异构边缘设备集群上的解耦式大语言模型服务可扩展放置

Truong-T hanh Le†、Hoang-Loc La§、Amir Taherkordi†、Frank Eliassen† 和 Phuong Hoai Ha §

† 信息学系奥斯陆大学挪威奥斯陆{truongl,amirhost,frank}@ifi.uio.no

§ 计算机科学系挪威北极大学挪威特罗姆瑟{hoang.l.la,phuong.hoai.ha}@uit.no

arXiv:2611.32881v1 [cs.DC] 24 Nov 2026

摘要—**在边缘端为大语言模型（LLM）提供服务，对于隐私敏感型应用以及要求数据保留在本地设备上的场景而言，正变得越来越重要。然而，边缘环境受限于有限的计算资源，且设备异构性显著，这使得高效的大语言模型服务面临挑战。在突发需求下，这些限制尤为突出，因为拥塞和队列积压会显著增加响应延迟。将模型复制为多个流水线实例，可以提升服务能力并减少拥塞。然而，如何将设备分组为副本、为副本分配预填充或解码功能，以及如何在异构设备间划分模型层，这些问题共同构成了一个规模庞大的组合优化问题。我们提出 SPADE，这是一种可扩展且运行时高效的放置规划器，用于在异构边缘集群上通过流水线复制实现解耦式大语言模型服务。SPADE 引入了一种基于支配关系的剪枝技术，可丢弃劣质的副本候选项，并递归地剔除包含这些候选项的更大候选项，从而大幅缩小搜索空间。此外，它还结合了用于层划分的注水法，以及有界多维背包问题建模，以在拥塞感知目标下优化放置。在一个由 30 台设备组成、涵盖五种硬件类别的合成设备集群上，SPADE 能在 441 毫秒内找到最优放置方案。相比之下，采用动态规划划分增强的遗传搜索基线需要约两小时，且所得方案的拥塞目标仍差 19.7%。在包含十种设备类型的设备集群上进行评估时，SPADE 的剪枝策略消除了 99.6% 的候选放置方案，同时得到与穷举枚举相同的保留候选集。**

索引词—**可扩展放置规划器、搜索空间剪枝、解耦式大语言模型服务、排队系统**

I. 引言

大语言模型（LLM）正日益从集中式数据中心转向边缘部署 [1], , 其原因是隐私要求禁止组织和个人用户将敏感数据发送至第三方 API[2]。这些部署不再运行于同质化集群，而必须使用可用硬件 [3, 4, 5]，并通过普通局域网连接，而非 NVLink 等高速互连。因此，执行环境在架构、内存容量和通信带宽方面天然具有异构性 [6, 7, 8]。

在此类设备集群上部署现代大语言模型颇具挑战，因为模型规模、激活值和 KV 缓存的使用量往往超过任何单个设备的内存容量 [3, 5]。因此，内存是决定模型可放置位置以及可同时服务多少请求的主要因素之一 [9]。因此，流水线并行是实现大语言模型服务的基础机制 [10, 11, 12]。此外，大语言模型服务本身也具有异构性。预填充 是计算密集型操作，会一次性处理整个提示词；而 解码 是内存密集型操作，会迭代生成词元。这种异构性促使数据中心系统采用大语言模型解耦 [13, 14]。在边缘端，设备差异显著，使解耦面临更大挑战。

边缘设备集群还必须在计算和内存容量有限的情况下服务大量并发请求，因此请求很容易以快于处理能力的速度到达；此时，首令牌时间（TTFT）将主要由拥塞决定，即请求在队列中等待的时间 [15]。部署多个流水线副本，每个副本都是部署在一组设备上的完整大语言模型副本，可以缓解拥塞，但也会将部署转变为一个放置问题：必须在多个模型副本之间分配设备，为每个模型副本分配预填充或解码角色，并将层划分到每个副本的设备上。现有解决方案只处理了该问题的部分内容。Halda [3]和 LOIP [5]针对单请求延迟优化单流水线部署，而 Helix [6] 和 Parallax [4]在不进行预填充–解码角色分配的情况下最大化总吞吐量。E2LLM [9] 与我们的工作最为接近，其目标是通过预填充和解码副本应对拥塞；然而，其目标函数将预填充服务时间与拥塞结合起来，这可能会偏向预填充

即使它无法缩短队列，也能提高容量；其遗传搜索则无法提供最优性保证。当前仍缺少的是一种能够单独衡量拥塞的目标，以及一种能够在设备集群规模下找到最优放置方案的搜索方法。

寻找这一最优解很困难，因为放置问题嵌套了两个子问题。外部问题将设备分配给副本，并为每个副本分配一个阶段；这是一个 NP-hard 问题，对于 N 设备，最多有 2^N 个候选副本 [16, 17], 。内部问题将模型层划分到候选方案的设备上，在内存约束下最大化吞吐量；现有解决方案通过对有序设备序列进行动态规划（DP），在 [9, 10, 11] 中以 O（KL^2）的时间复杂度求解，适用于 K 台设备

以及 L层。由于候选方案的吞吐量只有在求解其内部问题后才能确定，因此，精确最优解需要对指数级数量的候选方案使用内部求解器：在一个包含十种设备类型的30设备集群上，即使使用快速内部求解器，穷举评估也需要九个多小时，而且随着设备集群规模扩大，这一成本会呈指数增长。因此，规划时间成为实现最优性的障碍；现有规划器通过启发式搜索来应对这一问题，但不提供最优性保证，即使用禁忌搜索 [17],进化搜索 [18], 模拟退火 [19], 和遗传算法（GA） [9], ，或者限制问题范围：Parallax [4]将放置分解为连续阶段，而 Helix [6] 和 Halda [3] 的精确公式求解的是层放置（即内部问题），而不是副本构建。使用这些内部求解器也不会降低规划时间，因为每个候选方案都需要求解一个整数规划，这仍然比动态规划的成本更高。

本文提出的新方案 SPADE 避免了这两种折衷。它能精确求解外部问题，且其时间复杂度仅随设备类型数量呈指数增长；在异构边缘设备集群中，设备类型通常较少，因此这比随设备数量呈指数增长更可行。利用同一类型设备可互换这一特点，SPADE 用各类型设备的数量表示一个候选副本，并将基于生成的候选方案进行的分配建模为有界多维背包问题，再通过动态规划精确求解；若内存受限，则使用混合整数规划求解。SPADE 还通过引入支配规则 缩短规划时间：该规则会剔除其设备可分解为吞吐量相等或更高的较小候选方案的候选方案，并跳过包含该候选方案的更大候选方案的生成。由于拆分会复制模型权重，只有在吞吐量受延迟限制而非受内存限制时，分解方案才可能优于其父方案。我们假设包含该父方案的更大候选方案也具有这一限制，因此可以跳过。在所有能够进行穷举的实例中（表 III），这一假设没有排除任何最优候选方案。此外，由于副本流水线只有首端和尾端不同 [9], ，内部问题存在闭式注水解，可将复杂度从 $O(KL^2)$ 降至$O(K)$。总之，我们的贡献如下：

- 一种面向解耦式大语言模型服务的拥塞感知目标，将系统建模为流体串联队列，并将拥塞与预填充服务时间分离（III-A 节）。在一个由八台设备组成的 CUDA–Metal测试平台上，与最先进的目标相比，该目标选出的放置方案可将最坏情况下的等待时间最多降低 2.44× （IV-E 节）。
- 一种可扩展的放置方案，结合了按类型计数的背包问题表述、支配剪枝和闭式水填充分区（第三-B至第三-E节）。在合成的30设备集群上，与穷举相比，支配剪枝评估的候选方案最多减少了99.6%；SPADE 在441毫秒内达到经验证的最优解，而最先进的方法则需要6855秒，且拥塞高出19.7%（IV-D 节）。
- 一个基于 llama.cpp 构建的开源点对点服务框架。与 llama.cpp 和 E2LLM 使用的星型拓扑不同，后者会将所有激活值经由单个节点路由；该框架实现了完全去中心化通信（IV-F 节）。

