PromptTuner：面向服务级目标的弹性大语言模型提示词调优系统

Wei Gao[1], Peng Sun[2], Dmitrii Ustiugov[1], Tianwei Zhang[1], Yonggang Wen[1]1计算与数据科学学院，南洋理工大学，2无隶属机构 gaow0007@e.ntu.edu.sg，sunpeng1@outlook.com，{dmitrii.ustiugov,tianwei.zhang,ygwen}@ntu.edu.sg

arXiv:2603.05087v1 [cs.DC] 5 Mar 2026

摘要

提示调优已成为提升大语言模型（LLM）在下游任务中表现的一种重要策略。如今，许多 IT 企业提供提示调优即服务，以满足日益增长的需求，即为下游任务调优大语言模型提示。它们的首要目标是在满足用户服务等级目标（SLO）的同时降低资源配置成本。然而，我们对现有深度学习资源管理系统进行的特征分析表明，这些系统不足以针对大语言模型提示调优工作负载优化上述目标。

本文介绍 PromptTuner，一种用于优化大语言模型提示调优的 SLO 感知弹性系统。它包含两项创新。(1) 我们设计了提示库，以识别高效的初始提示，从而加速提示调优的收敛。(2) 我们开发了工作负载调度器，以实现快速资源分配，从而减少 SLO 违约和资源成本。在我们的评估中，与 INFless 和 ElasticFlow 相比，PromptTuner 分别将 SLO 违规降低了 4.0× 和 7.9×，并将成本降低了 1.6× 和 4.5×。

1 引言

大语言模型（LLM）凭借其卓越能力，正在众多场景中日益普及 [16, 40, 68]。大语言模型开发者采用一种引人注目且广受欢迎的技术，即提示调优，在不改变模型权重的情况下，针对多样化应用（尤其是智能体应用）定制大语言模型 [7, 19, 62, 67]。然而，手动提示调优过程既耗时又耗费资源 [28, 83, 95]，这促使许多 IT 公司提供提示调优即服务，以便在数秒到数分钟内实现自动提示调优 [8, 9, 15]。在这一业务模式中，用户提供初始提示和下游数据集，并选择基础大语言模型。随后，服务提供商必须高效分配 GPU，针对给定数据集优化提示，每天处理数以万计的大语言模型提示调优（LPT）请求 [15]，并将最终确定的提示返回给用户。

服务提供商在处理用户的 LPT 请求时需要考虑多个因素。首先，用户关注其 LPT 请求的准确率[1] 和延迟。他们会指定目标准确率和延迟的服务等级目标[2] (SLO) [6, 8, 9, 15]。其次，服务提供商从云平台 [3–5] 租用高端 GPU 资源来处理用户的 LPT 请求。鉴于 LPT 请求数量不断增加，且租用 GPU 的成本相当高，迫切需要设计能够优化 LPT 工作负载资源分配的系统。此类优化旨在满足用户服务等级目标的同时，降低服务提供商的资源成本。

我们在§2.2中给出了 LPT 工作负载的特征分析摘要，并发现它们同时呈现出类似训练和类似推理的特征。一种直接的方法是借鉴以往针对训练和推理工作负载的研究在集群管理系统中的相关研究，以应对 LPT 的需求。然而，我们在§3中的实证研究表明，这些方法在管理 LPT 工作负载方面效果不佳。首先，以往面向深度学习（DL）训练的 SLO 感知系统 [23, 34, 41] 会过度订阅一个**固定**规模的 GPU 集群以保证服务等级目标，从而导致资源成本增加。此外，普遍采用的频繁资源分配可能会为大语言模型带来近一分钟的资源分配开销 [49, 77]，并对为 LPT 工作负载实施分钟级延迟 SLO 构成重大障碍。其次，以往的推理系统 [72, 85, 91, 92] 采用了两种技术：（1）自动扩缩容所需的 GPU 数量，以降低资源成本；（2）在一段时间内预先将深度学习运行时（例如 CUD-A/框架运行时）和模型权重加载到 GPU 内存中，以减少分配开销并优化 SLO 达成。然而，这些方案遵循**固定** 的 GPU 分配策略，通常为每个作业分配一个 GPU，从而削弱了满足 LPT 作业不同 SLO 水平所需的适应性。如§3.2所示，即使将多 GPU 分配纳入深度学习推理系统，它们仍难以有效服务 LPT 作业。总体而言，以往的训练和推理系统在同时实现 LPT 的 SLO 满足度和成本降低方面存在不足。

此外，现有的深度学习集群管理系统和 LPT 服务忽略了 LPT 工作负载的一个独特特征：其模型收敛对初始提示高度敏感 (§2.2)。这种敏感性意味着，使用不同的初始提示达到目标准确率所需的迭代次数会有显著差异。例如，精心设计的初始提示比质量较差的提示所需的调优迭代次数更少，从而减少 SLO 违规和资源成本。在实践中，大语言模型开发者采用两种初始化方法。首先，目前 LPT 服务采用的是手动初始化。用户需要自行编写初始提示 [17, 87]。或者，也会建议用户直接复用公开可用的提示 [1, 8]。然而，这两种做法都依赖人类专业知识、消耗大量 GPU 资源，并需要投入时间进行繁琐的反复试错过程。其次，一些大语言模型研究 [88, 95] 和大语言模型服务 [2] 提出了归纳初始化方法，以引导大语言模型在无需人类专业知识的情况下自动生成初始提示。然而，生成的初始提示的质量在很大程度上取决于大语言模型自身的性能 [88, 95] (在 §6.3 中评估)。尽管这种方法具有潜在优势，但目前很少有系统性工作致力于为给定的 LPT 作业自动、高效地确定初始提示。

为弥合这些差距，我们设计了 PromptTuner，这是一个专为 LPT 打造的 SLO感知弹性集群管理系统。PromptTuner 包含两项设计。首先，我们设计了一个提示库作为查询引擎，用于自动、高效地为给定的 LPT 作业搜索初始提示。我们的观察是，为某个 LPT 任务优化的提示，也可以作为与其高度相似的另一项任务的有效初始提示；这一观察促成了提示库的设计 [75,

[1]我们使用“准确率”作为通用术语，指代任何评估指标。[2]服务等级目标的定义见 §4.2。

80]。随着针对各类任务优化的公开提示词显著增多 [29], ，我们收集了数千条高质量提示，作为新到 LPT 作业的初始提示词候选项。我们采用两层数据结构，以便快速搜索有效的初始提示，将选择时间缩短至 10 秒以内。

其次，我们设计了一种工作负载调度器，支持为 LPT 工作负载快速、弹性地分配 GPU，以满足服务等级目标并降低资源成本。工作负载调度器允许基于同一大语言模型的 LPT 作业复用热 GPU 池中的 GPU ；该池由预加载了相同作业专用大语言模型运行时和权重的 GPU 组成，从而实现快速 GPU 分配。工作负载调度器为每个大语言模型维护一个专用热 GPU 池，并通过从共享冷 GPU 池中添加 GPU，动态调整池的大小。它包含两种用于管理这些 GPU 池的算法。第一种算法将热 GPU 池中的 GPU 快速分配给 LPT 作业，从而减少多 GPU 执行所需的巨大初始化开销（§3.2）。第二种算法则根据 LPT 作业的动态流量，动态调整每个热 GPU 池中的 GPU 数量，以平衡 SLO 达成与资源成本之间的权衡。此外，工作负载调度器会智能地决定是将传入的 LPT 请求路由至提示库，还是直接执行，以避免资源争用。

我们基于 Knative 实现了 PromptTuner，并选取三个大语言模型（GPT2-Base [18],GPT2-Large [18],Vicuna-7B [24]），在由 32 张 NVIDIA A100-80GB GPU 组成的集群上，将 PromptTuner 与最先进的 SLO感知深度学习推理系统 INFless [85] 和训练系统 ElasticFlow [41] 进行比较。PromptTuner 将 SLO 违约最多降低 4.0×（相较于 INFless）和 7.9×（相较于 ElasticFlow），并将成本最多降低 1.6×（相较于 INFless）和 4.5×（相较于 ElasticFlow）。我们还针对 LLaMA-30B 和 DeepSeek-R1-Distill-Qwen-7B（Qwen7B-R1）开展实验，并在 96-GPU集群中进行大规模实验，以证明 PromptTuner 在高负载场景下的优越性。我们的贡献如下：

⋆ 我们深入分析了 LPT 工作负载的特征，并开展了详细的实证研究，以揭示现有系统在处理 LPT 时的低效之处。⋆ 我们提出了 PromptTuner，这是一种面向 LPT 工作负载的弹性系统

能够为用户保障服务等级目标，并降低服务提供商的资源成本。⋆ 我们进行了大量评估，以验证提示库 和工作负载调度器的效率。

2 LPT 工作负载特征分析

2.1 提示调优

提示调优是一种通过附加提示前缀（简称为提示），从大语言模型获取特定任务高质量响应的方法，从而节省重新训练模型权重的高昂成本。LPT 作业会优化一个提示，使其在前置到输入查询时从大语言模型引出最佳响应。图1展示了使用基于梯度的LPT算法 [58]将自然语言转换为SQL语言这一任务的示例。用户发送一个LPT请求，其中包含大语言模型、初始提示（“将自然语言描述优雅且正确地转换为*SQL*”），以及由输入查询和相应目标响应组成的任务特定数据集。一些LPT服务提供商 [1, 8]

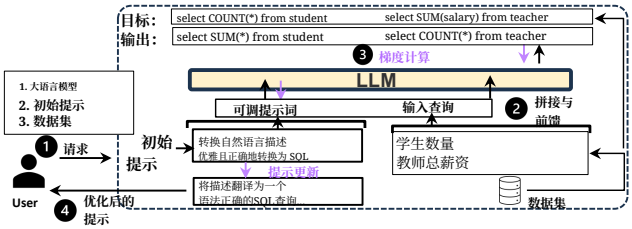


图 1：大语言模型提示调优示例。用户首先准备大语言模型、初始提示和任务特定数据集，该数据集由一批输入查询和目标响应组成。在执行阶段，从初始提示开始，在给定数据集上优化可调提示词。

表 1：提示技术在各任务上的平均得分（↑）。

提示技术	GPT-3.5	GPT-4	Vicuna-7B	LLaMA-30B	Qwen7B-R1
少样本提示	29.8	37.0	14.9	20.8	34.1
提示调优	70.1	75.8	80.5	84.1	85.7

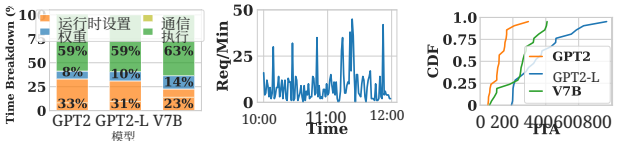
建议用户根据自身专业知识指定其初始提示（ ）。要执行LPT作业，LPT系统会将这组输入查询输入到大语言模型中（ ）。系统运行给定的LPT算法，计算生成与输出句子与目标响应之间的损失。然后反向传播文本梯度，并在可调提示词上对其进行更新（ ）。经过多次迭代后，生成优化后的提示：“将描述翻译成语法正确、针对速度和准确率优化的*SQL*查询”，并返回给用户（ ）。

