MQLserve：基于量化的多任务大语言模型服务系统

符芳诚 夏义扉 崔 斌
（北京大学计算机学院 北京 100871）

摘 要 随着大语言模型（LLMs）的不断发展，针对各种各样下游任务进行微调并部署的需求也与日俱增，以 LoRA 为代表的高效微调技术和以 GPTQ、AWQ 为代表的模型量化技术发挥着至关重要的作用。然而，尽管这些技术在单任务场景下已经有了众多成熟应用，但在多任务场景却鲜有研究。具体来说，由于主流的量化方法会导致基座大模型无法在任务之间共享，现有多任务服务系统难以结合量化进行部署，限制了其在资源受限场景下的可用性。此外，现有多任务服务系统缺乏灵活的动态任务实时添加能力和针对多任务场景的调度算法支持，往往导致其系统的低吞吐、高延迟、低响应时间和极差的灵活性。本文针对现有 LLM 服务系统在多任务场景下的不足，创新地提出了一个多 LoRA 任务服务系统 MQLserve。一方面，本系统设计了一种灵活高效的动态多任务量化算法，支持多任务的模型联合量化，赋能多任务场景下量化模型的共享，显著降低了模型部署的显存需求；同时支持实时的动态任务添加，提升了线上服务的稳定性和灵活性。另一方面，本系统针对多任务场景，提出了一种新型的基于输出长度预测和聚类的调度算法，有效地解决了传统调度算法在多任务场景下存在的高昂显存开销和频繁显存切换等问题，提升了系统性能。实验结果表明，与现有多任务服务系统相比，MQLserve 在不同负载场景下，吞吐提高了 7.5%～58.1%，延迟降低了 9.6%～43.3%，平均响应时间缩短了 84.2%，平均 SLO 满足率提高了 330%。

关键词 多 LoRA 任务服务系统；LoRA；量化；多任务调度
中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2025.00517

MQLserve：Quantization-Based Multi-Task LLM Serve System

FU Fang-Cheng XIA Yi-Fei CUI Bin
（School of Computer Science，Peking University，Beijing 100871）

Abstract With the advancement of Large Language Models（LLMs），the need for efficient fine-tuning and deployment in downstream tasks has grown. Technologies like LoRA（efficient fine-tuning）and GPTQ/AWQ（quantization）are crucial. However，current quantization methods prevent model sharing across tasks，limiting multi-task deployment. Additionally，existing multi-task systems lack support for dynamic task addition and effective scheduling algorithms in the multi-task serving scenario，resulting in low throughput and high latency. To address these issues，this paper introduces MQLserve，a multi-LoRA task-serving system. It features a dynamic multi-task quantization algorithm that allows joint quantization across tasks，reducing memory use through model sharing，and supports real-time task addition for stability. A new output length prediction-based scheduling algorithm tackles memory and switching issues，enhancing performance. The experimental results show that compared to existing multi-task service systems，MQLserve

收稿日期：2024-08-13；在线发布日期：2025-01-03。本课题得到新一代人工智能国家科技重大专项（2022ZD0116315）、国家自然科学基金（62402011）、博士后创新人才支持计划（BX20230012）、中国博士后科学基金（2024M750103）、北京市自然科学基金（4244080）、智能平行技术国家级重点实验室（IPT-2024JK29）资助。**符芳诚**，博士，中国计算机学会（CCF）会员，主要研究方向为机器/深度学习系统、大数据分析与处理。E-mail：ccchengff@pku.edu.cn。**夏义扉**，博士研究生，中国计算机学会（CCF）会员，主要研究方向为深度学习系统。**崔 斌**（通信作者），博士，教授，博士生导师，中国计算机学会（CCF）会士，主要研究领域为数据库、大数据管理与分析、机器/深度学习系统。E-mail：bin.cui@pku.edu.cn。

improves throughput by 7.5%−58.1%, reduces latency by 9.6%−43.3%, shortens average response time by 84.2%, and increases the average SLO attainment rate by 330%.

Keywords multi-LoRA task serve system; LoRA; quantization; multi-task schedule

1 引 言

随着大语言模型(Large Language Models, LLMs)[1-3]在众多领域展现出惊人的效果,其在文本摘要[4-5]、机器翻译[6-7]、对话系统[8-9]等下游任务中的应用需求日益增长。由于模型规模的爆炸性增长和设备资源的限制,"高效微调[10-13]-量化部署[14-18]"已经成为大语言模型针对下游任务最常见的应用范式。一方面,以 LoRA[10]为代表的高效微调技术仅对基座大模型训练额外的小规模适配器(adapter)以适应特定任务,显著降低了模型微调的成本。另一方面,以 GPTQ[14]、AWQ[15]为代表的模型低比特量化技术可以在维持模型效果的情况下大幅降低大模型部署的显存需求,并降低推理过程中的访存开销。

尽管许多主流 LLM 服务系统(如 vLLM[19])已集成了对微调模型进行量化部署的支持,但这些主流系统专注于单个下游任务的场景。随着各类下游任务需求的上涨,如何高效地支持多任务服务场景逐渐成为了一个至关重要的问题,并催生出了 S-LoRA[20]和 Punica[21]等面向多 LoRA 的多任务服务系统。这些系统通过在不同任务之间共享统一的基座大模型并根据服务请求激活不同的 LoRA adapter,能够在单次批处理中同时处理多个不同的任务。然而,在多任务场景下,现有系统仍然面临着三方面的挑战。

第一,现有的多任务服务系统无法很好地结合 GPTQ[14]、AWQ[15]等主流模型量化方法。具体来说,由于主流模型量化方法需要使用与下游任务有关的数据集对数值分布进行矫正,并且不同任务的量化过程需要激活对应的 LoRA adapter,因此,在量化后不同任务的基座大模型存在差异、无法统一共享。这一缺陷使得现有多任务服务系统在设备资源受限的场景下存在性能不足、甚至不可用的问题。

第二,在实际的多任务服务场景下,可能需要实时增加新的任务,而现有系统仅支持静态任务数量,无法动态增删不同任务的 LoRA adapter;此外,在量化部署场景下,现有工作均无法支持在不影响现有任务的前提下引入对新任务的量化与部署。因此,当需要增加新任务时,现有系统需要对服务进行中止与重启,严重影响了线上服务的稳定性。

第三,不同任务的服务请求必然存在负载差异(如请求长度、处理时间等),并且需要加载不同的 LoRA adapter 进行处理。现有系统在执行服务调度时未对这些问题进行考虑,因此往往需要在单次调度中加载大量 adapter,并在相邻调度之间频繁地切换需要加载的 adapter,导致性能受限。

针对以上问题与挑战,本文提出了一个新颖的多 LoRA 任务服务系统 MQLserve(Multi-task Quantization-based LoRA serve system)。如表 1 所示,本系统通过采用创新的联合量化、动态任务添加以及调度技术,旨在资源受限环境下提供灵活、高效的多 LoRA 部署服务。本文的主要贡献如下:

首先,本文提出了一种创新性的动态多任务量化算法 MLGPTQ,可以利用多任务的数据来对基座大模型进行联合量化,使量化后的基座大模型可以在多任务之间进行共享;同时可以支持无损地量化动态添加的新任务,且几乎不影响在线服务性能。

其次,本文提出了一种新型的基于输出长度预测和聚类的调度算法,有效降低了多任务场景下的显存占用和显存切换开销,极大地提升了系统性能。

基于此,本文开发了一个端到端的多 LoRA 任务服务系统 MQLserve,该系统专为资源受限环境设计,采用先进的调度算法和动态任务添加策略,融合了 MLGPTQ 量化,显著提高了其在资源受限环境中的运行效率。

最后,本文通过丰富的实验证明,相比现有系统,MQLserve 可以在几乎不损失精度的情况下,在不同负载场景下吞吐提高 7.5%～58.1%,延迟降低 9.6%～43.3%,平均响应时间缩短了 84.2%,平均 SLO 满足率提高了 330%,并且在相同资源限制下可以支持更大规模的大语言模型。

