面向预填充-解码解耦式大语言模型推理的感知服务等级目标的计算资源分配

解耦式大语言模型推理

Luchang Li*、Dongfang Li、Bozhao Gong、
Yu Zhang 金山云

摘要
预填充-解码（P/D）解耦已成为大语言模型（LLM）推理中广泛采用的一种优化策略。然而，目前尚无成熟的方法能够在满足总吞吐量、服务等级目标（SLO）和请求特征（具体而言，即输入和输出长度）约束的前提下，确定 P/D 硬件资源的最优数量。为弥补这一空白，我们提出了一种结合理论建模与实证基准测试的混合方法。首先，我们提出一个用于计算 P/D 资源数量的理论模型，该模型以总吞吐量要求、请求的输入和输出长度，以及预填充和解码吞吐量为依据。接着，为了获得 SLO 约束下实际的预填充和解码吞吐量，我们使用 M/M/1 排队论对预填充过程进行建模，根据基准测试测得的最大预填充吞吐量和首个令牌时间（TTFT）推导出实际达到的预填充吞吐量。对于解码阶段，我们确定满足每输出 token 时间（TPOT）要求的解码批次大小，并通过实测获得相应的解码吞吐量。实验结果表明，所提出的方法能够准确预测真实大语言模型推理场景中的最优 P/D 资源分配。

引言

大语言模型（LLM）在广泛的任务中展现出了卓越的能力[9, 16, 20, 24], ，推动了推理服务需求的激增。在满足服务等级目标（SLO）的同时，以高性能部署大语言模型，对于确保用户体验并降低推理成本至关重要。传统上，大语言模型推理采用非解耦方式部署，即由同一组 GPU 资源依次处理计算受限的预填充和内存带宽受限的解码。这两个截然不同的阶段相互干扰，使得同时优化首个令牌时间（TTFT）和每输出令牌时间（TPOT）指标变得困难[12，23]。这也阻碍了对两个阶段进行独立优化，例如采用不同的硬件[2, 21], 、多样化的并行策略以及其他提升吞吐量的方法。

为缓解这些干扰问题，预填充-解码（P/D）分离已成为一种被广泛采用的优化范式[2, 5, 12, 21, 23]。通过将两个阶段解耦为独立的预填充实例和解码实例，并在它们之间传输 KV 缓存，两个阶段的服务等级目标与吞吐量均可独立优化。vLLM[7]， SGLang[22],TensorRT-LLM[25] 等现有框架已成功实现预填充/解码分离机制。然而，一个关键的运维挑战仍然存在：如何

* 通讯作者：liluchang@kingsoft.com

为给定工作负载确定硬件资源的最优供给（即预填充 GPU 与解码 GPU 的具体数量）。不当的预填充/解码资源分配要么导致资源利用不足，要么无法满足服务等级目标要求。

目前，行业尚缺乏一种成熟的方法来精确计算预填充/解码所需的资源数量。SGLang 在 P_3D_9 实例 [26]，上部署了 DeepSeek V3.1，但尚未披露具体的预填充 / 解码实例比例。NVIDIA 的 AIConfigurator[19] 采用基于搜索的方法，根据特定用户的服务等级目标要求，推导出预填充/解码的最优部署策略，例如张量并行（TP）、注意力数据并行（DP）和专家并行（EP）等参数设置。尽管如此，它仍无法根据用户的吞吐量和服务等级目标要求，为预填充/解码提供全面的资源分配方法。其他一些研究[8, 13, 17]也对预填充/解码资源进行调度，但没有充分考虑服务等级目标要求，也未充分考虑系统实际的预填充吞吐量和解码吞吐量。

为弥补这一空白，我们提出了一种新方法，将理论建模与实证基准测试相结合，以精确确定最优的预填充/解码资源分配。我们的方法并非针对单个预填充/解码实例寻求最优部署，而是在预先确定的部署方案下，根据用户指定的吞吐量和服务等级目标要求，确定分配给预填充/解码的资源数量。