LPT 工作负载的普遍性。如今，LPT 工作负载已成为 GPU 的重要使用场景，使提示调优服务成为一项不可或缺的业务实践 [8]。当用户发送 LPT请求时，系统会将其登记为 LPT 作业，并安排每个 LPT 作业在 GPU 上运行，同时满足用户设定的严格服务等级目标。

LPT 工作负载的普遍性体现在三个方面。首先，许多提示调优服务 [1, 8, 9, 15, 31, 69, 76]致力于将大语言模型拓展到各个领域，使大语言模型提示市场日益火热、不断增长。大语言模型开发者每天会生成数万条提示调优请求 [8, 9, 15] ，并申请大量高端 GPU [10, 11] ，以快速响应这些 LPT请求。其次，大语言模型开发者利用精选提示，引导商业大语言模型服务在下游任务 [95]上超越人工设计的提示。如表 1 的第二列和第三列所示，在 10 项大语言模型任务 [44], 中，提示调优优于少样本提示技术；使用 GPT-3.5 和 GPT-4 时，平均提升分别为 2.5× 和 1.8× 。第三，大语言模型开发者依靠提示调优方法，提高开源大语言模型在特定任务 [14, 22, 81]上的准确率。如表 1 的第四、第五和第六列所示，在 Vicuna-7B、LLaMA-30B 和 Qwen7B-R1 的各项任务中，提示调优的平均得分分别提升了 5.4×、4.0×和 2.2× 。开源大语言模型允许访问 logits层 的输出，因此与商业大语言模型服务相比，能够达到更高的准确率。

2.2 提示调优工作负载特征

接下来，我们研究 LPT 工作负载特征，这些特征有助于设计高效的 LPT 集群管理系统。我们使用三个热门大语言模型（GPT2-Base、GPT2-Large、Vicuna-7B）和 SAMSUM 数据集 [39] ，在一台配备 8 块 Nvidia



(a) 时间分解。 (b) 轨迹模式。 (c) ITA累积分布函数。

图 2：LPT 工作负载的特征：(a) 不同大语言模型中 LPT 作业端到端执行时间的分解。(b) 集群中一段为期 2 小时的 LPT 工作负载轨迹。(c) 使用 SAMSUM 数据集 [39] 时，不同大语言模型下各种初始提示的迭代至准确率（ITA）分布。

A100-80GB GPU 的服务器上进行实验。我们发现，LPT 工作负载与训练和推理工作负载具有一些共同特征，这些特征已在先前的研究中得到广泛探讨 [46, 50, 72, 73, 84, 91]。**同步跨 GPU 通信。**与深度学习训练工作负载类似，执行 LPT 作业需要进行多次前向传播和反向传播迭代，并在每次迭代后同步交换提示梯度。不过，LPT 的跨 GPU 通信量远低于深度学习训练。图 2a 展示了三种 LPT 工作负载的时间分解：通信开销仅占总执行时间的 0.4%–0.5%。因此，增加分配的 GPU 数量时，LPT 工作负载的吞吐量几乎能够线性提升。

动态流量。 LPT 是一项面向用户的服务，具有高度波动的动态流量。我们分析了从某大型研究机构（已匿名）的 64-GPU 集群中采样的 LPT 作业轨迹。图 2b 展示了两小时内针对 Vicuna-7B 进行提示调优的 LPT 作业到达时间。我们观察到 LPT 流量出现大幅峰值，每分钟请求数的最大值为 5× 平均值。此模式表明，高效的 LPT 系统需要具备高度灵敏的自动扩缩容能力。

GPU分配与计算比高。 LPT 工作负载具有动态特性，因此需要快速供给 GPU，与推理工作负载类似 [72, 73, 91]。我们测量了 GPU 分配开销（包括容器搭建、框架初始化和 GPU 运行时创建），发现其占总执行时间的 37–41%。这表明需要快速供给 GPU，并在不同 LPT 作业之间重复利用。

对初始提示高度敏感。 我们发现，LPT 工作负载的收敛速度高度依赖于初始提示的选择。我们使用 SAMSUM 数据集 [39] 中随机选取的 20 个提示，针对不同的大语言模型，通过迭代至准确率（ITA）指标测量收敛速度。图 2c 展示了 ITA 指标的累积分布函数（CDF）。ITA 的中位数和最大值比最小值高出 1.7–4.5× 倍，表明在 LPT 开始时选择有效初始提示十分重要。鉴于有大量公开提示词 [29]，可用，我们认为可以为特定任务寻找并重复利用这些提示作为初始提示。

特征总结。表 2 总结了 LPT、深度学习训练和推理工作负载的特征。首先，LPT 工作负载与深度学习训练类似，需要在每次迭代后进行同步通信。其次，LPT 工作负载具有高度动态性，并且面临较长的 GPU 分配延迟，这与深度学习推理工作负载类似。同时，LPT 工作负载具有一个独特特征：其处理时间高度取决于初始提示的选择。

表 2：LPT、深度学习推理和训练工作负载的比较。

特征	LPT	推理	训练
GPU 间同步通信	✓	✗	✓
动态流量	✓	✓	✗
高分配开销	✓	✓	✗
提示敏感性	✓	✗	✗

3 现有集群管理系统的特征分析

由于 LPT 工作负载与深度学习训练和推理工作负载具有相似的执行特征，一种自然的策略是将现有的训练和推理集群管理系统扩展到 LPT。在本节中，我们使用与 §6.1 相同的实验设置，定量评估最先进的推理和训练系统的效率。我们使用图 2b 中轨迹的前 20 分钟，运行基于 Vicuna-7B 模型的提示微调作业。

3.1 深度学习训练系统的低效性

以往的研究提出了许多系统设计 [23, 34, 35, 41, 86]，用于优化深度学习训练工作负载的执行。这些系统配置固定规模的 GPU 集群（下文称为 GPU 池），并频繁地从池中为作业分配 GPU，以最大限度地提高 GPU 利用率。

我们评估了最先进的 SLO 感知训练系统 ElasticFlow [41]。该系统会动态调整分配给每个作业的 GPU 数量，以提高作业吞吐量和 SLO达成率。然而，在 ElasticFlow 中，无论实际使用情况如何，所有静态配置的 GPU 的单位时间资源成本都保持不变。图 3a 展示了 ElasticFlow 的 GPU 集群利用率。平均而言，ElasticFlow 的 GPU 集群利用率仅为 56%，使 GPU 集群成本几乎翻倍。

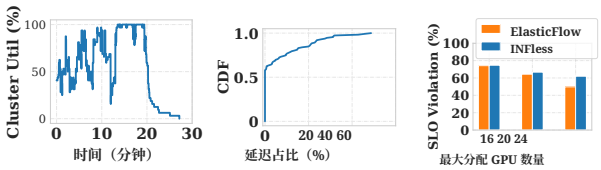
> **低效之处 1**：现有深度学习训练系统采用固定规模的 GPU 集群进行静态配置，运行 LPT 工作负载时会导致资源成本高昂。

3.2 深度学习推理系统的低效之处

现有推理系统 [72, 85, 91, 92] 通常采用无服务器自动扩缩容架构。收到推理作业后，它们通常会为每个到达的作业从提供商的大型可用 GPU 池中分配一个配备 GPU 的容器，也称为实例，它是扩缩容的基本单位。为减少 GPU 容器漫长的启动开销，提供商会让空闲实例保持就绪状态，以便处理未来同一模型的推理作业，但这会长时间占用昂贵的 GPU 内存。这些系统通过自动扩缩容，根据推理流量的变化调整每个模型的实例数量。

尽管这些设计避免了训练系统的静态资源配置，但其性能仍受其他限制。首先，它们独立扩缩每个模型的资源，而不考虑全局最优调度。其次，先前的系统 [72, 85, 91, 92] 仅限于为每个模型实例分配一块 GPU。最后，它们不支持同步的跨 GPU 通信，而这正是 LPT 作业所需要的。

我们在评估中选择 INFless [85]，这一具有代表性的深度学习推理系统。然而，在单个实例上运行 LPT 作业



（a）ElasticFlow。 （b）INFless。 （c）SLO 违规。

图 3：现有深度学习系统的特征分析：(a) ElasticFlow 中随时间变化的集群利用率（%）（y 轴）。(b) 累积分布函数（y 轴）展示了实例初始化导致的端到端延迟中等待延迟所占的比例（x 轴）。(c) 在不同最大 GPU 数量下，ElasticFlow 和 INFless 的服务等级目标违约率（%）。

不足以提升作业吞吐量并满足紧迫的延迟 SLO。为解决这一限制，我们扩展了 INFless，使其能够通过 Memcached [12], 支持同步跨 GPU 通信，这种做法在无服务器系统中十分常见 [52, 82]。多 GPU 执行的实现细节见 §5.1。因此，单个 LPT 作业可以使用多个实例（即多个 GPU）来加快完成速度。不过，在 INFless 和其他推理系统中，部分实例可能需要数十秒才能初始化，因而 LPT 作业在多个实例上运行时会经历较长的等待时间。图 3b 显示，实例初始化平均占端到端LPT作业延迟的 11%，最坏情况下可达 50%。

低效性 2：深度学习推理系统针对单 GPU 实例采用的初始化技术，在初始化多GPU实例时会产生大量延迟。

不出所料，由于上述低效性，ElasticFlow 和 INFless 都表现出极高的服务等级目标违约率。图 3c 展示了在 ElasticFlow 和 INFless 上运行 LPT 工作负载，并设置不同的最大分配 GPU 数量时出现的服务等级目标违约率（最高可达 70%）。这些结果表明，现有的集群管理系统并不适用于 LPT 工作负载，因此需要设计一种针对第 2.2 节所述 LPT 工作负载特征的新系统。

4 系统设计

我们介绍了 PromptTuner，这是一个面向 LPT 工作负载的 SLO感知弹性集群管理系统。我们首先介绍设计洞见和系统概览，随后说明其两个关键组件：提示库和工作负载调度器。

4.1 设计洞见

PromptTuner 的设计 受到两个洞见的启发。我们的第一个洞见是，*LPT* 任务可以复用针对相似任务优化的提示，将其作为初始提示，从而减少达到期望准确率所需的调优迭代次数。

大量关于迁移学习 [75, 80] 的实证研究和理论分析 [79] 表明，复用提示可以显著加快模型收敛。然而，成功复用提示的关键在于自动且及时地识别出最有效的提示。

我们的第二个洞见是，*LPT* 工作负载可以复用基于相同大语言模型的作业的运行时。事实上，许多 LPT 作业会在执行前将相同的运行时状态加载到 GPU 中 [20, 74]。

