



MTT S5000 Prefill-as-a-Service

技术白皮书

发布日期：2026 年 8 月

[摘要]

大模型推理服务正从短上下文、单请求的简单形态，演进为以 Agent、AI Coding 与长文档分析为代表的超高并发、长上下文在线系统。在此背景下，Prefill（预填充）与 Decode（解码）两个阶段在计算特性与硬件资源诉求上的根本差异日趋凸显：Prefill 为计算密集型负载，受浮点算力约束；Decode 为访存密集型负载，受显存带宽约束。在传统同构混部架构中，两者共享同一 GPU、同一显存、同一调度域，由此引发双重困境——长 Prefill 挤占调度窗口导致尾延迟（P99/ITL）离散化恶化，而同构扩容因两类负载资源诉求相背离而只会加剧错配。

本文首先论证了 Prefill as a Service^[1] 计算池统计口径与同构资源池双向浪费的结构成因，据此提出基于 Prefill-Decode 分离（PD 分离）的架构演进路径，并以 MTT S5000 作为 Prefill as a Service 算力池、GPU A* 作为 Decode 生成池，构建了异构 PD 分离方案。该方案基于 SGLang 主线的 PD 分离与分页 KV Cache 机制，并通过 Mooncake 框架实现跨节点传输；在 KV 页大小、布局与精度完全对齐的特定条件下，可规划页粒度直传路径以消除转换与带宽开销，实现 Prefill 与 Decode 的解耦供给、独立扩缩容与差异化的 SLO 保障。以智谱 GLM-5.2 为测试模型、在 90% 前缀缓存命中率与 P50 TTFT $\leq 6000\text{ms}$ 的 SLO 约束下，单机 8 卡 MTT S5000 于 64K 输入场景实现 Prefill 吞吐 95,920.5 tok/s，满载月收入约 64.6 万元；面向典型 Agent 混合上下文负载，单台满载月收入约 56.4 万元。本文结论认为，P/D 分离方案并非仅依赖调度层面的优化，而是推理基础设施的结构性架构演进。MTT S5000 以高密度 FP8 算力、原生精度支持及 MUSA 生态的 CUDA 源码级兼容，将 Prefill 从推理流程中的环节剥离为可独立供给、独立计费、独立优化的算力产品，为超大参数模型长上下文推理提供了一条围绕物理瓶颈分层收敛、具备良好可行性且可规模化落地的成本优化路径。

第一章

大模型推理服务的现状与挑战

随着生成式 AI 在问答、代码开发与智能代理等场景加速落地，面向高并发、低时延和高资源利用率的 Token 服务将成为大模型基础设施的重要增长方向。当前大模型推理服务的生产形态，已突破早期“单请求、单卡、即时一问一答”的简单抽象约束，演进为高并发、低延迟、长上下文的在线服务系统。理解其挑战，需先厘清一次请求在物理层的两个阶段：

Prefill（预填充）：对输入 prompt 执行一次性并行前向计算，并建立后续生成所需的 KV cache。其计算量与输入 token 数、模型结构及批处理形状正相关，属计算密集型负载。

Decode（解码）：Prefill 完成后，以自回归方式逐 token 生成后续输出，每步读取当前 attention 机制所需的历史状态。在标准全注意力及较小 Decode batch 条件下，每步计算量有限，瓶颈通常落在权重与历史 KV 的读取带宽上，属访存密集型负载。

两类负载计算特性和硬件资源的需求的差异，是后续一切结构性挑战的起点。

1.1 当 AI 助手突然“卡住”：一次停顿背后的调度问题

在生产环境的监控面板上，一个现象反复出现：模型在生成流畅的输出时，会周期性出现可见中断——相邻输出 token 的间隔（ITL）出现尖峰，用户侧的观感是“生成卡住了几秒”。

让运维团队困惑的，是这一现象与平均指标的矛盾：系统平均时延一直健康，但用户体验在恶化。答案不在这对矛盾本身，而在统计口径。

在线推理的请求负载呈现显著的双峰分布：

负载类型	上下文长度	请求占比	资源消耗占比
短请求（查询型）	~数百 token	高	低
长请求（文档/仓库分析）	数万 token	低	高

表 1-1 在线推理请求负载的双峰分布

由于长请求占比极低，它们对平均时延的影响被稀释到几乎不可见。平均指标在此处不是“失真”，而是“失效”——它无法反映尾部分位数（P99）的真实状况。

在单体推理引擎中，为最大化 GPU 利用率，调度器采用连续批处理（Continuous Batching）：将多个请求的解码步骤（Decode）打包为连续批次，同时将新请求的预填充（Prefill）挤入同一调度域。

当长请求进入时，其 Prefill 连续占用多个调度时间片。这段时间内，所有在途请求的下一次 Decode 只能排队。关键洞察在于：等待并不均匀分布——它集中于少数长 Prefill 窗口。

绝大多数时间系统维持低延迟，只有特定调度窗口出现尖峰。统计后果是：P99 被拉长，均值不动。这正是“平均指标健康”陷阱的完整解释。

1.2 同构集群的隐性成本：一种放大错配的架构选择

在线推理服务的采购决策中，存在一个常被平均指标掩盖的事实：持续攀升的硬件成本与两类推理负载的实际资源消耗之间，存在系统性错配。这一错配源于 Prefill 与 Decode 物理特性上的根本差异，并在同构 GPU 集群下被固化为长期低下的资源利用率。

问题的起点，是两类负载对资源的要求截然相反。Prefill 一次性处理完整输入，受浮点算力约束，属计算密集型；Decode 逐 token 生成，每步都要读取权重与 KV Cache，受显存带宽约束，属访存密集型。二者对硬件的核心诉求互异，却被迫共享同一套物理资源。

这一矛盾因芯片演进的非对称性而愈演愈烈。近几代 GPU 中，浮点算力代际提升接近翻倍，而显存带宽增速不足一倍，两者长期背离。

Prefill 以大规模 GEMM 为主体，算术强度随 batch 内 token 数近似线性增长，通常已越过 roofline 拐点，落在 compute-bound 区间——额外带宽难以兑换为等比例的吞吐增益，真正的驱动力是算力密度；带

宽只是“够用即止”的约束，而非“堆满”的目标。与此同时，旗舰 GPU 单价又被显存容量与先进制程成本共同推高。单一资源池混跑两类负载，必然双向浪费：按 Decode 带宽选型，则 Prefill 的大容量显存长期低效；按 Prefill 算力选型，则 Decode 计算单元大面积空转。这是物理规格层面无法共存的结构性价位，而非调度可以消解的低效。