本文的主要贡献包括：
- 我们建立了一个理论模型，根据总吞吐量、服务等级目标要求、请求输入/输出长度以及 P/D 阶段可实现的吞吐量，计算 P/D 实例的数量。

- 为确定满足首个令牌时间（TTFT）约束的可实现预填充吞吐量，我们使用 M/M/1 排队论对预填充过程建模。由此，我们可以推导出满足首令牌时间要求、且受基准测试所得最大预填充吞吐量限制的有效预填充吞吐量。

- 与此同时，我们通过实测确定满足每个输出令牌时间要求的最大解码批次大小，进而推导出有效解码吞吐量。

- 我们证明，该方法能够准确预测真实场景推理中的最优预填充/解码资源分配，同时确保成本效益和严格满足服务等级目标。

方法

预填充/解码硬件资源数量计算

首先，用户指定的总吞吐量是指每秒处理的输入和输出令牌总数。因此，总吞吐量 TP_{total} 的计算如下：

$$TP_{total} = \frac{N_{req} * (L_{in} + L_{out})}{T_{total}} \quad (1)$$

此处，N_{req} 表示请求数量，而 L_{in} 和 L_{out} 分别表示每个请求的平均输入和输出序列长度。T_{total} 是指完成所有请求的预填充和解码计算所需的总时间。

在预填充/解码分离系统中，请求的输入处理和输出生成分别由

预填充和解码组件负责。对于这两个阶段，预填充吞吐量 $TP_{prefill}$ 和解码吞吐量 TP_{decode} 分别定义为单个预填充或解码实例每秒处理的输入词元数和生成的输出词元数。因此，预填充阶段和解码阶段的处理时间如下：

$$T_{prefill} = \frac{N_{req} * L_{in}}{TP_{prefill} * N_{prefill}} \quad (2)$$

$$T_{decode} = \frac{N_{req} * L_{out}}{TP_{decode} * N_{decode}} \quad (3)$$

此处，$N_{prefill}$ 和 N_{decode} 分别表示预填充实例和解码实例的数量。预填充/解码分离系统采用流水线推理，因此总计算时间取预填充和解码计算时间中的较大值。为避免预填充或解码组件空闲，两个阶段的计算时间应相等，即：

$$T_{total} = T_{prefill} = T_{decode} \quad (4)$$

将公式 (2) 和公式 (3) 代入公式 (1)，可分别推导出预填充实例和解码实例的数量：

$$N_{prefill} = \frac{TP_{total} * L_{in}}{(L_{in} + L_{out}) * TP_{prefill}} \quad (5)$$

$$N_{decode} = \frac{TP_{total} * L_{out}}{(L_{in} + L_{out}) * TP_{decode}} \quad (6)$$

将公式 (5) 除以公式 (6)，即可在无需总吞吐量的情况下得出预填充/解码比：

$$R_{P/D} = \frac{L_{in} * TP_{decode}}{L_{out} * TP_{prefill}} \quad (7)$$

在这些公式中，总吞吐量、输入长度和输出长度均为用户指定的已知要求。要确定 P/D 实例的数量和比例，首先需要获得单个预填充实例和解码实例的吞吐量。以下章节将介绍如何根据具体的服务等级目标要求以及给定的请求输入长度和输出长度，计算相应的预填充和解码吞吐量。

在首令牌时间约束下获取预填充吞吐量

本章介绍如何获得满足目标首个令牌时间的实际预填充吞吐量与经验测得的最大预填充吞吐量之间的关系。首先，对于预填充阶段，在给定特定硬件和部署参数（例如张量并行、专家并行等并行度，以及分块预填充大小[1]）和请求输入长度的情况下，我们可以——通过充分利用预填充 GPU 且不留空闲时间来测量最大预填充吞吐量 $TP_{prefill}$。然而，不同的首个令牌时间要求会导致不同的预填充计算利用率，从而产生不同的有效预填充吞吐量。