II. 相关工作

A. 异构设备上的分布式大语言模型推理

将模型划分到多个设备上已是成熟做法。PipeEdge 和 EdgeShard 在内存约束下划分 Transformer 层 [10, 11]；LOIP、Parallax 和 Helix 将放置扩展到异构部署，并考虑通信、调度和优化约束 [5, 4, 6]；prima.cpp（Halda）则通过整数规划放置和磁盘卸载，面向消费级硬件 [3]。LOIP 和 prima.cpp 针对单流水线部署优化 TPOT 和延迟等单请求指标，而 Parallax 和 Helix 则最大化总吞吐量。然而，这些系统均未将拥塞明确建模为放置目标，也没有为不同副本分配不同的预填充和解码角色。E2LLM 虽然在异构边缘设备集群上实例化了多个预填充和解码副本 [9], ，但其目标将拥塞与预填充服务时间耦合在一起。我们还发现，prima.cpp 的多轮 SSD–CPU–GPU 流水线仅在 CPU 执行时有效；GPU 执行会退化为单轮、单微批次，从而消除了该设计所要实现的流水线和重叠执行（IV 节）。

B. 预填充–解码解耦

预填充–解码解耦如今已成为大语言模型服务中的常见实践。Splitwise 将两个阶段分别部署到专用机器池上 [13], ，DistServe 联合优化二者的放置与并行度 [14],，而 HexGen-2 则将这一方法扩展到异构 GPU 集群 [16]。近期的研究通过工作负载感知的路由和分析式资源配置进一步改进了资源分配 [15, 20]。因此，我们采用解耦方案，而不是重新引入这一方案。关键区别在于，现有系统主要将角色分配视为资源配置决策；而异构边缘设备集群中的设备在计算性能和内存容量方面存在显著差异，而这恰恰是预填充和解码分别最为依赖的资源。因此，将设备分配给某一阶段本身就成为部署位置问题的一部分。

C. 大语言模型服务中的排队与竞争

随着大语言模型服务系统日趋成熟，关注重点已从执行延迟转向负载下出现的拥塞。分析模型在首令牌时间目标 [20]，下配置预填充和解码容量，而测量研究表明，在高负载设备集群中，排队延迟和KV缓存传输主导了尾部首令牌时间 [15]。后续工作通过调度来应对拥塞，包括 Peace [21] 和APEX[22],，但基本将设备异构性视为与优化正交的因素。E2LLM将放置优化扩展到异构边缘设备集群，并支持多个副本 [9],，但通过结合预填充服务时间与

瓶颈阶段拥塞来估计延迟。正如第I节所述，这种方法将服务成本与排队效应耦合起来，并可能在不减少拥塞的情况下，通过增加预填充容量获得更优结果。我们的工作同样立足于边缘端服务场景，但将预填充和解码建模为流体串联队列，将拥塞与服务成本隔离开来，并围绕由此产生的排队行为直接优化放置。

D. 多集群放置问题

一旦能够同时部署多个大语言模型副本，放置就会成为一个组合问题，因为必须在各个副本之间分配设备。选择哪些设备组成一个副本，本质上是一个具有指数级搜索空间的聚类问题，而 ThunderServe 明确将其认定为 NP-hard [17]。因此，现有系统通常依赖启发式搜索来应对这种复杂度。HexGen 通过进化搜索构建非对称群组 [18]；HexGen-2 [16] 则用谱划分取代进化搜索，并通过 Kernighan–Lin 对其进行优化；随后利用最大流公式评估每个候选方案，从而为每个群组选择并行策略和 KV缓存通信路径，并进一步通过最大流引导的边交换改进方案。HexGen-3 采用模拟退火这一元启发式优化算法来搜索部署计划 [19]。ThunderServe 采用禁忌搜索 [17], , 而 E2LLM 使用遗传算法（GA） [9]。这些方法显著降低了搜索成本，并能够扩展到更大规模的部署，但无法提供最优性保证。

人们也对精确优化进行了研究。Helix 将放置建模为最大流混合整数线性规划 [6]，而 Haldadecomposes 则针对层分配建立整数规划 [3]。然而，这些方法优化的是流水线内部的层放置，而不是从异构设备集群中选择并组合多个副本这一更高层次的问题。此外，难点不在于表达优化问题，而在于评估其生成的大量候选方案。在将一个副本与另一个副本进行比较之前，必须先通过求解内部的层划分问题来估计其吞吐量。因此，副本评估的成本会迅速成为规划时间的主要部分（见 IV-D 节）。所以，降低这一成本是确保规划器可行性和可扩展性的关键。现有工作因此往往将精确性与可处理性视为相互竞争的目标。我们的方法则在外层问题中实现了二者的统一。第三节介绍的支配规则消除了评估绝大多数候选方案的必要，从而移除了主要的计算瓶颈，使设备集群规模下的放置优化变得可处理。

III. 所提方法

图 1 概述了 SPADE。首先，SPADE 为外部问题构建候选方案列表，并使用注水公式求解内部问题（第 III-B 节）。此外，SPADE 在构建列表时应用支配规则，以剔除前景不佳的候选方案，从而缩小搜索空间（第 III-D 节）。最后，它基于拥塞模型求解外部问题（第 III-E 节）该模型描述了解耦式预填充/解码，以确定使模型化拥塞最小的放置方案（第 III-A 节）。

A. 预填充/解码部署的拥塞模型

我们对一种解耦部署进行建模，该部署至少包含一个预填充副本和一个解码器副本。每个请求都必须先经过预填充阶段，然后才能转发至解码器，因此该部署构成一个两阶段串联队列，而非两个相互独立的服务。设 PS 和 DS 分别表示部署中预填充和解码的总吞吐量（每秒令牌数）， N_P 表示每个请求的平均输入令牌数， N_D 表示平均生成令牌数， T_a 表示请求之间的平均到达间隔。总吞吐量可根据第 III-B 节所述的分区方案直接计算，即对分配到各阶段的所有副本进行求和。我们定义

$$t_P = N_P/PS, \qquad t_D = N_D/DS \tag{1}$$

分别作为预填充阶段和解码器阶段的服务时间。当请求到达速度快于预填充阶段的处理速度时，预填充阶段（C_P）就会出现拥塞；也就是说，当预填充时间 t_P 超过平均到达间隔时，就会发生拥塞。

$$C_P = \max(0,\ t_P - T_a)\,. \tag{2}$$

解码器阶段经历着不同的到达过程。一旦预填充达到饱和，请求离开第一阶段的速度就不会快于预填充服务速率。因此，解码器不会直接观察到外部到达间隔 T_a。相反，请求到达的间隔

$$T_D = \max(T_a,\ t_P)\,, \tag{3}$$

反映了原始请求流或饱和预填充阶段的受限输出。因此，若以 T_a 衡量解码器拥塞，就会将延迟重复计算。解码器拥塞 C_D 实际上取决于其服务时间超出实际工作到达周期的部分：

$$C_D = \max(0,\ t_D - T_D)\,. \tag{4}$$

一旦形成队列，后续请求所经历的延迟就会不断累积。每个到达的请求都会继承先前请求造成的积压，导致整个突发期间的拥塞呈线性增长。SPADE 会选择使 $W = C_P + C_D$最小化的放置方案。拥塞可能发生在一个阶段，也可能同时发生在两个阶段。对于前一种情况，在拥塞阶段分配更多资源会有所帮助。对于后一种情况，根据公式 2–4，当 $t_D > t_P > T_a$ 时，两个阶段都会发生拥塞。在这种情况下，解码器是主要瓶颈：无论预填充阶段是否拥塞，提高解码器吞吐量都能减少解码器拥塞和总等待时间。因此，相较于提高预填充容量，向解码器分配更多资源通常更有利。