表 1 不同服务系统支持功能对比

服务系统名称	支持多任务	支持联合量化	动态任务添加	多任务调度
vLLM[19]	×	×	×	×
S-LoRA[20]和 Punica[21]	√	×	×	×
MQLserve	√	√	√	√

2　研究背景

2.1　基于 LoRA 的高效微调

LoRA，全称 Low-Rank Adaptation，是一种低秩的参数微调方法[10]。与传统的微调方法相比，LoRA 仅微调基座大模型的一小部分参数，减少了所需的资源和训练时间。其原理是通过引入两个小型的低秩矩阵，将模型的原始权重矩阵的变化（$\Delta \boldsymbol{W} \in \mathrm{R}^{m \times n}$）替换为两个小型 LoRA adapter（$\boldsymbol{A} \in \mathrm{R}^{r \times n}$，$\boldsymbol{B} \in \mathrm{R}^{m \times r}$，$r \ll m, n$）的乘积，即这两个矩阵相乘的结果可以近似模拟原始矩阵的变化，从而对模型进行微调训练。

2.2　低比特量化技术

低比特量化是一种十分常见的数据压缩技术，被广泛应用于模型部署、分布式通信优化、低精度计算等场景。在模型部署中，低比特量化技术常被用于模型量化压缩，可有效减少模型的存储和计算开销，提高推理效率[22-25]，尤其在边缘设备和资源受限的环境中具有显著优势[26-27]。在分布式计算中，许多工作均采用低比特量化技术对需要通信交互的数据进行压缩以降低通信量、从而提高端到端运行效率[28-31]。此外，在可容忍计算精度误差的情况下，低比特量化技术也被用于降低数值的位宽，从而降低存储和计算开销，例如，在大规模深度学习模型训练中，采用半精度（16 比特）浮点数而非单精度（即 32 比特）浮点数进行计算已经是一种常见做法[32]，且许多研究工作也在尝试使用 8 比特浮点数以进一步降低计算开销[33-36]。

本文关注低比特量化技术在模型部署特别是模型量化压缩的应用。模型量化压缩技术有很多种，其中以训练后量化（Post Training Quantization，PTQ）最为常见，下面是一个典型的 PTQ[14] 的过程：

$$X_{\mathrm{INT}} = \left\lfloor \alpha \, \mathrm{Clip}\left(\frac{X_R}{\alpha}, Q_{\min}, Q_{\max} \right) \right\rfloor,$$

其中，X_R 表示的是量化前的实数参数，X_{INT} 表示量化为整型数后的参数，$Q_{\min}$ 和 $Q_{\max}$ 分别表示量化范围的最小值和最大值，α 表示缩放比例。不同的 PTQ 方法通过不同的方法来计算 α，或者利用不同的近似方法。为了提升量化后的模型精度，通常需要基于目标数据集对数值分布进行矫正，以更好地获得 α 等参数。许多已有工作已经表明，带数据集矫正的量化精度往往显著高于无数据集矫正的量化[37]。因此，本文围绕带数据集矫正的量化展开研究。

常见的需要数据集矫正的量化算法包括 GPTQ[14]、SpQR[38]、AWQ[15] 和 SmoothQuant[39] 等。这些量化算法均利用矫正数据集捕获任务的数据分布，而后对模型权重进行矫正、使其适应分布后进行量化。GPTQ 利用矫正数据集逐层优化权重，最小化量化误差；SpQR 在 GPTQ 的基础上，使用了多级量化，进一步降低了存储空间；AWQ 利用激活感知的缩放策略，保护在特定数据集下显著的权重，从而减少量化误差；SmoothQuant 则通过平滑特定数据集的激活异常值，将量化难度从激活转移到权重，实现高效的 8 比特量化。然而，这些工作都没有考虑多任务数据分布下的量化。

2.3　大语言模型推理和调度服务

在大语言模型推理中，输入与输出均为由词元（token）组成的序列。给定每条请求的输入（prompt，又称提示词），大语言模型推理包含两个阶段：在预填充（prefill）阶段，对输入进行一次性的前向传播操作，并计算得到输出的第一个 token，该阶段仅执行一次；在解码（decode）阶段，根据输出中最后的 token 以及前序步骤中得到的中间结果（通常称为 KV cache），计算出下一个输出 token，该阶段需要执行多次，直到输出完成（如输出结束 token 或达到长度上限）。

在推理服务场景中，不同的请求往往有着不同的输出长度，故不同的请求需要执行解码阶段的次数存在差异。如果不对请求的执行顺序进行合理调度，则会出现一个批次（batch）里短输出请求的推理已经结束、但是要被迫等待相同批次里长输出请求的情况。因此，请求调度对大模型推理服务的性能至关重要。Orca[40] 提出了细粒度、token 级别的调度方法，支持以先来先到（First-In-First-Out，FIFO）的方式在 token 级别进行批处理。具体的，在当前正在处理的批次中某条请求完成后，从等待处理的请求队列中取出最先到达的请求，加入到当前正在运行的批处理中，从而提升整体的吞吐量。Fastserve[41] 则根据请求的部分信息（如输入长度），基于跳跃-加入（skip-join）多级反馈队列（Multi-Level Feedback Queue，MLFQ）的思想，为请求分配不同的优先级，从而允许抢占调度以优化平均请求完成时间。

然而，在多任务场景下，来自不同任务的请求需要加载不同的 LoRA adapter 以进行推理。现有调度策略均没有考虑请求的任务性质，在多任务场景

往往面临着严峻的挑战(将在 3.3 节详细说明)。

2.4 研究动机:资源受限情况下端到端多任务服务系统缺失

本文将不同 LLM 服务系统的功能特点列举在表 1 中。容易看出:(1)主流LLM服务系统,如 vLLM,不支持多任务服务;(2)新兴的多任务服务系统,如 S-LoRA 和 Punica,不支持量化部署、缺乏针对多任务场景的调度算法、不支持动态任务添加,且无法进行端到端的部署,在资源受限场景下可用性十分有限。针对上述问题,本文提出了 MQLserve 系统,解决了资源受限的情况下部署多任务 LoRA 服务系统的问题,使得其可以在资源受限的情况下保持高准确度、高吞吐、低延迟、高用户体验,赋能多任务服务系统在消费级 GPU 等资源受限场景中应用,真正将多任务服务系统推广到个人场景。

3 设计细节

MQLserve 的系统架构如图 1 所示。其中动态多任务量化模块用来对部署的模型进行初始多任务联合量化并部署,当后续有新任务需要被引入时,该模块也负责对模型进行重量化;请求池用来缓存各种客户端发来的请求,并交予后续的多任务请求调度模块;多任务请求调度模块基于本文所提出的调度算法,对多任务的请求进行调度编排;每当多任务请求调度模块给出一组输入,与现有服务系统类似,MQLserve 将该输入送入推理流程(Inference Pipeline)进行处理,获得对应输出的预测值(Logits)并解码,解码得到的词元(token)则会被流式输出并被返回给调度模块,等待进行下一轮调度。

