S-LORA: SERVING THOUSANDS OF CONCURRENT LoRA ADAPTERS

S-LoRA：服务于成千上万的并发的LoRA适配器

**太长不看版**

本文介绍了S-LoRA，为具有庞大数量的LoRA适配器的服务部署提供了一种新的方法，其实现主要有三个关键部分。

其一是批处理。在批处理阶段，不同于原始LoRA论文推荐的那样，S-LoRA没有将LoRA参数与基础模型的参数进行合并，而是将它们拆分开来计算。尽管这样会产生一些额外的成本，但与合并参数相比，仍然具有巨大的成本优势。此外，S-LoRA采用了一种token级别的调度策略，并将使用相同适配器的请求进行聚类，外加一种准入控制策略（详见原文附录B），实现了较为高效的批处理。

其二是内存管理。S-LoRA将所有的适配器权重都存储在内存中，只有当前批次需要时才调入到GPU显存中。在内存管理中，S-LoRA将PagedAttention的思想进行了扩展，同时管理KV缓存与适配器权重。此外，S-LoRA还采用了一种预测机制，根据当前的等待队列预测下一批次需要使用的适配器权重，并在下一批次开始之前提前载入，大大减少了I/O时间。最后，S-LoRA还为非连续内存的异构LoRA批处理提出了两个自定义CUDA内核。

最后是并行策略。S-LoRA将Megatron-LM进行了扩展，将新增的LoRA计算的输入和输出的分区策略与基础模型的分区策略对齐，从而减少了多GPU间的通信与内存成本。

本文也进行了详尽的实验，实验结果表明S-LoRA是一种优秀的、能够为大量基于同一基础模型的LoRA适配器提供服务的高效框架。

**摘要**

当部署大语言模型时，通常采用“先预训练后微调(pretrain-then-finetune)”的方法。LoRA是一种参数高效的微调技术，常用于将基础模型适应于多种任务，从而形成大量LoRA适配器，这体现出了在推理过程中进行批处理（将多个推理请求集中处理）的意义。为了实现这种批处理，作者提出了S-LoRA系统。S-LoRA将所有适配器都存储在内存中，并将当前运行的查询所需的适配器加载到GPU显存中。同时，为了有效利用GPU显存，并减少碎片化，S-LoRA还提出了统一分页技术，即利用统一的内存池管理具有不同秩的适配器权重和具有不同序列长度的KV缓存张量(Key-Value Cache Tensors)。此外，S-LoRA还采用了一种新的张量并行策略和高度优化的自定义CUDA核心，用于异构批处理LoRA计算。这些功能使得S-LoRA能够在单个GPU或多个GPU上为数千个LoRA适配器提供服务，并且保持高吞吐量和低额外开销。与HuggingFace PEFT和vLLM等相比，S-LoRA的吞吐量可以提高多达4倍，并且服务的适配器数量可以增加数个数量级。

**1.引言**

大语言模型在现代应用广泛，并且表现出了卓越的性能（尤其是针对特定任务微调后）。而“先预训练后微调(pretrain-then-finetune)”的方法使得单个基础模型产生了针对不同任务的大量变体。

当需要针对多个任务对大预言模型进行微调时（例如，训练个性化助手），成本可能会很高。为此，先前研究者提出了几种参数高效微调的方法，其中一个就是LoRA。LoRA通过更新低秩附加矩阵来实现微调，这些矩阵参数相较于整个大模型而言很少，并且能够实现与对参数进行完全微调相媲美的性能。然而，尽管对微调进行了大量的研究，但如何大规模地应用这些大语言模型的变体仍然是一个未解决的问题。

LoRA论文中的一个关键创新点是直接将适配器与模型参数合并来消除推理延迟。同时，为了能够在单台机器上支持多种变体，论文中也提出了通过向基础模型添加或删去LoRA权重来更换适配器的方法。虽然这样可以实现对单个适配器的低延迟推理和跨适配器的串行执行，但它：