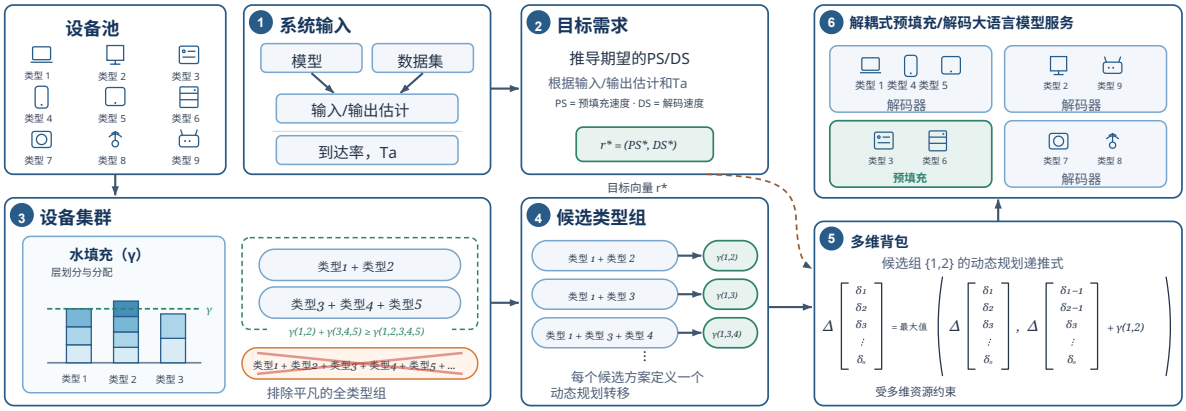

图 1：SPADE 系统　概览。 γ：采用层划分后的总体吞吐量。 δ_i：该类型的设备数量　　i

B. 使用注水法进行模型划分

在评估放置方案之前，我们必须确定 Transformer 的层如何分布在各个副本上，并估算每个副本能够维持的吞吐量。流水线副本是部署在一部分设备上的大语言模型完整副本。分配给它的设备共同承载全部L Transformer 层，并以流水线并行环的方式执行这些层。设备 i存储一个由 ℓ_i 层组成的连续块：

$$\sum_i \ell_i = L, \qquad \ell_i \geq \ell_{\min}, \tag{5}$$

这将分配给一个副本的设备数量限制为最多

$$K_{\max} = \left\lfloor \frac{L}{\ell_{\min}} \right\rfloor \tag{6}$$

与集中式部署不同，流水线序列中的位置并不可互换。首设备承载嵌入表，尾设备承载最终归一化和输出投影，因此固定内存开销$c_i(r)$和固定阶段延迟成本 $\alpha_i(r)$均取决于设备角色$r \in \{$首、中、尾$\}$。因此，一个包含 K 个设备的副本有$K(K-1)$种不同的首尾设备分配 [9], ，每种分配都会产生不同的最优划分。我们枚举这些分配，同时将中间设备视为可互换设备。

a) 内存限制： 副本吞吐量面临的第一个约束是内存。每台设备都必须存储其所分配层的权重，以及与所有活动请求关联的键值缓存。如果一个副本服务 s个并发请求，则角色为 r 的设备 i 必须满足

$$\ell_i w + s\,\ell_i \kappa + c_i(r) \leq M_i, \tag{7}$$

其中， w 表示每层的权重占用， κ 表示每个请求的每层键值占用， M_i 表示设备的可用内存容量。式（7）揭示了一个直接的权衡：为设备分配更多层，会同时增加其权重占用和其分摊的键值缓存，从而减少该设备可支持的并发请求数量。重新整理该约束可得

$$s \leq \frac{(M_i - c_i(r))/\ell_i - w}{\kappa},$$

这是设备 i 可容纳的最大请求数。由于该数值会随 ℓ_i 增大而减小，副本的容量最终取决于内存最受限的设备。因此，层分配不均会导致部分设备的内存闲置，同时限制其他设备的并发度。

最优划分通过在各设备之间均衡内存限制的槽位数（s_{mem}）来避免这种不平衡。这将产生一个注水解，其中 $\ell_i \propto M_i - c_i(r)$，因此有效内存预算较大的设备会按比例获得更多层。将这一关系与约束$\sum_i \ell_i = L$ 结合可得

$$s_{\text{mem}}(S) = \frac{\widetilde{M}(S) - wL}{\kappa L}, \qquad \widetilde{M}(S) = \sum_{i \in S} \big(M_i - c_i(r_i)\big) \tag{8}$$

其中， $\widetilde{M}(S)$表示扣除特定角色开销后副本可用的总内存。式 8 描述的是一个内存容量上界，而不是规划器最终选定的分区。它回答了一个更简单的问题：如果内存是唯一的限制因素，该副本能够支持多少请求？III-D节利用这一上界比较副本，并找出可以安全舍弃的候选方案。

b) 并发限制：内存决定了副本可以容纳多少请求，但最大并发度最终由吞吐量决定。随着并发度增加，可分配给每个请求的吞吐量会下降，因为可用资源需要由更多正在处理的请求共享。因此，只有在每个请求的吞吐量仍高于部署目标 τ（即每秒最低令牌数）时，我们才会接纳请求。令 $\rho(S, s)$ 表示候选方案 S 在处理 s个请求时达到的每请求吞吐量。因此，允许的最大并发度为

$$s_{con} = \max\{s : \rho(S, s) \geq \tau\}, \tag{9}$$

因此，最终可实现的服务请求数取决于哪个约束最先成为活动约束。副本可能在吞吐量降至 τ 以下之前耗尽其内存，或者在内存仍有剩余时达到吞吐量上限。在这两种

情况下，活动约束决定了评估副本时所采用的并发度。候选方案 S 的总体吞吐量为

$$\gamma(S) = \rho\big(S, s\big)\, s, \quad s = \min\{s_{mem}, s_{con}\} \tag{10}$$

c) 划分：现有规划器 [10, 11] 通过对有序设备序列进行动态规划，得到层分配。我们发现，这种分配具有简单得多的结构。吞吐量由最慢的流水线阶段决定，因此，任何导致一台设备比其他设备更慢的分配方案，都可以通过将层从该设备转移出去加以改进。因此，在最优解下，所有阶段都以相同的速率推进。

我们将设备 i的执行时间建模为其所分配层数的仿射函数

$$t_i(\ell) = \alpha_i(r_i) + \beta_i \ell_i, \tag{11}$$

其中， $\alpha_i(r)$表示与角色 r_i 相关的固定每令牌延迟成本，而 β_i 是该设备测得的每层延迟成本。由于 t_i 随ℓ_i单调递增，最大化吞吐量等价于最小化最慢阶段的时间；当所有设备达到共同的阶段时间 $t^\star$时，该时间最小。因此，层分配满足$\ell_i = (t^\star - \alpha_i(r))/(\beta_i)$。将此条件与守恒约束 $\sum_i \ell_i = L$ 结合，可得

$$L = \sum_i \big(\frac{t^\star - \alpha_i(r_i)}{\beta_i}\big) \Rightarrow t^\star = \frac{L + \sum_i \alpha_i(r_i)/\beta_i}{\sum_i 1/\beta_i} \tag{12}$$

因此，划分可以通过闭式形式得出。已知 $t^\star$ 后，便可直接计算各层的数量，随后对其进行取整和修正，以同时满足 $\ell_{\min}$ 以及式 7 的内存约束。由于首部和尾部设备产生的固定开销不同， $\alpha_i(r)$ 和 $c_i(r)$ 都取决于所选的角色分配。因此，对于 $K(K-1)$ 个可行的首尾设备分配，需要分别求解一次式 12，并保留吞吐量最高的分配方案。

d) 复杂度：对于含有 K 台设备的副本，水填充法可在 $O(K)$ 时间内求解式 12。动态规划则通过考察每台设备处的每个层边界，需要 $O(KL^2)$ 时间。枚举 $K(K-1)$ 种首尾设备分配后，水填充法的复杂度为 $O(K^3)$，而动态规划的复杂度为 $O(K^3L^2)$，具有 L^2 的优势；由于 $L \gg K$，这一优势在实践中意义显著。

C. 副本的类型计数表示

使外层搜索空间易于管理的关键观察是：硬件类型相同的设备可以互换。具有相同类型的设备拥有相同的内存预算和相同的实测性能特征，因此，任意两个副本只要从各类型中选取的设备数量相同，其吞吐量估计和得分就完全相同。规划器无需区分单个设备。