真正的解法，不是在同一资源池里用调度去弥合错配，而是把它还给硬件本身：让 Prefill 与 Decode 各自运行在最契合其计算特性的硬件平台。通过清晰的资源分层，实现硬件和软件的最佳协同。

Prefill 池，主打算力：高密度低精度算力、适度带宽、KV Cache 可迁出故容量从简、强化 KV 传输互联、弹性伸缩；

Decode 池，主打带宽与容量：以充足显存资源，承接生成阶段的访存压力。

第二章

大模型推理异构 PD 分离背景

2.1 什么是 PD 分离

PD 分离（Prefill Decode Disaggregation）架构的核心思想，是将大模型推理中的 Prefill 和 Decode 两个阶段解耦，并调度至相对独立的计算资源池中。Prefill 阶段主要对输入上下文进行并行计算并构建 KV Cache，通常直接影响首 Token 延迟（TTFT）；Decode 阶段负责后续 Token 的逐步生成，重点优化每 Token 生成时间（TPOT）。通过针对两个阶段的不同计算特征进行独立扩缩容与资源配置，PD 分离能够提升集群整体吞吐和资源利用率。

2.2 为什么需要异构 PD 分离

2.2.1 长上下文 Prefill 挤占资源，推高在线推理尾延迟

在 AI Coding、长文档分析、知识库问答和 Agent 等场景中，代码上下文、检索结果、工具调用记录及多轮历史会显著增加输入长度，使 Prefill 阶段的计算与显存开销随之上升。以编码 Agent 为例，密歇根大学和斯坦福大学的实测数据显示，其端到端执行过程中输入 Token 与输出 Token 的累计比例可达到约 154:1^[2]；该指标反映的是多轮模型调用的总体 Token 消耗，而非单次生成时的即时输入输出比例。

主流场景	典型输入长度	典型输出长度	Prefill 计算压力
Coding Plan / AI Coding	10K-200K tokens, 读取项目上下文	1K-2K tokens	极高
长文档分析	50K-1M tokens, 整本书或技术文档	500-5K tokens	极高
知识库问答 RAG	5K-50K tokens, 检索结果加用户问题	200-2K tokens	高
Agent / 工具调用	2K-20K tokens, 多轮历史与工具结果	500-5K tokens	中高
批量推理	1K-10K tokens/请求, 叠加高并发	200-1K tokens	总量极高

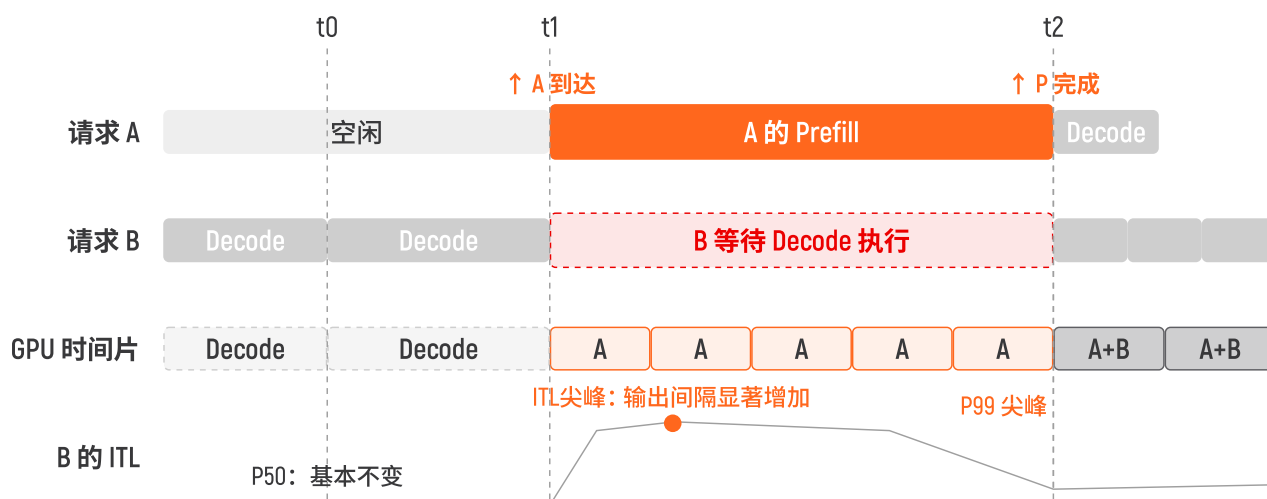
表 2-1 长上下文业务场景与 Prefill 压力

在单体部署中，长请求的 Prefill 与在线 Decode 共享 GPU、显存和调度域。长 Prefill 往往需要占用多个执行时间片，可能推迟短请求的逐 Token 生成，进而抬高 Token 间延迟（ITL）的尾延迟。

对于长输入请求，系统需完成上下文计算并构建 KV Cache。若长 Prefill 与延迟敏感的 Decode 请求在同一资源池中混合调度，可能产生队首阻塞，同时增加首 Token 延迟（TTFT）和 Token 间延迟。P/D 分离通过隔离 Prefill 与 Decode 负载，可缓解两者之间的资源竞争；分块 Prefill、连续批处理和优先级调度等机制同样能够降低这一影响。