* 显著降低了整体服务的吞吐量；
* 在同时为多个适配器提供服务时增加了总延迟；
* 没有考虑利用机器内存来增加单台机器托管的适配器的数量。

作者观察到，众多LoRA适配器使用同一个共享的基础模型，这为批处理推理提供了很高的可行性。

虽然在基础模型中运用批处理十分简单（所有查询共享一个基础模型），但在适配器中使用批处理仍然具有一些挑战。同时，为了实现高吞吐率，也需要将基础模型的计算与单个LoRA的计算分割开来。

首先，同时为多个LoRA适配器提供服务需要高效的内存管理。由于GPU显存有限，将所有适配器权重都存储在显存中是几乎不可能实现的，因此权重必须存储在显存之外，并在需要的时候动态获取。然而，适配器的大小不尽相同，动态地加载与卸载存在一定的困难。此外，为不同序列长度的请求动态分配和释放KV缓存张量(Key-Value Cache Tensors)可能导致严重的内存碎片化和高昂的I/O开销。

此外，在非连续内存中分离计算具有不同秩的许多适配器仍然是一个困难，需要开发新的计算核心。最后，利用单台机器上的多个GPU需要新的并行策略来适应增加的LoRA适配器，其中的通信与内存开销的降低也是一个难题。

为此，作者提出了S-LoRA，引入了统一内存池，采用了高度优化的自定义CUDA核心，并引入了一种新的张量并行策略。在Llama-7B/13B/30B/70B上的评估结果显示，与Huggingface PEFT相比，S-LoRA的吞吐量可提高多达30倍；与vLLM相比，S-LoRA的吞吐量可以提高多达4倍，并且适配器数量可以增加数个数量级。

**2.背景**

LoRA(Low-Rank Adaptation)是一种参数高效的微调方法，动机源于在适应过程中模型更新的低内在维度。在训练阶段，LoRA将预训练的基础模型的权重固定，并为每一层添加可训练的低秩矩阵；在推理阶段，原始论文中建议将低秩矩阵与基础模型的权重合并。因此，在推理过程中不会增加额外的开销。

对于一个预训练的权重矩阵，LoRA会将其更新为，其中，，并且秩。如果使用定义基础模型的前向传播过程，那么，在引入LoRA后，前向转播就相应地转换为：



通常，这种调整仅适用于自注意力模块中的Q(query)、K(key)、V(value)和输出投影矩阵，并不包含前馈模块。

**2.1服务于大语言模型**

大多数的大预言模型(LLMs)都基于Transformer结构，其中的参数从数十亿到数万亿不等，相应的磁盘大小也从数千兆字节到数TB不等，这导致LLM的服务具有显著的计算和内存需求。

此外，LLM的推理过程需要迭代自回归解码(iterative autoregressive decoding)。模型首先进行一次前向传播来编码提示，随后，逐个token地解码输出，这个过程使得解码很慢。由于每个token都需要注意到其前面所有token的隐藏状态，因此很有必要存储所有先前token的隐藏状态，这种存储称为“KV缓存(Key-Value Cache Tensors)”。这种机制增加了内存的开销，并且导致解码过程需要比计算资源更多的内存资源。

当提供在线服务时，具有不同序列长度的请求会动态到达。为了处理这种请求，之前的研究引入了一种精细的、迭代级的调度方法。这种方法并不在请求级别进行调度，而是在token级别进行批处理。这种方法允许持续将新请求添加到当前运行的批次中，从而提高吞吐量。vLLM通过PagedAttention进一步优化了上述方法的内存效率。vLLM采用了操作系统中虚拟内存和分页的概念，并以分页方式管理动态KV缓存张量(Key-Value Cache Tensors)的存储和访问，从而有效地缓解了碎片化，并提高了批处理大小和吞吐量。

当模型需求的GPU显存不能由单个GPU满足，或者存在更严格的延迟要求时，需要在多个GPU上并行优化模型。目前已经提出了几种并行方法：tensor parallelism（张量并行）、sequence parallelism（序列并行）、pipeline parallelism（管道并行）等。