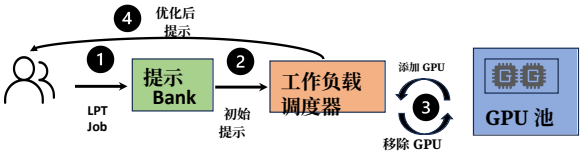


图 4：PromptTuner 的工作流程。该流程由两个关键组件组成：（1）提示库以最低成本为即将到来的 LPT 作业识别有效的初始提示；（2）工作负载调度器从 GPU 池中动态为每个 LPT 作业添加 GPU，以减少 SLO 违约，同时最大限度地降低资源成本。

表 3：PromptTuner 中的作业属性描述。

属性	描述
模型	大语言模型的名称。
终止条件	作业完成标准，包括最大数量迭代次数和准确率目标。
截止时间	LPT 工作负载应完成的预期时间。
数据集	存储数据样本的路径（例如 AWS S3）。
超参数	包括初始提示和批量大小等参数，优化算法。
提示	优化后的提示。

该状态包括 CUDA 和深度学习框架依赖项以及模型权重。复用此状态可以大幅降低 GPU 分配开销（§2.2）。然而，如 §3.2 所强调的，有效复用运行时的关键在于缓解需要多个 GPU 的 LPT 作业所面临的显著延迟。

4.2 系统概述

PromptTuner 包含两个关键组件：提示库利用 提示复用 为传入的 LPT 作业识别有效的初始提示（§4.3）；工作负载调度器利用运行时复用 快速为每个 LPT 作业分配 GPU，在满足服务等级目标要求的同时降低资源成本（§4.4）。

图 4 展示了 PromptTuner 的工作流程。首先，用户向服务提供商提交一个 LPT 作业（ ）。提示库为该作业识别有效的初始提示（ ）。接下来，工作负载调度器根据传入流量的 GPU 需求，动态地向 GPU 池添加或移除 GPU。工作负载调度器还会定期动态调整每个作业所用的 GPU 数量（ ）。最后，LPT 服务提供商在 LPT 作业完成后向用户返回优化后的提示（ ）。

在 PromptTuner 中，一个作业等同于 LPT 服务用户发送的 RPC 请求，以及系统随后返回的 RPC 响应。表 3 总结了作业属性及其描述。前五个属性是用户指定的作业参数。最后一个参数是系统返回给用户的、包含优化后的提示的响应。作业的服务等级目标定义为 LPT 作业满足终止条件的最长时间。

4.3 提示库

提示库通过提示复用 提升传入 LPT 作业的 ITA 性能。它包含一组由所有 LPT 作业共享的提示，供其选作初始提示。我们的目标是平衡识别初始提示所带来的加速收益与查询延迟成本。为此，我们将提示库设计为具有两层数据结构的查询引擎，以便高效查找新 LPT 作业，并支持无缝插入

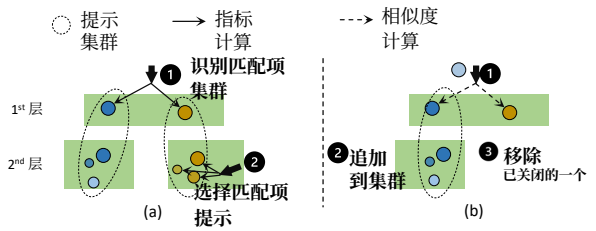


图5：在两层数据结构上执行（a）查找以及（b）插入与替换的示意图。

和替换新的初始提示词候选项。下面详细介绍该数据结构的构建过程，以及对其执行查找、插入和替换操作的过程。本节使用的符号定义见表4。

表4：提示库中的符号汇总。

Sym.	定义
D_{eval}	评估数据集
d_i^{in}	输入查询样本
d_i^{tgt}	目标响应样本
拼接	将两个文本序列拼接起来的操作
$\mathcal{L}$	输出与目标样本之间的损失
C	提示库中提示候选项的总数
K	K-中心点算法的簇数量
C_{sim}	代表性提示与新初始提示最接近的簇

4.3.1 数据结构构建。我们首先从公共来源 [8, 29] 汇集数千个提示候选项，构成一个全面的集合，从而最大限度地提高选出有效初始提示的可能性。要为给定的 LPT 作业找出有效的初始提示，对整个提示候选集进行暴力搜索，计算开销很大，往往需要数小时。我们的实证研究（§6 中的图10a）以及现有的大语言模型研究 [75] 都表明，提示相似性十分普遍。这为我们提供了机会，可以排除对大量劣质提示候选项进行不必要评估的情况，并提高查询效率 [29]。

为此，我们为提示候选集构建了一个两层数据结构。受 [55], 启发，我们根据所有提示候选项的激活特征相似度，将它们划分为多个簇。首先，我们使用大语言模型（例如 Vicuna-7B）提取每个提示候选项的激活特征。然后，根据激活特征之间的余弦距离衡量提示相似度。我们还将在§5.2讨论其他相似度指标。最后，我们采用 K-中心点聚类，将激活特征相似的提示归入同一个簇。图5(a)展示了这一数据结构的示例。第一层保留每个簇的中心点，后文将其称为该簇的代表性提示。第二层存储这些簇中的每个提示。下文将详细介绍查找、插入和替换操作。

4.3.2 查找。查找操作旨在为给定的 LPT 作业在这个两层数据结构中找到有效的初始提示。对于每个初始提示候选项 p，我们引入一个指标分数(p)，其计算方式是在评估样本上计算平均损失，无需在训练样本上进行额外调优。我们将 score(p) 定义如下：

$$\text{score}(p) = \frac{1}{\mathcal{D}_{\text{eval}}} \sum_{(d_i^{\text{in}}, d_i^{\text{tgt}}) \in \mathcal{D}_{\text{eval}}} \mathcal{L}\left(\text{concat}(p, d_i^{\text{in}}), d_i^{\text{tgt}}\right). \quad (1)$$

分数越低，表示初始提示越好。我们仅使用少量评估样本（例如 16 个）来评估提示。如果缺少评估数据集，这只需付出极少的标注工作。无需进行任何调优，我们就能选择分数最低的提示作为最有效的提示。

两层数据结构有助于减少执行式 1 中指标计算所需的提示候选项数量。图 5 (a) 展示了查找操作的过程。首先，我们在第一层计算每个代表性提示的得分，以确定匹配的集群。我们选取得分最低的集群。接下来，我们在第二层计算匹配集群中每个提示的得分，以选出匹配的初始提示。我们将得分最低的提示选为最优提示。假设每个集群包含相同数量的提示候选项，那么该两层数据结构可将指标计算次数从 C 减少至 $K + C/K$。理想情况下，当最优集群为 $K = \sqrt{C}$ 时，指标计算次数最少为 $2\sqrt{C}$。根据经验，两层数据结构最多可将查找操作的开销降低 40×，同时保持性能不变（§6.3）。

4.3.3 插入与替换。当服务提供商插入新的初始提示候选项时，图 5 (b) 展示了插入和替换操作的过程。首先，我们确定一个相似的集群。我们提取新候选项的激活特征，并在第一层计算新候选项与每个集群的代表性候选项之间激活特征的余弦距离。与查找操作不同，此步骤不涉及指标计算（式 1）。余弦距离最小的集群记为 C_{sim}。其次，我们在第二层将该初始提示追加到 C_{sim}。第三，如果插入后初始提示候选项的数量超过阈值（例如，3000），则触发替换操作。我们需要选择一个提示候选项并将其移除。为了最大限度地提高集群内提示候选项的多样性，我们选择与 C_{sim} 的代表性提示之间余弦距离最小的提示候选项，并将其移除以完成替换。

4.3.4 双层结构讨论。提示之间普遍存在相似性，这表明对相似提示进行聚类可以避免不必要的分数评估，同时只需牺牲少量加速收益。§6.3 中的研究表明，双层数据结构可以在 10 秒内识别出有效的初始提示。此外，我们使用 K-中心点聚类构建了一个三层结构，但在 Vicuna-7B 上遇到了收敛问题，且构建开销极高（长达数十分钟）。双层结构可在五分钟内构建完成，并且在不同大语言模型上均不存在收敛问题，因此是更合适的选择。

4.4 工作负载调度器

工作负载调度器通过实现运行时复用 来缓解高昂的 GPU 分配开销（§2.2），从而减少服务等级目标违约并最大限度地降低资源成本。图 6 展示了工作负载调度器的概览，它管理两种类型的 GPU 池：一个共享冷 GPU 池，以及一组按大语言模型划分的预热 GPU 池。每个预热池都包含已初始化、用于处理某个特定大语言模型作业的 GPU，也就是说，每个 GPU 都预先加载了

表 5：工作负载调度器中的符号汇总。

Sym.	定义
i	LPT 作业的索引
l	大语言模型的索引
k	热 GPU 池中 GPU 的索引
a	分配的 GPU 数量
L	大语言模型的数量。
R_l	每个大语言模型 l的热 GPU 池中的 GPU 数量
A	每个作业在热 GPU 池中分配的 GPU 数量
B	每个热 GPU 池从冷 GPU 池中添加的 GPU 数量
T_i^{slo}	作业 i的服务等级目标
$T_i^{\text{warm}}(a)$	作业 i 在热 GPU 池中分配 a 个 GPU 时的预计完成时间
T^{cold}	将 GPU 从冷 GPU 池添加到热 GPU 池的开销
$\mathcal{P}$	所有待处理的 LPT 作业
E_l	用于存储大语言模型 l的热 GPU 池中每个 GPU 的最早时间戳的列表
E	用于存储每个大语言模型的列表 E_l

PyTorch/CUDA 运行时和大语言模型权重。共享冷 GPU 池包含未预加载任何 GPU 上下文的 GPU[3]。

将每个大语言模型的预热池与共享冷池分开管理，可以显著降低 GPU 分配开销，而且无需像 Elas- ticFlow（§3.1）那样静态配置一个规模庞大的固定集群。当调度器从相应的预热池为 LPT 作业分配 GPU 时，作业可以立即开始执行，避免了预加载所需运行时和大语言模型权重造成的延迟。因此，工作负载调度器支持同一大语言模型的 LPT 作业进行运行时复用 。由于许多用户使用相同的大语言模型 [24, 70, 78], ，系统只需保留少量配备最少 GPU 的预热池，就能高效运行并最大限度地降低运营成本。与预热池中的 GPU 不同，冷池中的 GPU 不会产生任何成本，因此服务提供商可以将其分配给数据中心运行的任何服务。

对于待处理队列中的每个新作业，工作负载调度器会根据其服务等级目标和预测的执行时间，确定要分配的 GPU 数量。随后，它会从与作业属性中指定的大语言模型类型相对应的预热池中分配 GPU。为确保服务等级目标合规性，我们将作业执行时间的上界预测为：在给定分配的 GPU 数量时，最大剩余迭代次数与每次迭代的最大时间成本的乘积，再加上额外的 GPU 分配开销。LPT 作业完成后，会将分配给自己的 GPU 归还至相应的预热池。调度器会监控各池的 GPU 使用情况，并从冷池向以下预热池添加更多 GPU