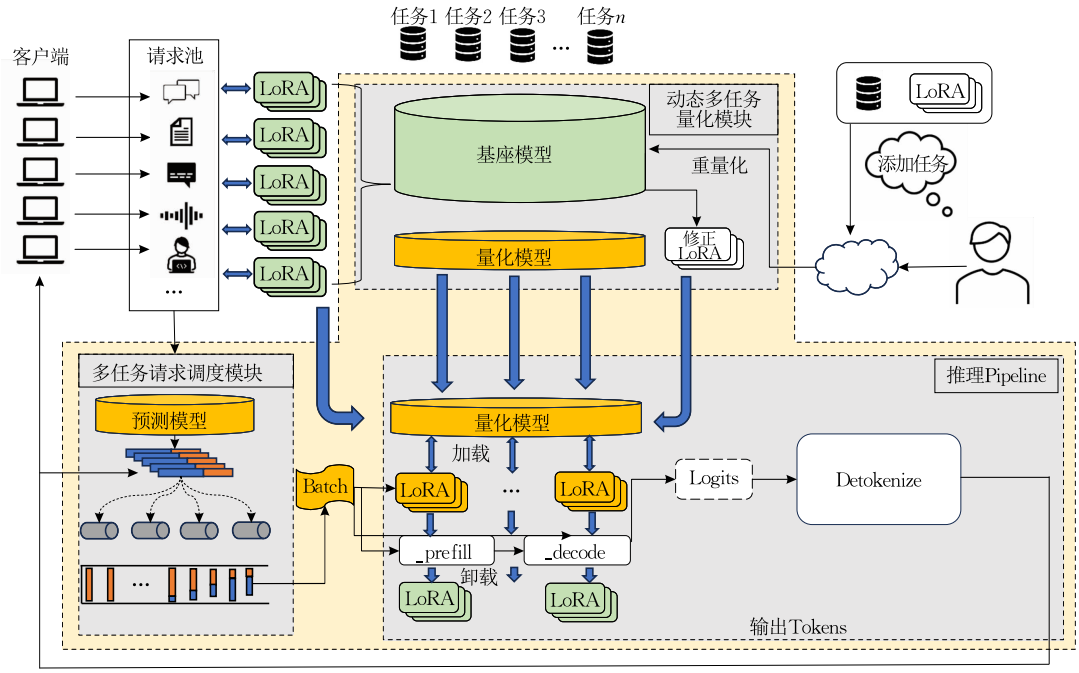

图 1 MLQserve 系统设计架构(其核心为两个模块:动态多任务量化模块,负责模型的初始多任务量化部署以及动态添加新的任务;多任务请求调度模块负责利用多任务的特性对请求进行调度。其他模块包含请求池,用来暂存未经处理的请求以及推理 Pipeline,用来执行请求的推理过程)

接下来,本文将介绍 MQLServe 的三个核心功能,分别是多任务量化、动态任务添加以及多任务请求调度。

3.1 多任务量化

如前文所述,由于主流量化方法需要使用任务相关的目标数据集进行矫正,且需要加载不同的 LoRA adapter 进行计算,因此对不同的任务进行量化所得到的基座大模型存在差异。为了可以在多任务服务场景下共享量化模型,存在两种朴素的做法,即不使用数据集矫正、或将所有任务的数据集进行混合以统一量化。然而,这两种做法无法精准地为每种任务捕捉单独的数值分布,因此会导致严重的模型精度损失(详见实验章节 4.2)。

为此,本文提出多任务场景下的量化算法——MLGPTQ,使得可以在只保留一份量化后的基座大模型的基础上对多任务联合量化,在保持准确度几

乎不损失的情况下，高效地服务多个任务。

3.1.1 MLGPTQ 量化算法

基于单任务量化算法 GPTQ[14]，本文提出了 MLGPTQ(Multi-LoRA GPTQ)算法，根据输入数据的激活值来适应对应的 LoRA adapter 以及基座大模型权重矩阵，使得量化后的激活值损失最小。其核心在于量化前向传播时根据当前批次的任务种类动态加载对应的 LoRA adapter，以准确预估数值分布，从而降低量化损失。

对于 n 个任务，要使得量化后的激活值损失最小，如下述公式所示：

$$\arg\min_{\hat{\boldsymbol{W}}} \left\| (\boldsymbol{W}+\boldsymbol{B}_i\boldsymbol{A}_i)\boldsymbol{X}-(\hat{\boldsymbol{W}}+\boldsymbol{B}_i\boldsymbol{A}_i)\boldsymbol{X} \right\|_2^2,\quad i\in\{0,1,\cdots,n-1\},$$

其中，$\boldsymbol{W}$ 表示基座大模型某一线性层的权重，$\hat{\boldsymbol{W}}$ 表示量化后的权重，$\boldsymbol{B}_i$ 和 $\boldsymbol{A}_i$ 表示第 i 个 LoRA adapter 的低秩适应矩阵，n 表示系统中服务的任务种类的数量，每种任务都有一个 LoRA adapter。MLGPTQ 的目的就是通过逐层量化，为每一个线性层的权重 $\boldsymbol{W}$ 找到一个 $\hat{\boldsymbol{W}}$，使得其激活值变化最小。

对于一个已经完成预训练和微调的模型，服务于某个任务 i 时，其 $W_i+\boldsymbol{B}_i\boldsymbol{A}_i$ 不会再改变。为方便叙述，本文记此 $W_i+\boldsymbol{B}_i\boldsymbol{A}_i$ 为 Wl_i。记模型目标函数为 E，由于其已预训练完成收敛，会达到一个局部极小值，其泰勒展开式为

$$\nabla E=\left(\frac{\partial E}{\partial Wl_i}\right)^{\mathrm{T}}\nabla Wl_i+\frac{1}{2}\nabla \boldsymbol{W}\boldsymbol{l}_i^{\mathrm{T}}\cdot\boldsymbol{H}\cdot\nabla Wl_i+O(\nabla\|Wl_i\|^3),$$

其中，$\boldsymbol{H}=\partial^2E/\partial W^2$ 为海森矩阵。由于其已经收敛，其一阶偏导数可近似为 0，且忽略高阶项 $O(\|\nabla Wl_i\|^3)$，得到误差为

$$\nabla E\approx\frac{1}{2}\nabla \boldsymbol{W}\boldsymbol{l}_i^{\mathrm{T}}\cdot\boldsymbol{H}\cdot\nabla Wl_i,$$

量化过程将 Wl_i 量化为 $Q(Wl_i)$，于是有 $\nabla Wl_{iq}=Q(Wl_{iq})-Wl_{iq}$，其中 Wl_{iq} 表示第 i 个任务的权重矩阵的第 q 个元素。于是得到优化的目标函数：

$$\arg\min_{q,\nabla Wl_i}\frac{1}{2}\nabla \boldsymbol{W}\boldsymbol{l}_i^{\mathrm{T}}\cdot\boldsymbol{H}\cdot\nabla Wl_i,$$

$$\text{s.t. }\ \boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}=Q(Wl_{iq}),$$

其中，$\boldsymbol{e}_q^{\mathrm{T}}$ 代表一个单位向量，其位置 q 处为 1，其余位置为 0。这是一个有约束的凸优化问题，利用拉格朗日乘数法，只需要求解如下等式：

$$L=\frac{1}{2}\nabla \boldsymbol{W}\boldsymbol{l}_i^{\mathrm{T}}\cdot\boldsymbol{H}\cdot\nabla Wl_i+\lambda(\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq})。$$

接下来对 ∇Wl_i 和 λ 求偏导，因为此时是在求稳定状态下的解，所以令偏导数为 0，得到(详细推导过程见附录 A)：

$$\begin{cases}\frac{1}{2}(\boldsymbol{H}+\boldsymbol{H}^{\mathrm{T}})\nabla Wl_i+\lambda\boldsymbol{e}_q=0,\\ \boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}-Q(Wl_{iq})=0,\end{cases}$$

解得

$$\lambda=\frac{Wl_{iq}-Q(Wl_{iq})}{(\boldsymbol{H}^{-1})_{qq}},$$

$$\nabla Wl_i=-\frac{Wl_{iq}-Q(Wl_{iq})}{(\boldsymbol{H}^{-1})_{qq}}\boldsymbol{H}^{-1}\boldsymbol{e}_q,$$

$$\nabla E=\frac{Wl_{iq}-Q(Wl_{iq})}{2(\boldsymbol{H}^{-1})_{qq}},$$

其中，$(\boldsymbol{H}^{-1})_{qq}$ 表示海森矩阵逆矩阵的对角线位置 (q,q) 的值。

以上便是 MLGPTQ 的原理，在实现中，每次获得一个批次的任务的少量数据，然后加载这个任务对应的 LoRA adapter，而后计算其海森矩阵并更新 Wl_i。在实际的模型部署中，由于 LoRA adapter 权重 $\boldsymbol{B}_i$ 和 $\boldsymbol{A}_i$ 的参数量相比基座大模型权重 $\boldsymbol{W}$ 来说可以忽略不计，所以仅需要量化基座大模型权重 $\boldsymbol{W}$，而 LoRA adapter 权重 $\boldsymbol{B}_i$ 和 $\boldsymbol{A}_i$ 不会被量化以保留微调过程所学习到的任务知识。即使用公式

$$\nabla \boldsymbol{W}=-\frac{Wl_{iq}-Q(\boldsymbol{W}+(\boldsymbol{B}_i\boldsymbol{A}_i)_q)}{(\boldsymbol{H}^{-1})_{qq}}\boldsymbol{H}^{-1}\boldsymbol{e}_q$$