由于副本完全由分配给它的设备决定，因此我们使用类型计数向量 $\mathbf{c} = (c_1, \ldots, c_t)$ 表示副本的候选设备分配，其中 c_t 表示从类型 t 中选取的设备数量， $0 \le c_t \le m_t$表示该类型的可用设备数量。因此，搜索从单个设备子集转向设备类型配置。这种表示方式将候选方案的数量从一个包含 N台设备的设备集群的 $2^N - 1$ 个非空子集减少为

$$\prod_{t=1}^{T} (m_t + 1) - 1, \tag{13}$$

其中统计非空类型计数向量的数量。按构造方式，包含超过 $K_{\max}$ 个设备的候选方案不可行，无需评估即可丢弃。

D. 搜索空间剪枝

即使经过公式（13）的缩减，搜索空间仍然很大。因此，我们识别被支配的候选方案。如果一个候选方案可以分解为有效的子候选项，且这些子候选项的总吞吐量不小于其自身吞吐量，则该候选方案被支配。此类候选方案可以丢弃。支配关系根据此前生成的候选方案递归评估，从而避免显式枚举所有分解方式。

a) 丢弃被支配的候选方案不会造成损失： 将被支配的候选方案替换为其最佳分解中的可行副本，能够保持可行性，因为二者消耗相同的设备集群资源。由于该分解在候选方案所承担的角色中提供了至少同等的总吞吐量，且目标仅取决于这些总量，因此替换不会增加拥塞。因此，存在一种不包含被支配候选方案的最优放置方案，可以在不损失最优性的情况下将其移除。该论证不适用于 TPOT 等逐请求指标。

b) 被支配候选项受限于延迟，而非内存：支配关系本身节省的是搜索，而非评估，因为在检验支配关系之前，必须先知道候选方案的吞吐量。关键在于内存账本。每个副本都会存储一份完整的模型权重，因此将设备集合 $\mathbf{a}$分解为 $\mathbf{b} + \mathbf{c}$需要支付两次权重开销。根据公式 8，

$$s_{\text{mem}}(\mathbf{b}) + s_{\text{mem}}(\mathbf{c}) = s_{\text{mem}}(\mathbf{a}) - w/\kappa. \tag{14}$$

因此，与其取代的候选方案相比，这种分解一开始拥有更少的槽位。如果它仍能实现更高的吞吐量，那么这种提升必然来自更高的服务速率，而非额外的内存。由此，支配关系可以识别出吞吐量受延迟而非内存限制的候选方案。

c) 将规则扩展至更大的候选方案： 丢弃被支配候选项可以节省搜索时间，但无法节省评估时间，因为必须先知道其取值才能判断支配关系。真正的收益在于支配关系还能扩展到更大的候选方案。设将一组设备 d 添加到 $\mathbf{a}$，并假设改为将同一设备放入其支配分解的某一部分中。被支配候选方案的规模已经超过了增加内存还能转化为有效并发度的临界点。因此，再添加一台设备主要只会增加一个阶段，而较小的副本仍能将新增内存转化为吞吐量。因此，与已被支配的候选方案相比，这台额外设备在分解方案中更有价值。

由于成本取决于角色、设备具有异构性，而且分解方案与其父候选方案不一定处于相同运行区间，要给出一般性证明会复杂得多。因此，我们将这一扩展作为候选项生成过程中使用的一项假设，而非定理。尽管违反该假设可能会使最优性保证失效，但实验结果表明，与现有方法相比，由此产生的规划器仍能生成高质量解，同时显著缩短求解时间。该假设只影响生成哪些候选方案。外层求解器对这些候选方案仍会进行精确求解，且移除被支配候选项本身不会造成损失（见 III-D.a 节）。这一扩展只是避免生成那些已被证明不具竞争力的候选方案的后代。在 IV 节中，我们将展示，只要穷举比较的开销可接受，穷举枚举与支配剪枝就会得到相同的幸存者集合、帕累托前沿和最终放置方案。

E. 放置方法

剩余候选方案构成一个**有界多维背包**问题：在每种类型的可用设备数量限制下，选择每个剩余类型计数向量要实例化多少个副本，以及将其分配到哪个角色。由于目标 W仅通过 (PS, DS) 取决于放置，问题便归结为枚举可实现吞吐量对的帕累托前沿，并在该前沿上计算 W 。

我们考虑两种精确公式。第一种是多维背包动态规划 [23] ，其状态为设备消耗向量 $\mathbf{u} \leq (m_1, \ldots, m_T)$。每次转移都会在两个角色之一中加入一个剩余候选方案，并更新可达的 (PS, DS) 前沿。这种动态规划不同于分层分区动态规划。第二种是基于副本数量 $x^r_{\mathbf{c}}$ 的混合整数规划：

$$\begin{aligned} \min \quad & W(PS, DS) \\ \text{s.t.} \quad & \sum_r \sum_{\mathbf{c}} x^r_{\mathbf{c}}\, c_t \leq m_t \quad \forall t, \qquad (15) \\ & PS = \sum_{\mathbf{c}} x^P_{\mathbf{c}}\, \gamma_P(\mathbf{c}), \qquad DS = \sum_{\mathbf{c}} x^D_{\mathbf{c}}\, \gamma_D(\mathbf{c}). \quad (16) \end{aligned}$$

这两种公式在时间与内存之间有所权衡。动态规划速度更快，因为它通过一次前向遍历构建整个帕累托前沿，并且在构造上保持精确。不过，其速度是以工作内存为代价的：状态表包含 $\prod_t (m_t + 1)$ 个条目，因此会随着设备类型数量的增加而呈指数增长。对于异构边缘设备集群而言，这通常可以接受，因为即使设备数量很多，设备类型的数量仍然较少；但最终，内存会成为限制因素。

混合整数规划采取了相反的权衡。由于完全避免使用状态表，其内存占用会随着类型数量增加而保持适中。然而，代价是运行时间：必须通过反复求解来恢复帕累托前沿，每次求解都对应于 $(PS,\ DS)$ 的一种不同标量化。因此，只要动态规划的状态空间能够轻松放入内存，我们就使用动态规划；一旦内存成为瓶颈，就改用混合整数规划。第 IV-D 节报告了我们各项部署中的内存交叉点。

表 I：设备信息。（U.M：统一内存）

设备名称	VRAM	RAM	U.M	类型标识	设备数量
RTX5070 Ryzen 7 9700X	12GB	64GB	No	类型1	1
RTX3060M Ryzen 5 5800H	6GB	64GB	No	类型2	1
Apple M2 Max	22GB	32GB	Yes	类型3	2
Apple M1	12GB	16GB	Yes	类型4	2
DGX Spark*	12GB	128GB	Yes	类型5	2

表II：输入词元和生成令牌的平均数量。

模型	数据集	N_P	N_D	比率
DeepseekR1	LZByteDance	576	579	0.99
	雅思	2282	1173	1.94
Qwen3MoE Q4_K_M	LZByteDance	576	746	0.77
	雅思	2291	1957	1.17

IV. 评估与讨论

我们的评估分为四个步骤。首先，我们考察现有解决方案中哪些可以在我们的部署场景中作为有意义的基线，并解释为何其他方案无法直接进行比较（IV-B 节）。随后，我们验证支配剪枝所依赖的单副本成本模型（IV-C 节）。接下来，我们将 SPADE 与所选基线的部署规划器在解的质量和规划时间方面进行比较，同时隔离支配规则的贡献（IV-D 节）。最后，我们研究随着模型、工作负载、到达率和设备集群规模的变化，排队感知的目标如何影响最终的部署（IV-E 节）。

A. 实验设置

我们的测试平台由八台设备组成，涵盖五种硬件类型，并通过局域网连接（表 I）。为模拟资源受限的边缘设备，我们将 DGX Spark 的显存限制为 12GB。遵循 E2LLM [9], ，我们在局域网（约 900Mbps）上部署测试平台，以减少传输时间对流水线并行性的影响。我们评估了两个具有对比性的模型：DeepSeek R1 Distill Qwen 14B Q6 K 和 Qwen3-30B-A3B-Q4 K M,– 分别称为 – – *DeepSeekR1* 和*Qwen3MoE*。DeepSeekR1 存储 11.3GB 的权重，但通过稠密前馈神经网络 [24], 激活全部 14B 个参数；而 Qwen3MoE 存储 17.3GB 的权重，但通过混合专家架构 [25]仅激活约 3B 个参数。因此，这两个模型分别代表受延迟约束和受内存约束的部署场景。工作负载取自 LZByteDance [26] 和雅思 [27] 数据集，二者共同代表表 II 中所示的不同 N_P/N_D 比率。此外，服务目标设定为每个请求 $\tau = 15$ 词元/秒，是人类阅读速度平均值的两倍 [28]。请求到达间隔服从正态分布，均值为 T_a ，标准差为 $\sigma = 0.1T_a$，并在零处截断。

