AIConfigurator：面向多框架大语言模型服务的闪电般快速配置优化

Tianhao Xu Yiming Liu Xianglong Lu YijiaZhao Xuting Zhou Aichen Feng Yiyi Chen Yi Shen Qin Zhou Xumeng Chen IlyaSherstyuk HaoruiLi Rishi Thakkar BenHamm Yuanzhe Li XueHuang Wenpeng Wu AnishShanbhag Harry Kim Chuan Chen JunjieLai英伟达

arXiv:2601.06288v1 [cs.LG] 9 Jan 2026

摘要

在生产系统中优化大语言模型（LLM）推理正变得愈发困难，这源于动态工作负载、严格的延迟/吞吐量目标，以及迅速扩大的配置空间。这种复杂性不仅涉及分布式并行策略（张量/流水线/专家并行），还包括复杂的、特定于框架的运行时参数，例如 CUDA 图的启用、可用 KV 缓存内存占比和最大令牌容量等；这些参数会显著影响性能。现代推理框架（如 TRT-LLM、vLLM、SGLang）种类繁多，且各自采用不同的内核和执行策略，因此手动调优不仅因框架而异，在计算上也难以承受。我们提出 AIConfigurator，这是一套统一的性能建模系统，无需基于 GPU 的性能分析，即可快速进行与框架无关的推理配置搜索。AIConfigurator 结合了：（1）一种方法论，将推理分解为可通过解析方式建模的基本操作——通用矩阵乘法、注意力、通信和内存操作，同时捕捉特定于框架的调度动态；（2）一个经过校准的内核级性能数据库，涵盖这些基本操作在多种硬件平台和热门开放权重模型（GPT-OSS、通义千问、DeepSeek、LLama、Mistral）上的表现；以及（3）一个抽象层，可自动为目标后端确定最优启动参数，并无缝集成到生产级编排系统中。在生产环境的大语言模型服务工作负载上进行的评估表明，AIConfigurator 能够找到更优的服务配置，使稠密模型（如 Qwen3-32B）的性能最高提升 40%，混合专家架构（如 DeepSeek-V3）的性能最高提升 50%；同时，平均仅需 30 秒即可完成搜索，从而能够快速探索广阔的设计空间——从集群拓扑到引擎专属标志。

1 引言

大语言模型（LLM）的迅速发展，对计算基础设施提出了前所未有的需求。随着最先进模型的参数量从数亿增长到数千亿，推理部署的效率已成为决定经济可行性的关键因素。然而，优化这些部署过程充满复杂性。服务提供商必须应对配置参数的组合爆炸——从张量并行、流水线并行和专家并行策略，到批次大小和量化的细粒度设置。传统的性能调优通常依赖手动基准测试和穷举测试，如今已越来越难以为继。随着现代 GPU 成本不断上涨，手动穷举测试变得极其昂贵。这些方法需要投入大量工程精力才能找到解决方案，而最终结果往往仍不够理想，导致大量性能潜力未能释放。

解耦式服务的兴起 [19, 28]——将预填充和解码阶段分配到不同的计算资源上——进一步加剧了这一局面的复杂性。尽管解耦有望优化“有效吞吐量”（在严格延迟约束下的吞吐量），但它并非普遍优越的解决方案；其收益在很大程度上取决于模型架构、硬件拓扑和网络带宽之间的相互作用。实践者面临着艰难的权衡：对于特定工作负载，解耦带来的调度灵活性是否足以抵消通信开销？此外，要配置此类系统以满足严格的服务等级协议（SLA）——尤其是首令牌时间（TTFT）和每输出令牌时间（TPOT）——其设计空间很容易超过 10,000 种组合。

除了架构决策之外，推理引擎本身的复杂性也进一步加剧了这一难题。TensorRT-LLM [2],vLLM [13], 和 SGLang [27] 等现代框架提供了大量可调节的运行时标志，例如是否启用 CUDA 图、可用的 KV 缓存内存占比以及最大令牌容量等，这些选项会对性能产生重大影响。针对特定工作负载实现吞吐量最大化的配置，随着服务环境的变化，往往会变得脆弱或低效。因此，开发者经常

放弃调优过程，转而采用保守的设置，导致大量性能潜力未能发挥。

虽然像 Vizier 这样的黑盒优化方法 [11]或 Morphling 这类自动化服务框架 [23] 能够加快寻找最优配置的速度，但要针对每种特定场景收敛到解决方案，仍需要消耗大量 GPU 小时。

为应对这些挑战，我们推出了 AIConfigurator，这是一款专为优化多种推理框架中的大语言模型推理而设计的工具包。不同于依赖理论抽象的通用模拟器，AIConfigurator 采用基于操作级性能建模的数据驱动方法。该工具包将推理分解为基本内核——例如通用矩阵乘法计算、注意力机制和通信原语（如全归约、点对点通信）——并利用真实系统数据进行插值，从而针对英伟达平台（Ampere、Ada、Hopper 和 Black well）实现高保真度估算。我们的贡献包括：

- 设计一个系统，能够在复杂的推理配置空间中进行搜索，以高精度在数秒内找出最优设置。
- 通过与主流推理框架（包括 vLLM、SGLang、TRTLLM 和英伟达 Dynamo）无缝集成，展示该系统的有效性，并提供切实可行、可用于生产环境的建议。
- 通过与真实硅片数据进行基准测试，开展全面评估。

2 背景

2.1 大语言模型推理优化

大语言模型推理优化涉及三个相互依存的支柱。高级调度——包括连续批次处理 [24], 分页注意力 [13], 分块预填充 [5], 和解耦式服务 [28]——通过针对预填充（计算受限）与解码（内存受限）阶段各自不同的计算特征，最大限度地提高硬件利用率。模型并行通过张量并行（TP） [21], 流水线并行（PP） [12],和专家并行（EP） [16]，将大型模型分布到多个 GPU 上，以支持混合专家架构。配置调优则用于探索由此产生的组合空间；尽管 Vidur [4] 和 APEX [15] 等模拟器能够快速探索，但它们依赖理论屋顶线模型，因此得出的结果往往与生产环境性能存在偏差。

2.2 聚合式服务与解耦式服务

解耦式服务 [19, 28] 将预填充和解码分配到不同的 GPU 池中，从而支持独立扩展，但会引入 KV缓存传输开销。Splitwise [18] 表明对于短上下文，这种开销可能抵消其优势。最佳架构取决于工作负载组合（预填充密集型还是解码密集型）、互连带宽和集群规模——对于较小规模的部署，采用分块预填充的聚合式服务往往优于解耦。正是这种复杂性促使 AIConfigurator 对这两种架构进行建模，以确定最佳配置。

3 动机

传统启发式方法（如“节点内采用 TP、节点间采用 PP”）无法捕捉计算与网络带宽之间的非线性交互。研究 [17, 26] 表明，自动化搜索在成本效益方面可比手动调优高出 >2×。

框架异构性。生产环境中的推理涉及多种框架，各自具有不同的性能特征：**vLLM** [13]（分页注意力、基于 Python 的调度）、**SGLang** [27]（基数注意力、Triton 内核）、**TensorRT-LLM** [2]（静态图优化、自定义内核）以及**NVIDIA Dynamo** [1]（与后端无关的编排）。每种框架都表现出独特的性能断崖，这些断崖受大量框架特定运行时标志控制，而通用模型无法有效捕捉这些因素。

配置调优差距。当前方法——夜间基准测试 [20] 和精心整理的配方 [3]——是静态查找表，无法满足动态生产环境的需求。AIConfigurator 提供算法搜索，可在并行度、批次大小和服务架构构成的多维空间中识别符合 SLA 的配置。

4 设计与实现

AIConfigurator 采用严谨的数据驱动方法，在庞大的大语言模型推理配置空间中进行探索。系统不会依赖理论屋顶线模型或穷举基准测试，而是将推理分解为基本操作，收集这些基本操作在真实硬件上的测量数据，并通过定义明确的性能模型组合出端到端性能估算。

该工具包通过统一的后端抽象支持多种推理框架。每个后端（TensorRT-LLM、SGLang、vLLM）均实现了特定于框架的内存估算、聚合式服务模拟和基于约束的优化逻辑，同时共用操作建模基础设施。