需求量高的大语言模型，并从这些大语言模型的热池中移除多余的 GPU。

接下来，我们将详细介绍两种用于执行 LPT 作业的关键资源分配算法，以及一种用于执行提示库的算法。表 5 定义了本节中使用的符号。

4.4.1 从预热池分配 *GPU*。 该算法通过确定为待处理队列中的每个作业从热 GPU 池分配的 GPU 数量，优化服务等级目标达成情况。当 LPT 作业到达时，调度器会将其加入待处理队列。随后，调度器会定期调整队列中每个作业的 GPU 分配，并在需要时从相应的预热池分配更多 GPU，以实现该作业的服务等级目标。算法 1 展示了这一过程。该算法首先根据服务等级目标按升序对待处理队列中的 LPT 作业进行排序，然后逐步增加每个 LPT 作业的分配 GPU 数量，以满足其服务等级目标，直到预热池耗尽（第 7—9 行）。

[3]虽然云服务提供商可以自由使用冷池中的 GPU 来处理其数据中心内运行的任何作业，但为简单起见，我们假设冷 GPU 池的规模固定，且分配 GPU 不会造成任何时间延迟。

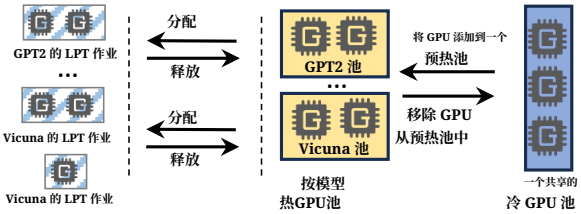


图 6：工作负载调度器由一个共享冷 GPU 池和一组按大语言模型划分的热 GPU 池组成。它会迅速从热 GPU 池中为 LPT 作业分配 GPU，以优化 SLO达成情况。它还会根据流量和 GPU 可用性，动态调整从共享冷 GPU 池添加到热 GPU 池中的 GPU 数量。

算法 1从预热池分配 GPU。

1：输入： R_l，即 LLM l 的预热池中的 GPU 数量； P_l，即 LLM l 的待处理队列。2：输出： A，即待处理队列中为每个作业分配的 GPU 数量。3：4：根据 T_i^{slo} 按升序对作业进行排序 5：对于 P_l 中的每个作业 i，执行 6：设置初始分配 $A_i = 1$ 7：当 T 预热 $_i(A_i) > T_i^{\text{slo}}$ 且 $A_i \le R_l$ 时 8：$A_i = A_i + 1$ // 为该作业分配 A_i GPU 9：结束循环 10：如果 T预热$_i(A_i) \le T_i^{\text{slo}}$，则 11：$R_l = R_l - A_i$ // 更新热 GPU 池中的 GPU 数量 12：否则 13：$A_i = 0$ 14：结束条件 15：结束循环

4.4.2 冷池*GPU*分配。工作负载调度器可以定期将 GPU 从冷 GPU 池添加到相应的热 GPU 池中，或从中移除，具体取决于相应大语言模型的需求。该算法的主要目标是确保每个热池至少拥有满足作业服务等级目标所需的 GPU 数量，同时最大限度地降低资源成本；资源成本与作业使用的 GPU 数量以及热池中的 GPU 数量成正比。因此，该算法优先处理服务等级目标较短的作业，并延迟执行服务等级目标较长以及预计无法达成服务等级目标的作业。

算法 2 详细说明了如何将 GPU 从冷池分配到热池。首先，算法根据所有待处理作业的服务等级目标对其进行排序。其次，通过调用 DelaySchedulable 函数（第 23–35 行），找出 LPT 作业 i，即使延迟调度该作业，仍能满足其服务等级目标。第三，如果系统无法满足该作业的服务等级目标，算法便逐步从冷 GPU 池向该作业分配更多 GPU，直到确保满足其服务等级目标。算法在判断系统能否满足该作业的服务等级目标时，会考虑 GPU 分配开销T_l^{cold}（第 12 行）。最后，如果系统能够满足作业 i 的服务等级目标，算法便将从冷 GPU 池新增的 GPU 数量累加到相应的热 GPU 池中（第 16–20 行）。

DelaySchedulable 函数用于判断作业的服务等级目标是否可以通过将其执行推迟到未来某一时刻来满足，届时已完成的作业会向预热池释放足够的 GPU，而无需立即向预热池添加更多 GPU。我们使用 $E_{l,k}$ 来记录最早时间戳，即当 k 个 GPU 处于热

算法 2 冷池 GPU 分配。

输入： L 大语言模型，包含作业的待处理队列 $\mathcal{P}$，GPU 的最早时间戳位于 the 热GPU池 E。
输出： 分配给每个大语言模型热 GPU 池的 GPU 数量 B。

按 T_i^{slo} 升序对 $\mathcal{P}$ 排序
对于每个作业 i 及其对应的大语言模型 l 在$\mathcal{P}$ **中执行**
  // 通过延迟作业的执行，评估系统是否能够满足该作业的服务等级目标
  如果 DelaySchedulable(E, i, l) **则**
    继续
  结束 if
  设置初始分配的 GPU 数量 $A_i = 1$
  //确定满足作业服务等级目标所需的 *GPU* 数量
  当 $T_i(A_i) + T_{冷l} > T_{\text{服务等级目标}i}$ **且** $T_{\text{服务等级目标}i} < T_{冷}$ **时执行**
    $A_i = A_i + 1$
  结束循环
  如果 $T_i(A_i) + T_{冷l} \leq T_{\text{服务等级目标}i}$ **则**
    // 更新每个大语言模型预热池中的 *GPU* 数量
    $B_l = B_l + A_i$
    // 更新热 *GPU* 池中 *GPU* 的最早时间戳。
    重复 A_i 次，将 $T_{热l}(A_i) + T_{冷l}$ 推迟到 E_l 中。
  结束 if
结束 for

函数 DelaySchedulable(E, i, l)
  $k = 1$, T^{cur} = 当前时间戳
  按升序排序 E_l
  当 $k \leq E_l.len$ **和** $T_i(k) - T_{\text{cur}} + E_{l,k} > T_i^{\text{slo}}$ **时**
    $k = k + 1$
  结束循环
  如果 $k < E_l.len$ **和** $T_i(k) - T_{\text{cur}} + E_{l,k} \leq T_i^{\text{slo}}$，**则**
    $E_{l,\ 1:k} = T_i(k) + E_{l,\ k} - T^{\text{cur}}$
    按升序排序 E
    返回 真
  结束 if
  返回 假
结束函数

用于大语言模型 l 的 GPU 池可用。该信息来自对每个正在运行的 LPT 作业完成时间的预测，以及随后将 GPU 释放到相应预热池的过程。此外，为降低资源成本，如果 GPU 在一个时间窗口内未服务任何作业，工作负载调度器会将其从预热池中移除，该时间窗口的大小设为一分钟（§6.3）。

为什么DelaySchedulable 函数。 许多感知 SLO 的资源分配策略 [41, 72, 85] 希望及时调度作业，以确保作业满足 SLO 合规性。延迟执行作业可能会导致 SLO 违约。得益于提示库带来的加速收益，许多 LPT 作业能够提前完成，从而使许多 GPU 闲置。工作负载调度器利用这一点，在无需启动额外 GPU 资源的情况下，有策略地延迟执行 LPT 请求。这样，系统就能用更少的 GPU 满足更多 LPT 请求的服务等级目标。

4.4.3 提示库的延迟预算。工作负载调度器需要分配 GPU 来执行提示库。尽管我们支持提示库与 LPT 的顺序执行（§ 5.2），并将提示库的开销控制在 10 秒以内，但这项开销仍可能影响短请求的 SLO 合规性。我们通过实测发现，与归纳初始化 [88], 自动提示词初始化基线相比，提示库可带来 1.2-4.7× 倍的加速（详见 §6.1 ）。因此，我们将提示库的延迟预算设为延迟 SLO 的 20%，以确保即使按最低估算，加速收益仍超过提示库带来的开销。

5 实现

5.1 多 GPU 执行

我们使用 2000 行 Python 代码，在 Transformers 2.4.1 和 PyTorch 2.1 的基础上实现 LPT 作业，并将其部署为容器化的 GPU Knative 函数，以便在 GPU 中预加载 LPT 运行时和大语言模型权重。每个 Knative 函数都接受一组表 3 所述的参数，并向用户返回优化后的提示。我们采用了 [57] 中的提示调优算法。请注意，PromptTuner 具有通用性，支持其他 LPT 作业实现和算法。

一个 LPT 作业需要多个函数实例来实现多 GPU 执行。我们在 Lamb-daML [52], 之上实现多 GPU 执行，该系统使用 Memcached 作为存储通道，以实现函数实例之间的同步跨 GPU 通信。属于某个 LPT 作业的每个函数实例都会分配一个 IP 地址和端口，以便与其他函数实例连接，最多会产生 2 秒的开销。由于存储通道的数据量很小，因此其通信开销可以忽略不计。

5.2 提示库

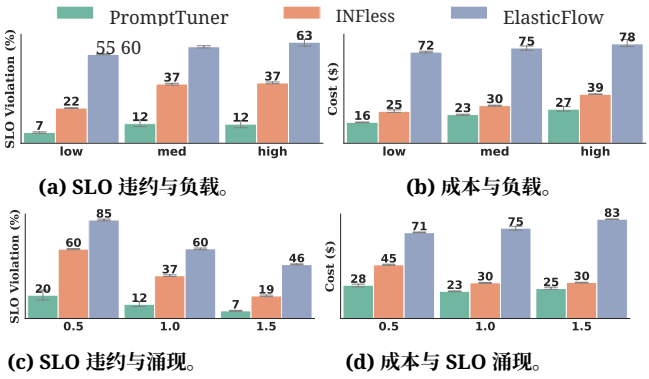
我们基于 Transformers 2.4.1 和 PyTorch 2.1，使用 ~1000 行 Python 代码实现了提示库。该提示库还作为一个配备单个 GPU 的 Knative 函数进行部署，接受包括表 3 所述数据集和初始提示在内的参数，并返回经过优化的初始提示，以供后续提示调优使用。

离线阶段。 对于每个 LPT 作业，我们使用相应的大语言模型提取收集到的提示的激活特征，并根据经验将两层数据结构中的簇数设为 50。此外，我们使用 Scipy 1.10.1 执行 K-中心点聚类。尽管我们尝试了包括曼哈顿距离和欧几里得距离在内的其他距离度量，但遇到了收敛问题。无法收敛可能是由于不同提示的激活特征中数值尺度不平衡所致。每个大语言模型的存储空间均小于 5 GB。我们在 §4.3 中详细介绍了插入和替换操作。如果服务提供商引入新的大语言模型，则需要重新提取所有已收集提示的激活特征，以构建两层数据结构。