B. 所选基线

无法在此场景下执行的系统。一些已发表的系统无法直接在我们的测试平台上进行评估。

PipeEdge 面向不使用键值缓存的编码器模型，其划分成本模型无法扩展到自回归解码。此外，HexGen、HexGen-2、HexGen-3、ThunderServe 和 Splitwise 依赖 CUDA 和 NCCL，而 Mac 设备以及通过局域网连接的边缘设备集群不支持这些组件。因此，我们无法与它们的完整实现进行比较。相反，正如下文所述，我们将其规划器和目标纳入比较。

在我们的规划器中评估的目标。放置目标和放置搜索解决的是不同的问题，将两者放在一起评估会混淆各自的影响。因此，我们在自己的规划器中评估以下放置目标：完成时间（来自 HexGen[18] -微调）、角色固定的解聚（来自 Splitwise [13]，称为 Adapted Splitwise - SPW），以及 E2LLM 中的拥塞目标。对于 SPW，我们根据 Splitwise 报告的部署选择制定角色固定策略：始终将吞吐量较高的资源分配给预填充，将吞吐量较低的资源分配给解码。因此，最终放置方案的差异可归因于目标本身，而不是实现方式的差异。因此，应将这些实验视为目标层消融实验。

直接比较的搜索方法。为隔离搜索算法的影响，我们比较了三种方案：遗传算法（GA）与我们的注水式分区器结合、GA 与 E2LLM[9]，中使用的分层分区动态规划结合，以及 ThunderServe [17] 采用的禁忌搜索。所有方法均采用其作者报告的参数运行；我们未针对自己的设备集群对搜索方法进行专门调优。

此外，prima.cpp [3] 是已发表系统中最接近在我们的硬件上运行的系统。Halda 作为其规划器，依赖流水线环并行（PRP）；在这种并行方式中，权重传输会与计算重叠，并跨越多个执行轮次（k >1）。我们使用 GPU 后端重新构建 prima.cpp，并在我们的设备上复现 Halda 的参数选择后发现，该机制在我们的设置中从未被激活。要让 PRP 隐藏存储延迟，设备的计算速率必须满足

$$rate \leq (N-1)\, I\, disk_bw \tag{17}$$

其中，I 表示模型的算术强度，$disk\ bw$ 表示存储带宽。对于其原始评估中使用的 Llama-3-70B Q4 K_M 模型，在四设备部署（$I = 3.32$，SSD $diskbw = 6GB/s$）下，该上限约为每个设备 60 GFLOPS；而根据其工具测得的结果，我们设备集群中最慢的中央处理器可达到 66.4 GFLOPS，最快的图形处理器可达到 87,207 GFLOPS。因此，整个设备集群都不满足重叠条件，导致 prima.cpp 退化为单轮流水线，且在途微批次仅有一个。尽管仍采用流水线并行，但这种执行模式已无法从流水线填充或权重获取重叠中获益，而这两者对于高效的流水线并行至关重要 [11, 10, 12]。由于在本文研究的场景中，prima.cpp 处于效率显著低下的运行状态，我们不将其纳入端到端比较。

C. 流水线并行性能预测

支配规则会在不进行评估的情况下剪枝候选方案，而外层求解器会对无需实际执行的放置方案进行排序。因此，两者都依赖于每个副本成本模型的准确性。参照 E2LLM [9]和 EdgeShard [11]，我们使用其瓶颈阶段对完全填充的流水线进行建模：

$$TPOT = T_{\text{slowest}} \times N_{\text{stage}}. \tag{18}$$

为验证式 18，我们从设备集群中抽取了多个部署样本，其中也包括非最优放置方案。使用测得的阶段时间和性能进行评估时，式 18 对所有部署的吞吐量预测误差均低于 0.5%，这证实了瓶颈阶段模型能够很好地解释吞吐量。接下来，我们结合阶段时间预测器（式 11）和式 18 中的瓶颈阶段模型来估算副本吞吐量。所有部署的预测平均误差为 5.22%。图 2 展示了误差分布。

D. 规划器比较

表 III 评估了支配剪枝对合成的 30 台设备的影响。这些设备基于我们的 8 台设备测试平台，通过复制观测到的硬件类型，并为其配置不同的内存容量和计算吞吐量而生成。评估采用注水法。未进行剪枝时，外层求解器运行前必须生成并定价每个候选方案。进行剪枝后，仅评估通过支配测试的候选方案。节省的开销十分显著。当设备类型数量为 $T = 5$ 时，穷举生成会评估 16554 个候选方案，整个过程需要 153 秒；而剪枝生成器仅评估 238 个候选方案，整个过程只需 0.44 秒，运行时间减少了 $347\times$ 。当数量为 $T = 10$ 时，差距进一步扩大：耗时从 35 252 秒（超过九小时）降至 11.24 秒，减少了 $3\,136\times$；在 1 045682 个候选方案中，实际评估的仅有 3 710 个。随着数量增加，这一优势还会进一步扩大

设备类型，而这恰恰是搜索空间呈指数级增长的维度。这些节省并未造成可观察到的解的质量损失，因为在这两个规模下，穷举生成与剪枝生成得到的幸存者集合和帕累托前沿完全相同，最终放置结果也相同。这就是 III-D节中所述的实证验证：尽管一般情况下尚未证明扩展规则不会丢失候选方案，但穷举枚举证实，它保留了恢复最优解所需的所有候选方案。

a) 搜索方法：表 IV 比较了两种设备清单上的放置搜索：实体设备集群，以及通过复制实测设备类型构建的 30 台设备清单。

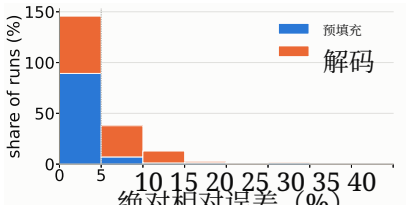

图 2：流水线并行性能预测的误差分布。每个阶段的分布总和为 100%。

在实体设备集群上，所有方法都达到了相同的最优解，差别仅在于规划成本。我们的规划器耗时 109.4 ms，而禁忌搜索耗时 384.4 ms，采用水填充分区的遗传算法耗时 548.0 ms，与层分区动态规划配合的遗传算法则耗时 8 257 ms。在八台设备的情况下，搜索空间已经足够小，解的质量不再具有区分度，因此我们报告这一结果，而不是只选择规模更大的实例。

当设备数量达到 30 台时，情况发生了变化。我们的规划器在 441.3ms 内找到了经验证的最优解。只要权重网格足够精细，混合整数公式也能找到最优解，每个帕累托点耗时 31 ms（PS，DS）。因此，要找出完整帕累托前沿需要 11.3 s，耗时约为动态规划的 $25.6\times$ 倍。这种差异源于方法本身，而非实现上的偶然因素：动态规划通过一次反向遍历即可找出整个帕累托前沿，而混合整数线性规划必须针对每个帕累托点分别求解。启发式方法在运行时间和解的质量上都逊于前两者。遗传算法耗时 230.7s，得到的拥塞高出 18.8%；用分层分区动态规划替代水填充后，运行时间增至 6 855 s，但拥塞仍高出 19.7%。禁忌搜索在 13.4 s 内完成，但拥塞高出 20.4%。值得注意的是，调整禁忌搜索的参数并不能显著缩小差距；遗传算法也有同样的情况，这是不提供最优性保证的启发式搜索的典型表现。因此，在这一规模下，最优规划是可行的：SPADE 能在毫秒级找到最优解，而启发式方法则要么更慢、精度更低，要么两者兼有。

b) 内存： 这两种精确公式体现了第三节E中所预测的权衡。多维背包动态规划是速度更快的求解器，但其稠密状态表会随着设备类型数量的增加呈指数增长。在我们规模最大的设备清单中，它需要 4.75 GB 的内存。混合整数规划则采取相反的权衡：仅使用 122MB，便可重现相同的前沿，但需要更长的求解时间。当设备类型规模更大时，例如有 20 种类型、共 50 台设备，动态规划将无法装入内存，使混合整数公式成为唯一可行的精确求解器。

E. 放置目标比较

我们现在固定规划器，仅改变目标，因此所得部署中的任何差异都可归因于目标本身。遵循E2LLM，我们从四个维度对它们进行评估：模型架构、到达率、工作负载和设备集群规模。表V汇总了四个目标在所有评估维度上的结果。在表VI、VII和VIII中，**R** 表示副本角色（P = 预填充，D = 解码），**Sl.** 表示总槽位数，**G-C** 表示图形处理器和