本节将介绍 AIConfigurator 的架构、核心机制和实现细节。

4.1 AIConfigurator 工作流程

图 1 展示了两种服务模式的帕累托曲线对比——**聚合** 和 **解耦**——针对一款运行在 64 个 **通义千问-235B** 模型上的 **H200** GPU。这体现了 AIConfigurator 的一项关键洞见

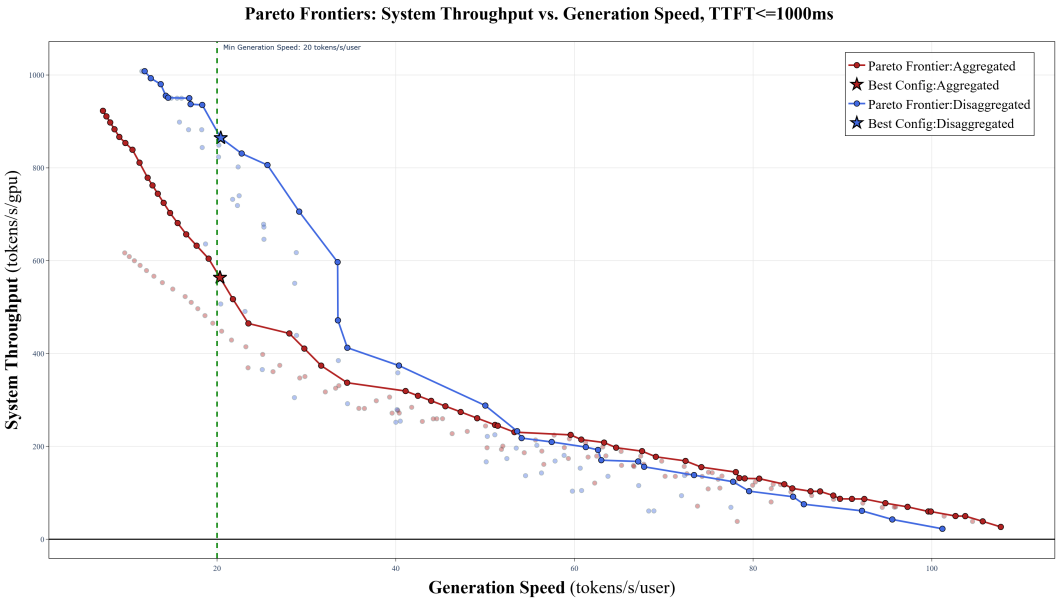


图 1：AIConfigurator 对 Qwen3-235B 在 64 个 H200 GPU 上运行时的吞吐量与速度帕累托前沿进行了预测。图中绘出了所有能够达到首词元时间（TTFT）$\leq$ 1000毫秒的服务配置。

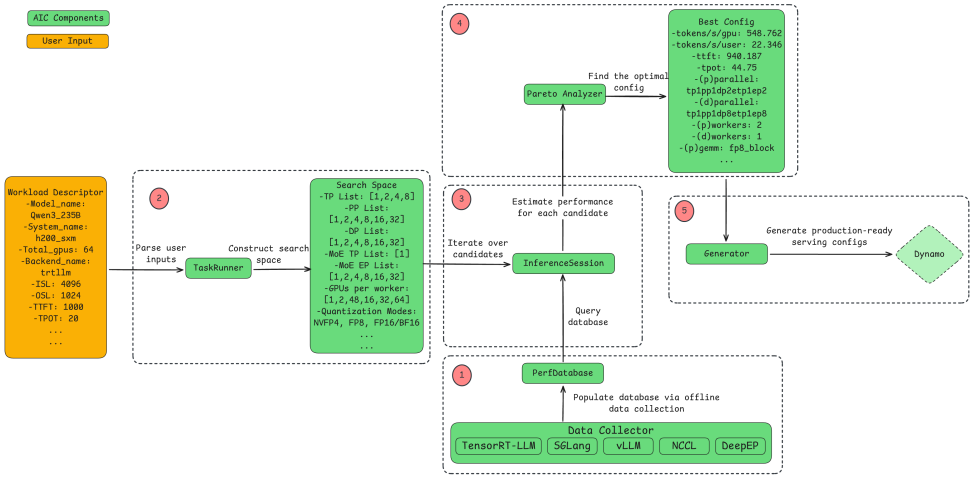


图 2：K ey 是 AIConfigurator 的关键组件，以及寻找最优配置的通用工作流程 on.

来自 AIConfigurator 的模拟结果。横轴表示**生成速度**（每个用户请求每秒生成的令牌数），纵轴表示**系统吞吐量**（每个 GPU 每秒生成的令牌数）。每个点都代表一种满足首令牌时间约束的服务配置。

最佳配置是在满足目标生成速度的同时，实现最高系统吞吐量的配置。例如，输入序列长度为 4096、输出长度为 1024 时，如果要求每位用户的速度至少达到 20 令牌/秒，那么带星号的配置就是最优配置，因为它们在超过这一速度阈值的同时，最大限度地提高了每个图形处理器的吞吐量。值得注意的是，此处解耦式服务更具优势：其最佳配置可达到 823 令牌/秒/GPU，比相同速度约束下的最佳聚合配置（564 令牌/秒/GPU）高约**53%**。

总之，AIConfigurator 能够确定最优服务配置，在满足特定服务等级协议目标（例如，TTFT 为 <1 秒、生成速度为 >20 令牌/秒/用户）的同时最大化系统吞吐量。

图 2 描述了使用 AIConfigurator 的通用工作流程，通常包括五个步骤，每个步骤都围绕 AIConfigurator 的一个关键组件展开：

- **PerfDatabase**：首先，执行离线数据收集流程，

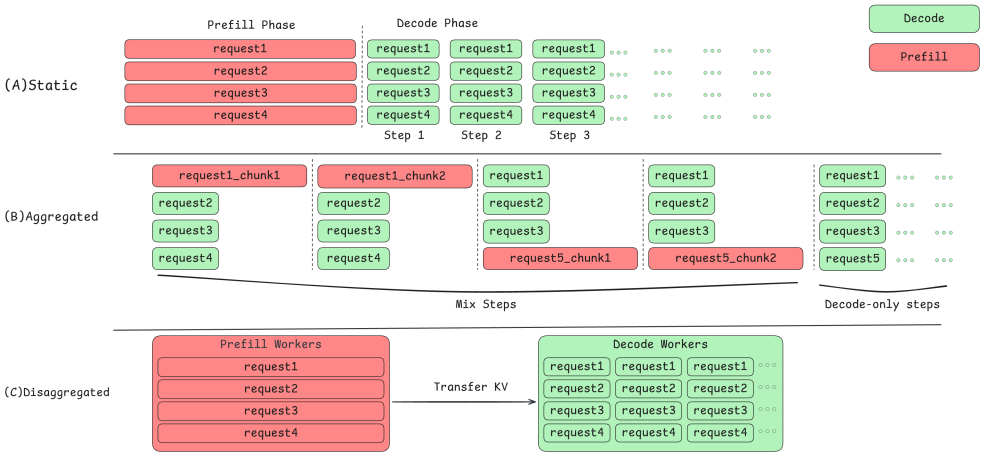


图3：AIConfigurator 建模的三种服务模式。(A) **静态**：图形处理器工作节点端到端处理固定的推理请求。(B) **聚合**：混合处理不同请求的预填充和解码（连续批处理）。(C) **解耦**：为预填充和解码阶段分别设置独立的图形处理器池。

以构建一个数据库，其中包含各种常用大语言模型算子在不同指定硬件平台上的全面性能数据。在此流程中，每种受支持的推理框架都将单独处理。

• **TaskRunner**：在第二步中，TaskRunner 将根据用户提供的工作负载描述符构建一个搜索空间，其中包含所有有效的候选服务配置。该描述符将包含用户期望的环境设置、服务等级协议以及请求特定特征等信息。

• **InferenceSession**：随后，InferenceSession 将遍历所有候选服务配置，并针对每个候选配置，将迭代级建模与直接从 Perf-Database 查询的算子级性能数据相结合，以估算关键性能指标。