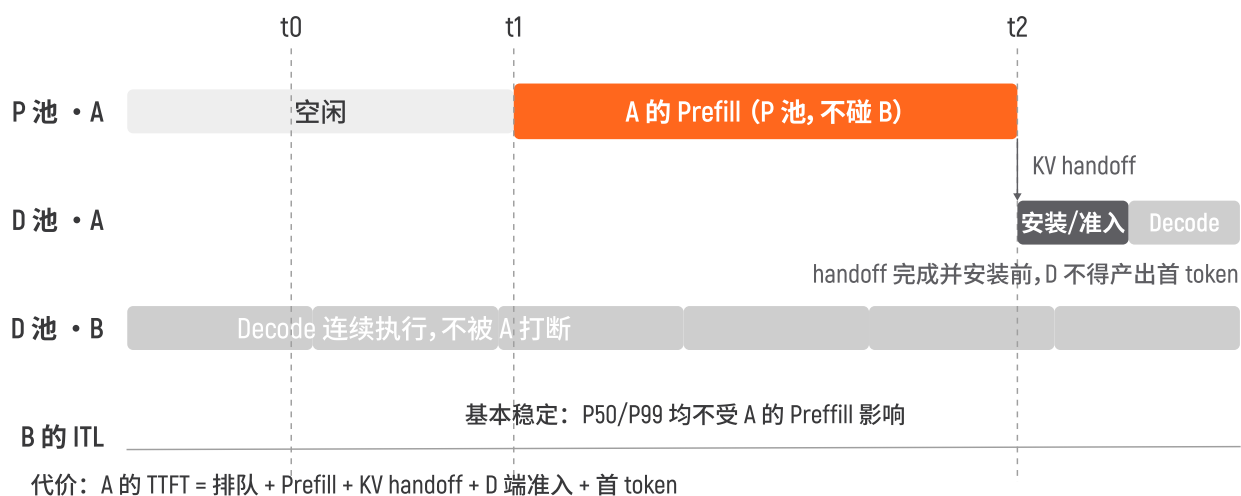
单位服务：共享GPU调度域

共享对象：GPU算力、带宽、显存KV容量、连续批处理调度域



归因：B 的 ITL 尖峰与 A 的 Prefill 窗口对齐；不是平均 ITL 的轻微上升，也不能仅凭现象归因于网络。

P/D 分离：P 池与 D 池隔离



请求 B/Decode 请求 A/Prefill KV handoff B 等待执行窗口 空闲/等待

表 2-1 长上下文业务场景与 Prefill 压力

单体连续批处理将 A 的 Prefill 与 B 的 Decode 放在同一调度域，A 的执行时间可能延后 B 的下一次 Decode，表现为 ITL 尖峰。P/D 分离将 A 的 Prefill 放入 P 池、将 B 的 Decode 放入 D 池，从调度域上隔离两类阶段；A 的首 token 路径则新增 KV handoff、D 端安装和准入步骤。

2.2.2 Prefill 与 Decode 对计算资源的需求差异

本节以稠密 Transformer、标准因果（causal）全注意力模型结构为基础，单个输入序列为例，说明 Prefill-Decode（P/D）分离对资源需求差异的理论依据。

各变量命名：输入 Token 长度为 S ；生成 Token 长度为 T ；模型层数为 L ；隐藏维度为 d ；Query 头数为 n_q ；KV 头数为 n_{kv} ；单头维度为 d_h ，对于 GQA/MQA $d = n_q d_h$ ；KV Cache 单元字节数为 b ，例如 FP16/BF16 为 2，FP8/INT8 为 1（暂不考虑 KV 量化额外计入的 scale、zero-point 等元数据）。

若批量大小为 B ，线性层计算、KV Cache 容量以及 KV 读写量通常近似乘以 B ；变长批次应按序列分别求和。模型权重读取量不随请求数线性增长，这是 batching 能提升权重复用、改善 Decode 权重侧算术强度的根本原因。下文先按单序列进行理论推导，暂忽略批量 B 。

从系统视角看，KV Cache 的作用是保存历史 Token 的 Key/Value，避免 Decode 阶段每生成一个 Token 都重新计算完整前缀。它显著减少重复计算，但会把一部分压力转移到显存容量、带宽以及跨节点传输带宽上。因此，Prefill 与 Decode 对算力、显存、带宽的需求天然不同。

Prefill 阶段

Prefill 在逻辑上一次性处理 S 个输入 Token，完成各层前向计算，并将输入对应的 Key/Value 写入 KV Cache。工程实现中可采用 chunked prefill，也可与 Decode 请求混合成批。因此“同时处理 S 个 Token”是计算逻辑描述，不要求实现上必须把整个 Prompt 放在同一个调度片中。在上述前提下，Prefill 的主导计算由两部分组成：

1. **线性层与 MLP 计算**：包括 Q/K/V/O 投影、MLP 等，随 Token 数线性增长。
2. **注意力计算**：包括全注意力中 QK^T 与 Attention-Value 计算，随上下文长度的平方增长。

忽略 LayerNorm、残差、激活函数等低阶逐元素操作后，单序列 Prefill 的总计算量可近似写为：

$$F_{prefill} = O(L(Sd^2 + S^2d))$$

- ▶ Sd^2 表示线性层与 MLP 的计算量级，隐藏维度大小固定，故随 S 线性增长。
- ▶ Sd^2d 表示注意力矩阵乘法的计算量级，随 S^2 线性增长。
- ▶ 若 MLP 中间维度约为 $4d$ ，或采用 SwiGLU 等变体，只会改变常数因子，不改变上述量级结论。
- ▶ GQA/MQA 会降低 K/V 投影输出维度与 KV Cache 写入规模，但 Query 侧注意力仍以 n_q 个 Query 头参与计算，因此注意力计算量级不按 n_{kv}/n_q 等比例下降。

Prefill 除计算之外，需要写入的 KV Cache 量： $KV\ Write\ Bytes = 2LSn_{kv}d_hb$ 。其中系数 2 表示 Key 与 Value 各存一份。标准 MHA 下 $n_{kv}=n_q$ ，因此 $d=n_{kv}d_h$ ；GQA/MQA 下 $n_{kv} < n_q$ ，KV 写入量按 n_{kv}/n_q 近似降低。

小结： Prefill 中的两类计算均以大规模 GEMM 形态存在、算术强度高。上下文长度达一定量级后，注意力项逐渐成为主导，因此 Prefill 整体呈计算密集（compute-bound）特征；KV 写入的访存开销相对计算量较小，不构成主要瓶颈。FlashAttention 等融合实现通过减少中间注意力矩阵落显存，进一步强化了 Prefill 的计算密集特征。

KV Cache 容量

Prefill 完成后，单个序列、长度为 S 的 KV Cache 逻辑容量为：

$$KV\ Cache\ Bytes = 2LSn_{kv}d_hd$$

生成 T 个 Token 后，若缓存完整上下文，容量相应增长为：

$$KV\ Cache\ Bytes = 2L(S + T)n_{kv}d_hd$$

该公式说明 KV Cache 容量与层数、上下文长度、KV 头数、单头维度和元素字节数线性相关。GQA/MQA 的核心收益之一，就是把从 n_{kv} 从 n_q 降低到更小值，从而直接减少 KV Cache 容量和后续 Decode 的 KV 读取带宽。

需要注意，上述容量为逻辑容量，工程显存占用可能高于或低于该值。比如 PagedAttention 等分页管理引入块分配和内部碎片，前缀共享（prefix caching）又会降低实际占用，MLA 缓存的是压缩潜向量，