在预填充/解码分离部署中，首个令牌时间不仅包括预填充阶段的请求排队时间 $T_{prefill_queuing}$ 和计算时间 $T_{prefill_computation}$，还包括一定的开销时间 $T_{overhead}$。该 $T_{overhead}$ 由用户终端与GPU服务器之间请求的输入和输出传输时间，以及P实例与D实例之间的KV 缓存传输时间组成。$T_{overhead}$ 通常可以预先确定。因此，首个令牌时间的计算公式为：

$$\text{首个令牌时间} = T_{\text{prefill_queuing}} + T_{\text{prefill_computation}} + T_{\text{overhead}} \tag{8}$$

其次，我们将单个预填充实例的请求排队与计算过程建模为一个M/M/1排队问题[3, 6]。需要注意的是，当预填充部署中启用数据并行[27] 时，每个数据并行组会独立处理请求；因此，M/M/1模型应分别应用于每个数据并行组。为简便起见，本文仅讨论禁用数据并行的场景。

根据 M/M/1 排队模型，服务率 μ 定义为最大预填充吞吐量与输入序列长度之比：

$$\mu = \frac{\widehat{TP_{\text{prefill}}}}{L_{\text{in}}} \tag{9}$$

到达率 λ 定义为实际请求率，必须小于 μ。根据排队理论，系统利用率 ρ 计算如下：

$$\rho = \frac{\lambda}{\mu} \tag{10}$$

然后，我们根据系统利用率定义实际达到的预填充吞吐量：

$$TP_{\text{prefill}} = \widehat{TP_{\text{prefill}}} * \rho \tag{11}$$

根据 M/M/1 排队模型，预填充排队时间与计算时间之和的公式为：

$$T_s = T_{\text{prefill_queuing}} + T_{\text{prefill_computation}} = TTFT - T_{\text{overhead}} = \frac{1}{\mu - \lambda} \tag{12}$$

最后，基于以上推导，我们可以根据目标首个令牌时间和最大预填充吞吐量推导出实际吞吐量，如下所示：

$$TP_{\text{prefill}} = \widehat{TP_{\text{prefill}}} - \frac{L_{\text{in}}}{TTFT - T_{\text{overhead}}} \tag{13}$$

从该公式可以得出两个关键结论：第一，目标首个令牌时间越短，实际可实现的预填充吞吐量就越低；第二，在目标首个令牌时间相同的情况下，最大预填充吞吐量越高，可实现的系统利用率就越高。

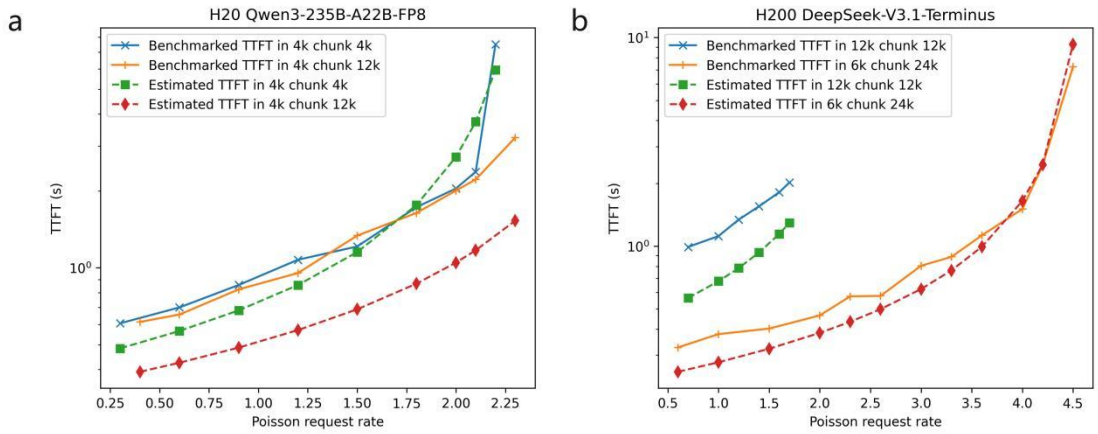

图 1 基准测试测得的首个令牌时间与 M/M/1 排队论预测的首个令牌时间的对比。请注意，本图中预测的首个令牌时间不包括从预填充到解码的KV缓存传输时间。在本图中，in 表示输入序列长度，chunk 表示分块