• **Pareto Analyzer**：之后，帕累托分析器将根据上一阶段生成的性能预测，对所有有效的服务配置进行筛选和排序，并输出排名靠前的配置及其全面的性能预测。

• **生成器**：终于，AIConfigurator 的生成器模块可以将帕累托分析器识别出的服务建议直接转换为与版本兼容的启动文件，适用于 TensorRT-LLM、vLLM 和 SGLang 中的任意一种推理引擎，并自动设置最优服务标志，例如 --enable_cuda_graph、--kv_cache_- free_gpu_mem_fraction 和 --enable_chunked_- context。NVIDIA Dynamo 等服务框架也可以直接利用启动文件来配置最优的推理服务器。

4.2 性能建模

AIConfigurator 不会直接估算系统吞吐量和生成速度，而是从并发数处于稳态的服务器所接收的平均请求这一角度推导这两项指标。

$$\text{Generation Speed} = 1000/TPOT \tag{1}$$

$$\begin{aligned}\text{System Throughput} = {} & \frac{1000}{TTFT + (OSL-1)\times TPOT} \\ & * \text{Batch Size} \\ & * OSL \\ & /\text{Total Number of GPU}\end{aligned} \tag{2}$$

无论是 **TTFT**（首个令牌时间）还是 **TPOT**（每输出令牌时间），单位都是毫秒。**OSL**（输出序列长度）被视为由用户提供的工作负载描述符指定的固定值。至于**批次大小**，AIConfigurator 会在估算给定服务配置的性能时，遍历一系列预定义值。

为了高效估算 TPOT 和 TTFT，我们首先考虑推理服务器处理典型请求的完整生命周期。在大语言模型服务中，任何请求都会经历两个不同的处理阶段：

预填充/上下文阶段会处理整个用户输入提示词，计算并存储对应的 KVcache，然后生成第一个输出词元。此阶段通常是计算密集型的，通常会使用 FlashAttention 等融合式多头注意力风格的内核 [8] 来加速注意力计算。此外，上下文

分块 [5] 还可用于将长输入提示的预填充分割为多个迭代步数。

解码/生成阶段 以逐步的自回归方式逐个生成剩余的输出词元。新生成的词元会被用作后续步骤的输入，并使用缓存的键值对来避免重新计算过去词元的键值对。该阶段通常对内存要求较高，并会在此阶段使用 XQA [2]等专用内核。

虽然预填充和解码构成了大语言模型推理的基本计算阶段，但实际性能在很大程度上取决于服务引擎如何对二者进行编排。简单的抽象不足以捕捉不同调度策略的细微差异。因此，AIConfigurator 明确对三种不同的服务模式进行了建模——静态、聚合和解耦；每种模式处理资源争用和阶段交错的方式各不相同。我们首先定义静态模式下的基准行为。

算法 1 静态模式推理性能估算 要求：ISL（输入长度）、OSL（输出长度） 要求：B（批次大小）、P（前缀长度） 要求：步长 S_{stride} （默认值：32） 要求：函数 GETSTEPLATENCY(batch_size, seq_len, phase) 1：阶段 1：上下文延迟（TTFT） 2：$ISL_{eff} \leftarrow ISL - P$ 3：计算处理输入提示的延迟 4：$TTFT \leftarrow$ GETSTEPLATENCY(B, ISL_{eff}, '预填充') 5：阶段 2：生成延迟（总生成时间） 6：$T_{gen} \leftarrow 0$ 7：如果 $OSL > 1$ 则 8：$k \leftarrow 0$ 9：当 $k < OSL - 1$ 时执行 10：$S_{seq} \leftarrow ISL + k + 1$ ▷当前总序列长度 11：查询当前长度下单个解码步的延迟 12：$T_{step} \leftarrow$ GETSTEPLATENCY(B, S_{seq}, '解码') 13：$R \leftarrow \min(S_{stride}, OSL - 1 - k)$ ▷为接下来的 R 个令牌进行插值 14：$T_{gen} \leftarrow T_{gen} + (T_{step} \times R)$ 15：$k \leftarrow k + S_{stride}$ 16：结束循环 17：结束条件 18：阶段 3：TPOT 计算 19：如果 $OSL > 1$ 则 20：$TPOT \leftarrow T_{gen}/(OSL - 1)$ 21：否则 22：$TPOT \leftarrow 0$ 23：结束条件 24：返回 TTFT、TPOT

4.2.1 静态模式

如图 3（A）所示，在静态服务模式下，工作负载以严格的顺序方式处理，并采用固定批大小。在此模式下，TTFT 等同于预填充阶段的延迟。TPOT 通过对生成整个输出序列所需的后续解码步延迟取平均值来近似计算。

算法 1 概述了计算这些指标的过程。该过程采用基于步幅的优化（步骤13），通过对区间内的成本进行插值，而不是为每个令牌步骤查询数据库，从而降低估算生成延迟的计算开销。

4.2.2 聚合模式

也称为在途批处理或连续批处理 [24]，这种服务模式的特点是能够在单次推理迭代中混合处理来自不同请求的预填充和解码步，如图 3（B）所示。与静态服务相比，这种灵活性显著提高了 GPU 资源利用率，从而带来更高的系统吞吐量。

我们的性能模型详见算法 2，通过将执行划分为两个不同阶段来近似这一行为：

混合阶段： 该阶段表示连续批处理系统的稳态运行状态，其中预填充请求和解码请求并发执行。调度器优先利用可用的上下文容量（C_{ctx}）来处理预填充请求。默认情况下，一个预填充请求包含数量为 ISL 的令牌，也可以选择启用上下文分块，将完整 ISL 数量的上下文令牌拆分为多个预填充请求。剩余的批次槽位将分配给解码任务（N_{mix}^{gen}）。至关重要的是，当预填充工作负载较重时（上下文处理时间超过生成时间），我们的模型（算法 2，第 6–10 行）会使用速率匹配启发式方法限制并发解码流的数量。这可以避免"饥饿"场景，即解码请求完成的速度快于新请求被预填充的速度。该阶段的步延迟（L_{mix}）主要由计算密集型的预填充注意力所决定。

纯生成阶段： 此阶段模拟工作负载的尾部阶段，或低到达率期间，此时预填充队列已清空。系统转入纯解码模式，将所有批次槽位（B）专用于自回归生成。此处的步延迟（L^{gen}）显著较低，通常受内存带宽而非计算能力限制。

我们根据混合阶段的延迟估算首令牌时间，并引入一个以分段线性函数建模的经验修正因子（F_{corr}）。该因子考虑了基础调度开销（常数项）、随着上下文积压增加而产生的成比例排队延迟，以及用于反映系统级准入控制的饱和上限。对于每输出令牌时间，我们采用两个阶段延迟的加权平均值。值得注意的是，

在计算权重时，我们会对混合阶段持续时间施加一个小偏移量（3 步）（算法 2，步骤 5）。这种启发式方法可以过滤批次初始步数中经常出现的调度抖动，从而更稳健地估算稳态解码性能。该方法无需进行完整的离散事件仿真，即可对预填充与解码操作之间的非线性干扰进行建模。

算法 2 聚合模式（连续批处理）性能估算

要求： 输入序列长度，输出序列长度

要求： 批次大小 （批次大小），C_{ctx} （上下文令牌容量）

要求： 辅助函数:

GETMIXLAT(N_{ctx}，N_{gen}，输入序列长度，输出序列长度)

GETGENLAT(N_{gen}，输入序列长度，输出序列长度)

步骤 1：阶段持续时间（以步数计）

$T_{total_ctx} \leftarrow \lceil$(输入序列长度 $\times B$)$/C_{ctx}\rceil$ ▷ 处理全部内容所需的总步数上下文