**3.S-LoRA预览**

S-LoRA包括三个创新组件：批处理策略、PagedAttention的推广以及一种新的张量并行策略。

**4.批处理和调度策略**

**4.1批处理**

对于单个LoRA适配器，推荐的做法是将适配器权重合并到模型的基础权重中，从而产生一个新的模型。这样做可以保证新旧模型参数量相同，从而保证在推理过程中没有额外的开销。然而，对于多个LoRA适配器，这种合并意味着需要维护多个完整的大模型副本，会导致批处理难以进行。为此，在原始的LoRA论文中，也提出了动态添加和删去LoRA权重的方法，但并不支持单独的LoRA适配器的并发推理，从而不能直接应用于批处理。

在研究中，作者证实了直接将适配器权重进行合并是低效的，并建议直接按照的方式来进行前向传播的计算。这样可以避免原始大模型权重的复制，并且使得计算成本更高的部分可以进行批处理。尽管这种方法也带来了一定的额外开销（更多的参数），但是，由于的成本要远远小于，额外的开销要明显低于节省的成本。

在S-LoRA中，会批量计算 基础模型的结果，然后使用自定义的CUDA内核执行的计算。使用自定义的CUDA内核，是因为序列长度与适配器的秩不尽相同，直接使用BLAS库中的batch GEMM kernel实现基础模型与LoRA适配器的分解计算需要进行大量的填充，导致硬件利用率降低。

手机屏幕截图

中度可信度描述已自动生成

S-LoRA示意图

虽然LoRA适配器的权重被存储在内存中，可以存储的数量很大，但当前运行的批次中所需的适配器数量是可管理的，同时也受到GPU显存大小的限制。在每个批次的推理中，仅仅将当前批次所需要的适配器载入到GPU显存中，那么可服务的最大适配器数量就会受到内存大小的限制。为了提高吞吐量，S-LoRA采用了迭代级调度批处理策略(iteration-level scheduling batching strategy)，将请求在token级别进行调度，如果空间可用，那么就立即将请求纳入运行中的批次，一旦达到最大token数或者满足其他停止条件，那么就令该请求退出批次。这个过程减少了GPU显存的使用，但引入了新的内存管理挑战。

图示

描述已自动生成

S-LoRA推理过程中内存与显存的使用情况

**4.2适配器聚类**

为了增强批处理的效率，可以减少运行批次中适配器的数量，从而分配更多的GPU显存给KV缓存，从而实现更大的批处理大小。由于GPU的显存有限，并且在解码过程中利用率不高，因此，更大的批处理大小会带来更高的吞吐量。为了减少运行批次中的适配器数量，一个直接的方法就是优先考虑使用相同适配器的请求，这种策略被称为“适配器聚类(Adapter Clustering)”。然而，适配器聚类可能会影响适配器之间的平均延迟或计算资源分配的公平性。在原文附录A中，消融实验的结果说明了不同集群大小下，吞吐量和延迟的变化。

**4.3准入控制**

S-LoRA也采用了一种准入控制策略，以在流量高于服务系统容量时维持良好的表现。一个服务系统的表现通常由服务水平目标(service level objective, SLO)体现，该目标代表了请求处理的期望延迟。S-LoRA中采用了一种早期中止策略，即估计可以在SLO中提供服务的最新请求集合，然后按照到达时间的顺序为它们提供服务。具体细节请参阅原文附录B。

**5.内存管理**

由于需要动态加载和卸载不同大小的适配器权重，带来了内存碎片化的问题与额外的延迟。为了解决这两个问题，S-LoRA采用了统一分页的方法，通过预取适配器权重来重叠I/O和计算。

**5.1统一分页**

动态加载和卸载适配器的权重类似于动态KV缓存。KV缓存的大小会根据序列长度的不同进行波动，而适配器权重的大小也会随着选择的适配器不同而变化；同时，KV缓存张量的形状为，而适配器的权重形状为，它们的维度大小都为。