预填充大小。

我们将实际测得的首个令牌时间与公式（12）预测的值进行比较，以验证这一理论的正确性。我们使用 1 个 NVIDIA H20 节点部署一个 P 实例和一个 D 实例，采用 Qwen3-235B-A22B-FP8 模型，每个实例占用 4 个 GPU。此外，我们使用 2 个 NVIDIA H200 节点部署一个 P 实例和一个 D 实例，采用 DeepSeek-V3.1-Terminus 模型，每个实例占用 1 个节点和 8 个 GPU。我们注意到，预填充部署有一个名为分块预填充大小的参数。在一定范围内，分块预填充大小越大，峰值预填充吞吐量就越高，直到达到饱和。M/M/1 排队模型要求依次处理请求；当请求长度大于或等于分块预填充大小时，这一条件成立。当分块预填充大小远大于请求长度时，队列中的多个请求可能会被批量合并并一起处理，这与 M/M/1 排队理论并不完全一致。

为完全符合 M/M/1 排队论的假设，我们首先评估了分块预填充大小等于请求输入长度的情况。我们测量了 H20 部署中输入长度为 4096、H200 部署中输入长度为 12288 时，请求首个令牌时间随请求率的变化情况。随后，我们评估了相应的最大性能，并使用公式（12）预测理论首词元时间；结果如图 1 所示。从图中可以看出，预测的首个令牌时间与测量结果的变化趋势完全一致，仅存在微小差距。该差距主要归因于 KV 缓存传输时间。请注意，图 1 中预测的首个令牌时间不包括从预填充到解码的 KV缓存传输时间，并且 KV缓存传输时间会随着请求输入长度的增加而增加。

如图 1 所示，我们还评估了分块预填充大小远大于请求长度的情况。例如，我们评估了输入长度为 4096、分块预填充大小为 12288，以及输入长度为 6144、分块预填充大小为 24576 的情况。我们发现，基于我们的方法得出的结果仍然是很好的近似。

总之，我们的方法首先需要评估已部署的预填充实例在非空闲条件下的最大预填充吞吐量。随后，我们将预填充过程建模为一个 M/M/1排队问题，并推导出公式（13），该公式利用最大预填充吞吐量、所需的首个令牌时间和请求输入长度，计算实际达到的预填充吞吐量。请注意，我们的方法不考虑前缀缓存[4, 10, 11, 18]。如果预填充计算时间与未命中前缀缓存的部分的长度成正比，只需将本文中的输入长度替换为未命中 KV 缓存的输入长度即可。

在每输出令牌时间约束下获取解码吞吐量

本节介绍如何获取满足目标每输出令牌时间（TPOT）的实际解码吞吐量。在解码阶段，大语言模型引擎利用连续批处理并行计算多个请求。我们发现，解码 TPOT 和解码吞吐量都与解码批次大小正相关。具体而言，批次越大，解码吞吐量越高，但 TPOT 也会增加。因此，要获得满足目标 TPOT 的解码吞吐量，只需先对实际解码吞吐量和 TPOT 随解码批次大小变化的曲线进行基准测试，然后确定解码