步骤 2：工作负载分布（步数和令牌）

如果 $B > 1$ **则**

  若 T_{total} ctx $\geq$ 输出序列长度 **则**

    ▷ 上下文占主导；生成槽位有限

    $T_{mix} \leftarrow T_{total}$ 上下文

    $T_{gen} \leftarrow 0$

    $N^{ctx}_{mix} \leftarrow C_{ctx}$ ▷ 每步令牌数

    $N^{gen}_{混合} \leftarrow \max(1, \lfloor B/(T_{total_ctx}/$输出序列长度$)\rfloor)$ ▷ 令牌每步

  else

    ▷ 标准连续批处理

    $T_{mix} \leftarrow T_{total}$ 上下文

    $T_{gen} \leftarrow$ 输出序列长度 $- T_{mix}$

    $N^{ctx}_{mix} \leftarrow C_{ctx}$

    $N^{gen}_{混合} \leftarrow B - \lceil C_{ctx}/$输入序列长度$\rceil$ ▷ 填充剩余槽位

    断言 $N^{gen}_{混合} \geq 1$

  结束 if

否则

  ▷ 单批次（$B = 1$）

  $T_{mix} \leftarrow 1$, $T_{gen} \leftarrow$ 输出序列长度 $- 1$

  $N^{ctx}_{mix} \leftarrow C_{ctx}$, $N^{gen}_{混合} \leftarrow 0$

结束 if

步骤 3：延迟计算

$L_{mix} \leftarrow$ GETMIXLAT(N^{ctx}_{mix}, $N^{gen}_{混合}$，输入序列长度，输出序列长度)

$L_{gen} \leftarrow$ GETGENLAT(B，输入序列长度，输出序列长度)

步骤 4：首令牌时间估算

最小值$_{corr} \leftarrow$ 最小值 $\left(2 + \frac{\text{总计_上下文}}{20}, 3, 4\right)$

首令牌时间 $\leftarrow L_{mix} \times \lceil$输入序列长度$/C_{ctx}\rceil \times F_{corr}$

步骤 5：每输出令牌时间估算

$T'_{mix} \leftarrow \max(1, T_{mix} - 3)$

若 $B > 1$ 则

  每输出令牌时间 $\leftarrow \frac{Lmix \times T'mix + Lgen \times Tgen}{T'mix + Tgen}$

否则

  每输出令牌时间 $\leftarrow L_{gen}$

结束 if

返回首令牌时间 ，TPOT

4.2.3 解耦模式

算法3 解耦模式性能估算

要求： 候选配置：C^{pre} (预填充)，C_{dec} (解码)

要求： 约束：最大首令牌时间（L^{TTFT}_{limit}），最大每输出令牌时间（L^{TPOT}_{limit}）

要求： 有效 GPU 总数：G_{valid}

要求： 退化因子：$\alpha_{pre} = 0.9$，$\alpha_{dec} = 0.92$

要求： 首令牌时间校正因子：$\beta_{TTFT} = 1.8$

步骤 1：按延迟筛选候选项

  ▷ 筛选预填充配置（c^{pre}）和解码配置（c_{dec}）

$C'_{pre} \leftarrow \{c \in C_{pre} \mid (L^{pre}(c) \times \beta_{TTFT}) \leq L^{TTFT}_{limit}\}$

$C'_{dec} \leftarrow \{c \in C_{dec} \mid L_{dec}(c) \leq L^{TPOT}_{limit}\}$

步骤 2：速率匹配（查找最优 x、y）

最佳配置 $\leftarrow \emptyset$，最大吞吐量$_{GPU} \leftarrow 0$

对于 $c_{dec} \in C'_{dec}$ **执行**

  对于 $c_{pre} \in C'_{pre}$ **do**

    吞吐量$_{pre} \leftarrow$ 序列吞吐量(c^{pre})

    序列吞吐量$_{dec} \leftarrow$ SeqThroughput(c_{dec})

    图形处理器$_{pre} \leftarrow$GPUs(c^{pre})，图形处理器$_{dec} \leftarrow$GPUs(c_{dec})

    ▷ 扫描工作进程数量 x （预填充）和 y（解码）

    对于 $x \in [1,32]$、$y \in [1,64]$ **执行**

      $G_{total} \leftarrow x \cdot G_{pre} + y \cdot G_{dec}$

      如果 $G_{total} \notin G_{valid}$ **，则继续**

      结束如果

      $R_{pre} \leftarrow Thru_{pre} \cdot x \cdot \alpha_{pre}$

      $R_{dec} \leftarrow Thru_{dec} \cdot y \cdot \alpha_{dec}$

      $R_{sys} \leftarrow$ 最小值（R^{pre}，R_{dec}） ▷ 系统速率

      吞吐量$_{GPU} \leftarrow R_{sys}/G_{total}$

      如果 吞吐量$_{GPU} >$ 最大吞吐量$_{GPU}$ **则**

        最大吞吐量$_{GPU} \leftarrow$ 吞吐量$_{GPU}$

        最佳配置 $\leftarrow$ {首令牌时间：L_p 再(c_{pre})，每输出令牌时间：$L_{dec}(c_{dec})$ ...}

      结束条件判断

    结束循环

  结束循环

结束循环

返回 $BestConfig$

与静态模式和聚合模式不同，解耦模式采用两个不同的工作池，各自专用于大语言模型推理的特定阶段。如图 3(C) 所示，传入的请求首先由专用的预填充工作节点处理。预填充阶段完成后，计算得到的键值（KV）缓存和中间状态将传输至解码工作节点，后者以自回归方式生成后续令牌。

这种解耦带来了显著的架构优势：它消除了预填充-解码干扰，并允许每个资源池根据各自的工作负载特征采用不同的模型并行策略，从而分别优化计算受限的预填充和内存受限的解码。

先前研究 [28]表明，这种分离能够显著提升整体系统的有效吞吐量。

从建模角度来看，AIConfigurator 分两个阶段估算解耦系统的性能。首先，分别对预填充和解码配置的搜索空间进行扫描，将每个候选配置视为独立的静态实例，并使用算法1估算其基础延迟。对于预填充延迟，AIConfigurator 会应用校正因子（β_{TTFT}），以计入解耦架构固有的 KV缓存传输开销。

其次，通过**速率匹配**流程构建有效的复合服务器，表示为（x）P（y）D，其中x 和y 分别表示预填充和解码实例的数量。该算法通过最大化有效的**每GPU吞吐量**（$Thru_{GPU}$）来确定最优配置，该吞吐量由系统速率R_{sys}推导得出：

$$R_{sys} = \min(R_{pre}, R_{dec})$$

其中，R 表示相应工作池的请求速率（每秒请求数），并根据干扰因子进行折减α。确定最优的$(x)P(y)D$ 配置后，系统的**首令牌时间** 根据所选预填充工作池的延迟（包括传输开销）得出，而**每输出令牌时间**则由解码工作池的延迟决定。算法3详细介绍了这一优化流程。

4.3 迭代级建模

迄今为止介绍的这三种算法，其有效性都取决于我们能否准确预测给定推理迭代步骤的延迟。例如，在算法 1 中，我们依靠通过 GETSTEPLATENCY(批次_大小，序列_长度，阶段) 获取的仅预填充步骤和解码步骤的延迟数据进行进一步估算；而在算法2 中，除了仅解码步骤的延迟（GETGENLAT(N_{gen}，输入序列长度，输出序列长度）之外，我们还必须准确测量混合步骤的延迟（GETMIXLAT(N_{ctx}，N_{gen}，输入序列长度，输出序列长度）），才能继续推进。因此，为了建立一种快速而稳健的方法来估算给定推理步骤的延迟，AIConfigurator 充分利用了现代大语言模型推理可分解的特性来解决这一问题。

4.3.1 将迭代分解为算子

由于现代大语言模型由重复的 Transformer 层构成，因此每个推理迭代步骤都可以建模为按固定顺序运行一系列算子若干次，运行次数通常取决于模型所包含的层数。引入现代并行策略不会改变这一基本特性，只会在迭代执行路径中的固定位置插入少量定义明确的通信算子，并通过将输入分片到多个计算设备上来缩减计算算子的规模。

例如，图 4 展示了使用混合专家模型进行推理时算子的一种典型组合。整个推理步骤本质上是按一定次数运行 4 种算子，即嵌入、通用矩阵乘法、注意力（根据阶段不同，执行预填充或解码）以及混合专家。若加入专家并行，则会缩小混合专家算子的规模，同时增加一对通信算子；具体使用哪一对，取决于生产环境所用的推理引擎后端。