表 III：穷举生成与剪枝生成之间的性能比较（$N = 30$，规划器动态规划）。总时间还包括其他操作所需的时间，例如文件写入和比较

T	生成器	候选方案	幸存者	生成	规划器	总计
5	穷举	16 554	79	152.0 s	389.3ms	152.9 s
	剪枝	**238**	79	338.3毫秒	3.2 毫秒	**0.44 秒**
10	穷举	1 045 682	750	25 949 秒	8 896 秒	35 252 秒
	剪枝	**3 710**	750	8.8 秒	2.2 秒	**11.2 秒**

表 IV：规划器性能。越低越好

搜索	N=8, T=5 time	N=8, T=5 差距（%）	N=30, T=5 time	N=30, T=5 差距（%）
SPADE（背包动态规划）	**109.4 毫秒**	**best**	**441.3 毫秒**	**best**
SPADE（混合整数线性规划，完整帕累托点）	223.0 毫秒	**best**	11.3 秒	**best**
禁忌搜索（ThunderServe）	384.4 毫秒	**best**	13.4 秒	+20.4
遗传算法 + 水填充	548.0 毫秒	**best**	231 秒	+18.8
遗传算法 + 副本动态规划（E2LLM）	8.26 秒	**best**	6855 秒	+19.7

分配给该副本的中央处理器层数。第一个设备始终是头节点，最后一个设备始终是尾节点。对于角色固定放置（SPW），我们不应用支配剪枝，因为这两种技术配合效果较差。需要注意的是，在表V中，PS 和 DS 是按平均值测量的。因此，即使采用相同的部署计划，平均性能也可能因系统饱和程度不同而有所差异。

a) 模型变化：我们首先改变模型，同时固定设备集群（八台设备）、工作负载（LZByteDance）和到达周期（100ms）。我们的拥塞目标在两种模型上都优于所有基线，但改进幅度很大程度上取决于模型架构。在 Qwen3MoE 上，微调、E2LLM 和 Splitwise 产生的最大等待时间分别仅比 SPADE 高 4.7%、4.7% 和 14.2%（表 V）。尽管这些差异看似适中，但等待时间会随请求数呈二次增长，因此，在持续负载下，每请求的细微差异会转化为截然不同的积压增长。在 DeepSeekR1 上，差距明显扩大：微调、E2LLM 和 Splitwise 的最大等待时间分别是 SPADE 的 2.44×、2.44× 和 3.13× 倍。该对比反映了两种模型不同的运行状态。Qwen3MoE 每个词元仅激活一小部分参数，因此能够维持相对较高的服务速率。随着并发度增加，部署会持续接纳请求，直到内存容量成为限制因素，如表V的 DS 列所示。相比之下，DeepSeekR1 通过密集前馈网络激活所有模型参数，导致服务速率显著降低。因此，在内存耗尽之前就会达到吞吐量目标（公式 8、9 和 10），使部署受延迟而非内存限制。因此，忽略阶段特定拥塞的目标在 DeepSeekR1 上会受到更严重的惩罚。

b) 工作负载变化： 固定模型为 Qwen3MoE、设备集群规模为八台设备，并将到达周期固定为 100ms 后，我们将工作负载从雅思切换为 LZByteDance（见表VII）。这使预填充与解码词元比从 1.17 变为 0.77，跨过了解码变得比预填充更高负载的临界点。固定阶段划分的部署策略无法自然适应这一变化，而我们的拥塞目标可以。在雅思工作负载下，SPADE 使用单台 Apple M2 Max 进行预填充，并为解码专门分配 25 个槽位。在 LZByteDance 工作负载下，它通过将解码池增加到 27 个槽位来重新配置部署。因此，SPADE 以过剩的预填充容量换取更高的解码并发度。尽管每请求解码速率低于微调条件下的速率（28.9 对 31.3 词元/秒），更大的解码池提升了总解码吞吐量

表V：各目标所选择的放置方案的测量指标。 PS/DS 分别为每请求预填充/解码吞吐量（令牌/秒）； $W_{\max}$ 为最差等待时间（秒）。 PS/DS 越高、 W_{max} 越低越好；每行中的最佳值以粗体显示。

配置	SPADE PS	SPADE DS	SPADE $W_{\max}$	FT PS	FT DS	FT $W_{\max}$	E2LLM PS	E2LLM DS	E2LLM $W_{\max}$	SPW PS	SPW DS	SPW $W_{\max}$
模型 — 8 个设备，LZByteDance，100 毫秒												
Qwen3-MoE	183.2	28.9	**148**	366.3	31.3	155	366.3	31.3	155	1315.1	25.4	169
DeepSeekR1	198.6	16.2	**324**	761.1	15.2	791	761.1	15.2	791	752.9	19.5	1014
工作负载 — 8 个设备，Qwen3-MoE，100 毫秒												
雅思	694.8	23.4	**372**	394.7	25.6	400	394.7	25.6	400	1510.9	21.5	458
LZByteDance	183.2	28.9	**148**	366.3	31.3	155	366.3	31.3	155	1315.1	25.4	169
到达间隔 — 8 个设备，Qwen3-MoE，LZByteDance												
100 毫秒	183.2	28.9	**148**	366.3	31.3	155	366.3	31.3	155	1315.1	25.4	169
1000 毫秒	183.8	28.9	**59**	518.7	31.2	72	518.7	31.2	72	1210.0	25.4	69
3000 毫秒	213.2	59.0	7	508.7	46.8	**1**	213.2	59.0	7	1210.1	33.1	2
集群大小— Qwen3-MoE、雅思、100 毫秒												
4 个设备	264.1	21.5	**1477**	235.1	18.7	1729	235.1	18.7	1729	1475.6	25.0	3838
6 个设备	235.5	31.5	**598**	315.6	32.7	666	315.6	32.7	666	1476.6	17.2	804
8 个设备	694.8	23.4	**372**	394.7	25.6	400	394.7	25.6	400	1510.9	21.5	458

表 VI：模型变化时的部署方案

（a）Qwen3MoE / SPADE。

R	型	Sl.	G	–C	
P	类型2		2		8–20
	类型4				20–0
D	类型5		15		16–0
	类型5	16		–0	
	类型1	15		–1	
D	类型3	22		–0	
	类型4	7	12	–0	
	类型3				19–0

（b）Qwen3MoE / 微调 ≡ E2LLM。

R	型	Sl.	G	–C	
P	类型4		2		9–0
	类型3				39–0
P	型	类型3	1	4	8–0
D	类型5		16		–0
	类型5	16		–0	
	类型1				15–1
D	类型4		4		19–0
	类型2				5–24

(c) Qwen3MoE / Splitwise。

R	型	Sl.	G	–C	
P	型	类型1	1	2	5–23
D	类型2		18		3–3
	类型4				3–0
	类型4				3–0
	类型3				10–0
	类型5				13–0
	类型5				13–0
D	型	类型3	2	4	8–0

(d) DeepSeekR1 / SPADE。

R	型	Sl.	G	–C	
P	型	类型3	1	4	8–0
D	类型5		15		15–0
	类型5	15		–0	
	类型1	18		–0	
D	类型2		2		15–0
	类型4				6–0
	类型4				6–0
	类型3				21–0

(e) DeepSeekR1 / 微调 ≡ E2LLM。

R	型	Sl.	G	–C	
P	类型2		2		13–0
	类型5				35–0
D	类型5		6		24–0
	类型1				24–0
D	类型4		2		5–0
	类型3				20–0
	类型3				20–0
	类型4				3–0

(f) DeepSeekR1 / Splitwise。

R	型	Sl.	G	–C	
P	类型5		2		13–0
	类型5				35–0
D	型	类型3	1	4	8–0
D	类型4		2		2–0
	类型2				15–0
	类型4				2–0
	类型1				29–0
D	型	类型3	1	4	8–0

表VII：改变工作负载时的部署方案（a）雅思 / SPADE。

R	型	Sl.	G	–C	
P	型	类型3	1	4	8–0
D	类型5		15		15–0
	类型5	15		–0	
	类型1				14–4
D	类型4		6		12–0
	类型3				36–0
D	类型4		4		18–0
	类型2				5–25

(b) 雅思 / 微调 ≡ E2LLM.