滑动窗口注意力通常只保留窗口 W 内的历史状态。

Decode 阶段

Decode 在 Prefill 之后逐 Token 生成。生成第 t 个新 Token 时，历史长度约为 $S+t-1$ 。由于历史 Token 的 K/V 已经保存在 Cache 中，当前步只需计算当前 Token 的 Q/K/V、MLP、输出投影，并让当前 Query 与历史 K/V 做注意力计算。单步主导计算量如下：

$$F_{\text{decode_step}(n)} = O(L(d^2 + (S+t-1)d))$$

- ▶ d^2 项来自当前 Token 的线性层与 MLP。
- ▶ nd 项来自当前 Query 对历史上下文的注意力。
- ▶ GQA/MQA 主要减少 KV Cache 的存储和读取量，并不按同样比例降低 Query 与历史 Key 之间的注意力计算量级。

生成 T 个 Token 的总计算量为：

$$F_{\text{decode_total}} = O(L(Td^2 + (ST + T(T-1)/2)d))$$

如果 T 相对 S 较小，注意力项常近似为： $O(LTSd)$ ，若生成长度也很长，则 $T(T-1)/2$ 项不可忽略。

单步读取历史 KV 的逻辑数据量为： $KV \text{ Read Bytes/step} = 2L(S + t - 1) n_{kv} d_h b$ 。当前 Token 还会写入新的 KV，写入量为： $2Ln_{kv}d_h b$ 。注意：实际显存流量还取决于缓存命中、kernel 实现与 KV 布局。

小结： Decode 往往呈 memory-bound，原因有两类：

1. **权重侧带宽压力：** 低批量时每步只处理少量 Token，矩阵乘法退化为接近 GEMV 的形态，模型权重需要被反复从显存读取，算术强度较低；增大 batch 或 continuous batching 可以提高权重复用。
2. **KV 侧带宽压力：** 每步都要读取随上下文长度线性增长的历史 K/V，且不同请求的 KV 不能像权重一样共享；因此长上下文和大并发下，瓶颈会逐渐从权重带宽转向 KV Cache 带宽与容量。

作为对照，若不使用 KV Cache，Decode 每生成一个 Token 都要对完整前缀重新前向，线性层与注意力都将对历史 Token 重复计算。KV Cache 的核心价值是避免这种重复计算，使每步只对当前 Token 做线性层计算，并复用历史 K/V；代价则是需要持续维护和读取 KV Cache。

因此，Prefill 更需要高算力和高并行 GEMM 能力，直接影响 TTFT；Decode 更依赖显存容量、带宽、KV Cache 布局 and 调度效率，直接影响 TPOT。P/D 分离正是利用这种资源需求差异：Prefill 侧可选择更偏计算吞吐的配置，Decode 侧可选择更重视显存容量、带宽和稳定低延迟的配置。其收益是 TTFT 与 TPOT 指标解耦，减少长 Prefill 对 Decode 尾延迟的干扰；其代价是 Prefill 产生的 KV Cache 需要传输到 Decode 侧，引入额外互联带宽和延迟开销。因此，P/D 分离不是无条件最优，而应结合 chunked prefill、continuous batching、SLO 目标、请求长度分布以及节点间互联能力综合权衡。

2.2.3 异构 PD 分离在大模型推理服务中的价值

异构 PD 分离的价值，不在于单纯拆分推理流程或增加硬件数量，而在于将资源供给、调度策略与服务目标按阶段解耦，并进一步为两类负载匹配各自最优的硬件规格——以算力型和显存平衡型 GPU 承载 Prefill、以大显存高带宽的 GPU 承载 Decode。

1. 资源利用率提升：从“通用冗余”到“按需匹配”

在统一资源池中，系统须以“兼顾两类负载”的方式配置 GPU，难以使计算、显存容量与带宽长期处于高效利用状态：算力型硬件在 Decode 阶段被带宽拖累，其昂贵算力又被大量闲置。

异构 PD 分离后，可依据各阶段的实际瓶颈独立选择硬件规格、批处理策略与并发规模：

- ▶ Prefill 池：追求高 MFU（Model FLOPs Utilization，模型算力利用率）与低 TTFT；
- ▶ Decode 池：追求满带宽与高并发下的稳定 ITL。

同时，两个资源池的利用率可分别观测、分别优化，避免一侧紧张而另一侧闲置。由此提升的是集群整体的有效吞吐（goodput）——即满足 SLO 约束下的达标 Token 产出，而非单纯的理论吞吐。

2. 弹性伸缩能力：解耦后的定向扩缩容

不同业务形态、模型规格与流量周期下，Prefill 与 Decode 的负载比例并不固定。统一扩容需同时增加两类资源，易造成冗余供给。PD 分离后，系统可基于实时指标定向扩缩容：

- ▶ 当请求排队增加、TTFT 恶化时，优先扩展 Prefill 资源；
- ▶ 当生成并发升高、KV Cache 显存吃紧或 Decode 批处理排队导致 ITL 抖动时，优先扩展 Decode 资源；

针对不同模型、租户或优先级业务，动态调整两类资源池的容量配比。

这种更细粒度的调度，提高了集群对流量波动与业务结构变化的适应能力。

3. 增强服务等级（SLO）保障能力

PD 分离为不同类型请求建立了更清晰的物理隔离边界，使系统能够围绕 TTFT、ITL、TPOT 等指标实施差异化服务策略：

- ▶ 为实时交互、付费用户或关键 Agent 任务，预留 Decode 池的并发名额与显存配额；
- ▶ 对超长上下文、离线批处理等非实时任务，实施限流、排队或降级。

此外，Decode 资源池可基于 KV Cache 水位、活跃请求数与尾延迟情况进行准入控制，降低资源耗尽引发排队扩散与级联延迟的风险。

这一隔离机制的本质，是将“尽力而为的平均吞吐调度”转化为“面向尾延迟的可预测资源预留”，使系统优化目标从“提升平均吞吐”升级为“稳定保障尾延迟与用户体验”。

4. 成本优化能力：围绕瓶颈的分层投入

异构 PD 分离使硬件采购与扩容能够围绕实际瓶颈展开，不再要求所有节点同构同规格。其成本逻辑的关键在于：将最昂贵的算力只投放在真正算力受限的 Prefill 上，而对于带宽受限的 Decode 阶段，则交由配备匹配算力和高显存规格的 GPU 来承载。

对于规模化在线推理业务，这种资源分层有助于减少闲置能力与过度配置，降低单位 Token 的基础设施成本。

第三章

异构 PD 分离方案

3.1 异构 PD 方案总体架构

本章介绍一种以 MTT S5000 作为 Prefill 计算池、GPU A*^①作为 Decode 生成池的异构 Prefill/Decode 分离（PD Disaggregation）架构。该架构基于 SGLang 的 Prefill-Decode Disaggregation 能力，将长提示词的 Prefill 阶段与逐 Token 生成的 Decode 阶段解耦：由 Router 统一接收请求，并分别调度至 Prefill 实例与 Decode 实例。其中，MTT S5000 池承担长上下文 Prompt 编码、Attention 计算与 KV Cache 生成；GPU A* 池则承担对延迟更敏感的逐 Token 生成（Decode loop），以充分发挥其在推理吞吐与性价比方面的优势。Prefill 阶段生成的 KV Cache，并通过 Mooncake Transfer Engine 跨节点传输至对应 Decode 节点，结合 GPUDirect RDMA 的直连能力，绕过主机内存、减少数据拷贝，有效降低端到端传输延迟与开销。