根据上述观察，在 AIConfigurator 中，我们通过汇总构成推理迭代步骤的各个算子的性能配置文件，对该步骤的延迟进行建模。

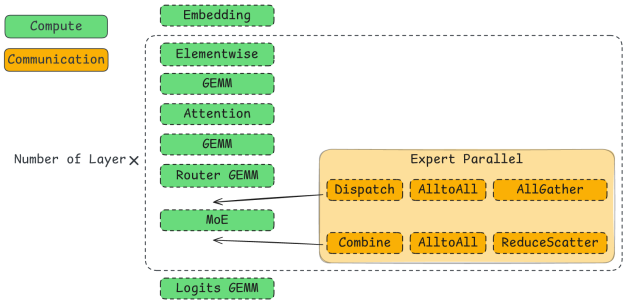


图4：一个大语言模型推理步骤可分解为少数几个关键算子的重复执行。例如，典型混合专家模型的推理步骤通常涉及上图所示的算子，而专家并行的实现方式取决于所使用的具体后端。

4.4 算子数据库

AIConfigurator通过在实际GPU硬件上进行离线性能分析来构建性能数据库，目前支持TensorRT-LLM、vLLM和SGLang。

数据库覆盖范围。数据库包括：(1) 通用矩阵乘法操作，按维度(M,N,K)和量化方式（FP16、FP8、INT8、INT4）进行参数化；(2) 注意力操作，涵盖计算受限的上下文注意力和内存受限的生成注意力，支持多头注意力 [22],分组查询注意力 [7], 和多头潜在注意力 [9]；(3)通信原语，包括不同消息大小和图形处理器数量下的AllReduce、全收集、全互换及点对点传输；(4) MoE操作，包括分发/合并 [25]模式；以及(5)硬件规格（内存带宽、计算吞吐量、互连带宽）。

数据收集。我们结合了三种策略：穷举式性能分析使用框架原生工具扫描参数（批次大小、序列长度、隐藏维度），每个平台与框架组合耗费 ~30 GPU 小时；插值利用已分析的数据点估算中间配置的延迟；光速估计则通过屋顶线模型，为未分析算子提供理论边界。

4.4.1 混合专家的幂律校正

混合专家算子的性能高度依赖令牌分布。先前研究 [10, 14] 表明，在推理期间，某些专家接收的令牌数量远超其他专家——对 Qwen3-235B 的观察显示，~70 % 的计算由仅占活跃专家 20% 的专家承担。为应对这种不均衡，AIConfigurator 实现了一种受控的令牌分配过程，用于模拟生产环境中观察到的幂律分布（图5）。

步骤 1：采样专家负载权重。我们通过从幂律分布中采样 E 个权重（每个专家对应一个），生成负载配置。使用逆变换采样法，并令 $U \sim$ 服从均匀分布 Uniform(0,1)，即可计算每个原始权重：

$$x_i = \left[(x_{\max}^{1-\alpha} - x_{\min}^{1-\alpha}) \cdot U + x_{\min}^{1-\alpha} \right]^{\frac{1}{1-\alpha}} \tag{3}$$

其中，$x_{\min}$ 和 $x_{\max}$ 定义了分布边界，α控制不均衡程度。随后对这些权重进行归一化，以获得每个专家的令牌数：

$$N_i = \text{round}\left(\frac{x_i}{\sum_{j=0}^{E-1} x_j} \times T_{\text{total}} \times K \right) \tag{4}$$

其中，T_{total} 是批次大小，K 是 top-k 路由因子（每个令牌路由至 K 个专家），N_i 是分配给专家 i 的令牌数。通过分配剩余令牌来平衡总数。参数 α 控制偏斜程度：$\alpha \approx 0$ 会产生近乎均匀的负载（理论上的理想情况），而 $\alpha \approx 1.2$ 会产生重尾分布，少数“热门”专家会接收大多数令牌——这与 Qwen3-235B 等模型中的观察结果相符。

步骤 2：构造合成路由器分配。在正常推理过程中，学习型门控网络会将每个令牌路由至其对应的专家。为了进行可控基准测试，我们绕过该路由器，直接注入合成分配矩阵 $\mathbf{L} \in \mathbb{R}^{T_{\text{total}} \times E}$，其中恰好有 N_i 个令牌被确定性地路由至专家 i。这样可以消除路由器带来的随机方差，并确保硬件执行步骤 1 中设定的精确工作负载形态，从而测得由负载最重的专家造成的“尾延迟”；在实际应用中，它决定了整体吞吐量。

5 评估

我们从三个维度评估 AIConfigurator：(1) 与真实硬件测量结果相比的预测保真度，(2) 相较于穷举基准测试的搜索效率，以及 (3) 在生产环境场景中的实际性能提升。每个小节都会介绍各自的实验设置。

5.1 聚合服务保真度

我们首先评估 AIConfigurator 在聚合（连续批处理）服务模式下的预测准确率；这是生产环境大语言模型推理中最常见的部署模式。

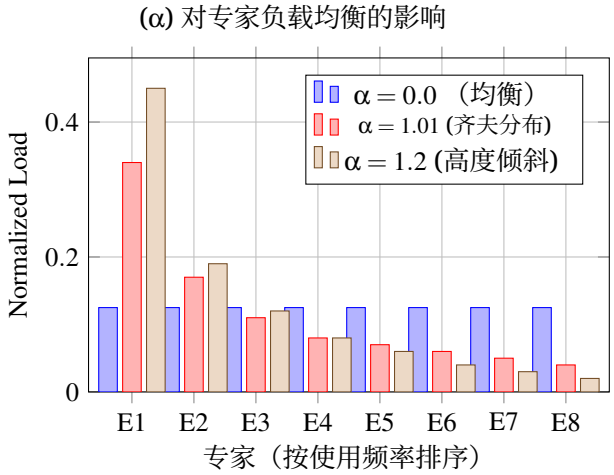


图 5：可视化 α 的影响。随着 α 增加，路由分布从完全均衡（均匀分布）转变为高度倾斜，此时排名靠前的专家（E1、E2）处理了大多数令牌。

实验设置。实验在一台配备 8 个图形处理器（每个显存为 80GB HBM3）、通过 NVSwitch 互连的 NVIDIA H100 SXM 节点上进行。我们评估了两种规模和架构各异的模型：**通义千问 3-32B**（320 亿参数、稠密模型、FP8）和**通义千问3-235B**（2350 亿参数、混合专家模型，包含 128 个专家、FP8）。所有模型均使用 TensorRT-LLM v1.0.0 提供服务。为评估跨框架泛化能力，我们还使用**vLLMv0.11.0**对**通义千问3-32B** 进行了评估。

我们全面遍历了配置空间，以在各种运行条件下对预测准确率进行压力测试：

- **输入序列长度（ISL）：** 128–4096 个令牌
- **输出序列长度（OSL）：** 128–512 个令牌
- **并发数：** 4–128 个并发请求
- **张量并行（TP）：** 1、2、4、8 个 GPU
- **专家并行（EP）：** 1、2、4、8 个 GPU（适用于 Qwen3-235B）

这为 TensorRT-LLM 提供了**960 种独特配置**（Qwen3-32B 有 360 种，Qwen3-235B 有 600 种），此外还为 vLLM 提供了 128 种配置，涵盖从注重延迟的聊天到注重吞吐量的批处理等工作负载。

指标。我们使用平均绝对百分比误差（MAPE）衡量预测保真度，针对两项关键延迟指标：每输出令牌时间（TPOT），它决定生成吞吐量；以及首令牌时间（TTFT），它决定用户感知的响应速度。

结果。图 6 展示了两个框架的保真度分析。对于每输出令牌时间预测，AIConfigurator 的总体平均绝对百分比误差为 **7.8%**，且所有模型均呈现出较强的相关性：Qwen3-32B-TRTLLM（8.2%，r=0.96）、Qwen3-235B-MoE-TRTLLM（6.8%，r=0.98）以及 Qwen3-32B-VLLM（11.9%，r=0.99）。混合专家模型的误差最低，验证了我们针对专家负载不均衡的幂律工作负载建模。在