在线阶段。 工作负载调度器利用延迟预算来决定是否为传入请求执行提示库。我们注意到，提示库和 LPT 的实现与运行时可以共享。因此，我们将提示库整合到相应的 LPT 作业中。换言之，提示库和 LPT 作业会在分配给它们的 GPU 上依次运行。

5.3 工作负载调度器

预加载运行时。 每个作业都需要多个函数实例来执行多 GPU 任务。Knative 提供自动扩缩机制，以维持函数实例，从而为未来的请求提供服务。**从冷池分配 GPU。** 该算法旨在快速从预热池为 LPT 作业分配 GPU。因此，我们每 50 毫秒执行一次基于轮次的 GPU 分配；与分钟级的延迟 SLO 相比，这一间隔可以忽略不计。该算法在 Knative 的分布式控制平面内运行，将热 GPU 池中的 GPU 分配给相应的 LPT 作业。



(a) SLO 违约与负载。 (b) 成本与负载。

(c) SLO 违约与涌现。 (d) 成本与 SLO 涌现。

图 7：不同负载 (a-b) 和不同 SLO 涌现 (c-d) 下的端到端性能。

从冷池分配 GPU。 该算法在 Knative 的分布式控制平面内部实现，间隔设为 50 毫秒，以便及时为热池添加和移除 GPU。此外，该算法会跟踪分析所得的信息，包括每个大语言模型的分配开销和作业吞吐量。随后，它会持续将这些信息更新至调度器，以避免出现较大的估算误差。

6 评估

6.1 实验设置

测试平台。 我们在物理 GPU 集群上搭建了 PromptTuner。每台 GPU 服务器配备八块 NVIDIA A100-80GB GPU 和一张 200Gbs HDR InfiniBand 网卡。服务器搭载 Intel Xeon 8369B 2.90GHz CPU，拥有 64 个核心、256 GB 内存和 PCIe-III。PromptTuner 最多可配置 4 台 GPU 服务器。我们采用 Memcached 1.5.22 搭建弹性缓存服务，用于 GPU 服务器间的通信。**工作负载构建。** 我们在 12 个数据集上评估了三种具有代表性的大语言模型（GPT-Base、GPT-Large 和 Vicuna-7B），如表 6 所示。为进一步提高 LPT 工作负载的多样性，我们将每个数据集采样为十个互不重叠的分区，并为每个大语言模型构建 120 个任务。对于表 6 中的每个 LPT 任务，我们将从提示库中随机选取的 20 个初始提示的平均准确率作为目标准确率。这样做主要是为了确保在 §2.2 的提示敏感性分析中使用不同初始提示的 LPT 作业能够达到该准确率。我们还评估了 LLaMA-30B 和 Qwen7B-R1[4]，以展示系统在大规模模型和长序列输入场景下的效率。

在实验中，我们从数据中心采样了三段时长为 20 分钟的 LPT 跟踪数据，为不同的大语言模型（GPT-2 基础版/GPT2-L/V7B）构造低负载（41/55/42）、中负载（77/71/65）和高负载（99/85/76）。对于 LLaMA-30B 和 Qwen7B-R1，我们分别采样 59 个和 70 个请求作为中负载。这些跟踪数据包含提交时间、分配的 GPU 数量以及每个 LPT 作业的持续时间。作业持续时间从几秒到几分钟不等。我们按照提交时间属性的分钟粒度，以指数分布调用请求。作业持续时间与分配的 GPU 数量的乘积用于为每个作业分配 LPT 任务。系统会随机选择表 6 中的一个任务

[4]我们使用软提示调优算法 [58] 和 GSM8K 任务 [25] 评估 Qwen7B-R1。选定的初始文本提示置于软提示之前。

表 6：LPT 任务和目标准确率：[B]和 [R] 分别表示 BLEU分数和 ROUGE分数。

任务描述	数据集	准确率	任务描述	数据集	准确率
对话	DA [57]	54 [B]	摘要	CNNDM [57]	34 [B]
	PC [93]	19 [B]		SAMSUM [39]	46 [B]
问答	COQAQG [71]	51 [B]		XSUM [57]	40 [B]
	QUORA [53]	21 [B]	故事生成	CMV [47]	26 [R]
文本生成	WIKIBIO [57]	70 [R]		WP [30]	20 [R]
	WIKIP [13]	22 [R]		ROC [64]	25 [R]

以匹配此类作业的 GPU 时间。我们将每个作业的服务等级目标（SLO）设为其持续时间乘以参数 S，再加上资源分配开销。我们将 S 称为 SLO 涌现度。S 越小，表示 SLO 越具涌现性。

基线方法。 PromptTuner 是首个面向 LPT 工作负载的 SLO感知系统。我们选取了两种最先进的深度学习集群管理系统作为基线方法：(1) **INFless** [85]：这是一种高效、SLO感知且成本效益高的深度学习推理系统。它支持基于流量的自动扩缩容和运行时复用。为确保公平比较，我们为 INFless 增加了多GPU执行和提示库。(2) **ElasticFlow** [41]：这是一种 SLO感知的深度学习训练系统。它会动态调整每个作业使用的 GPU 数量。然而，它不支持运行时复用。我们也将提示库集成到了 ElasticFlow 中。

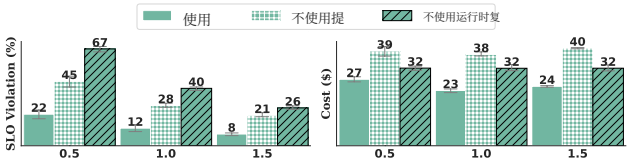
为评估提示库中初始提示的质量，我们考虑了两个基线方法：(1) **理想提示词**：这是 ITA 性能最佳的提示。为便于计算，我们使用评分筛选出 20 个提示，并根据其 ITA 性能选出最佳提示。然而，这种方法在实践中计算开销过高，难以实现。(2) **Induction** [88]：这是一种自动提示词初始化方法，它利用一组示范示例引导大语言模型生成合适的初始提示。然而，它只适用于简单任务，且大语言模型需要具备较强的能力。

评估指标。我们考虑两项评估指标：（1）满足服务等级目标的工作负载比例。我们使用服务等级目标违约作为指标。（2）资源总成本。我们根据 AWS p4de.24xlarge 实例的价格估算成本。存储成本按 GB/小时计费（AWS 弹性缓存）。考虑到通信时间很短，我们采用存储传输数据的最低可能价格。对于提示库，我们选择 ITA，以展示所选初始提示的高质量。

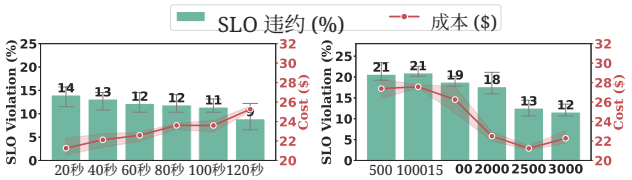
6.2 端到端性能

我们在物理集群的各种环境下，将 PromptTuner 的端到端性能与两种基线方法（INFless 和 ElasticFlow）进行比较。在图 7 所示的实证评估中，本实验同时为三个大语言模型处理请求。首先，图 7a 和图 7b 分别展示了这些系统在不同作业负载下的服务等级目标违约率和成本。与 INFless 相比，PromptTuner 将服务等级目标违约率降低了 15%–25%；与 ElasticFlow 相比，则降低了 48%–51%。有趣的是，负载增加为执行运行时复用提供了更多机会。因此，服务等级目标违约率并未随着负载从中等升至较高而显著增加。繁重的作业负载会提高服务等级目标违约率和成本，而在更繁重的作业负载下，PromptTuner 相比基线方法展现出更明显的优势。

其次，我们探究了这些系统在不同服务等级目标紧急程度下的 SLO 违约情况和成本，并以中等作业负载为重点



(a) SLO 违约与共享。 (b) 成本与共享。



(c) 窗口大小。 (d) 银行大小。

图 8：特征评估：(a-b) 在不同 SLO 等级下，提示复用（P.R.）和运行时复用（R.R.）对 SLO 违约和成本的影响。(c-d) PromptTuner 在不同窗口大小 (c) 和提示库大小 (d) 下的 SLO 违约和成本。

表 7：重负载评估。

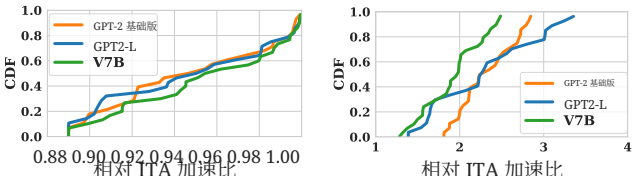
重负载设置	指标	PromptTuner	INFless	ElasticFlow
LLaMA-30B	SLO 违约 ↓(%)	28.4	38.9	82.3
	成本 ↓($)	38.8	46.4	69.4
Qwen7B-R1	SLO 违约 ↓(%)	23.1	36.2	74.9
	成本 ↓($)	30.7	42.8	70.1
大规模	SLO 违约 ↓(%)	25.4	57.1	78.2
	成本 ↓($)	57.2	65.9	99.1

以简化分析。如图 7c 和 7d 所示，在不同 SLO 等级下，PromptTuner 的表现始终优于基线系统，SLO 违约率至少降低 10%。当 SLO 涌现程度设为 0.5 时，更多 LPT 作业会在多个 GPU 上执行。因此，正如第 3.2 节所述，INFless 更容易受到实例初始化造成的长时间等待延迟影响。因此，INFless 的 SLO 违约率甚至与 ElasticFlow 一样高。在资源成本方面，与 INFless 相比，PromptTuner 在 SLO 等级为 $S = 0.5$、1.0 和 1.5 时，分别降低了 38%、23% 和 17% 的开支。与 ElasticFlow 相比，PromptTuner 的成本节省更为显著：在 $S = 1.5$ 时最高可达 70%。总之，PromptTuner 在降低 SLO 违约率和提升成本效率方面均表现出色。

LLaMA-30B 评估。 LLaMA-30B 的单个副本部署在四块 GPU 上，并采用张量并行来支持提示调优。由于 GPU 资源有限，LLaMA-30B 的实验单独进行。表 7 对比了 PromptTuner 与另外两种基线方法的 SLO 违约率和资源成本。与另外两种基线方法相比，PromptTuner 将 SLO 违约率降低了 1.36-2.90×，并将资源成本降低了 1.20-1.79×。这些结果表明，在管理繁重的大语言模型工作负载时，PromptTuner 仍能保持其优越性能。

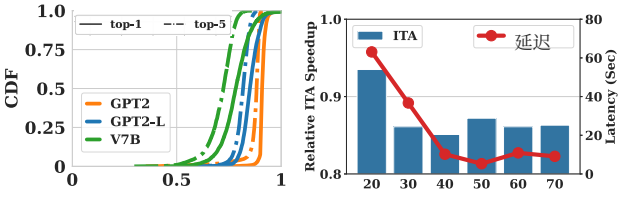
Qwen7B-R1 评估。 Qwen7B-R1 的最大序列长度为 32,768 个令牌。为满足这一要求，我们使用四块 GPU 通过张量并行部署一个 Qwen7B-R1 副本。此外，我们还使用一个由 32 块 GPU 组成的集群来评估 Qwen7B-R1 的性能。表 7 对比了 SLO 违约率和资源