受到这样类比的启发，作者对PagedAttention的思想进行了扩展，同时管理KV缓存与适配器权重。首先，静态地在显存池分配一个大缓冲区，该缓冲区可以使用在基础模型权重与临时激活张量以外的所有空间。KV缓存和适配器权重都以分页方式存储在这个内存池中，每个页面对应于一个大小为H的向量。也就是说，序列长度为S的KV缓存张量使用S个页面，而秩为R的LoRA权重张量占用R个页面。同时，KV缓存和适配器权重交错存储，而非连续存储。这样的方法显著减少了碎片化，确保了不同秩的适配器权重能够以结构化和系统化的方式与动态KV缓存共存。

图表, 图示, 条形图

描述已自动生成

统一分页示意图

**5.2预取与重叠**

虽然统一内存池减少了碎片化，但加载和卸载的I/O开销仍然是一个问题，尤其是适配器数量庞大或适配器权重庞大地时候。

为此，S-LoRA采用了一种动态预测机制，在运行当前解码批次的同时，根据当前等待队列预测下一个批次所需的适配器，从而在运行下一个批次之前，将大部分需要的适配器提前准备好，进而减少了适配器交换的I/O时间。

**5.3用于非连续内存的异构LoRA批处理的自定义内核**

由于统一内存池的设计，适配器权重存储在非连续内存中。为了在这种设计下高效运行计算，S-LoRA实现了支持在非连续内存布局中批处理具有不同秩和序列长度的LoRA计算的自定义CUDA内核。

在预填阶段，内核处理一系列标记，并从显存池中收集具有不同秩的适配器权重。这个内核被称为多尺寸批量聚集矩阵-矩阵乘法(Multi-size Batched Gather Matrix-Matrix Multiplication,MBGMM)，采用了瓦片技术进行实现。

在解码阶段，内核处理单个标记，并从显存池中收集具有不同秩的适配器权重。这个内核被称为多尺寸批量聚集矩阵-向量乘法(Multi-size Batched Gather Matrix-Vector Multiplication ,MBGMV)，是在先前研究基础上改进的。

**6.张量并行策略(TENSOR PARALLELISM)**

作者涉及了新的张量并行策略，用于支持大型Transformer模型中多GPU的LoRA批处理。张量并行(tensor parallelism)是最广泛使用的并行方法，因为其实现简单，并且易于与现有系统集成。张量并行可以降低在服务大型模型时每个GPU的内存使用量和延迟。在S-LoRA中，额外的LoRA适配器引入了新的权重矩阵和矩阵乘法，这需要对这些新增项采用新的分区策略。

**6.1分区策略**

由于基础模型使用了Megatron-LM张量并行策略，S-LoRA旨在将新增的LoRA计算的输入和输出的分区策略与基础模型的分区策略对齐，从而最小化通信成本。

以2层MLP为例，在基础模型中，Megatron-LM将第一个权重矩阵按列分区，第二个则按行分区。之后，通过全归约(all-reduce)操作将各个设备上的部分和进行汇总。

类似地，在LoRA中，第一层权重中，与都是按列进行划分的，而在第二层权重中，将按行分区，将按列分区，并使用全归约(all-reduce)操作总结中间结果。

最后，将LoRA计算的结果(matmul\_4)与基础模型的结果进行叠加，再通过全归约(all-reduce)操作得出最终结果。整体过程如下图所示：

图形用户界面, 图示

描述已自动生成

S-LoRA分区策略

那么，与Megatron-LM策略类似，对自注意力层的头维度进行分区，Q、K与V的投影权重矩阵可以视为示例中的，输出投影权重矩阵可以视为示例中的，就实现了MLP到自注意力机制的迁移。

**6.2通信与内存成本**

记为 设备数，为标记数，为隐藏大小，为适配器秩。

基础模型的通信成本是一个全归约(all-reduce)，可表示为 。LoRA计算的通信成本是三个全归约(all-reduce)（Q、K、V）外加一个全归约(all-reduce)（输出投影），可以表示为。与基础模型的通信成本相比，LoRA计算的通信成本是可以忽略不计的()。