R	型	Sl.	G	–C	
P	类型4		2		10–0
	类型3				38–0
P	型	类型3	1	4	8–0
D	类型5		15		15–0
	类型5	15		–0	
	类型1				14–4
D	类型4		4		18–0
	类型2				5–25

(c) 雅思 / Splitwise。

R	型	Sl.	G	–C	
P	类型1	1	2	5	–23
D	型3	2			48–0
D	类型2		18		3–3
	类型4				3–0
	类型4				3–0
	类型3				10–0
	类型5				13–0
	类型5				13–0

(d) LZByteDance / SPADE。

R	型	Sl.	G	–C	
P	类型2		2		8–20
	类型4				20–0
D	类型5		16		–0
	类型5	16		–0	
	类型1				15–1
D	类型3		7		22–0
	类型4	7	12	–0	
	类型3				19–0

(e) LZByteDance / 微调 ≡E2LLM。

R	型	Sl.	G	–C	
P	类型4		2		9–0
	类型3				39–0
P	型	类型3	1	4	8–0
D	类型5		16		–0
	类型5	16		–0	
	类型1				15–1
D	类型4		4		19–0
	类型2				5–24

(f) LZByteDance / Splitwise。

R	型	Sl.	G	–C	
P	型	类型1	1	2	5–23
D	型	类型3	2	4	8–0
D	类型2		18		3–3
	类型4				3–0
	类型4				3–0
	类型3				10–0
	类型5				13–0
	类型5				13–0

约降低 31%（27 个槽位对比 19 个槽位），从而减少拥塞。

相比之下，微调和 E2LLM 的目标在两种工作负载下都保留了相同的三个预填充设备，仅将层划分边界调整了一层。E2LLM 的表现类似，因为其目标仍包含预填充服务时间项。尽管工作负载发生了显著变化，预填充服务时间的减少仍足以抵消跨阶段重新分配设备带来的收益，因此优化器始终保留原有的放置方案。不过，工作负载变化仍会影响底层执行特性。有趣的是，从雅思数据集切换到 LZByteDance 数据集时，PS 减少，而 DS 增加。由于 Transformer 推理依赖自注意力，层的开销会随着 KV 缓存中已存储的位置数量增加。因此，雅思和 LZByteDance 会产生不同的执行深度，这反映在输入和输出词元数量的差异上，进而导致阶段时间不同。因此，在这两种工作负载下，预填充与解码之间的最佳平衡并不相同。

有趣的是，微调和 E2LLM 仍然收敛到相同的部署。由于搜索空间是离散的，改变放置需要替换整个副本组合，而不是进行增量调整。工作负载变化足以改变目标值，但不足以将解移动到不同的搜索空间盆地。因此，微调和 E2LLM 选择相同的方案，仅在赋予该方案的值上有所不同。Adapted Splitwise 更为僵化，在两种工作负载下都将 RTX5070 保留在预填充中，因为其规则始终将吞吐量最高的副本分配给预填充，而不考虑工作负载特征。

c) 到达率变化：固定模型、工作负载和设备集群分别为 Qwen3MoE、LZByteDance 和八台设备，将到达周期从 100 ms 调整至 3000 ms（表 VIII）。这是唯一一种从过载状态转为欠载状态的配置，交叉点位于 1 s 到 2 s 之间。在过载状态下，放置方案保持不变：SPADE 和 E2LLM 在到达周期不超过 1000 ms 时保持原有部署，并在 3000 ms 时收敛到相同的放置方案；而完成时间和 Adapted Splitwise 则始终保持相同的部署。转变之后，吞吐量增益依然明显。在 3000 ms 时，SPADE 和 E2LLM 的每请求

解码速率达到 59.0 令牌/秒，而完成时间和 Adapted Splitwise 分别为 46.8 令牌/秒和 33.1 令牌/秒。当多余的预填充容量不再减少拥塞时，SPADE 会将资源重新分配给解码。再加上其更高的解码吞吐量和饱和时更大的解码池，即使等待时间降至零，也能将闲置容量转化为更高的每请求吞吐量。最值得关注的是，SPADE 和 E2LLM 最终收敛到相同的部署。当到达负载充分低于容量时，增加预填充资源不再降低排队延迟，因此，尽管两者的目标函数来自不同的公式，它们都会将资源转向解码。

在到达周期为 3000 ms 时出现了一个例外，此时 SPADE 的最大等待时间仍约为 7 s（表V）。实验日志显示，测得的预填充吞吐量比预测值最高低 30%，从而导致残余拥塞。该差异源于规划器使用具有代表性的序列长度而非完整的请求长度分布来表示工作负载。由于延迟会随序列长度呈非线性变化，在代表性长度下评估的吞吐量通常不同于整个工作负载的平均吞吐量。这种近似显著降低了规划复杂度，并且与以往部署规划器中常用的工作负载抽象一致 [13, 17, 19]。重要的是，这种影响仅限于预填充。解码吞吐量仍能被准确预测。在过载配置中也存在同样的过度预测，但在这些配置下，预测的瓶颈与测得的瓶颈一致，因此它只会改变拥塞的程度。在 3000 ms 时，模型预测 $W = 0$，但测量结果显示仍存在预填充拥塞。尽管采用相同的预填充，我们仍观察到 SPW 的预填充吞吐量存在细微差异。这些差异可能是由不同空闲时间下的设备电源管理引起的。由于这涉及运行时行为而非放置，我们不对此作进一步分析。

d) 可扩展性：我们评估由四台、六台和八台设备组成的嵌套设备清单，从而避免设备集群规模与设备集群组成相混淆。四台设备的设备集群由一台类型1设备、一台类型2设备和两台类型4设备组成；六台设备的设备集群在此基础上增加一台类型3设备和一台类型5设备；八台设备的设备集群则使用全部设备。在所有设备集群规模下，SPADE 始终实现最低的最大等待时间。这一优势在最小规模的设备集群中最为明显：与 SPADE 相比，完成时间和 E2LLM 目标产生的最大等待时间长 17%，而 Adapted Splitwise 产生的最大等待时间长 160%。

F. 框架比较

SPADE 构建于 llama.cpp 之上，但将原始的星型拓扑通信替换为点对点流水线。因此，我们同时与原始 llama.cpp 实现以及同样采用星型拓扑的 E2LLM 进行比较。在使用三个设备和九个槽位（类型1、类型5、类型2）的情况下，SPADE 达到 37.13 令牌/秒，而 E2LLM 和 llama.cpp 分别为 25.23 令牌/秒和 12.69 令牌/秒。在此

表 VIII：改变到达率时的部署方案。（a）SPADE，100 和 1000 毫秒。

R	型	Sl.	G–C	
P	类型2	2		8–20
	类型4			20–0
D	类型5	15		16–0
	类型5			15–0
	类型1			15–1
D	类型3	7	2	22–0
	类型4			7–0
	类型3			19–0

(b) SPADE ≡E2LLM，3000 毫秒。

R	型	Sl.	G–C	
P	型2	1	8	40
D	类型5	16–0		
	类型5	15–0		
	类型1			15–1
D	类型4	6		12–0
	类型3			36–0
D	类型4	6		12–0
	类型3			36–0

(c) 微调，所有间隔（≡ E2LLM 为 100 和 1000 毫秒）。

R	型	Sl.	G–C	
P	类型4	2		9–0
	类型3	3		9–0
P	型3	1	4	8–0
D	类型5	16–0		
	类型5	15–0		
	类型1			15–1
D	类型4	4		19–0
	类型2			5–24

（d）Splitwise，所有间隔。

R	型	Sl.	G–C	
P	型1	1	25	23
D	类型2	18		3–3
	类型4			3–0
	类型4			3–0
	类型3			10–0
	类型5			13–0
	类型5	13–0		
D	类型3	2	48	0

随着规模扩大，E2LLM 得益于分布式流水线执行，而原始 llama.cpp 无法执行网络化流水线并行。随着流水线不断扩展，情况发生了变化。在六台设备和 18 个槽位的配置下（第一个解码器副本，表 VIII.d），SPADE 可维持 23.92 tokens/s 的吞吐量，而 E2LLM 和 llama.cpp 分别仅达到 5.02 和 6.29 tokens/s。其原因在于通信争用。在 E2LLM 中，每个阶段都通过中心节点交换激活值，迫使多个流水线阶段竞争同一节点的上行和下行带宽。随着阶段数量增加，通信逐渐成为主要瓶颈，最终抵消流水线并行本身带来的收益，导致 E2LLM 的性能甚至不如原始 llama.cpp。相比之下，P2Pring 将流量分散到各台设备，避免集中于单个节点，使吞吐量能够随着流水线深度增加而更加平稳地扩展。

