xin Jin 和 Hao Zhang。2024年。DistServe：解耦预填充与解码，实现面向有效吞吐量优化的大语言模型服务。载于第*18*届 *USENIX* 操作系统设计与实现研讨会（*OSDI*）。