PromptTuner 与两种基线方法的成本。与另外两种基线方法相比，PromptTuner 将服务等级目标违约率降低了 1.56-3.24×，并将资源成本降低了 1.39-2.28×。这些结果



(a) 分数与理想值的对比。 (b) 分数与归纳值的对比。

图9：分数候选项相对于(a)理想提示词候选项；(b) Induction候选项的相对ITA加速比分布。



(a) 提示相似度。 (b) 变化 K。

图10：双层结构的性能：（a）提示相似度分布；（b）不同组数下的延迟和平均相对 TTA。

表明 PromptTuner 在处理长序列样本时仍能保持其卓越性能。

可扩展性评估。 我们测量了 PromptTuner 在大规模 GPU 集群中的性能。由于可用 GPU 数量有限，我们在一个最多包含 96 个 GPU 的集群上为每个系统各进行一次实验。我们按比例增加图7c中的中等作业负载，以匹配已配置 GPU 的最大数量。表 7 比较了 PromptTuner 与另外两种基线方法的 SLO 违约情况和资源成本。随着已配置 GPU 数量的增加，PromptTuner 相较于其他基线方法的性能优势也随之扩大。借助更多工作负载和 GPU，PromptTuner 可以利用动态资源分配来获得更优的调度决策。此外，其平均/最大调度开销为 13/67 毫秒，因此不会成为 Pro mptTuner 的性能瓶颈。较低的调度开销进一步增强了我们的信心：PromptTuner 能够在大规模 GPU 集群中实现令人满意的性能。

6.3 核心组件评估

提示与运行时复用。 图8a和图8b展示了提示复用（提示复用）和运行时复用（运行时复用）在不同 SLO 等级下对保障服务等级目标和提升成本效益的作用。首先，提示复用可将 SLO 违规减少 13-23%，并节省 30-40% 的成本。对于严格的 SLO，提示库（即提示复用）通过满足更多 LPT 作业的服务等级目标来节省 GPU 时间。在宽松SLO场景中，提示库可以减少分配给热 GPU 池的 GPU 数量。PromptTuner 尤其能从运行时复用中受益，因为它能减少 GPU 分配开销并提升 SLO达成。不过，运行时复用带来的成本节省无法与提示复用相提并论。

从热 GPU 池分配的影响。 从热 GPU 池分配 GPU 可实现同时分配多 GPU，从而有效降低与多 GPU 实例相关的初始化开销，正如 § 3.2 所述。我们实现了一种基线策略（不使用预热分配器），该策略会立即为每个函数实例分配热 GPU（§5.1），而不考虑同一 LPT 中同时分配的限制

表 8：工作负载调度器中关键组件的影响。

指标	工作负载调度器	w/o 预热分配器	w/o DelaySchedulable	w/o 延迟预算
SLO 违约↓(%)	12.4	27.8	15.6	16.3
成本 ↓($)	22.9	20.9	26.6	23.2

请求。表 8 的第二列和第三列表明，在服务等级目标为 1.0 且作业负载为中等时，我们提出的分配策略将 SLO 违约率降低了 2.24 倍，而成本仅略有增加。这些结果表明，热 GPU 分配器对 SLO达成有积极影响。

DelaySchedulable 函数的影响。 DelaySchedulable 函数会延迟某些 LPT 请求的执行，以充分利用热 GPU 池中的 GPU，从而在尽量降低 SLO 违约率的同时减少热 GPU 池中的 GPU 数量。如表 8 的第二列和第四列所示，在 SLO 水平为 1.0 且作业负载为中等时，该函数分别将 SLO 违约率和资源成本降低了 1.27× 和 1.16×。这从经验上证明了 DelaySchedulable 函数在同时优化 SLO 违规与资源成本方面的有效性。

延迟预算的影响。 为评估延迟预算的有效性，我们建立了一个基线，在该基线中，每当收到请求时都会触发提示库。如表 8 所示，延迟预算分别使 SLO 违规和成本降低了 1.31× 和 1.02×。事实证明，延迟预算是 PromptTuner 中一项有益的机制。

冷 GPU 池分配窗口大小。 我们研究冷 GPU 分配器的窗口大小如何影响 PromptTuner 的性能。窗口较小时，GPU 会频繁地从热 GPU 池中移除，从而增加 SLO 违约。窗口较大时，PromptTuner 对流量的响应可能不够及时，导致资源成本上升。图 8c 展示了不同的窗口大小，并表明将窗口设为 60 秒能够在 SLO 违约与成本之间取得令人满意的平衡。

改变提示库大小。由于提示库的评估成本很高，我们仅选用 GPT2-Base、GPT2-Large 和 Vicuna-7B 进行研究。我们分析了提示候选项数量对 PromptTuner 调度性能的影响。由于可免费使用的高质量提示数量有限，我们将最大规模设为 3,000。图 8d 显示，SLO 违约和成本会随提示库大小而变化。较大的提示库会带来更高的执行开销，而较小的提示库则可能减少从有效初始提示中获得的潜在加速收益。当规模降至 2,000 时，SLO 违规和成本均显著上升，这凸显了保持提示库多样性的重要性。

评分指标。 我们将由所提出的指标（公式 1）、理想基线和归纳基线分别选出的提示称为分数候选项、理想候选项和归纳候选项。图 9a 展示了来自 3 个大语言模型的 120 个 LPT 任务中，分数候选项与理想候选项之间相对 ITA 性能的分布。大多数分数候选项的 ITA 性能都超过理想候选项的 90%。图 9b 展示了分数候选项与归纳候选项之间相对 ITA 性能的分布。分数候选项优于归纳候选项，对于 GPT-Base、GPT-Large 和 Vicuna-7B，分别至少带来 1.81×、1.38×、1.28× 的 ITA 加速比。GPT-B 获得最高的 ITA 加速收益（1.8–2.8×），因为与 Vicuna-7B 相比，其通用性较弱，导致其自身生成初始提示的效果较差。

相比之下，V-7B 相较于归纳候选项实现的最低 ITA 加速比为 1.28×。这一分析表明，我们的评分方法能够识别近最优初始提示，在各种任务和大语言模型上，与归纳初始化相比实现更优的 ITA 性能。

两层数据结构。图10a展示了在不同大语言模型上，我们整理的提示候选集中激活特征的前 1（实线）和前 5（虚线）余弦相似度的累积分布函数。这种较高的相似度促使我们设计两级数据结构，以对相似的提示候选项进行分组。此外，我们验证了对相似的提示候选项进行聚类是否会降低所识别初始提示的 ITA 性能并减少选择延迟。我们将评估样本数量固定为 16，并使用 GPT2-Base。图10b展示了聚类数量对相对于理想候选项的 ITA 加速比以及平均选择延迟的影响。增加组数并不会造成明显的 ITA 性能损失。对于 GPT2-Large 和 Vicuna-7B，聚类数量对 ITA 加速比的影响呈现出类似趋势。此外，考虑到延迟开销，我们将 PromptTuner 的簇数设为 50。此时，GPT2-Base、GPT2-Large 和 Vicuna-7B 的平均延迟分别为 5.3 秒、6.1 秒和 9.2 秒。作为参考，当 K 设为 1 时，GPT-Base、GPT-Large 和 Vicuna-7B 分别需要约 2.5、2.9 和 4.5 小时。总体而言，两层数据结构有效地平衡了加速收益与初始提示选择的开销。

7 相关工作

工作负载调度。 许多调度器旨在满足延迟 SLO。训练系统 [23, 33–36, 38, 89?] 会考虑资源弹性，以调整 GPU 分配。推理系统 [26, 42, 72, 85, 91] 利用 GPU 共享和请求批处理来满足服务等级目标，同时提高 GPU 利用率。我们的系统借鉴了这些设计，包括资源弹性和运行时复用。

大语言模型系统。 大语言模型的重要性吸引了研究人员设计专用系统来支持其运行。许多大语言模型系统 [21, 43, 48, 51, 65, 66, 94] 致力于自动探索并行策略，以便在数千个 GPU 上部署大语言模型训练。在我们的场景中，与大语言模型训练相比，LPT 的通信开销和 GPU 请求量（最多几十个）都显著更小。因此，这些策略并不适用于 LPT。许多大语言模型推理相关工作 [20, 32, 37, 54, 63, 74, 90] 主要处理计算受限的预填充阶段与内存受限的解码阶段之间的不匹配，以及庞大的 KV 缓存。然而，由于许多提示调优算法不涉及解码阶段和 KV 缓存管理，PromptTuner 无法直接采用这些推理系统提出的解决方案。相反，PromptTuner 利用提示共享来加速 LPT。

参数高效微调。 近期工作为大语言模型设计了多种参数高效微调方法，包括 LoRA [45], 前缀微调 [58], P-Tuning [59], 以及提示调优 [56, 60]。前缀微调和 P-Tuning 是提示调优的扩展，PromptTuner 可将它们视为 LPT 工作负载，以确定 GPU 分配。大语言模型社区还引入了

其他方法，例如基于梯度的方法 [57, 58], 零阶方法 [61], 以及强化学习 [27]，以优化提示。

8 结论

本文介绍了 PromptTuner，这是一种用于管理 LPT 工作负载的 SLO感知弹性系统。我们利用提示复用开发了提示库，以加速 LPT 工作负载。我们还利用运行时复用降低 GPU 分配开销，从而实现资源弹性。大量实验表明，PromptTuner 在 SLO达成和降低成本方面表现出色。