就内存使用而言，S-LoRA将所有权重矩阵分配到所有设备上，并且没有重复的权重矩阵，是十分高效的。

**7.评估**

S-LoRA的评估建立在LightLLM之上，这是一个基于PyTorch和Triton的单模型LLM服务系统。在评估中，使用到的适配器数量达到了2,000个。

**7.1设置**

**模型**

采用的是Llama模型，具体的模型参数与适配器秩在下图中呈现：

表格

描述已自动生成

实验参数

**硬件**

实验中分别使用了单个Nvidia A10(24GB)、单个Nvidia A100(40GB)、单个Nvidia A100(80GB)和多个Nvidia A100(40/80GB) 进行了实验，内存则根据GPU不同，在64GB到670GB之间变化。

**基线**

* **HuggingFace PEFT**是用于训练和运行参数高效微调模型的库，但缺乏高级批处理和内存管理。实验中，使用其构建了一个服务器，使其可以批处理单个适配器的请求，也可以在批处理之间切换适配器权重。
* **vLLM -packed**是基于vLLM的简单多模型服务解决方案。由于其不支持LoRA，实验中，所以将LoRA参数合并到了基础模型之中。为了为个适配器提供服务，在实验中，在单个GPU上运行了个vLLM工作进程，这些进程由Nvidia MPS管理，并根据每个进程的平均请求率比例分配GPU内存。
* **S-LoRA**是完整的S-LoRA，并且遵循先请求先服务的策略。
* **S-LoRA-no-unify-mem**则在S-LoRA中去除了统一分页策略。
* **S-LoRA-bmm**在S-LoRA中去除了统一分页和自定义CUDA内核。它将适配器权重复制到连续的显存空间中，并进行带填充的批量矩阵乘法。

**指标**

按照通常的做法，在实验中统计了吞吐量、平均请求延迟、平均第一个token延迟和SLO达成率，SLO达成率是在6秒内返回第一个标记的请求的百分比。此外，还引入了“用户满意度(user satisfaction，详情参阅原文附录B)”。简单来说，较短的第一个token延迟会带来更高的满意度，而如果第一个token延迟超过SLO，那么满意度为0。

**7.2虚拟工作负载的端到端结果**

**工作负载追踪**

在实验中，使用Gamma过程进行工作负载的追踪。具体来说，对于第个适配器（共有个适配器），设定其平均速率为，变异系数(coefficient of variance, CV)为。其中，平均速率遵循幂律分布，指数为。同时，设定总请求速率为。实验中采用的基础模型、适配器秩等参数均如上一节（7.1）中表格所示。对于每个请求，输入和输出长度分别从均匀分布和中采样，默认追踪的持续时间是5分钟。为了实验的全面性，首先采用了一组默认参数，之后变化、、与中的一个，以观察其对性能带来的影响。默认参数如下图所示：

表格

描述已自动生成

默认参数

**与其他系统的比较**

在实验中，将S-LoRA与vLLM-packed和HuggingFace PEFT进行了比较，结果如下图所示。

表格

描述已自动生成

S-LoRA与其他方法比较结果

S-LoRA可以同时为2,000个适配器提供服务，对于添加的LoRA计算只需要最小的额外开销，而vLLM-packed需要维护多个权重副本，只能服务于不多于5个适配器，HuggingFace PEFT缺乏先进的批处理方法，内存管理性能明显较差。总体而言，S-LoRA在为少量适配器提供服务时的吞吐量最多比vLLM-packed高出4倍，并且比HuggingFace PEFT高出30倍，同时支持更大数量的适配器。