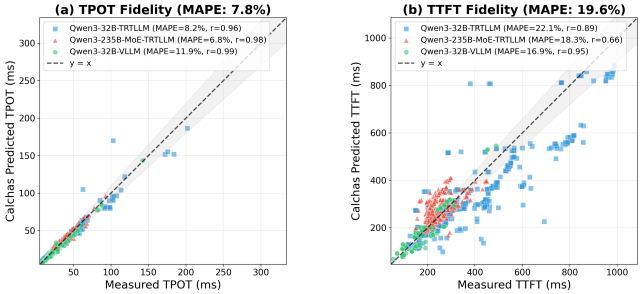


图6：H100 SXM 上 TensorRT-LLM 与 vLLM 聚合式服务的预测保真度。每个点代表一种配置；对角线表示完美预测。TTFT 值为 >1000ms 的数据被作为离群值过滤。

TensorRT-LLM 和 vLLM 之间保持一致的准确性，表明我们的算子级建模具有良好的泛化能力——无论使用何种框架，核心计算模式（通用矩阵乘法、注意力、通信）都是可预测的。

TTFT 预测同样表现出很高的保真度：Qwen3-32B-TRTLLM 的平均绝对百分比误差为 22.1%（r=0.89），Qwen3-235B-MoE-TRTLLM 为 18.3%（r=0.66），Qwen3-32B-VLLM 为**16.9%**（平均绝对百分比误差 r=0.95）。值得注意的是，vLLM 的 TTFT 预测准确率最高，这可能是因为与 TensorRT-LLM 的批处理启发式策略相比，其预填充调度更可预测。极端离群值（TTFT >1000ms）已被排除，因为它们代表的是异常排队延迟，而非稳态运行。两项指标均呈现出很强的相关性，证实了 AIConfigurator 在异构框架部署中进行配置选择的实用性。

5.2 解耦式服务保真度

除了聚合模式之外，我们还评估了 AIConfigurator 对多节点解耦式服务的预测保真度，其中预填充和解码阶段分别在独立的 GPU 池上运行。

设置。 实验在两个计算节点上进行，每个节点均配备 8 个通过 NVLink 互连的 NVIDIA Hopper GPU。一个节点专用于预填充，另一个节点负责解码，并分别为每个节点独立配置并行度。我们使用完整规模的 **DeepSeek-V3** [16] 模型（671B 参数，MoE），并采用 **TensorRT-LLM** 作为服务后端。

不同于聚合模式下测量每个请求的 TTFT 和 TPOT，解耦式服务需要采用系统级指标：*生成速度*（令牌/秒/用户）和*系统吞吐量*（总令牌/秒）。我们分两步评估 AIConfigurator：1. **配置搜索：** AIConfigurator 在两个输入配置（ISL 为 5k 和 6k 令牌，OSL 为 1k 令牌）下，在 5 秒 TTFT 约束条件内探索配置空间。搜索空间包括：

- **张量并行（TP）：** 1, 2,4,8
- **数据并行（DP）：** 1、2、4、8
- **专家并行（EP）：** 1、2、4、8

超出显存容量的配置会被自动剪枝。此次扫描生成了吞吐量与速度权衡的最优帕累托前沿。

2.**真实值验证：** 我们在实际的 TensorRT-LLM 上对每种帕累托最优配置进行了基准测试，并测量了 AIConfigurator预测与真实值测量结果之间的平均绝对百分比误差。

结果。 图 7 比较了 AIConfigurator 的预测与 TensorRT-LLM 的测量结果。在所有配置中，系统吞吐量的平均绝对百分比误差为 **25.49%**，生成速度的平均绝对百分比误差为 **14.94%**。值得注意的是，在每秒每用户 25–50 个令牌的交互速度区间（由绿色虚线标示）内——这一速度对大多数用户而言足以舒适阅读——预测吻合度显著提高：吞吐量的平均绝对百分比误差降至 **13.19%**，生成速度的平均绝对百分比误差降至 **3.35%**。这表明，AIConfigurator预测在最关键的区域最为准确，也就是生产部署通常瞄准的运行区间。

5.3 搜索效率

表1：H100 SXM 上 AIConfigurator 与 GPU 基准测试的配置搜索效率。

模型	AIConfigurator	GPU基准测试	加速比
总耗时（所有配置）			
Llama3.1-8B（339 个配置）	0.52s	24.4 小时	171,000×
Qwen3-32B FP8（358 个配置）	0.72 秒	35.4 小时	177,000×
Qwen3-235B FP8（506 个配置）	0.84 秒	99.5 小时	427,000×
每种配置的中位时间			
Llama3.1-8B	1.5毫秒	4.0 分钟	162,000×
通义千问3-32B FP8	1.5毫秒	5.4分钟	214,000×
Qwen3-235B FP8	1.5毫秒	11.5分钟	459,000×

AIConfigurator 的一项关键优势是能够在 CPU 上探索配置空间，从而无需进行穷尽式 GPU 基准测试。为量化这一效率提升，我们比较了 AIConfigurator 评估配置搜索空间所需的时间与基准测试相同配置所需的实际耗时。

表1 展示了三种复杂度各异的模型之间的效率对比。请注意，“GPU 时间”表示端到端实际耗时，其中包括权重加载、服务器启动和基准测试执行。AIConfigurator 在 CPU 上不到一秒即可完成配置搜索，而在 GPU 上进行同等的基准测试则需要数天的 GPU 时间。例如，对于 Qwen3-235B（506 种配置），AIConfigurator 实现了 427，000× 倍加速比（0.8 秒，而 GPU 基准测试需要 99.5 GPU 小时）。

值得注意的是，无论模型规模如何，每种配置的模拟时间中位数都保持在 ∼1.5ms，因为 AIConfigura- tor 的算子级数据库查询随模型架构而变化，而非随参数量变化。相比之下，每种配置的

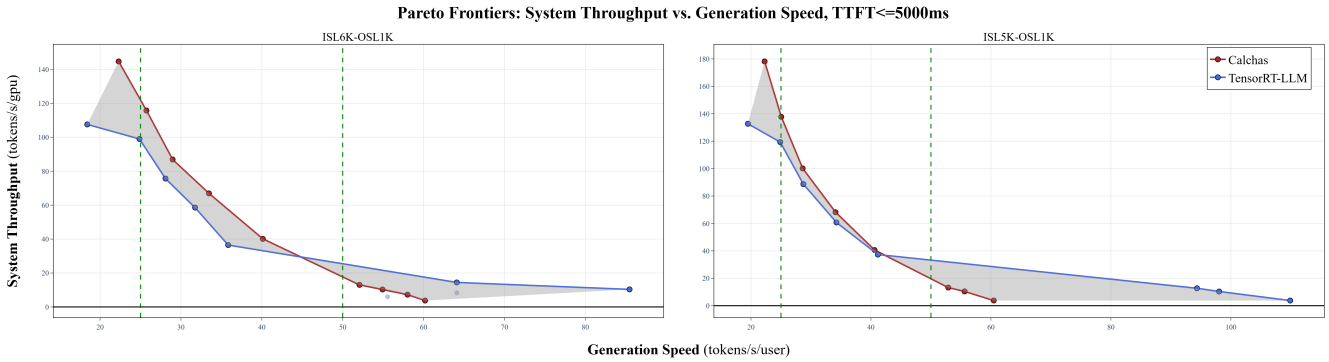


图7：在两个节点上采用预填充/解码分离部署 DeepSeek V3 时，AIConfigurator 预测的帕累托前沿（红色）与 TensorRT-LLM 的真实值（蓝色）对比。阴影区域表示 AIConfigurator 的预测与框架实际表现之间的差异。