图 3-1 Mooncake 异构 KVCache 传输架构：Prefill S5000 (MUSA) → GPU A* (CUDA)

① GPU A* 为 Decode 侧所用 GPU 的代称，指某款具备高显存带宽的通用 GPU，具体型号从略；本文测试结论不依赖于该型号的具体选择。

3.2 关键技术与工程可行性

异构 Prefill/Decode 分离的工程可行性，建立在三类成熟技术组件之上。

- 1. KV Cache 协同层。** 该层负责目标 KV 缓存预分配、地址索引的映射与同步、跨节点传输及数据一致性校验。在会话建立阶段，系统先校验模型结构、分词器、层数、KV Head 数量、页大小及数据类型等兼容性描述符；随后以校验结果为依据，根据页大小、内存布局与存储数据类型三者的对齐情况，自动选择以下三条路径之一完成传输：当发送端（MTT S5000）与 Decode 池两侧的 KV Cache 在页大小、内存布局及存储数据类型上完全一致（且两端页表索引已就绪，如均采用 FP8 格式）时，走页粒度直传路径，绕过重排与类型转换，避免额外的计算与带宽开销；当存储数据类型一致但内存布局或页粒度不同时，通过地址映射与布局重排完成传输；当存储数据类型不一致（如 FP16 → FP8）或存在复合不匹配（布局与数据类型同时不同）时，则由专用转换 Kernel 完成格式转换与必要的布局重排后再行传输。
- 2. 队列化生命周期管理。** SGLang 在 Prefill 与 Decode 两侧通过握手、预分配及传输完成队列，保障 KV Cache 生命周期的安全性，并实现 chunked prefill 计算与分块传输的流水线重叠。该能力已合入 SGLang 主线版本持续演进，不依赖任何自研私有组件。
- 3. 高速传输通路。** Mooncake 提供以 KV Cache 为中心的统合高性能传输框架；MTT S5000 与 GPU A* 均支持 GPUDirect RDMA，配合 400G RDMA 组网，为 KV Cache 跨节点传输提供高带宽通路，显著降低传输开销。

上述组件均经过开源社区与生产环境验证，为成熟生态。MTT S5000 支持 FP8 至 FP64 的全精度算力，是国产 GPU 中最早支持 FP8 的产品之一；其 MUSA 工具链提供对 CUDA 的源码迁移级兼容，并支持 GPUDirect RDMA，并已完成面向 GLM-5.2 的全链路适配与实证验证。第四章实测的 P50 TTFT 指标均已计入 KV 传输开销，并与非分离 / 同构分离基线进行对照，证明该方案具备直接落地的工程可行性。

3.3 MTT S5000：Prefill as a Service 场景的高性价比算力选择

MTT S5000 是一款通用 AI 加速卡，适用于训练以及各类模型的推理场景。在异构 PD 分离架构中，其算力特性与 Prefill 阶段的任务需求高度契合，是构建 Prefill as a Service 算力池的高性价比选择。S5000 在以下五个维度匹配 Prefill 阶段的任务特性：

1. **高计算密度，匹配 Prefill 的计算密集特征。** Prefill 以大规模矩阵计算为核心。MTT S5000 提供匹配 Prefill 负载同构的计算密度，以及高稠密峰值算力，同时支持 FP8 精度计算。在指定集群配置下，GLM-5.2 于 64K 输入场景实现 Prefill 单机 8 卡吞吐 95,920 tok/s，同时能够有效压缩首 Token 生成时延。
2. **长序列吞吐能力强，适配长上下文场景。** 在 64K–400K 输入区间，S5000 单机 8 卡 Prefill 吞吐从 95,920 tok/s 平缓降至 44,206 tok/s，且 P50 TTFT 全程稳定在 6 秒 SLO 以内——吞吐随长度可预测、时延不随长度失控，可直接支撑 Agent、长文档分析等业务的容量规划与 SLO 承诺。
3. **原生支持 FP8，降低 KV Cache 传输开销。** S5000 原生支持 FP8 计算，KV Cache 同为原生 FP8 精度，可与 Decode 池在 KV 精度上保持兼容，避免跨节点传输前的格式转换开销，降低传输时延。实现零转换开销的直传还需保证 FP8 格式（E4M3/E5M2）、缩放系数（scale）、KV 页大小、布局与元数据在 Prefill 与 Decode 两侧完全一致。
4. **支持大模型原生精度，Prefill 不引入额外精度损失。** S5000 对主流大模型的原生精度具备天然支持能力，在给定精度与量化标定条件下，Prefill 计算不引入额外精度损失。
5. **MUSA 生态兼容 CUDA，迁移成本低。** MUSA 生态兼容 CUDA，支持 PyTorch、SGLang、vLLM 等主流框架与推理生态，客户可基于现有 CUDA 代码快速迁移适配，GLM-5.2 等主流模型已完成实测验证。

3.4 异构 PD 组网最小部署实例

本组网图为例参考，实际交付将依据服务器规模、网络带宽、业务场景按需规划配置。方案配置 1 台 128 口 400G RoCE 交换机、1 台 25G 业务交换机、1 台千兆带外管理交换机，可承载 12 台 MTT S5000 服务器与 1 台 GPU A * 服务器。服务器分别通过 400G 光纤接入 RoCE 交换机、25G 光纤接入业务交换机，BMC 管理口接入千兆带外交换机，实现算力、业务、管理三网流量隔离，优化网络性能、可靠性与运维安全。

计算平面

计算平面为异构 PD 推理集群高性能算力互联平面，承载 Prefill、Decode 节点及 GPU 服务器间高速数据交互，支撑节点协同计算、KV Cache 与中间数据传输，对带宽、时延要求严苛。本方案采用 400G