满足 TPOT 要求的吞吐量。

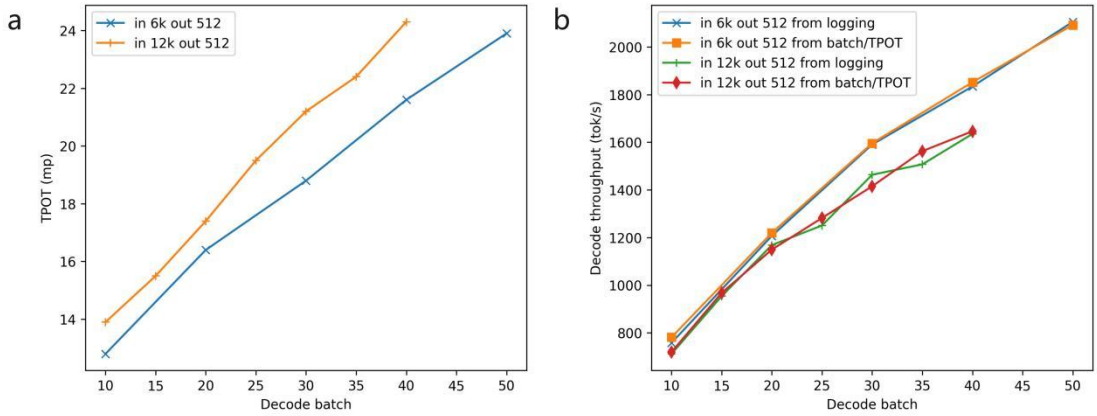

图2：DeepSeek-V3.1-Terminus 模型部署在 H200节点上时，TOPT 和解码吞吐量随批次大小变化的基准测试曲线。请注意，基准测试中启用了多令牌预测。

如图2所示，我们评估了使用 8 个 NVIDIA H200 GPU 部署的 DeepSeek-V3.1-Terminus 模型在不同解码批次大小下 TPOT 和解码吞吐量的变化曲线。我们测试了两种输入序列长度（6144 和 12288），输出长度固定为 512。TPOT 根据基准测试工具 EvalScope[15],的结果得出，而解码吞吐量则通过两种独立方法测量：一种是从 SGLang 推理引擎打印的运行时日志中提取，另一种是用解码批次大小除以对应的 TPOT 计算得出。我们观察到，这两种方法得出的吞吐量结果高度一致。因此，无需记录和解析运行时日志，只需对 TPOT 和对应的批次大小进行基准测试，即可轻松获得解码吞吐量。值得注意的是，这种吞吐量计算方法要求解码计算批量大小与基准测试时的并发量大致一致。

总之，为了达到满足 TPOT 要求的解码吞吐量，我们首先在真实的用户输入–输出工作负载下，刻画解码 TPOT 随批次大小变化的曲线。然后确定满足目标 TPOT约束的批次大小，并通过将该有效批次大小除以相应的 TPOT，得出对应的解码吞吐量。

评估

我们通过一个真实的推理场景验证了我们的方法。用户要求如下：

- 大语言模型：DeepSeek-V3.1-Terminus
- 首个令牌时间：2 s
- 每输出令牌时间：20 ms
- 平均输入序列长度：6144
- 平均输出序列长度：512
- 总吞吐量：每分钟 500 万令牌（百万令牌/分钟）。所

用硬件和软件如下：

- 部署硬件：NVIDIA H200 GPU

- 推理引擎：SGLang v0.5.8
- 基准测试工具：EvalScope v1.4.2

为满足 SLO 约束，我们采用 TP 和 EP 相结合的方式部署预填充实例，分块预填充大小为 24576；双批次重叠优化已禁用。对于解码实例，我们仅使用 TP，并禁用了数据并行和 EP。每个预填充/解码实例均运行在独立节点上，且所有实例均启用了多令牌预测（MTP）。

我们评估得出，对于该输入序列长度，最大预填充吞吐量为 28300 tokens/s。考虑到 100 ms 的 KV 缓存传输时间（即 $T_{overhead} = 100$ ms），根据公式（13），我们推导出有效预填充吞吐量约为 25000 tokens/s。