来更新基座大模型参数矩阵 $\boldsymbol{W}$。同时，由于每一次更新都会改变 $\boldsymbol{W}$，导致 ∇E 也会改变，所以也要更新 E，从而更好地进行下一批次的更新。

3.1.2 算法工程实现

对于 $\boldsymbol{W}\in\mathrm{R}^{m\times n}$，MLGPTQ 的复杂度是 $O(mn^3)$，同时具有较低的计算访存比。本文采取了以下实现优化方法，将复杂度降低至 $O(\max\{mn^2,n^3\})$，并优化其计算访存比。算法伪代码见算法 1。

算法 1. MLGPTQ 量化算法

输入：$\{X_i\}_{i=1}^n$，$\{\boldsymbol{A}_i\}_{i=1}^n$，$\{\boldsymbol{B}_i\}_{i=1}^n$，$C$

▷分别表示任务 i 的输入激活(activation)、LoRA adapter权重 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，n 表示任务数，C 表示分块大小

1. $\boldsymbol{X}\leftarrow\sum_{i=1}^{n}(\boldsymbol{W}+\boldsymbol{B}_i\boldsymbol{A}_i)X_i$ ▷前向传播
2. $\boldsymbol{H}^{-1}\leftarrow\mathrm{Cholesky}((2\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}+\lambda I)^{-1})^{\mathrm{T}}$ ▷计算海森逆矩阵
3. $Q\leftarrow 0_{d_{\mathrm{row}}\times d_{\mathrm{col}}}$ ▷存储量化后的结果
4. $E\leftarrow 0_{d_{\mathrm{row}}\times C}$ ▷存储块内量化误差
5. for $j\leftarrow 0,C,2C,\cdots$ do
6. for $k\leftarrow j,\cdots,j+C-1$ do
7. $Q_{:,k}\leftarrow\mathrm{quant}(W_{:,k})$ ▷行内量化
8. $E_{:,k-j}\leftarrow(W_{:,k}-Q_{:,k})/(\boldsymbol{H}^{-1})_{kk}$ ▷更新块内量化误差
9. $W_{:,k:(j+C)}\leftarrow W_{:,k:(j+C)}-E_{:,k-j}\cdot(\boldsymbol{H}^{-1})_{:,k:(j+C)}$ ▷更新当前块权重
10. $W_{:,(j+C):}\leftarrow W_{:,(j+C):}-E\cdot(\boldsymbol{H}^{-1})_{j:(j+C),(j+C):}$ ▷更新剩余块权重

(1)随机顺序优化:上述 MLGPTQ 算法的一个核心是,按照产生量化误差最小的权重的顺序进行量化,然而,在实际实现中,调整量化顺序并不会影响量化方法的有效性。于是,本文利用 GPU 的并行计算能力,进行逐行计算,从而将复杂度从 $O(mn^3)$ 降低至 $O(\max\{mn^2, n^3\})$。

(2)批处理:由于同一个处理的矩阵 $\boldsymbol{W}$ 不同列之间的权重更新并不冗余,本文采用了批处理的方法,一次处理 C 个列,称为一个 block,提升了计算速度,同时采用了延迟更新策略,以减少频繁的访存。

(3)乔姆斯基分解:为增加数值计算稳定性,也即为了解决海森矩阵的逆出现不定的情况,本文使用数值稳定的乔姆斯基分解(Cholesky Decomposition)提前计算所需信息。

3.2 动态任务添加

在多任务场景下,存在动态任务添加(即增加 LoRA adapter)的需求,然而现有的多任务服务系统均不支持该功能。此外,由于 MQLserve 采用了多任务量化机制,为动态任务添加带来更多额外挑战:

(1)挑战 1:模型固定。MLGPTQ 利用预设固定种类的任务进行量化后,固定量化模型。新的任务加入后,由于旧的量化模型没有捕捉到它的信息,因此无法直接使用。

(2)挑战 2:服务打断。如果加入新的任务重新量化并更新模型,会长时间打断正在进行服务的量化模型,降低线上服务的稳定性。

(3)挑战 3:挤占显存。重新量化需占用大量的显存,减少了可用于推理服务的存储空间,进而降低了整体吞吐。

本文创新性地采用了逐层的异步 LoRA 更新机制,利用量化误差来调整新任务的 LoRA,而不打断正在线上服务的模型。如图 2 所示,动态任务添加的流程包含三个核心步骤。

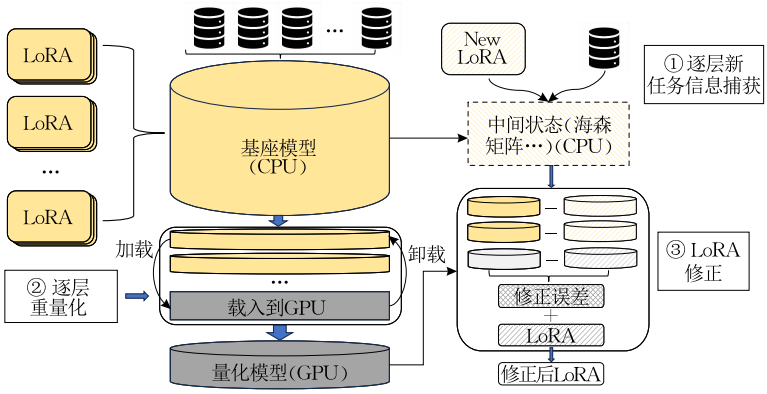

图 2 异步 LoRA 更新机制(该更新由多任务量化模块完成,涉及三部分数据,分别是新任务的 New LoRA 以及样本数据,暂存在 CPU 中的未量化的基座模型权重和初始服务任务的中间状态(海森矩阵),在线上服务的 GPU 中的量化模型)

(1)逐层新任务信息捕获:首先,MQLserve 利用新任务的 LoRA adapter、新任务的矫正数据集以及保存在 CPU 主存中的(未量化的)基座模型,进行前向传播,得到新任务的海森矩阵,捕获其数据信息。值得注意的是,MQLserve 逐层加载基座模型、进行前向传播、并释放该层的模型权重,从而降低 GPU 显存占用。

(2)逐层重量化:其次,在部署初始量化时,MQLserve 将初始任务的中间状态信息(即初始任务的海森矩阵)暂存在 CPU 主存中,在动态任务添加时,MQLserve 将此中间状态加载入 GPU 显存中,并与新任务的信息进行聚合。通过这种方式,MQLserve 不仅为新任务计算出中间状态信息,同时无损地保留了原有任务的所有信息。随后,MQLserve 将模型逐层载入到 GPU 进行重量化。

(3)LoRA 修正:最后,为了防止打断正在服务的量化模型,本工作不直接更新线上服务模型,而是选择修正 LoRA,从而降低更新代价。具体来说,本工作将重量化模型与线上服务模型作差,将误差修正到新 LoRA 上,得到修正后的 LoRA,无感知地发送给线上,而不涉及更新模型,在不打断服务且几乎不消耗 GPU 资源的同时,动态地将新任务的 LoRA 添加到线上服务的 LoRA 池中。

此外,动态任务添加的所有步骤,包括新任务信息捕获、逐层重量化以及 LoRA 修正,都是另外开启后台线程、异步进行的,因此,任务添加过程不会打断正在进行的服务,保证了线上服务的稳定性。