参考文献

[1] https://promptbase.com/。访问日期：2025年1月。[2] https://claude.ai/。访问日期：2025年1月。[3] 阿里云。https://www.alibabacloud.com/。2023年。[4] 亚马逊云科技。https://aws.amazon.com/。2025年。[5] Azure 云。https://azure.microsoft.com/。2025年。[6] mergeflow。https://mergeflow.com/。[7] Midjourney。https://www.midjourney.com。[8] Prompthero。https://prompthero.com/。[9] Promptperfect。https://promptperfect.jina.ai/home。[10] Nvidia a100。https://www.nvidia.com/en-sg/data-center/a100/，2022年。[11] Nvidia h100。https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper- hopper，2022年。[12] Memcached。https://memcached.org/，2025年。[13] Wikiplots。https://github.com/markriedl/WikiPlots，2025年。[14] Toyin D. Aguda、Suchetha Siddagangappa、Elena Kochkina、Simerjot Kaur、Dong- sheng Wang 和 Charese Smiley。《大语言模型作为金融数据标注者：对有效性和效率的研究》。载于 Nicoletta Calzolari、Min-Yen Kan、Veronique Hoste、Alessandro Lenci、Sakriani Sakti 和 Nianwen Xue 编，《*2024*年计算语言学、语言资源与评估联合国际会议论文集（*LREC-COLING 2024*）》，第 10124–10145 页，意大利都灵，2024年5月。欧洲语言资源协会和国际计算语言学委员会。[15] Portkey AI。Portkey ai，2025年。访问日期：2025年1月。[16] Anthropic。Claude ai。https://claude.ai/，2025年。[17] Maciej Besta、Nils Blach、Ales Kubicek、Robert Gerstenberger、Lukas Gianinazzi、Joanna Gajda、Tomasz Lehmann、Michal Podstawski、Hubert Niewiadomski、Piotr Nyczyk 等。《思维图：利用大语言模型解决复杂问题》。*arXiv* 预印本 *arXiv:2308.09687*，2023年。[18] Tom Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared D Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell、Sandhini Agarwal、Ariel Herbert-Voss、Gretchen Krueger、Tom Henighan、Rewon Child、Aditya Ramesh、Daniel Ziegler、Jeffrey Wu、Clemens Winter、Chris Hesse、Mark Chen、Eric Sigler、Mateusz Litwin、Scott Gray、Benjamin Chess、Jack Clark、Christopher Berner、Sam McCandlish、Alec Radford、Ilya Sutskever 和 Dario Amodei。《语言模型是少样本学习器》。载于《神经信息处理系统进展》，神经信息处理系统会议 '20，2020年。[19] Guangyao Chen、Siwei Dong、Yu Shu、Ge Zhang、Jaward Sesay、Börje F Karlsson、Jie Fu 和 Yemin Shi。《Autoagents：自动智能体生成框架》。*arXiv* 预印本 *arXiv:2309.17288*，2023年。[20] Lequn Chen、Zihao Ye、Yongji Wu、Danyang Zhuo、Luis Ceze 和 Arvind Krish- namurthy。《Punica：多租户 LoRA 服务》。*arXiv* 预印本 *arXiv:2310 18547*，2023。。[21] Qiaoling Chen、Shenggui Li、Wei Gao、Peng Sun、Yonggang Wen 和 Tianwei Zhang。《SPPO：通过自适应序列流水线并行卸载实现高效的长序列大语言模型训练》。*arXiv* 预印本 *arXiv:2503.10377*，2025年3月。[22] Wenhu Chen、Xueguang Ma、Xinyi Wang 和 William W. Cohen。《思维程序提示：将计算与推理分离以完成数值推理任务》，2023年。[23] Zhaoyun Chen、Wei Quan、Mei Wen、Jianbin Fang、Jie Yu、Chunyuan Zhang 和 Lei Luo。《深度学习研发平台：GPU 集群上的服务质量保障特征分析与调度》。*IEEE*并行与分布式系统汇刊，2020年。[24] Wei-Lin Chiang、Zhuohan Li、Zi Lin、Ying Sheng、Zhanghao Wu、Hao Zhang、Lianmin Zheng、Siyuan Zhuang、Yonghao Zhuang、Joseph E. Gonzalez、Ion Stoica 和 Eric P. Xing。《Vicuna：一款以 90%* ChatGPT 质量媲美 GPT-4 的开源聊天机器人》，2023年3月。[25] Karl Cobbe、Vineet Kosaraju、Mohammad Bavarian、Mark Chen、Heewoo Jun、Lukasz Kaiser、Matthias Plappert、Jerry Tworek、Jacob Hilton、Reiichiro Nakano、Christopher Hesse 和 John Schulman。《训练验证器以解决数学文字题》。*arXiv* 预印本 *arXiv:2110.14168*，2021年。[26] Weihao Cui、Han Zhao、Quan Chen、Ningxin Zheng、Jingwen Leng、Jieru Zhao、Zhuo Song、Tao Ma、Yong Yang、Chao Li 和 Minyi Guo。实现深度神经网络同时

基于确定性算子重叠和精确延迟预测的服务。在高性能计算、网络、存储与分析国际会议论文集（SC '21）中，2021年。[27] Mingkai Deng、Jianyu Wang、Cheng-Ping Hsieh、Yihan Wang、Han Guo、Tianmin Shu、Meng Song、Eric P Xing和Zhiting Hu。Rlprompt：使用强化学习优化离散文本提示。arXiv预印本arXiv:2205.12548，2022。[28] Viet-Tung Do、Van-Khanh Hoang、Duy-Hung Nguyen、Shahab Sabahi、Jeff Yang、Hajime Hotta、Minh-Tien Nguyen和Hung Le。大语言模型的自动提示选择。arXiv预印本 arXiv:2404.02717，2024。[29] Fatih Erikli。Awesome chatgpt prompts。https://github.com/f/awesome-chatgpt-prompts/，2022。[30] Angela Fan、Mike Lewis和Yann Dauphin。分层神经故事生成。载于第56届计算语言学协会年会论文集（第1卷：长论文），第889–898页，澳大利亚墨尔本，2018年7月。计算语言学协会。[31] FlowGPT。Flowgpt：面向未来的提示工程。https://flowgpt.com/，2024年。访问日期：2025年1月。[32] Yao Fu、Leyang Xue、Yeqi Huang、Andrei-Octavian Brabete、Dmitrii Ustiugov、Yuvraj Patel和Luo Mai。Serverlessllm：面向大语言模型的低延迟无服务器推理。载于第18届USENIX操作系统设计与实现研讨会，第135–153页。USENIX协会，2024年。[33] Wei Gao、Zhuoyuan Ouyang、Peng Sun、Tianwei Zhang和Yonggang Wen。Icefrog：GPU集群中的深度学习训练层弹性调度系统。IEEE并行与分布式系统汇刊，36(6)，2025年6月。[34] Wei Gao、Zhisheng Ye、Peng Sun、Yonggang Wen和Tianwei Zhang。Chronus：一种新型截止期限感知调度器，用于深度学习训练作业。载于ACM云计算研讨会论文集，SoCC '21，2021年。[35] Wei Gao、Zhisheng Ye、Peng Sun、Tianwei Zhang和Yonggang Wen。Unisched：面向不同用户需求的深度学习训练作业统一调度器。IEEE计算机汇刊，2024年。[36] Wei Gao、Xu Zhang、Shan Huang、Shangwei Guo、Peng Sun、Yonggang Wen和Tianwei Zhang。Autosched：用于调度深度学习训练工作负载的自适应自配置框架。ICS '24，第473–484页，美国纽约州纽约市，2024年。计算机协会。[37] Wei Gao、Xinyu Zhou、Peng Sun、Tianwei Zhang和Yonggang Wen。重新思考大语言模型服务中的键值缓存压缩技术。载于机器学习与系统年会（MLSys）论文集，2025年5月。[38] Wei Gao、Weiming Zhuang、Minghao Li、Peng Sun、Yonggang Wen和Tianwei Zhang。Ymir：数据中心基础模型微调工作负载的调度器。载于第38届ACM国际超级计算大会论文集，第259–271页，美国纽约州纽约市，2024年。[39] Bogdan Gliwa、Iwona Mochol、Maciej Biesek和Aleksander Wawer。SAMSum语料库：用于抽象式摘要的人工标注对话数据集。载于第2届摘要新前沿研讨会论文集，第70–79页，中国香港，2019年11月。计算语言学协会。[40] 谷歌。Gemini ai。https://gemini.google.com/，2025年。[41] Diandian Gu、Yihao Zhao、Yinmin Zhong、Yifan Xiong、Zhenhua Han、Peng Cheng、Fan Yang、Gang Huang、Xin Jin和Xuanzhe Liu。Elasticflow：用于分布式深度学习的弹性无服务器训练平台。载于第28届ACM编程语言与操作系统架构支持国际会议论文集，第2卷，第266–280页，2023年。[42] Arpan Gujarati、Reza Karimi、Safya Alzayat、Wei Hao、Antoine Kaufmann、Ymir Vigfusson和Jonathan Mace。像发条一样服务深度神经网络：自底向上的性能可预测性。载于第14届USENIX操作系统设计与实现研讨会（OSDI 20），2020年。[43] Jihu Guo、Tenghui Ma、Wei Gao、Peng Sun、Jiaxing Li、Xun Chen、Yuyang Jin和Dahua Lin。Adaptis：在异构模型上通过自适应流水线并行减少流水线气泡。arXiv预印本arXiv:2509.23722，2025年9月。[44] Or Honovich、Uri Shaham、Samuel R Bowman和Omer Levy。指令归纳：从少量示例到自然语言任务描述。arXiv预印本arXiv:2205.10782，2022年。[45] Edward J Hu、Yelong Shen、Phillip Wallis、Zeyuan Allen-Zhu、Yuanzhi Li、Shean Wang、Lu Wang和Weizhu Chen。LoRA：大语言模型的低秩适配。arXiv预印本arXiv:2106.09685，2021年。[46] Qinghao Hu、Peng Sun、Shengen Yan、Yonggang Wen和Tianwei Zhang。大规模GPU数据中心中深度学习工作负载的表征与预测。载于高性能计算、网络、存储与分析国际会议论文集，SC '21，2021年。[47] Xinyu Hua和Lu Wang。PAIR：预训练Transformer中的规划与迭代优化，用于长文本生成。载于2020年自然语言处理实证方法会议（EMNLP）论文集，第781–793页，线上，2020年11月。计算语言学协会。[48] Yanping Huang、Youlong Cheng、Ankur Bapna、Orhan Firat、Dehao Chen、Mia Chen、HyoukJoong Lee、Jiquan Ngiam、Quoc V Le、Yonghui Wu等。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。