根据图2所示的 TPOT 和解码吞吐量随解码批大小变化的基准测试曲线，我们得出，在满足 TPOT约束的情况下，解码吞吐量约为 1700 tokens/s。根据公式（7），我们计算得出预填充解码比为 0.82:1。利用公式（5）和（6），我们确定需要三个预填充实例和四个解码实例才能满足总吞吐量需求。本文使用 mPnD 表示包含 m 个预填充实例和 n 个解码实例的部署。因此，我们针对这一场景提出 3P4D 部署方案。

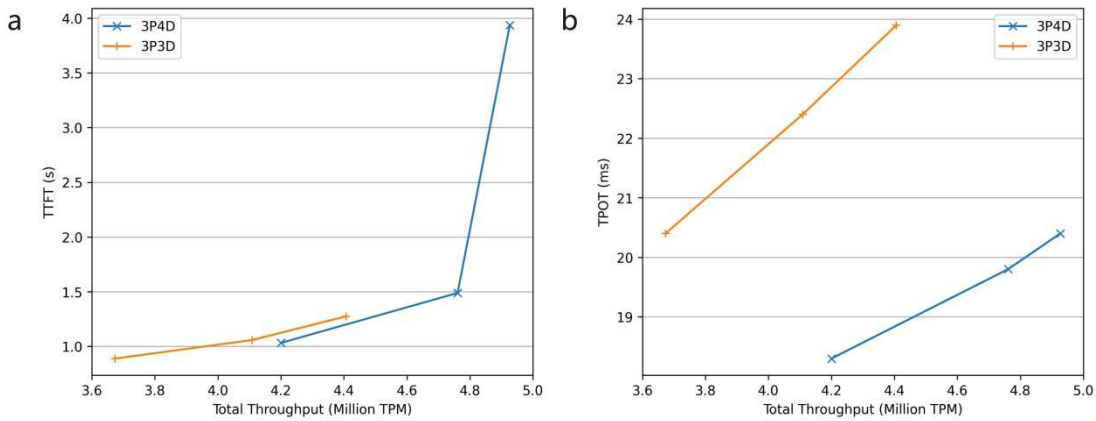

图 3 不同 P/D 资源数量下，首个令牌时间和每输出令牌时间随总吞吐量变化的曲线。

我们评估了 3P4D 部署下首个令牌时间和每输出令牌时间随总吞吐量变化的曲线，并评估了 3P3D 场景以作对比。结果如图 3 所示。从中可以观察到，采用 3P4D 部署时，2 秒首个令牌时间阈值与 20 毫秒服务等级目标阈值可在约 4.8 百万令牌/分钟处同时达成。这表明我们的方法计算出的预填充/解码比较为均衡，所实现的总吞吐量非常接近 5 百万令牌/分钟的目标值。相比之下，采用 3P3D 部署时，服务等级目标阈值仅能在约 3.6 百万令牌/分钟处满足，主要受每输出令牌时间制约，而首个令牌时间仍有相当余量。在该服务等级目标下，3P4D 的单节点平均吞吐量为 0.69 百万令牌/分钟，而 3P3D 为 0.6 百万令牌/分钟，表明我们的方法计算出的部署效率更高。

结论

本文针对 P/D 解耦式大语言模型推理中的计算资源分配这一关键挑战展开研究。我们提出一种将理论建模与实证基准测试相结合的混合方法。首先，我们推导出一个公式，依据总吞吐量要求、SLO 约束、请求输入/输出长度以及 P/D 吞吐量来确定所需的 P/D 资源数量。在该公式中，大多数参数为用户指定的已知值，而 P/D 部署实际可达到的吞吐量仍不明确。其次，我们采用 M/M/1 排队论对预填充阶段进行建模，从而推导出满足 TTFT 约束的有效吞吐量。对于解码阶段，我们通过实证测量确定满足 TPOT 要求的最优批次大小，从而在避免违反性能保证的同时确保资源利用率最大化。最后，我们通过真实场景推理验证了该方法，表明其能够准确预测 P/D 资源数量，从而在给定的 SLO 约束下最大化 P/D 资源利用效率。