RoCE 网络搭建计算平面，服务器经由 400G 光纤接入 RoCE 交换机。平面独立承载算力通信流量，不受业务、管理流量干扰，保障推理通信低时延、高吞吐。

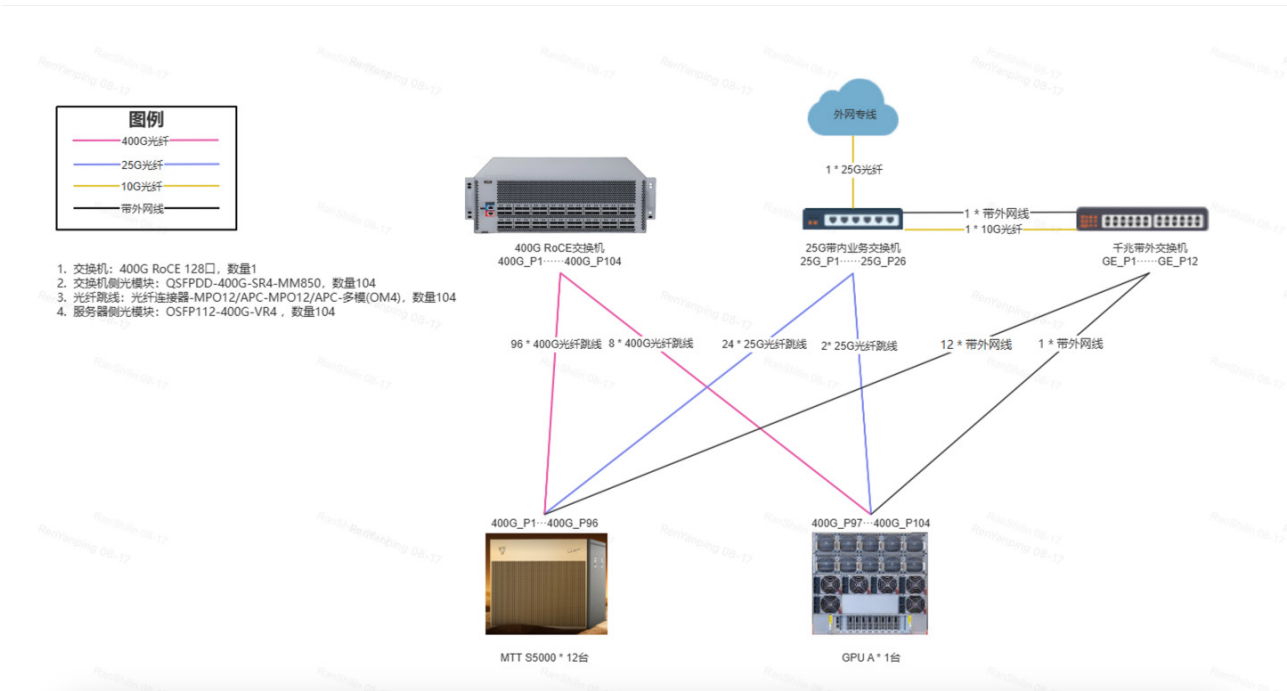
业务平面

业务平面承载集群内外业务访问、服务调度、南北向与常规东西向业务流量，对接调度平台、外部用户网络，是集群对外提供推理服务的核心通道。方案使用 25G 交换机组建业务平面，服务器通过 25G 光纤接入并连通外网。业务平面与计算平面物理隔离，业务流量不占用 400G 算力网络资源，保障算力网络稳定，便于业务流量管控。

带外管理平面

带外管理平面为独立运维网络，对接服务器 BMC 与设备管理接口，不承载业务、计算流量，专供远程开关机、硬件监控、故障排查、固件升级等运维应急操作。集群服务器同时接入三大网络平面，实现算力通信、业务交互、设备运维流量相互隔离。

各类服务器分别接入三个网络平面：通过 400G 光纤接入 RoCE 交换机，构建高速计算互连网络；通过 25G 光纤接入业务交换机，承载业务通信与外网访问；通过 BMC 带外管理接口接入千兆带外管理交换机，形成独立运维管理网络。



第四章

性能与单位算力收入测算

本章以智谱 GLM-5.2 为测试模型，评估 MTT S5000 在 PD 分离架构下提供 Prefill as a Service 的性能表现与收益水平。

4.1 测试方案

本测试基于同构 PD 分离架构，以 2 台 × 8 卡 MTT S5000（PP2/CP8/EP8）执行 Prefill、4 台 × 8 卡 MTT S5000（EP32/DP32）执行 Decode，通过 Mooncake Transfer Engine 经 RDMA 直传 KV Cache，基于 GLM-5.2-FP8 与 FP8_E4M3 KV Cache 精度，使用 vLLM bench serve 在 64K–400K 输入长度下压测。测试中 Decode 侧设置 output_token=1，使 Decode 计算开销趋近于零，从而将测量焦点收敛至 Prefill 阶段；在长上下文场景下，Prefill 计算耗时占 TTFT 的绝对主导地位，Decode 节点的硬件选型对 Prefill 性能基线的影响可忽略不计。每 Case 前清空 Prefix Cache 以确保冷启动，并执行预热请求直至系统达到稳态；最终输出总 Token 吞吐及 TTFT（Mean/P50/P90/P99），验证 P50 TTFT ≤ 6000ms SLO 下的长上下文 Prefill 性能基线。测试环境及核心参数详见附录《GLM-5.2-FP8 P2D4 Prefill-Only 实测环境及工具描述》。

4.2 测试 SLO 与定价口径

测试覆盖 64K 至 400K 五档输入长度，Prefix Cache 命中率 90%，输出 1 Token（仅评估 Prefill 阶段）的条件下，要求各档输入长度均满足 P50 TTFT ≤ 6,000 ms 的硬性服务等级目标。测试参数汇总见表 4-1。

参数	取值
输入长度	64K / 128K / 200K / 300K / 400K
输出长度	1 Token (仅评估 Prefill 阶段)
Prefix Cache 命中率	90%
P50 TTFT	≤ 6,000 ms
部署方案	P2D4 (2 台 S5000 Prefill + 4 台 Decode)

表 4-1 测试 SLO 参数

定价以 2026 年 8 月 20 日智谱官网 API 价格为基准，输入 Token 与缓存命中 Token 分别计价，见表 4-2。

上下文	输入单价 (百万 Tokens)	缓存命中 (百万 Tokens)
1M	8 元	2 元

表 4-2 定价口径（2026 年 8 月 20 日智谱官网 API 定价）

4.3 Prefill as a Service 性能实测数据

在上述 SLO 条件下，8 卡 MTT S5000 单服务器 Prefill as a Service 实测性能如表 4-3。

输入长度	P50 TTFT (ms)	Total Prefill (tok/s)	TPM (MTokens/分钟)	Cache Miss (tok/s)	Cache Hit (tok/s)
64K	5,705	95,920.5	5.8	9,592.1	86328.5
128K	5,677	90,177.5	5.4	9,017.8	81159.8
200K	5,416	75,028	4.5	7,502.8	67525.2
300K	4,952	58,412	3.5	5,841.2	52570.8
400K	5,088	44,206	2.6	4,420.6	39785.4