**与自身变体的比较**

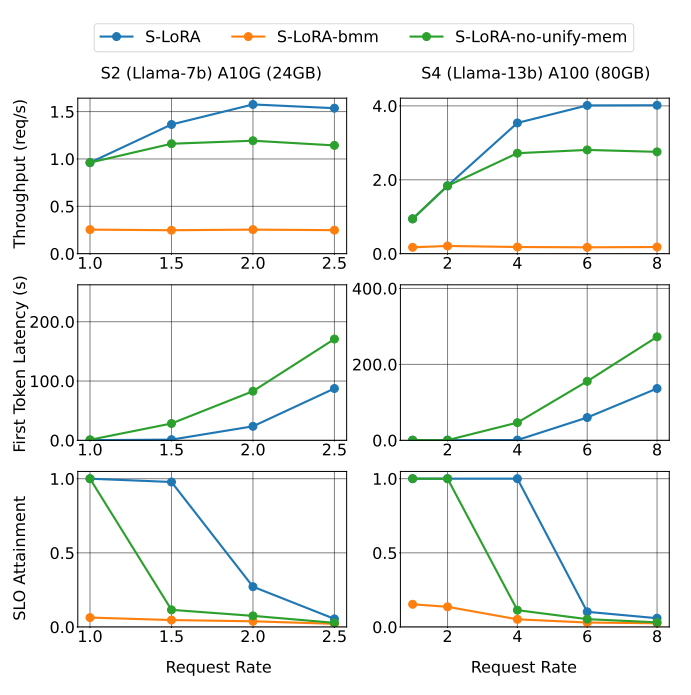
由于没有基准系统能够有效地扩展到大量的适配器，实验的重点放在了将S-LoRA与其自身变体进行比较。结果如下图所示：

图形用户界面, 应用程序, 表格, Excel

描述已自动生成

S-LoRA各变体比较

与S-LoRA-bmm和S-LoRA-no-unify-mem相比，S-LoRA的吞吐量明显更高，延迟更低。当适配器数量增加时，S-LoRA的吞吐量最初会由于LoRA的引入而略微下降。然而，一旦适配器数量达到一定阈值之后，S-LoRA的吞吐量不再下降。这种稳定性可能是由于随着适配器数量的增加，当前运行批次的激活适配器数量保持不变，从而保持了恒定的开销。因此，S-LoRA可以在不增加额外开销的情况下扩展到更大数量的适配器，仅受到可用内存的限制。下图展示了吞吐量、第一个token的延迟和SLO达成率相对于总请求速率的变化情况。



吞吐量、第一个token的延迟和SLO达成率相对于总请求速率的变化情况

**7.3实际工作负载的端到端结果**

**实际工作负载追踪**

在实验中，通过从LMSYS Chatbot Arena的追踪中进行降采样来构建实际的服务追踪。原始的Arena并不涉及LoRA适配器，但可以将不同基础模型的分布视为单个基础模型的不同适配器的分布，从而套用Arena的方式。为了将原始的结果采样成为对不同请求速率()和持续时间()的追踪结果，作者从原始的日志中抽取个请求，并将时间戳调整到的范围内，并将适配器的数量与模型的数量进行了对应。

由于采用的是实际的工作负载，并没有、等参数，为了保持一致性，就将持续时间设置到了5分钟，通过调整请求速率来研究其对性能的影响。平均输入长度为85个token，平均输出长度为165个token，适配器数量约为26个。

**结果**

下图展示了吞吐量和达成率的结果，结果与虚拟工作负载中的结果类似。

图表, 折线图

描述已自动生成

实际工作负载结果

**7.4多GPU张量并行**

在实验中，采用了如下两种情况进行验证：

1. 两个A100(40GB)和四个A100(40GB)上运行Llama-30B，适配器数量从10到100；
2. 两个A100(80GB)和四个A100(80GB)上运行Llama-70B，适配器数量为10

吞吐量结果如下图所示：

图表, 条形图

描述已自动生成

多GPU并行吞吐量结果

可以发现，有和没有LoRA通信的S-LoRA之间的差异很小，这表明LoRA通信具有非常小的开销，可以忽略不计。同时，也可以发现，由于LoRA引入的通信开销小于其引入的计算开销。此外，从2个GPU过渡到4个GPU时，服务吞吐量增加了两倍以上，这可能是由于系统主要受到内存不足的限制，从而在增加GPU时呈现出了超线性扩展。