G. SPADE 的局限性SPADE

如前所述，SPADE 能够准确预测各个运行点的预填充吞吐量，但单个平均请求的吞吐量不一定等于所有请求的平均吞吐量。因此，当请求长度呈现较高方差时，仅使用平均输入长度描述的工作负载可能会导致较大的预测误差。此外，SPADE 面向静态部署规划。优化器在规划期间假设工作负载特征描述、到达率和设备集群组成均保持不变。在到达率和请求大小发生变化时进行在线角色重新分配、副本迁移或重新优化等动态调整，是对我们工作的补充，也仍是未来的研究方向。最后，将支配关系扩展到后代副本目前仅获得了经验证据支持，尚未证明其在一般情况下不会造成损失。在验证计算成本仍可承受的情况下，穷举方法会产生相同的幸存者集合、帕累托前沿和最终放置方案，但一般性证明仍有待完成。

V. 结论

我们介绍了 SPADE，这是一种面向异构边缘设备集群、具有良好可处理性和可扩展性的放置规划器。SPADE 将解耦的预填充/解码服务建模为排队系统，并直接优化拥塞。为使优化具有可处理性，SPADE 将剪枝与用于层划分的注水法相结合，在保留与穷举方法相同幸存者集合的同时，剪除了多达 99.6% 的候选方案。结合注水法后，SPADE 在 30设备集群上可于数百毫秒内达到经验证的最优解；相比之下，即使运行数秒至数小时，启发式搜索的结果仍显著低于最优解。这些结果表明，在设备集群规模下，面向异构边缘服务的最优排队感知部署是可处理的。

致谢

本研究部分得到了挪威研究委员会322473号资助（AirQMan项目）以及欧盟委员会101086541号资助（MISO项目）的支持。

参考文献

[1] Zhixiong Chen等。“大语言模型的网络边缘推理：原理、技术与机遇”。载于：*ACM*计算调查58.12（2026），第1–35页。DOI：10.1145/3809166。[2] Biwei Yan等。“保护大语言模型（LLM）和大语言模型智能体数据隐私：文献综述”。载于：高置信计算5.2（2025），第100300页。DOI：10.1016/j.hcc.2025.100300。[3] Zonghang Li等。“Prima.cpp：在异构且资源受限的家庭集群上实现快速30–70B大语言模型推理”。载于：国际学习表征会议。第2026卷。2026年，第144376–144413页。[4] Chris Tong等。“Parallax：在去中心化环境中实现高效的大语言模型推理服务”。载于：*arXiv*预印本 *arXiv:2509.26182*（2025）。DOI：10.48550/arXiv.2509.26182。[5] Mingyu Sun等。“基于卸载的流水线并行，在边缘设备上实现协作式无损大语言模型推理服务”。载于：*arXiv*预印本 *arXiv:2512.21835*（2025）。DOI：10.48550/arXiv.2512.21835。[6] Yixuan Mei等。“Helix：通过最大流经由异构图形处理器和网络提供大语言模型服务”。载于：第*30*届*ACM*编程语言与操作系统架构支持国际会议论文集，第*1*卷。2025年，第586–602页。DOI：10.1145/3669940.3707215。[7] Juntao Zhao等。“POSTER: LLM-PQ：采用阶段感知划分和自适应量化，在异构集群上提供大语言模型服务”。载于：*ACM SIGPLAN*并行编程原理与实践年度研讨会第*29*届会议论文集。2024年，第460–462页。DOI：10.1145/3627535.3638480。

[8] Yixuan Mei 等。《Coral：异构云端 GPU 上高性价比的多大语言模型服务》。载于：*arXiv* 预印本 *arXiv:2605.04357*（2026）。DOI：10 . 48550 / arXiv.2605.04357。[9] Truong-Thanh Le 等。《E2LLM：迈向异构边缘/雾环境中的高效大语言模型服务》。载于：*arXiv* 预印本 *arXiv:2606.03770*（2026）。DOI：10.48550/arXiv.2606.03770。[10] Yang Hu 等。《Pipeedge：异构边缘设备上的大规模模型推理流水线并行》。载于：*2022*年第*25*届*Euromicro*数字系统设计会议（*DSD*）。IEEE。2022，第298–307页。DOI：10.1109/ DSD57027.2022.00048。[11] MingjinZhang 等。《Edgeshard：通过协同边缘计算实现高效的大语言模型推理》。载于：*IEEE*物联网期刊 12.10（2024），第13119–13131页。DOI：10.1109/JIOT.2024.3524255。[12] Truong Thanh Le 等。《面向高输入频率应用的机器学习模型在边缘侧的最优分布》。载于：*2024 IEEE*云计算技术与科学国际会议（*Cloud-Com*）。IEEE。2024，第143–150页。DOI：10 . 1109 / CloudCom62794.2024.00033。[13] Pratyush Patel 等。《Splitwise：利用阶段拆分实现高效的生成式大语言模型推理》。载于：*2024 ACM/IEEE*第*51*届年度国际计算机体系结构研讨会（*ISCA*）。IEEE。2024，第118–132页。DOI：10.1109/ISCA59077.2024.00019。[14] Yinmin Zhong 等。《{DistServe}：拆分预填充与解码，实现吞吐效率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 24*）。2024，第193–210页。[15] Shrikara Arun 等。《面向负载感知的解耦式大语言模型服务预填充转移》。载于：*arXiv* 预印本 *arXiv:2607.02043*（2026）。DOI：10 . 48550 / arXiv.2607.02043。[16] Youhe Jiang、Ran Yan 和 Binhang Yuan。《Hexgen-2：异构环境中的解耦式生成推理》。载于：国际学习表征会议。第2025卷。2025，第3985–4002页。[17] Youhe Jiang 等。《Thunderserve：云环境中的高性能且高性价比的大语言模型服务》。载于：机器学习与系统会议论文集 7（2025）。[18] Youhe Jiang 等。《HEXGEN：异构环境中的大语言模型生成推理》。载于：第*41*届机器学习国际会议论文集。ICML’24。奥地利维也纳：JMLR.org，2024。[19] Youhe Jiang 等。《HexGen-3：具备细粒度异构资源自动扩缩容的完全解耦式大语言模型服务框架》。载于：第四十三届机器学习国际会议。

[20] Luchang Li 等。《面向服务级别目标的预填充-解码解耦式大语言模型推理计算资源分配》。载于：*arXiv* 预印本 *arXiv:2603.04716*（2026）。DOI：10.48550/arXiv.2603.04716。[21] Zeyu Zhang 和 Haiying Shen。《PEACE：面向混合提示词大语言模型推理的抢占式高效集群调度》。载于：*2026*年*IEEE*国际并行与分布式处理研讨会（*IPDPS*）。IEEE。2026，第1272–1286页。DOI：10.1109/ IPDPS65963.2026.00105。[22] Jiakun Fan 等。《APEX：受限图形处理器上面向在线大语言模型推理的异步并行中央处理器-图形处理器执行》。载于：*2026*年*IEEE*国际并行与分布式处理研讨会（*IPDPS*）。IEEE。2026，第1190–1203页。DOI：10.1109/IPDPS65963.2026. 00099。[23] Hans Kellerer、Ulrich Pferschy 和 David Pisinger。《多维背包问题》。载于：背包问题。Springer，2004，第235–283页。[24] DeepSeek-AI。《*DeepSeek-R1*：通过强化学习激励大语言模型的推理能力》。2025。arXiv：2501.12948［cs.CL］。URL：https://arxiv.org/ abs/2501.12948。[25] Qwen团队。《*Qwen3*技术报告》。2025。arXiv：2505。09388［cs.CL］。URL：https : / / arxiv. org / abs / 2505.09388。[26] Ling Zhan 等。《LongReason：通过上下文扩展构建的合成长上下文推理基准》。载于：*arXiv* 预印本 *arXiv:2501.15089*（2024）。[27] chillies。《*DPO*雅思写作》。Hugging Face。2024。URL：https : / / huggingface . co / datasets / chillies / DPO 雅思写作（访问于2026年7月8日）。[28] Marc Brysbaert。《我们每分钟阅读多少单词？阅读速度的综述与元分析》。载于：*记忆*与语言期刊（2019）。