基准测试时间会因模型复杂度提高而延长，这是由于权重加载和推理时间更长，范围为 4 至 11.5 分钟。

这种效率使从业者能够快速迭代部署场景——调整SLA目标、探索不同的硬件分配，或评估新的模型变体——而无需承担高昂的GPU成本。

5.4 案例研究：寻找最优方案

我们通过为生产部署场景寻找最优服务配置，展示 AIConfigurator 的实际价值，并在真实的 SLA约束下比较聚合式服务与解耦式服务。

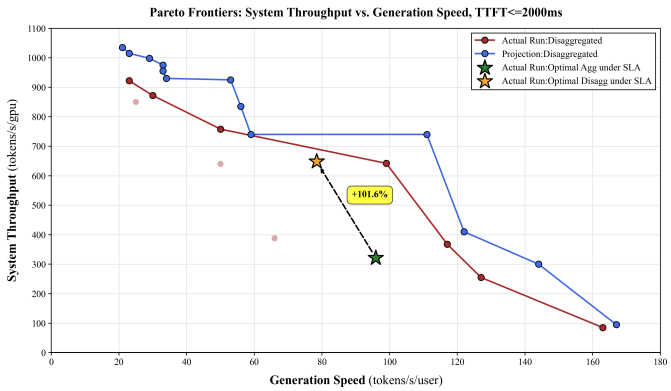


图8：Qwen3-32B-FP8 在 8 块 H200 GPU 上的 AIConfigurator 预测与真实测量结果对比。帕累托前沿展示了吞吐量与速度的权衡；在满足 SLA约束的同时，解耦式服务的吞吐量比最优聚合式配置高出 2×。

设置。我们的目标是满足以下服务等级协议要求：首令牌时间 ≤1200ms，生成速度 ≥ 60 令牌/秒/用户，使用 8 块 NVIDIA H200 SXM GPU。工作负载包含输入序列长度 = 4000 和输出序列长度 = 500 令牌。所有实验均使用 NVIDIA Dynamo 搭配 TensorRT-LLM 后端（v0.5.0）和 AI-Perf [6] 进行负载测试。请求并发度与最大批次大小相同，以最大化吞吐量，并采用 20×过采样来减轻预热效应对首令牌时间测量的影响。

聚合式基线。使用 AIConfigurator，我们确定了最优的聚合配置：单个 TP2 实例，批次大小为 8，实现了 321.5 令牌/秒/GPU 和 95.9 令牌/秒/用户（表 2）。通过对所有有效的 TP 和批次大小组合进行穷举基准测试，我们验证了该配置是全局最优解。

解耦优化。AIConfigurator 在数十秒内探索了解耦配置空间，并确定了预填充/解码拆分方案：4 个预填充副本（TP1）和 2 个解码副本（TP2），批次大小分别为 1 和 80。此配置实现了 **648.3令牌/秒/GPU**——相较于聚合式基线，**吞吐量提升 101.6%** ——同时满足所有 SLA约束（表2）。

预测准确度。为了验证 AIConfigurator 在帕累托前沿上的预测结果，我们在放宽至 2000ms 的首令牌时间约束下，对所有推荐配置进行了基准测试（图 8）。AIConfigurator 预测的前沿与真实测量结果高度吻合：在并发数相同的情况下，生成速度的最大偏差为 11.2%，系统吞吐量的最大偏差为 17.4%。

6 相关工作

大语言模型推理系统。现代服务架构已从静态批处理发展为通过 Orca 实现的动态调度 [24](连续批处理)、vLLM [13] (分页注意力)、SGLang [27](基数注意力)和 TensorRT-LLM [2] (优化内核)。模型并行技术——张量并行 [21], 流水线并行 [12],专家并行 [16]——通过复杂的性能权衡将大型模型分布到多个 GPU 上。解耦架构 [18,19,28] 将预填充/解码阶段分离，以便独立扩展。AIConfigurator 面向这些系统，自动选择配置，而非提出新的调度算法。

性能仿真。Vidur [4] 和 APEX [15] 使

表 2：在生产服务级别协议（首令牌时间 $\leq$ 1200ms、速度 $\geq$ 60 tokens/s/user）下，8 张 H200 GPU 上 Qwen3-32B-FP8 的最佳聚合与解耦配置。P = 预填充，D = 解码。

Mode	吞吐量（令牌/秒/GPU）	速度（令牌/秒/用户）	首令牌时间（毫秒）	批次大小	配置
聚合	321.5	95.9	1017.5	8	1 ×张量并行2
解耦	648.3	78.4	1068.9	P：1，D：80	P：4 × 张量并行1，D：2 ×张量并行2

能够通过离散事件仿真和成本优化快速探索配置。然而，这些方法依赖于解析屋顶线模型，无法体现特定框架的行为。AIConfigurator 的不同之处在于其数据驱动的基础：在目标硬件上测量实际执行时间，并组合这些测量结果来预测端到端性能，从而捕捉解析模型所忽略的实现特定开销。与 Inference- Max 等静态资源不同，[20] 基准测试和部署方案 [3],AIConfigurator 提供算法搜索，可推广至全新的工作负载组合。

7 结论

我们介绍了 AIConfigurator，这是一款数据驱动的工具包，用于优化 TensorRT-LLM、vLLM 和 SGLang 中的大语言模型推理配置。通过将推理分解为基本操作，并测量这些操作在目标硬件上的延迟，AIConfigurator 能够实现高保真度的性能预测，捕捉分析模拟器所忽略的框架特定开销。该工具包可在 CPU 上于数秒内评估数千种配置，从而无需进行成本高昂的 GPU 基准测试。

我们的评估表明，在稠密架构和混合专家架构中，预测准确率均表现出色（每输出令牌时间的平均绝对百分比误差为 6–12%）；生产案例研究显示，通过自动化解耦配置发现， 2× 吞吐量得以提升。与 NVIDIA Dynamo 集成后，还能顺畅部署优化配置。未来的工作包括扩展到更多硬件平台、纳入成本模型，以及支持推测解码和稀疏注意力等新兴技术。

参考文献

[1] GitHub - ai-dynamo/dynamo：数据中心级分布式推理服务框架 — github.com。 https://github.com/ ai-dynamo/dynamo。 [访问日期：23-11-2025]。[2] GitHub - NVIDIA/TensorRT-LLM：TensorRT LLM 为用户提供易于使用的 Python API，用于定义大语言模型（LLM），并支持最先进的优化方法，以便在 NVIDIA GPU 上高效执行推理。TensorRT LLM 还包含用于创建 Python 和 C++ 运行时的组件，这些组件能够以高性能方式编排推理执行。 — github.com。 https://github.com/NVIDIA/TensorRT-LLM。 [访问日期：23-11- 2025]。[3] GitHub - vllm-project/recipes：运行 vLLM 的通用方案 — github.com。 https://github.com/vllm-project/recipes。 [访问日期：23-11-2025]。

[4] AGRAWAL, A ., KEDIA, N ., MOHAN, J ., PANWAR, A ., KWATRA, N ., GULAVANI, B. S.、RAMJEE, R. 和 TUMANOV, A. Vidur：用于大语言模型推理的大规模模拟框架。《机器学习与系统会议论文集》第 *6* 卷（2024 年），351–366。

[5] AGRAWAL, A.、KEDIA, N.、PANWAR, A.、MOHAN, J.、KWATRA, N.、GULAVANI, B.、TUMANOV, A. 和 RAMJEE, R. 使用 {Sarathi- Serve} 驯服{吞吐量-延迟} 权衡，实现 {大语言模型} 推理。载于 第 *18* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 24*）（2024 年），第 117–134 页。

[6] AI Dynamo Team。Aiperf：用于生成式人工智能模型的综合基准测试工具。 https://github.com/ai-dynamo/aiperf，2025 年。访问日期：2025-12-11。

[7] AINSLIE, J., LEE-THORP, J., DE JONG, M., ZEMLYANSKIY, Y., LE- BRÓN, F., AND SANGHAI, S. 分组查询注意力：从多头检查点训练广义多查询变换器模型，2023年。

[8] DAO, T ., FU, D ., Y., ERMON, S ., RUDRA, A ., AND RÉ, C . FlashAtten - tion：具备 I/O 感知能力的快速且高效利用内存的精确注意力机制。载于神经信息处理系统进展（*NeurIPS*）（2022年）。