表 4-3 Prefill as a Service 性能实测数据（单台 8 卡 MTT S5000 服务器）

需要说明的是，表 4-3 以 8 卡 S5000 为统计口径，取各档输入长度下 P50 TTFT 不超过 6000 ms 对应最优 Total Prefill [tok/s] 作为对应档位的 Prefill 性能评估基线；表中数据实测环境详见第六章附录《GLM-5.2-FP8 P2D4 Prefill-Only 实测环境及工具》。

4.4 收入测算模型

基于 Prefill-only 性能数据构建收入测算模型。

Prefill 侧按实际处理的 Token 数量计费，缓存未命中按输入单价 $p_m = 8$ 元 /MTtokens 计，缓存命中按 $p_h = 2$ 元 /MTtokens 计。

设 Total Prefill 吞吐为 T (tok/s)，Prefix Cache 命中率为 r ($0 \leq r \leq 1$)，按 30 天 / 月计，每月 MTtokens 收入为：

$$R_m = 2.592 \times T \times [(1 - r) \cdot p_m + r \cdot p_h] \tag{1}$$

其中系数 $2.592 = 86,400 \times 30 \div 10^6$ ，用于将 tok/s 折算为 MTokens/ 月。方括号内为考虑命中率后的综合单价；代入 $r = 0.90$ ，得综合单价 $0.1 \times 8 + 0.9 \times 2 = 2.6$ 元 /MTokens。

4.5 单台服务器月收入测算

若单台 S5000 持续运行于单一上下文档位，按式 [1] 计算的满载月收入见表 4-4。

输入长度	月处理量 (MTokens/月)	月收入 (满载, 元)
64K	248,625.9	646,427.43
128K	233,740.1	607,724.21
200K	194,472.6	505,628.70
300K	151,403.9	393,650.15
400K	114,582.0	297,913.08

表 4-4 分档满载月收入（单台 S5000，利用率 100%）

实际业务为多种上下文长度的混合负载。假设典型 Agent 业务的上下文分布——P50 约 4K~64K Token（短任务，约占 50%）、P50~P90 约 64K~200K（多步自主循环，约占 40%）、P90~P99 约 200K~300K（长时自主运行，约占 9%）、P99+ 为 400K 以上（复杂代码 Agent，约占 1%）——分别映射至 64K / 200K / 300K / 400K 性能档加权，参考表 4-3 Tokens/s 数据，得到每秒业务等效吞吐：

$$T = 0.5 \times 95920.5 + 0.4 \times 75028 + 0.09 \times 58412 + 0.01 \times 44206 \approx 83,670.6 \text{ tok/s} \quad [2]$$

$$TPM = 83,670.6 \times 60 / 1000000 = 5 \text{ MTokens/min}$$

代入式 [1]，单台 S5000 满载月收入为：

$$R_m = 83,670.6 \times 2.6 \times 2.592 \approx 563,872.9 \text{ 元 / 月} \quad [3]$$

即单台 S5000 TPM 达到 5 MTokens，满载月收入约 56.4 万元。

上述结果为满载理论值。需要指出，式 [2] 为理想上界。在 P/D 分离部署下，KV Cache 跨节点传输的带宽占用与在途显存锁定、混合长度请求的调度效率损失、以及 Prefix Cache 在多节点间的命中率衰减等因素，均会使实际 Prefill 吞吐低于该值；具体折减幅度需结合实际业务流量特征与部署拓扑在实测中确定。

基于式（2）考虑实际负载波动，不同利用率下的月处理量与月收入见表 4-5。

利用率	TPM (MTokens/分钟)	月收入(万元)
100%	5	56.4 万元
70%	3.5	39.5 万元
50%	2.5	28.2 万元
40%	2	22.6 万元

表 4-5 不同服务器负载利用率下单台 8 卡 S5000 月收入估算（混合上下文场景）

上述收入按理想吞吐测算。在实际 P/D 分离的生产环境中，Prefill 节点吞吐能否逼近理论上限，取决于 Prefill-Decode 全局调度协同、KV Cache 传输链路（网络带宽、传输协议、在途显存管理）、Prefix Cache 跨节点共享策略、以及流量整形与 SLO 感知的并发控制等系统级联合优化；任一环节未充分调优，均可能导致实际吞吐与收入低于上表估计。

4.6 小结

综合上述测试与测算，MTT S5000 在 Prefill 场景下，于满足 6 秒 SLO 硬约束的前提下，把长上下文的 Prefill 转化为可稳定计费、可预测创收的服务能力——这是其单位算力收入优势最直接的体现。

从性能看，单台 S5000 在 64K 输入下，TPM 可以达到 5.8MTokens 的 Prefill 吞吐水平，即便延伸至 400K 长上下文，时延依然平稳压在与短序列几乎同等的水平。此处值得注意的不是某个峰值，而是整个长度区间内“吞吐可观、时延不失控”的一致性——它意味着运营商无需为长尾业务预留过度缓冲，硬件投入得以接近可预期的确定性产出。

从经济账看，在假定 agent 场景混合上下文场景下，单台 S5000 满载月收入约 56 万元，即便利用率回落到 40%，单台名义月收入仍约 23 万元。需要说明的是，本章测算尚未纳入设备折旧、功耗、机房与运维等成本项，完整的 TCO 结论需在补齐成本侧数据后得出。

在 Prefill as a Service 计算池方案中，MTT S5000 以匹配负载画像的硬件，兑现了可量化、可预期的收入回报。

第五章

总结

过去两年，生成式 AI 的推理负载发生了结构性迁移。以 Agent、AI Coding 与长文档分析为代表的新场景，将请求的输入长度推升至数万乃至数十万 Token。这一变化的意义，远不止于“上下文变长了”——它从根本上改变了推理两个阶段的资源权重配比。

在传统混部架构下，Prefill 与 Decode 共享同一 GPU。这种设计在短请求时代是自洽的：输入数百 Token 的 Prefill，与 Decode 之间不存在实质性的资源竞争，调度器足以在两者之间平滑切换。但当输入长度突破某一阈值后，这一前提被彻底打破。长 Prefill 是一个持续数秒的计算密集任务，它每占用一个时间片，就有一批在途请求的 Decode 被迫等待。其结果不是吞吐下降，而是尾延迟的离散化恶化——P50 纹丝不动，P99 持续拉长。更关键的是，这一恶化无法通过增加同构 GPU 解决：Prefill 受算力约束，Decode 受带宽约束，二者对硬件的诉求背道而驰，同构扩容只会加剧错配，而非弥合它。