[49] Insu Jang、Zhenning Yang、Zhen Zhang、Xin Jin和Mosharaf Chowdhury。Oobleck：使用流水线模板进行弹性分布式大模型训练。载于第29届操作系统原理研讨会论文集，第382–395页，2023年。[50] Myeongjae Jeon、Shivaram Venkataraman、Amar Phanishayee、Junjie Qian、Wen-cong Xiao和Fan Yang。面向深度神经网络训练工作负载的大规模多租户GPU集群分析。载于2019年USENIX年度技术会议，USENIX ATC '19，2019年。[51] Zhihao Jia、Matei Zaharia和Alex Aiken。超越深度神经网络的数据并行与模型并行。载于机器学习与系统论文集，MLSys '19，2019年。[52] Jiawei Jiang、Shaoduo Gan、Yue Liu、Fanlin Wang、Gustavo Alonso、Ana Klimovic、Ankit Singla、Wentao Wu和Ce Zhang。揭示无服务器机器学习训练的奥秘。载于2021年数据管理国际会议论文集，第857–871页，2021年。[53] Ashutosh Kumar、Kabir Ahuja、Raghuram Vadapalli和Partha Talukdar。语法引导的释义可控生成。计算语言学协会汇刊，8：329–345，2020年。[54] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph Gonzalez、Hao Zhang和Ion Stoica。使用PagedAttention实现大语言模型服务的高效内存管理。载于第29届操作系统原理研讨会论文集，第626页，2023年，第611页。[55] Fan Lai、Yinwei Dai、Harsha V. Madhyastha和Mosharaf Chowdhury。Model-keeper：通过自动化训练预热加速深度神经网络训练。载于USENIX网络系统设计与实现研讨会（NSDI），2023年。[56] Brian Lester、Rami Al-Rfou和Noah Constant。规模对参数高效提示调优的力量。arXiv预印本arXiv:2104.08691，2021年。[57] Junyi Li、Tianyi Tang、Gaole He、Jinhao Jiang、Xiaoxuan Hu、Puzhao Xie、Zhipeng Chen、Zhuohao Yu、Wayne Xin Zhao和Ji-Rong Wen。TextBox：用于文本生成的统一、模块化且可扩展框架。载于第59届计算语言学协会年会暨第11届自然语言处理国际联合会议：系统演示论文集，第30–39页，线上，2021年8月。计算语言学协会。[58] Xiang Lisa Li和Percy Liang。前缀微调：优化用于生成的连续提示。载于第59届计算语言学协会年会暨第11届自然语言处理国际联合会议论文集（第1卷：长论文），第4582–4597页，2021年。[59] Xiao Liu、Kaixuan Ji、Yicheng Fu、Weng Lam Tam、Zhengxiao Du、Zhilin Yang和Jie Tang。P-Tuning v2：提示调优能够在不同规模和任务中普遍达到可比微调的效果。arXiv预印本arXiv:2110.07602，2021年。[60] Xiao Liu、Yanan Zheng、Zhengxiao Du、Ming Ding、Yujie Qian、Zhilin Yang和Jie Tang。GPT也能理解。AI Open，2023年。[61] Sadhika Malladi、Tianyu Gao、Eshaan Nichani、Alex Damian、Jason D Lee、Danqi Chen和Sanjeev Arora。仅通过前向传播微调语言模型。arXiv预印本arXiv:2305.17333，2023年。[62] Manus。Manus。https://manus.im/，2025年4月。[63] ModelTC。LightLLM。https://github.com/ModelTC/lightllm/，2025年。[64] Nasrin Mostafazadeh、Nathanael Chambers、Xiaodong He、Devi Parikh、Dhruv Batra、Lucy Vanderwende、Pushmeet Kohli和James Allen。用于深入理解常识故事的语料库与完形填空评估。载于2016年北美计算语言学协会分会会议：人类语言技术论文集，第839–849页，美国加利福尼亚州圣迭戈，2016年6月。计算语言学协会。[65] Deepak Narayanan、Aaron Harlap、Amar Phanishayee、Vivek Seshadri、Nikhil R Devanur、Gregory R Ganger、Phillip B Gibbons和Matei Zaharia。PipeDream：用于深度神经网络训练的广义流水线并行。载于第27届ACM操作系统原理研讨会论文集，第1–15页，2019年。[66] Deepak Narayanan、Mohammad Shoeybi、Jared Casper、Patrick LeGresley、Mostofa Patwary、Vijay Korthikanti、Dmitri Vainbrand、Prethvi Kashinkunti、Julie Bernauer、Bryan Catanzaro、Amar Phanishayee和Matei Zaharia。使用Megatron-LM在GPU集群上进行高效的大规模语言模型训练。载于高性能计算、网络、存储与分析国际会议论文集，SC '21，2021年。[67] OpenAI。GPT-4技术报告，2023年。[68] OpenAI。ChatGPT。https://chatgpt.com/，2025年。[69] Promptrr。Promptrr。https://promptrr.io/，2024年。访问日期：2025年1月。[70] Alec Radford、Jeffrey Wu、Rewon Child、David Luan、Dario Amodei、Ilya Sutskever等。语言模型是无监督多任务学习器。OpenAI博客，2019年1月9日。[71] Siva Reddy、Danqi Chen和Christopher D. Manning。CoQA：对话式问答挑战。计算语言学协会汇刊，7：249–266，2019年。[72] Francisco Romero、Qian Li、Neeraja J. Yadwadkar和Christos Kozyrakis。INFaaS：自动化无模型推理服务。载于2021年USENIX年度技术会议，USENIX ATC '21，2021年。[73] Mohammad Shahrad、Rodrigo Fonseca、Inigo Goiri、Gohar Chaudhry、Paul Batum、Jason Cooke、Eduardo Laureano、Colby Tresness、Mark Russinovich 和 Ricardo Bianchini。真实环境中的无服务器：在大型云服务提供商处表征并优化无服务器工作负载。载于2020年USENIX年度技术会议，USENIX ATC '20，2020年。[74] Ying Sheng、Shiyi Cao、Dacheng Li、Coleman Hooper、Nicholas Lee、Shuo Yang、Christopher Chou、Banghua Zhu、Lianmin Zheng、Kurt Keutzer 等。S-lora：为数千个并发LoRA适配器提供服务。arXiv预印本，arXiv:2311 03285，2023年。。[75] Yusheng Su、Xiaozhi Wang、Yujia Qin、Chi-Min Chan、Yankai Lin、Huadong Wang、Kaiyue Wen、Zhiyuan Liu、Peng Li、Juanzi Li、Lei Hou、Maosong Sun 和 Jie Zhou。提示调优在自然语言处理中的可迁移性。载于2022年计算语言学协会北美分会会议论文集：人类语言技术，第3949–3969页，美国西雅图，2022年7月。计算语言学协会。[76] TextCortex。AI提示词市场——TextCortex。https://textcortex.com/ templates/ai-prompt-marketplace，2024年。访问时间：2025年1月。[77] John Thorpe、Pengzhan Zhao、Jonathan Eyolfson、Yifan Qiao、Zhihao Jia、Minjia Zhang、Ravi Netravali 和 Guoqing Harry Xu。Bamboo：使可抢占实例具备弹性，以实现经济实惠的大规模深度神经网络训练。载于第20届USENIX网络系统设计与实现研讨会（NSDI 23），第497–513页，2023年。[78] Hugo Touvron、Thibaut Lavril、Gautier Izacard、Xavier Martinet、Marie-Anne Lachaux、Timothée Lacroix、Baptiste Rozière、Naman Goyal、Eric Hambro、Faisal Azhar 等。Llama：开放且高效的基础语言模型，2023年。URL https://arxiv. org/abs/2302.13971。[79] Nilesh Tripuraneni、Michael Jordan 和 Chi Jin。迁移学习理论：任务多样性的重要性。神经信息处理系统进展，33:7852–7862，2020年。[80] Tu Vu、Tong Wang、Tsendsuren Munkhdalai、Alessandro Sordoni、Adam Trischler、Andrew Mattarella-Micke、Subhransu Maji 和 Mohit Iyyer。探索并预测跨自然语言处理任务的可迁移性。arXiv预印本，arXiv:2005.00770，2020年。[81] Hanbin Wang、Zhenghao Liu、Shuo Wang、Ganqu Cui、Ning Ding、Zhiyuan Liu 和 Ge Yu。Intervenor：通过交互式修复链激发大语言模型的编码能力，2024年。[82] Hao Wang、Di Niu 和 Baochun Li。采用无服务器架构的分布式机器学习。载于IEEE INFOCOM，第1288–1296页，2019年。[83] Shuyang Wang、Somayeh Moazeni 和 Diego Klabjan。面向大语言模型自动提示词工程的序贯最优学习方法。arXiv预印本，arXiv:2501.03508v1，2025年1月。[84] Qizhen Weng、Wencong Xiao、Yinghao Yu、Wei Wang、Cheng Wang、Jian He、Yong Li、Liping Zhang、Wei Lin 和 Yu Ding。真实环境中的机器学习即服务：大规模异构GPU集群中的工作负载分析与调度。载于第19届USENIX网络系统设计与实现研讨会，NSDI '22，2022年。[85] Yanan Yang、Laiping Zhao、Yiming Li、Huanyu Zhang、Jie Li、Mingyang Zhao、Xingzhen Chen 和 Keqiu Li。INFless：面向低延迟、高吞吐量推理的原生无服务器系统。载于第27届ACM编程语言与操作系统架构支持国际会议论文集，第768–781页，2022年。[86] Zichao Yang、Heng Wu、Yuanjia Xu、Yuewen Wu、Hua Zhong 和 Wenbo Zhang。Hydra：面向截止时间且面向效率的异构GPU深度学习作业调度。IEEE《计算机汇刊》，2023年。[87] Shunyu Yao、Dian Yu、Jeffrey Zhao、Izhak Shafran、Thomas L Griffiths、Yuan Cao 和 Karthik Narasimhan。思维树：利用大语言模型进行审慎的问题求解。arXiv预印本，arXiv:2305.10601，2023年。[88] Qinyuan Ye、Maxamed Axmed、Reid Pryzant 和 Fereshte Khani。提示工程师的提示工程。arXiv预印本，arXiv:2311.05661，2023年。[89] Zhisheng Ye、Peng Sun、Wei Gao、Tianwei Zhang、Xiaolin Wang、Shengen Yan 和 Yingwei Luo。Astraea：面向多租户GPU集群的公平深度学习调度器。IEEE并行与分布式系统汇刊，2021年。[90] Gyeong-In Yu、Joo Seong Jeong、Geon-Woo Kim、Soojeong Kim 和 Byung-Gon Chun。Orca：面向基于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，2022年。[91] Minchen Yu、Ao Wang、Dong Chen、Haoxuan Yu、Xiaonan Luo、Zhuohao Li、Wei Wang、Ruichuan Chen、Dapeng Nie 和 Haoran Yang。FaaSwap：通过模型交换实现SLO感知且GPU高效的无服务器推理。arXiv预印本，arXiv:2306 03622，2023年。。[92] Chengliang Zhang、Minchen Yu、Wei Wang 和 Feng Yan。MArk：利用云服务实现经济高效、SLO感知的机器学习推理服务。载于2019年USENIX年度技术会议（USENIX ATC 19），2019年。[93] Saizheng Zhang、Emily Dinan、Jack Urbanek、Arthur Szlam、Douwe Kiela 和 Jason Weston。个性化对话智能体：我养了一只狗，你养宠物吗？载于第56届计算语言学协会年会论文集（第1卷：长论文），第2204–2213页，澳大利亚墨尔本，2018年7月。计算语言学协会。

[94] Lianmin Zheng、Zhuohan Li、Hao Zhang、Yonghao Zhuang、Zhifeng Chen、Yanping Huang、Yida Wang、Yuanzhong Xu、Danyang Zhuo、Joseph E Gonzalez 等。Alpa：为分布式深度学习自动实现算子间与算子内并行。*arXiv*预印本 *arXiv:2201.12023*，2022年。

[95] Yongchao Zhou、Andrei Ioan Muresanu、Ziwen Han、Keiran Paster、Silviu Pitis、Harris Chan 和 Jimmy Ba。大语言模型是人类水平的提示工程师。*arXiv* 预印本 *arXiv:2211.01910*，2022。