[9] DEEPSEEK-AI、LIU, A.、FENG, B.、WANG, B.、WANG, B.、LIU, B.、ZHAO, C.、DENGR, C.、RUAN, C.、DAI, D.、GUO, D.、YANG, D.、CHEN, D.、JI, D.、LI, E.、LIN, F.、LUO, F.、HAO, G.、CHEN, G.、LI, G.、ZHANG, H.、XU, H.、YANG, H.、ZHANG, H.、DING, H.、XIN, H.、GAO, H.、LI, H.、QU, H.、CAI, J. L.、LIANG, J.、GUO, J.、NI, J.、LI, J.、CHEN, J.、YUAN, J.、QIU, J.、SONG, J.、DONG, K.、GAO, K.、GUAN, K.、WANG, L.、ZHANG, L.、XU, L.、XIA, L.、ZHAO, L.、ZHANG, L.、LI, M.、WANG, M.、ZHANG, M.、ZHANG, M.、TANG, M.、LI, M.、TIAN, N.、HUANG, P.、WANG, P.、ZHANG, P.、ZHU, Q.、CHEN, Q.、DU, Q.、CHEN, R. J.、JIN, R. L.、GE, R.、PAN, R.、XU, R.、CHEN, R.、LI, S. S.、LU, S.、ZHOU, S.、CHEN, S.、WU, S.、YE, S.、MA, S.、WANG, S.、ZHOU, S.、YU, S.、ZHOU, S.、ZHENG, S.、WANG, T.、PEI, T.、YUAN, T.、SUN, T.、XIAO, W. L.、ZENG, W.、AN, W.、LIU, W.、LIANG, W.、GAO, W.、ZHANG, W.、LI, X. Q.、JIN, X.、WANG, X.、BI, X.、LIU, X.、WANG, X.、SHEN, X.、CHEN, X.、CHEN, X.、NIE, X.、SUN, X.、WANG, X.、LIU, X.、XIE, X.、YU, X.、SONG, X.、Z HOU, X.、YANG, X.、LU, X.、SU, X.、WU, Y.、LI, Y. K.、WEI, Y. X.、ZHU, Y. X.、XU, Y.、HUANG, Y.、LI, Y.、ZHAO, Y.、SUN, Y.、LI, Y.、WANG, Y.、ZHENG, Y.、ZHANG, Y.、XIONG, Y.、ZHAO, Y.、HE, Y.、TANG, Y.、PIAO, Y.、DONG, Y.、TAN, Y.、LIU, Y.、WANG, Y.、GUO, Y.、ZHU, Y.、WANG, Y.、ZOU, Y.、ZHA, Y.、MA, Y.、YAN, Y.、YOU, Y.、LIU, Y.、REN, Z. Z.、REN, Z.、SHA, Z.、FU, Z.、HUANG, Z.、ZHANG, Z.、XIE, Z.、HAO, Z.、SHAO, Z.、WEN, Z.、XU, Z.、ZHANG, Z.、LI, Z.、WANG, Z.、GU, Z.、LI, Z.、以及 XIE, Z.《DeepSeek-V2：一种强大、经济且高效的混合专家语言模型》，2024年。

[10] FEDUS, W., ZOPH, B., AND SHAZEER, N. Switch Transformer：以简单高效的稀疏性扩展至万亿参数模型。机器学习研究杂志 *23*, 120 (2022年), 1–39.

[11] GOLOVIN, D., SOLNIK, B., MOITRA, S., KOCHANSKI, G., KARRO, J., AND SCULLEY, D. Google Vizier：一项用于黑盒优化的服务。载于第*23*届 *ACM SIGKDD* 知识发现与数据挖掘国际会议论文集(2017年), pp. 1487–1495.[12] HUANG, Y., CHENG, Y., BAPNA, A., FIRAT, O., CHEN, D., CHEN, M., LEE, H., NGIAM, J., LE, Q. V., WU, Y., ET AL. Gpipe：使用流水线并行高效训练巨型神经网络。神经信息处理系统进展 *32*(2019年).[13] KWON, W., LI, Z., ZHUANG, S., SHENG, Y., ZHENG, L., YU, C. H., GONZALEZ, J., ZHANG, H., AND STOICA, I. 使用分页注意力高效管理大语言模型服务的内存。载于第*29*届操作系统原理研讨会论文集(2023年), pp. 611–626.[14] LEPIKHIN, D., LEE, H., XU, Y., CHEN, D., FIRAT, O., HUANG, Y., KRIKUN, M., SHAZEER, N., AND CHEN, Z. GShard：通过条件计算和自动分片扩展巨型模型。载于国际学习表征会议(2021年).[15] LIN, Y.-C., KWON, W., PINEDA, R., AND PARAVECINO, F. N. APEX：面向大语言模型服务自动并行执行的可扩展动态感知模拟器。*arXiv* 预印本 *arXiv:2411.17651*(2024年).

[16] LIU, A., FENG, B., XUE, B., WANG, B., WU, B., LU, C., ZHAO, C., DENG, C., ZHANG, C., RUAN, C., 等. Deepseek-v3技术报告。*arXiv*预印本 *arXiv:2412.19437*（2024年）。

[17] MIAO, X., WANG, Y., JIANG, Y., SHI, C., ZHU, X., ZHANG, Z., 和 CUI, B. Galvatron：利用自动并行实现多GPU上的高效Transformer训练。*VLDB*基金会会刊 *16*，3 2022年 470 479（），–。[18] PATEL, P., CHOUKSE, E., ZHANG, C., SHAH, A., GOIRI, Í., MALEKI, S., 和 BIANCHINI, R. Splitwise：利用阶段拆分实现高效生成式大语言模型推理。载于*2024*年 *ACM/IEEE* 第*51*届年度国际计算机体系结构研讨会（*ISCA*）（2024年），IEEE，第118–132页。[19] QIN, R., LI, Z., HE, W., CUI, J., TANG, H., REN, F., MA, T., CAI, S., ZHANG, Y., ZHANG, M., 等. Mooncake：面向大语言模型服务、以KV缓存为中心的解耦架构。*ACM*存储汇刊（2024年）。

[20] SemiAnalysis。SemiAnalysis 推出的 InferenceMAX — inference-max.semianalysis.com。 https://inferencemax.semianalysis. com/。 [访问日期：23-11-2025]。

[21] SHOEYBI, M., PATWARY, M., PURI, R., LEGRESLEY, P., CASPER, J., 和 CATANZARO, B. Megatron-lm：使用模型并行训练数十亿参数语言模型。*arXiv* 预印本 *arXiv:1909.08053*（2019年）。

[22] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L., 和 POLOSUKHIN, I. Attention Is All You Need，2023年。[23] WANG, L., YANG, J., LUO, Y., 和 WANG, D. Morphling：面向云原生模型服务的快速、近乎最优自动配置。载于*2021*年*ACM*云计算研讨会论文集（2021年），第63–77页。[24] YU, G.-I., JEONG, J. S., KIM, G.-W., KIM, S., 和 CHUN, B.-G. Orca：一种用于 {基于Transformer的} 生成模型的分布式服务系统。载于第*16*届*USENIX*操作系统设计与实现研讨会（*OSDI 22*），第521–538页。[25] ZHAO, C., ZHOU, S., ZHANG, L., DENG, C., XU, Z., LIU, Y., YU, K., LI, J., 和 ZHAO, L. DeepEP：一种高效的专家并行通信库。 https://github.com/deepseek-ai/DeepEP，2025年。[26] ZHENG, L., LI, Z., ZHANG, H., ZHUANG, Y., CHEN, Z., HUANG, Y., WANG, Y., XU, Y., ZHUO, D., XING, E. P., 等。Alpa：自动化分布式深度学习的算子间与算子内并行。载于第*16*届*USENIX*操作系统设计与实现研讨会（*OSDI 22*）（2022年），第559–578页。

[27] ZHENG, L., YIN, L., XIE, Z., HUANG, J., SUN, C., YU, C., CAO, S., KOZYRAKIS, C., STOICA, I., GONZALEZ, J. E., 等。使用SGLang高效编程大语言模型。[28] ZHONG, Y., LIU, S., CHEN, J., HU, J., ZHU, Y., LIU, X., JIN, X., 和 ZHANG, H. DistServe：将预填充与解码分离，以实现吞吐效率优化的大语言模型服务，2024年。