未来，我们的方法可以与 AIConfigurator 等方法集成。AIConfigurator 能够估算各个 P/D 实例的最优部署，并采用基于仿真的方法预测 P/D 吞吐量。此外，我们的方法有望推广至多模态 EPD 分离系统[14]，，从而确定三个独立部署组件所需的资源数量。

参考文献

[1] Amey Agrawal、Ashish Panwar、Jayashree Mohan、Nipun Kwatra、Bhargav S Gulavani 和 Ramachandran Ramjee。2023。Sarathi：通过让解码搭载分块预填充，实现高效的大语言模型推理。*arXiv* 预印本 *arXiv:2308.16369*（2023）。[2] Xing Chen、Rong Shi、Lu Zhao、Lingbin Wang、Xiao Jin、Yueqiang Chen 和 Hongfeng Sun。2025。面向多厂商 GPU 上大语言模型服务的预填充与解码解耦推理系统。*arXiv* 预印本 *arXiv:2509.17542*（2025）。[3] Donald Gross 和 Carl M Harris。2008。《排队论基础》（John Wiley & Sons）。*Inc*：新泽西州（2008）。[4] Cunchen Hu、Heyang Huang、Junhao Hu、Jiang Xu、Xusheng Chen、Tao Xie、Chenxi Wang、Sa Wang、Yungang Bao 和 Ninghui Sun。2024。Memserve：借助弹性内存池，为解耦式大语言模型服务提供上下文缓存。*arXiv* 预印本 *arXiv:2406.17565*（2024）。[5] Yibo Jin、Tao Wang、Huimin Lin、Mingyang Song、Peiyang Li、Yipeng Ma、Yicheng Shan、Zhengfan Yuan、Cailong Li 和 Yajing Sun。2024。P/D-Serve：大规模提供解耦式大语言模型服务。*arXiv* 预印本 *arXiv:2408.08147*（2024）。[6] Leonard Kleinrock。1975。《排队系统》第一卷：理论。（1975）。[7] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody HaoYu、Joseph Gonzalez、Hao Zhang 和 Ion Stoica。2023。通过 PagedAttention 高效管理大语言模型服务的内存。2023。611–626。[8] Junhan Liao、Minxian Xu、Wanyi Zheng、Yan Wang、Kejiang Ye、Rajkumar Buyya 和 Chengzhong Xu。2025。DOPO：一种用于最大化大语言模型推理服务有效吞吐量的动态 PD 解耦架构。*arXiv* 预印本 *arXiv:2511.20982*