**7.5消融实验**

虽然S-LoRA不会合并适配器权重，并且每次都会即时计算LoRA矩阵，但作者还是将其与合并适配器与基础模型的方式进行了比较。结果如下图所示：

图表, 折线图

描述已自动生成

合并方法与分隔方法性能比较

只有一个适配器时，合并的方法要明显优于分隔的方法，这主要是因为一次性合并了所有成本。但是，当适配器数量增加时，其性能产生了下降，主要是因为在适配器之间切换需要耗费时间，从而延长了GPU利用率不足的时期。此外，较小的值会导致请求在适配器之间分配不均匀，从而减少批次大小和整体性能。

**早期终止策略实验**

实验中，比较了S-LoRA的早期中止策略与先来先服务（FCFS）和后来先服务（LCFS）策略。比较结果如下图所示：

图表, 折线图

描述已自动生成

各种策略比较结果

从结果中可以看出，S-LoRA的策略要优于两者，尤其时随着CV（coefficient of variance，变异系数）增加的时候。FCFS效果最差，通常会处理已经错过SLO的请求。LCFS类似于一种贪心算法，只优先考虑最新的请求，对于较小的CV效果很好，但是随着CV的增大，其性能会产生下降。具体信息请参阅原文附录B。

**8.相关工作**

**使用系统化技术优化LLM服务**

Transformer架构的重要性导致许多研究人员专门为其开发了服务系统。这些系统使用了先进的批处理机制、内存优化、GPU内核优化、模型并行性、参数共享和推测执行等技术，以实现高效的服务。其中，PetS与S-LoRA最接近。然而，PetS仅考虑了小型仅编码器BERT模型的服务，而不考虑生成式推理、适配器规模庞大和模型参数庞大的情况，因此无法解决本文中的问题。

在另一项并行工作中，探索了基模型和适配器的分解计算概念。自定义的CUDA核心也是基于此进行开发的，并支持不同等级和非连续内存的批处理。

**使用算法技术优化LLM服务**

除了系统级改进之外，还可以使用算法技术来增强推断效率，如量化、稀疏化和模型架构改进。这些方法可以减少内存消耗并加速计算，对模型质量有轻微的妥协。它们与本文中的技术是互补的。

**参数高效微调(PEFT,** **Parameter-efficient fine-tuning)**

最近的工作开发了用于大型预训练语言模型的参数高效微调方法，仅使用少量调整的参数即可进行微调，包括LoRA、Prefix-tuning、P-Tuning、Prompt tuning、AdaLoRA和。虽然本论文主要关注LoRA，但大多数技术也可以轻松应用于其他参数高效微调方法。

**通用模型服务系统**

多年来，通用模型服务领域取得了显着进展。早期研究中的显着系统包括Clipper、TensorFlow Serving、Nexus、InferLine和Clockwork。这些系统涉及批处理、缓存和模型布置等主题，满足单个和多个模型部署的需求。在更近期的发展中，DVABatch、REEF、Shepherd和AlpaServe等系统探索了多输入、多输出的批处理，优先权的设置和具有并行性的多路复用思想。尽管这些系统做出了重大贡献，但它们忽视了LLM服务中的自回归特性和参数高效适配器，存在潜在的优化空白。

**9.结论**

作者S-LoRA，能够从单台机器上为成千上万个LoRA适配器提供服务，并且其吞吐量比现有系统高得多。S-LoRA的实现得益于统一内存池设计、张量并行策略、适配器批处理和自定义CUDA核心。S-LoRA实现了大规模的、定制化的微调服务，对于部署适应各种需求的模型至关重要。S-LoRA的未来扩展将包括支持额外的适配器方法、增强的融合核心以及使用多个CUDA流来并行化基模型和LoRA计算。