3.3 多任务请求调度

尽管关于 LLM 服务请求的调度算法已有许多相关研究,但未考虑多任务场景中的特殊性质:(1)如

图 3 所示，不同任务的输入、输出总长度分布差别非常大，而同任务却呈现聚集效应。比如，对于翻译任务，其输入输出长度都很短；对于总结类任务，其输入很长，输出很短；对于生成类任务，往往输入很短，输出很长；(2) 由于任务到来的随机性，不固定性，单次调度内会有多种任务，且相邻两批次调度之间任务种类差别大。

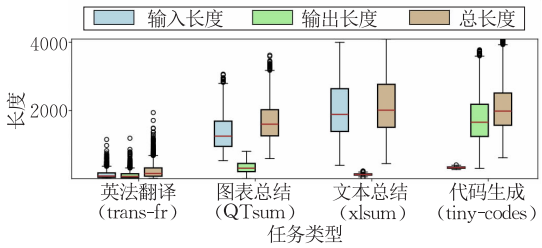

图 3　不同类型任务的输入、输出以及总长度分布

由于这些特性，现有请求调度算法在多任务场景下存在以下三方面的挑战：

(1) 挑战 1：输出长度差异显著导致平均任务完成时间长。现有 SOTA 的服务系统调度策略，比如 Fastserve[41] 的 skip join MLFQ 等，仅基于半透明信息进行近似调度，但缺乏了对输出长度信息的考虑，不适用于输出长度差异显著的多任务场景。

(2) 挑战 2：频繁换入换出 LoRA adapter 造成 IO 延迟。如图 4 所示，Fastserve 在相邻两批次之间任务数有很大起伏，这是由于 Fastserve 在调度过程中仅仅考虑了调度队列优先级，没有考虑任务种类变化，这种调度策略在多任务场景中相邻批次任务种类差异大会造成频繁的换入换出 LoRA adapter，这造成了不必要的 IO 延迟。

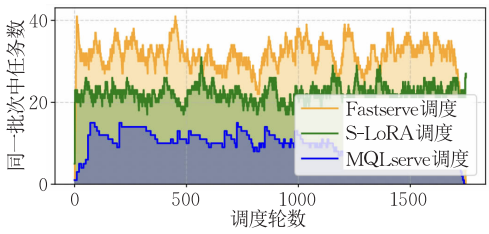

图 4　调度任务种类变化对比

(3) 挑战 3：批次中任务种类数目多，需要更多空间存储 LoRA adapter。仍从图 4 中看出，由于缺乏对任务种类数量的考虑，Fastserve 在每个批次中任务数非常高，这使得同一批次中需要在 GPU 中加载大量 LoRA adapter，浪费有效显存。

本文提出了基于输出长度预测和任务聚类的调度策略，很好地解决了上述挑战。

首先，本文受已有研究启发，利用蒸馏的小模型来预测请求的输出长度[42-43]；基于该预测，本文采取理论最优的调度策略——最短剩余时间优先策略 (SRTF) 算法进行调度；其次，在调度过程中，为了防止同一批次中的任务数量过多使得 LoRA adapter 占用显存造成浪费，本系统设置了聚类系数 β(默认值为 10)，令一个批次的任务种类数尽量小于 β，同时，系统会优先调度上一批次中的任务类型，避免相邻两次任务种类差别过大；最后，为了避免饥饿，系统设置了饥饿队列，会优先调度饥饿请求。具体调度策略算法流程见附录 D 的算法 2 和算法 3。

需要说明的是，S-LoRA[20] 也利用了聚类的思想以降低每个批次所包含的任务数量，然而，由于 S-LoRA 的调度算法是朴素的 FIFO 服务策略，其在服务过程中往往会出现负载不均的情况——由于来自长输出任务的请求会占用大量时间，使得来自短输出任务的请求迟迟得不到服务，使得不同任务之间存在出现不公平调度的问题，导致系统整体性能受限。相对地，MQLserve 采用了基于输出长度预测信息进行 SRTF 调度，加快了任务完成时间，同时，MQLserve 加入了饥饿队列，在保证负载均衡和减少任务完成时间的同时，很好地保证了公平性。此外，由于同种任务之间天然的负载相似性，SRTF 调度也有助于减少同一批次中的任务种类、提高相同任务被连续调度的概率，如图 4 所示，相比 Fastserve 和 S-LoRA，MQLserve 显著降低了同一批次中的任务种类数量，且前后两个批次的任务差异也更小。

4　实验结果与分析

4.1　实验设置

(1) 实验环境：本文使用 RTX 4090 和 RTX 3090 两种消费级显卡进行测试，两种显卡的详细参数配置如表 2 所示。

表 2　实验环境

实验平台	GPU 计算力/GFLOPS	显存容量/GB	访存带宽/(GB·s^{-1})
RTX 4090	51 640	24	1024
RTX 3090	35 580	24	936

(2) 数据集及负载：在测试模型精度时，本文选取了 6 个翻译任务[44] (trans-fr，trans-cs，trans-id，trans-nl，trans-da，trans-sw)、一个文本总结任务[45] (xlsum)、一个表格总结任务[46] (QTsum) 及一个代码生成任务[47] (tiny-codes) 共 9 个任务。任务数据集具体描述见附录 B。对于测试端到端的吞吐以及

延迟的负载，利用 Gamma process 生成大规模的虚拟负载[19-20]。

(3)模型：本文使用 Llama2-7b 和 Llama2-13b 模型进行测试。在测试模型精度时，本文对 9 种任务分别使用单独的 LoRA adapter，并分别评测模型精度。本文选用了开源的 LoRA adapter 进行模型精度测试，详见附录 B。在测试系统效率时，本文采用与 S-LoRA 相同的仿真方式，随机生成秩(rank)为 8、16、32、64 的 LoRA adapter，用以模拟任务数量更多的服务部署场景。

(4)评测指标：对于翻译任务，选取经典评测指标 sacrebleu[48]；文本总结、图表总结以及代码生成任务，选取 rouge[49]。其具体含义见附录 C。对于系统效率测试，本文考虑：吞吐率、平均请求延迟、任务完成时间、平均响应时间和 SLO(Service Level Object)满足率(多少请求在设置的期望延迟之前完成)。测试默认服务时间为 1 min，期望延迟为 6 s。在衡量方法 1 相对于方法 2 的提升时，本文计算提升率为 $(m_1-m_2)/m_2$，其中 m_1、m_2 分别代表方法 1 和方法 2 所达到的指标值；类似地，在衡量方法 1 相对于方法 2 的下降时，本文计算下降率为 $(m_2-m_1)/m_2$。

(5)对比对象：在测试模型精度时，本文将 MLGPTQ 与两类朴素方法进行对比，分别是无数据集矫正的 FP 量化方法(记为 FPquant)和将所有任务的数据集进行混合以统一 GPTQ 或 AWQ 量化的方法(记为 DPGPTQ 和 DPAWQ)。在测试系统效率时，本文测试多 LoRA 任务场景下的服务效率，选取了 vLLM 和 S-LoRA 作为对比对象。

4.2 模型精度及有效性评估

(1)模型精度：本实验以不量化(记为 noQuant)所达到的模型精度为基准，测试 MLGPTQ、DPGPTQ、DPAWQ 和 FPquant 相对于 noQuant 的模型精度下降率，详细结果如图 5 所示。在 4 bit 量化的情况下，MLGPTQ、DPGPTQ、DPAWQ 和 FP4 在 9 个数据集上平均的模型精度下降率分别为 1.20%、4.52%、3.72%和 3.18%。在 3 bit 的情况下，MLGPTQ、DPGPTQ、DPAWQ 和 FP3 的平均模型精度下降率分别为 9.93%、19.01%、20.11%和 90.92%。在所有数据集中，MLGPTQ 的模型精度下降都是最小的，充分体现了其稳定性。另外，在 4-bit 量化时，直接混合数据集进行量化的 DPGPTQ 和 DPAWQ 方法在大部分数据集上均不如 FPquant，说明了当 bit 数较多时，混合多个不同任务的数据分布是有害的，不如无需数据集矫正的 FP 量化；而在 3-bit 量化时，由于 bit 数过少，没有数据集矫正的 FPquant 几乎丧失所有能力，而利用数据集矫正则会增强其在极低比特下的表达能力。