因此，我们的判断是：Prefill 与 Decode 的分离，不仅仅是一项调度优化，而是推理基础设施的架构演进。在可预见的周期内，长上下文将成为推理负载的主流形态，而将两个物理特性迥异的阶段强行塞进同构资源池，将构成推理成本与延迟的双重天花板。异构 PD 分离——以算力型硬件承载 Prefill、以带宽容量型硬件承载 Decode——是破解这一天天花板的一种可行的结构性路径。

在这一框架下，MTT S5000 是 Prefill 侧算力供给的理想选择。作为一款通用 AI 加速卡，S5000 同样适用于训练与其他模型推理场景；而其在 Prefill-as-a-Service 场景中的突出优势，基于以下判断：

第一，Prefill 的算力需求是向 FP8 高密度矩阵计算收敛的。长上下文 Prefill 的计算主体是 QKV 投影与自注意力，均为大规模 GEMM，在长上下文与较大 batch 的典型条件下，算术强度通常已越过 roofline 拐点，落在 compute-bound 区间。MTT S5000 提供的高密度 FP8 稠密算力，与这一计算画像高度匹配。实测数据支撑了这一判断：在 64K 输入下，单台 S5000 的 TPM 达 5.8 MTokens/min，P50 TTFT 为 5,705ms——这一吞吐是在满足 6 秒 SLO 约束下、计入 KV 跨节点传输开销后的达标值，而非脱离服务等级的理论峰值。

第二，KV Cache 本身是可以被迁移的资产，而非必须驻留的资源。 这是异构 PD 分离得以成立的关键物理前提。在 PD 分离架构中，Prefill 侧生成的 KV Cache 经跨节点传输后，在 Decode 侧承担后续自回归的访存压力。S5000 以原生 FP8 KV 格式与 Decode 池对齐，使这一传输无需任何格式转换——页粒度直传路径消除了类型转换的带宽与延迟开销。这意味着 Prefill 卡所需的显存容量被显著降低，从而将成本进一步向算力本身收敛。

第三，单位算力收入是这套方案合理性的最终度量。 以 2026 年 8 月 20 日智谱官网 API 定价为基准，在 90% 前缀缓存命中率下，单台 S5000 的满载月收入约为 64.6 万元（64K 档位），折算为单位算力收入约 16.15 万元 / PFLOPS / 月。这一数字的意义，不在于绝对值本身，而在于它建立了一个可对标的坐标系：它将 Prefill 侧的投入产出，从模糊的“吞吐能力”转化为清晰的“每 PFLOPS 的月度经济产出”，使硬件选型决策得以围绕瓶颈展开，而非围绕平均指标展开。

方案的落地可行性，建立在成熟生态之上，而非自研封闭体系。KV Cache 协同层、队列化生命周期管理与 Mooncake 高速传输通路，均是与 SGLang 主线深度集成的开源组件，已经社区与生产环境验证。MUSA 工具链对 CUDA 的源码级兼容，使客户得以基于现有代码库快速迁移，GLM-5.2 的端到端实测已证明这条路径的可复现性。对于一个需要规模化部署的推理服务而言，这一点的重要性不亚于单卡性能本身——它决定了方案能否被纳入既有工程体系，而非成为一座技术孤岛。

基于 MTT S5000 构建的 Prefill as a Service 算力池，使 Prefill 从推理流程中的一个环节升级为可独立供给、独立计费、独立优化的算力服务。对推理服务运营商而言，这套方案的最终意义在于：它提供了一个将 long-context 推理成本结构从“水位随输入长度持续攀升”转变为“算力成本随瓶颈分层收敛”的可执行路径。在长上下文推理即将成为默认工作负载的节点上，这一路径的部署价值，将随着输入长度的每一次量级跃升而愈发显著。

第六章

附录

GLM-5.2-FP8 P2D4 Prefill-Only 实测环境及工具描述

测试环境及核心测试参数

参数	核心配置
CPU / OS	Intel(R) Xeon(R) Gold 6430 / Ubuntu 22.04.4
模型与精度	GLM-5.2-FP8 (ZhipuAI) , KV Cache Dtype=fp8_e4m3
硬件规模	MTT S5000 × 8卡/节点;2 Prefill节点 + 4 Decode节点
驱动版本	3.3.5-server
Prefill并行	PP2, CP8, EP8
Decode并行	EP32, DP32
服务架构	PD分离;Mooncake + RDMA进行KV传输
压测工具	vLLM bench serve v0.24.0;OpenAI Completions API
工作负载	random dataset;max_tokens=1;每点请求数=concurrency × 5
Cache策略	共享前缀目标90%;每点测试前执行flush_cache;warmup=1
统计范围	64K、128K、200K、300K、400K, 共51个有效数据点
输出指标	总Token吞吐、TTFT Mean/P50/P90/P99

Prefill 配置

参数	值
PP 层切分	40,38 (78 层)
PP /CP/ EP	2 / 8 / 8
Attention Backend	DSA (tilelang)
MoE A2A Backend	DeepEP (normal)
KV Cache Dtype	fp8_e4m3
Mem Fraction Static	0.80
Max Running Requests	36
Chunked Prefill Size	8,192
Max Prefill Tokens	524,288 (512K)
Disaggregation Mode	prefill (Mooncake + RDMA)
PP Async Batch Depth	1

Decode 配置

参数	值
EP / DP	32 / 32
MoE A2A Backend	DeepEP (low_latency)
EAGLE Speculative	steps=5, topk=1, draft=6
Disaggregation Mode	decode (Mooncake + RDMA)

压测工具

参数	值
工具	vLLM bench serve v0.24.0
API	OpenAI Completions (/v1/completions)
max_tokens	1(测 Prefill 延迟)
dataset	random
cache 命中率	90% (prefix : unique = 9 : 1)
每点请求数	concurrency × 5
warmup	1 request (--num-warmups 1)
flush_cache	清空服务端的 Prefix Cache, 避免上一测试点的缓存残留影响当点结果。

参考文献：

[1] QIN R, HE W, WANG Y, et al. Prefill-as-a-Service: KVCache of Next-Generation Models Could Go Cross-Datacenter[EB/OL].

[2] University of Michigan, Stanford University, et al. How Do AI Agents Spend Your Money? Analyzing and Predicting Token Consumption in Agentic Coding Tasks. 2026.