（2025）。[9] Aixin Liu、Bei Feng、Bing Xue、Bingxuan Wang、Bochao Wu、Chengda Lu、Chenggang Zhao、Chengqi Deng、Chenyu Zhang 和 Chong Ruan。2024。Deepseek-v3 技术报告。*arXiv* 预印本 *arXiv:2412.19437*（2024）。[10] Yuhan Liu、Yihua Cheng、Jiayi Yao、Yuwei An、Xiaokun Chen、Shaoting Feng、Yuyang Huang、Samuel Shen、Rui Zhang 和 Kuntai Du。2025。Lmcache：面向企业级大语言模型推理的高效 KV 缓存层。*arXiv* 预印本 *arXiv:2510.09665*（2025）。[11] Zaifeng Pan、Ajjkumar Patel、Zhengding Hu、Yipeng Shen、Yue Guan、Wan-Lu Li、Lianhui Qin、Yida Wang 和 Yufei Ding。2025。KVFlow：通过高效前缀缓存加速基于大语言模型的多智能体工作流。*arXiv* 预印本 *arXiv:2507.07400*（2025）。[12] Pratyush Patel、Esha Choukse、Chaojie Zhang、Aashaka Shah、Íñigo Goiri、Saeed Maleki 和 Ricardo Bianchini。2024。Splitwise：通过阶段拆分实现高效的生成式大语言模型推理。2024。IEEE，118–132。[13] Ruoyu Qin、Zheming Li、Weiran He、Jialei Cui、Heyi Tang、Feng Ren、Teng Ma、Shangming Cai、Yineng Zhang 和 Mingxing Zhang。2024。Mooncake：以 KV 缓存为核心的解耦式大语言模型服务架构。*ACM TransactionsonStorage*（2024）。[14] Gursimran Singh、Xinglu Wang、Yifan Hu、Timothy Yu、Linzi Xing、Wei Jiang、Zhefeng Wang、Xiaolong Bai、Yi Li 和 Ying Xiong。2024。利用 EPD 解耦高效服务大型多模态模型。*arXiv* 预印本 *arXiv:2501.05460*（2024）。[15] ModelScope团队。EvalScope：大模型评估框架，2024。网址 *https://github. com/modelscope/evalscope*。[16] Hugo Touvron、Thibaut Lavril、Gautier Izacard、Xavier Martinet、Marie-Anne Lachaux、Timothée Lacroix、Baptiste Rozière、Naman Goyal、Eric Hambro 和 Faisal Azhar。2023。Llama：开放且高效的基础语言模型。*arXiv* 预印本 *arXiv:2302.13971*（2023）。[17] Zhibin Wang、Zetao Hong、Xue Li、Zibo Wang、Shipeng Li、Qingkai Meng、Qing Wang、Chengying Huan、Rong Gu 和 Sheng Zhong。2025。预填充-解码解耦式大语言模型推理中的自适应重新调度。*arXiv* 预印本 *arXiv:2510.13668*（2025）。[18] Zhiqiang Xie、Ziyi Xu、Mark Zhao、YuweiAn、Vikram Sharma Mailthody、Scott Mahlke、Michael Garland 和 Christos Kozyrakis。2025。Strata：面向长上下文语言模型服务的分层上下文缓存。*arXiv* 预印本 *arXiv:2508.18572*（2025）。[19] Tianhao Xu、Yiming Liu、Xianglong Lu、Yijia Zhao、Xuting Zhou、Aichen Feng、Yiyi Chen、Yi Shen、Qin Zhou 和 Xumeng Chen。2026。AIConfigurator：面向多框架大语言模型服务的闪电般快速配置优化。*arXiv* 预印本 *arXiv:2601.06288*（2026）。[20] An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang 和 Chenxu Lv。2025。Qwen3 技术报告。*arXiv* 预印本 *arXiv:2505.09388*（2025）。[21] Hengrui Zhang、Pratyush Patel、August Ning 和 David Wentzlaff。2025。SPAD：面向解耦式大语言模型推理的专用预填充与解码硬件。*arXiv*

预印本 *arXiv:2510.08544*（2025）。[22] Lianmin Zheng、Liangsheng Yin、Zhiqiang Xie、Chuyue Livia Sun、Jeff Huang、Cody Hao Yu、Shiyi Cao、Christos Kozyrakis、Ion Stoica 和 Joseph E Gonzalez。2024。Sglang：高效执行结构化语言模型程序。神经信息处理系统进展 37，（2024），62557–62583。[23] Yinmin Zhong、Shengyu Liu、Junda Chen、Jianbo Hu、Yibo Zhu、Xuanzhe Liu、Xin Jin 和 Hao Zhang。2024。{DistServe}：通过分离预填充与解码，实现吞吐效能优化的大语言模型服务。2024。193–210。[24] ChatGPT。检索自 https://chatgpt.com/[25] TensorRT-LLM。检索自 https://github.com/NVIDIA/TensorRT-LLM[26] 在 96 个 H100 GPU 上通过预填充与解码分离及大规模专家并行部署 DeepSeek。检索自 https://lmsys.org/blog/2025-05-05-large-scale-ep/[27] SGLang v0.4：零开销批处理调度器、缓存感知负载均衡器、更快的结构化输出。检索自 https://lmsys.org/blog/2024-12-04-sglang-v0-4/