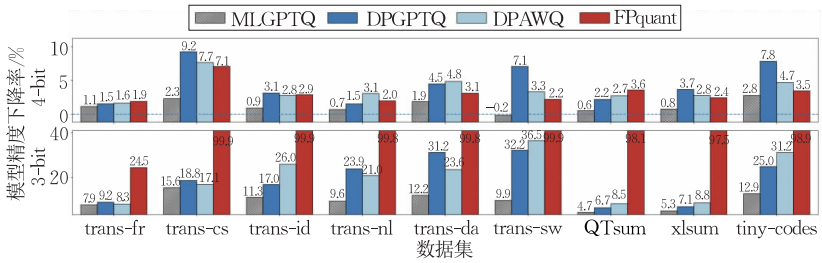

图 5 不同量化方法在每个数据集上的模型精度下降率(Llama2-7b)(模型精度下降率的计算公式为 $(m_{\text{noQuant}}-m)/m_{\text{noQuant}}$，其中 m_{noQuant} 代表不量化(noQuant)所达到的模型精度指标值，m 代表实验使用的量化方法所达到的模型精度指标值)

(2)有效性解释：本实验旨在解释 MLGPTQ 的有效性，为了使结果更加鲁棒，除了选取 sacrebleu 指标外，又另外选取了 5 个不同的衡量机器翻译质量的指标，指标详细解释见附录 C。量化的优劣取决于两点：① 量化前向过程中，是否对正确的任务模拟加载了正确的 LoRA adapter；② 量化前向过程中，是否出现了目标任务的数据分布。

本文对比如下四个方法，来对不同的任务进行 int4 量化：MLGPTQ(同时满足要求①和②)、DPGPTQ(不满足要求①)、no-Target GPTQ(noT GPTQ)，即在 DPGPTQ 量化过程中，特地省略了目标任务的数据，只利用其他杂数据进行量化(不满足要求②)、GPTQ 即普通的单任务 GPTQ 量化算法。结果如图 6 所示，雷达图极坐标表示相对于 noQuant 的模型精度下降率。可以直观看到，DPGPTQ 和 noT GPTQ 基本可以完全覆盖 MLGPTQ 和 GPTQ，这表明从任何评测指标上来看，DPGPTQ 和 noT GPTQ 的模型精度下降率都要大于 MLGPTQ 和 GPTQ，说明 MLGPTQ 很好地解决了 DPGPTQ 和 noT GPTQ 的问题，证明了其有效性。

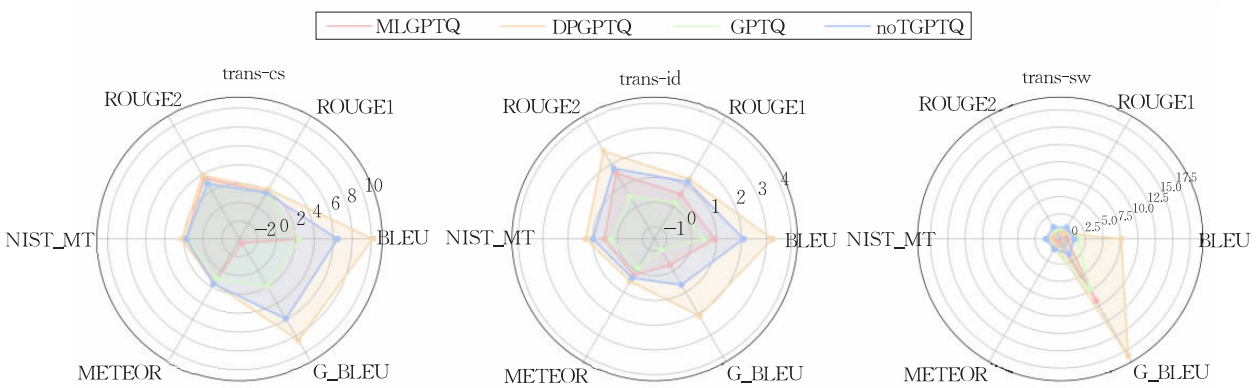

图 6　MLGPTQ 有效性验证(Llama2-7b)(以三个翻译数据集为代表进行了有效性验证，选取了 6 个衡量翻译质量的评价指标，极坐标代表相对于 noQuant 的模型精度下降率，越向里表示下降越小、效果越好)

(3) 任务数影响分析：为了验证随着任务数量增加对量化后模型精度的影响，本文以 trans-cs、trans-id、trans-da 三个数据集为目标任务，测试随着参与量化的任务数量的增加，MLGPTQ 和 DPGPTQ 的模型精度变化，结果如图 7 所示。当只有 1 个任务时，MLGPTQ 和 DPGPTQ 的模型精度相当。随着参与量化任务数的增加，MLGPTQ 的模型精度基本保持不变，而 DPGPTQ 则有着逐渐变差的趋势。此外，从实验结果可以看出，DPGPTQ 的精度下降率具有一定的波动性，其原因是所添加的任务与目标任务的数据分布(如语种相似性、输入和输出长度分布等)存在不同的相似性；然而，总体来说，随着任务数量的增加，DPGPTQ 将更多任务的矫正数据集进行混合，使得目标任务的数据分布逐渐被破坏，故模型精度下降越来越严重。MLGPTQ 则准确捕获了目标任务的数据分布，因此即使任务数量增加，其模型精度都维持稳定，充分证明了 MLGPTQ 关于任务数量的鲁棒性。

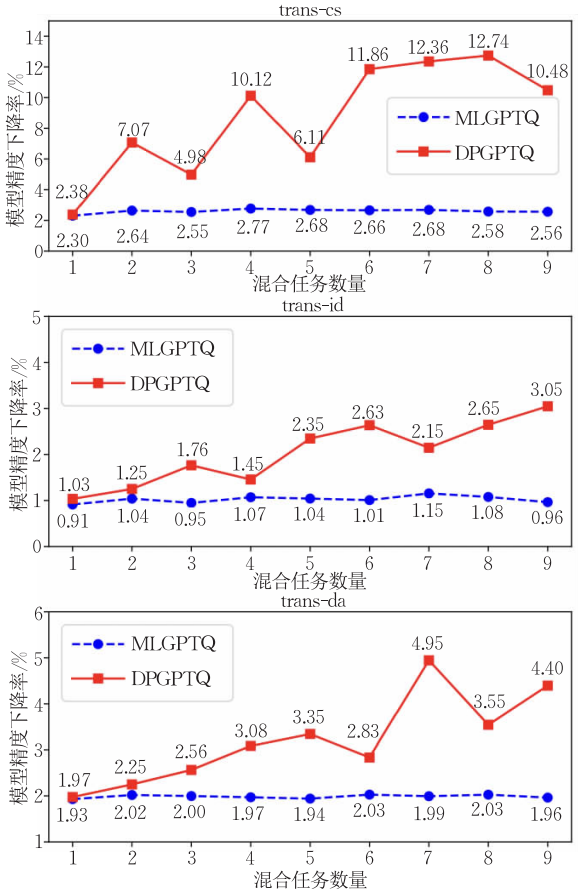

图 7　任务量化数对模型精度下降率的影响(Llama-7b)

4.3　端到端性能评测

4.3.1　实用性能：吞吐率、延迟、任务完成时间

本文在 RTX 4090 和 RTX 3090 上测试了 Llama2-7b、Llama2-13b 模型，均服务 100 个不同的 LoRA 任务，请求到达速率从 1 reqs/s 增加到 30 reqs/s，其吞吐率、延迟、任务完成时间见图 8。由于资源受限，S-LoRA 和 vLLM 在服务 Llama2-13b 时均出现了显存不足的错误(图中标注 OOM)，而 MQLserve 不会。图例中“-i”表明服务 i 个任务。总体来看，在不同负载场景下，相比于 S-LoRA，MQLserve 系统吞吐提高了 7.5%～58.1%，平均延迟降低了 9.6%～43.3%。

随着请求速率的不断升高，MQLserve 的平均吞吐不断上升，以 Llama2-7b@RTX4090 为例，最终稳定在的 4.93 reqs/s，而 S-LoRA 的仅为 3.79 reqs/s。在所有实验中，MQLserve 的平均吞吐率比 S-LoRA 提高了 30%(3.65 reqs/s vs. 2.81 reqs/s)，平均延迟比 S-LoRA 缩短了 27%(86.69 s vs. 119.29 s)，任务完成时间比 S-LoRA 缩短了 28%(226.14 s vs. 314.96 s)。同时，还可以得到如下结论：(1) 在所有配置下，无论请求速率如何，MQLserve 在吞吐量、平均延迟和任务完成时间上均优于 S-LoRA，充分证明

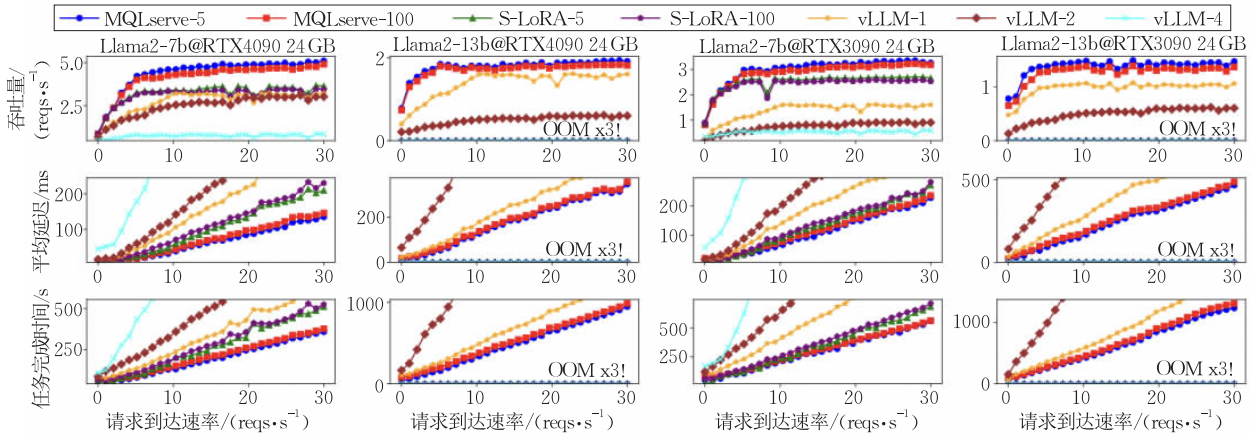

图 8 MQLserve、S-LoRA 和 vLLM 在不同请求到达速率情况下的吞吐量、平均延迟和任务完成时间（图中 OOM 表示显存不足、无法完成推理服务）

了其性能的稳定性和优越性；(2) 数据中偶尔出现异常抖动，比如 Llama2-7b@RTX3090 下的 S-LoRA 在请求到达速率为 8 reqs/s 时出现了较大的吞吐下滑，这是由于多任务场景下的请求分布不适合 S-LoRA 的 FIFO 的调度方式，容易出现一个批次里某些任务的请求长度过大而拖累长度较短任务的推理的现象，而利用多任务请求调度的 MQLserve 在处理同样负载时便没有出现此异常——由于 MQLserve 利用了基于预测的排序和聚类，并且利用了更加灵活的连续批处理，减少了同一批次内部请求长度的差异，很好地解决了此问题；(3) 在服务不同任务数量时，MQLserve 的吞吐量，平均延迟和平均任务完成时间十分接近，证明了 MQLserve 具有极强的鲁棒性和可扩展性。

4.3.2 用户体验度：响应时间以及 SLO 满足率

良好的用户体验度也是衡量系统优劣的重要指标，本文从响应时间和 SLO 满足率两方面对 MQLserve 和 S-LoRA 进行对比。

(1) 响应时间：如图 9 所示，在请求到达速率为 5 reqs/s 的情况下，在 RTX 4090 上测试了 Llama2-7b 的响应时间。在绝大多数情况下，MQLserve 的响应时间远远小于 S-LoRA，例如，MQLserve 对 90% 的请求的响应时间都在 4.74 s 以下，能够给予用户良好的体验。总的来说，MQLserve 的平均响应时间相比 S-LoRA 缩短了 84.2%（2.42 s vs. 15.27 s），对 90% 的请求的响应时间比 S-LoRA 缩短了 87.2%（4.74 s vs. 37.15 s），对 50% 的请求的响应时间比 S-LoRA 缩短了 89.7%（1.17 s vs. 11.37 s）。

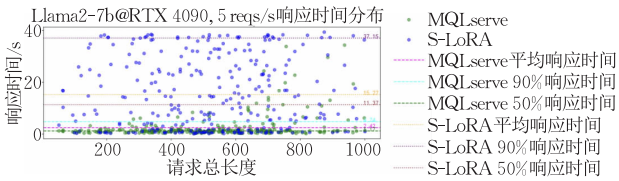

图 9 MQLserve 和 S-LoRA 响应时间分布（Llama2-7b@RTX 4090，任务数量为 100，请求到达速率为 5 reqs/s。每个数据点代表一个请求，虚线表示响应时间的统计数据）

(2) SLO 满足率：本文用不同的请求到达速率 (1,5,10) 和最大请求长度 (512,1024,2048) 组合，来测试在不同负载下的 SLO 满足率，结果如图 10 所示。总体来看，MQLserve 的平均 SLO 满足率相

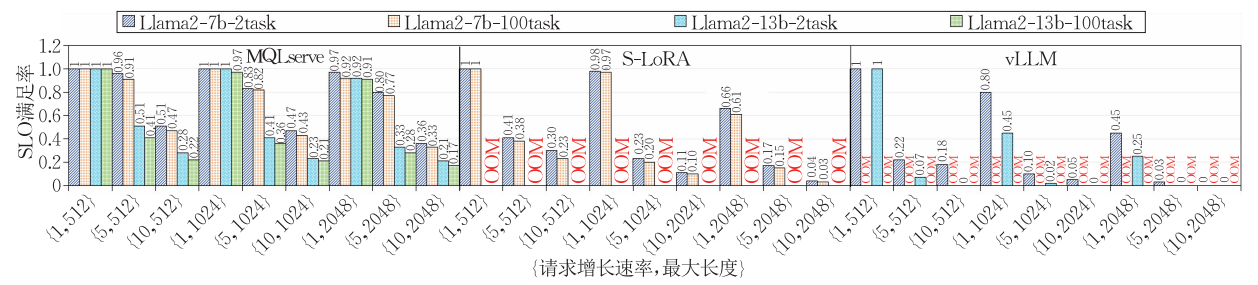

图 10 MQLserve、S-LoRA 和 vLLM 在不同请求到达速率、不同最大长度条件下的 SLO 满足率对比（RTX 4090）（图中 OOM 表示显存不足、无法完成推理服务）

比 S-LoRA 提高了 330%。具体地，在请求到达速率为 1 reqs/s 时，所有系统的 SLO 满足率都很高。但随着负载加大，MQLserve 的 SLO 满足率下降最小，在请求到达速率为 5 reqs/s 和 10 reqs/s 时，MQLserve 的 SLO 满足率最高比 S-LoRA 高 1000%(0.33 vs. 0.3)，最高比 vLLM 高 2566.7%(0.8 vs. 0.03)；同时，在保持速率不变，增大请求长度时，MQLserve 的 SLO 满足率几乎不会下降，请求长度翻倍时，SLO 满足率最多下降 23.4%(0.47 vs. 0.36)，而 S-LoRA 和 vLLM 却随着请求长度增大而急剧降低，其中 S-LoRA 在长度翻倍后 SLO 满足率最高下降 70%(0.1 vs. 0.03)，而 vLLM 甚至会出现 SLO 满足率变为 0 的情况。这些实验结果表明 MQLserve 有具备更好的适应力和可扩展性。

4.4　扩展性测试

4.4.1　任务数扩展

本文用 Llama2-7b 在 RTX4090 上测试了在不同任务数量和请求速率下，MQLserve、S-LoRA 和 vLLM 三个服务系统的吞吐率，结果见表 3，其中 OOM 表示显存不足、无法完成推理服务。

表 3　任务数扩展性测试(Llama2-7b@RTX 4090)(在不同任务数、不同请求到达率条件下，评估系统每条处理的服务请求数量。其中 OOM 表示显存不足、无法完成推理服务)

任务数	2			3			4			5			…	100		
请求到达速率	5	10	20	5	10	20	5	10	20	5	10	20	…	5	10	20
MQLserve	3.89	4.70	4.86	3.78	4.66	4.81	3.82	4.77	4.89	3.71	4.61	4.73	…	3.60	4.25	4.58
S-LoRA	2.93	3.45	3.51	2.97	3.38	3.54	2.91	3.36	3.58	2.97	3.40	3.55	…	2.87	3.35	3.36
vLLM	1.77	2.46	2.98	1.02	1.68	2.27	0.77	0.76	0.80	OOM	OOM	OOM	…	OOM	OOM	OOM

结果表明，MQLserve 吞吐随着请求速率升高而增加，且在任务数量达到 100 时，仍然保持着较高的吞吐(4.58 reqs/s)，吞吐率几乎不会随任务数量增加而下降。而 vLLM 在多任务服务几乎没有扩展性，在任务数量达到 5 时便出现内存不足的错误。且随着任务种类的增加，vLLM 的吞吐率出现明显下降，LoRA 任务从 1 增加到 4 时，其吞吐率下降了 68%(2.40 reqs/s vs. 0.78 reqs/s)。S-LoRA 虽然也支持大规模任务数量，但是 MQLserve 在所有测试场景中所达到的吞吐均高于 S-LoRA，取得了更好的性能表现。

4.4.2　模型扩展

为了验证本工作在更大规模的模型上的可用性，本文分别从模型精度和性能两方面，测试了 MQLserve 在 Llama2-70b 模型上的表现。由于该模型规模较大，本实验使用 NVIDIA A800 显卡进行测试，该显卡的显存容量为 80 GB，计算力为 312 TFLOPS，访存带宽为 2039 GB/s。

(1) 模型精度：为了测试 Llama2-70b 模型上的不同量化方法对模型精度的影响，本文选取了 3 个不同的数据集(alpaca、dolphin 和 GSM8k)，包含 2 个问答任务，一个数学推理任务，均选用开源的微调的 LoRA adapter 来进行测试，具体的数据集和 LoRA adapter 描述见附录 B。测试结果如图 11 所示。在 4-bit 量化时，MLGPTQ、DPGPTQ、DPAWQ 和 FP4 在 3 个数据集上的平均模型精度下降率为 3.96%、8.61%、9.25% 和 13.74%，可以看出 MLGPTQ 的模型精度下降显著小于其余方法；在 3-bit 量化时，平均模型精度下降率分别为 27.59%、31.67%、29.61% 和 32.09%，MLGPTQ 仍然是表现最好的。这些实验结果证明了 MLGPTQ 在更大规模的模型上依然具有优异的模型压缩能力。

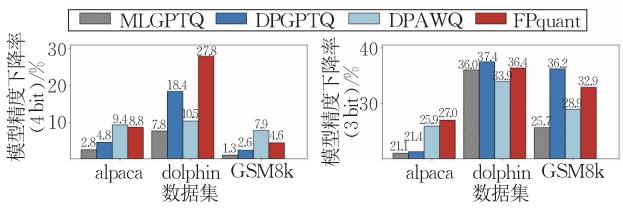

图 11　不同量化方法在每个数据集上的模型精度下降率(Llama2-70b)

(2) 端到端性能：本实验采用与 4.3.1 节同样的负载来测试不同系统在 A800 上服务 Llama2-70b 模型时的系统性能，由于 S-LoRA 不支持部署量化、无法在 80 GB 的显存条件下服务该大规模模型，故本实验仅对比 MQLserve 和 vLLM 的性能。实验结果如图 12 所示。在请求速率为 5 reqs/s 之前系统服务吞吐可稳定上升，后续则保持稳定。在服务多任务时，MQLserve 服务 5 个任务吞吐最高可达 1.57 reqs/s，服务 100 个任务吞吐最高仍可达 1.43 reqs/s；vLLM 在服务至 2 个任务时，吞吐已下降至最高只有 0.38 reqs/s、不及 MQLserve，进一步服务更多任务时则遇到显存不足、无法完成推理服务的问题。总的来说，在大规模的模型上，MQLserve 在吞吐量、平均延迟和平均任务完成时间这三个指标仍然优于现有服务系统。

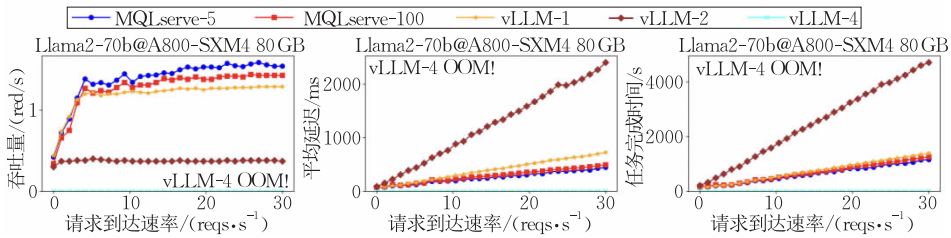

图 12 MQLserve 和 vLLM 在不同请求到达速率情况下的吞吐量、平均延迟和任务完成时间(Llama2-70b@A800)
(图中 OOM 表示显存不足、无法完成推理服务)

4.5 调度算法测试

本文在MQLserve上使用不同的调度策略,分别测试Llama2-7b和Llama2-13b在RTX 4090上的吞吐和SLO满足率。本实验考虑4种对比对象:(1)FIFO调度策略;(2)Fastserve调度策略;(3)MQLserve(w/o SRTF),即MQLserve调度策略在去除基于预测的SRTF后的变种;(4)MQLserve(w/o cluster),即MQLserve调度策略在去除任务聚类模块后的变种。值得说明的是,FIFO是S-LoRA所采用的调度策略,该对比对象反映了当S-LoRA支持多任务联合量化后的系统性能,通过与其相比,可以很好地反映本工作所提出的调度方案的有效性。

结果如表4所示,可以看出,MQLserve调度策略均优于FIFO和Fastserve,接下来将从吞吐和SLO满足率两方面进行分析。

表 4 调度算法对 MQLserve 吞吐和 SLO 满足率影响
(任务数量为 100,请求到达速率为 20 reqs/s)

调度策略	吞吐		SLO 满足率	
	7b	13b	7b	13b
FIFO	4.26	1.69	0.04	0.02
Fastserve	3.56	1.60	0.12	0.11
MQLserve (w/o SRTF)	4.49	1.78	0.05	0.02
MQLserve (w/o cluster)	4.33	1.72	0.26	0.15
MQLserve	4.58	1.80	0.25	0.15

在吞吐方面,MQLserve在Llama2-7b上吞吐为4.58 reqs/s,在Llama2-13b上为1.80 reqs/s。FIFO策略在两个模型上的吞吐分别为4.26 reqs/s和1.69 reqs/s,MQLserve的多任务调度相对于FIFO总共带来了6.51%～7.51%的吞吐提升率。而仅仅通过多级队列按照优先级进行排序的Fastserve调度,则由于没有考虑频繁的任务切换和控制同一批次中任务的数量,导致LoRA adapter过多且切换过于频繁,降低了有效显存,增多了无效IO,因此,MQLserve相比Fastserve带来了12.5%～28.65%的吞吐提升率。

在SLO满足率方面,MQLserve在Llama2-7b和Llama2-13b上的SLO满足率分别为0.25和0.15。由于FIFO调度策略最为朴素,故表现最差,在两个模型上仅达到0.04和0.02的SLO满足率;而Fastserve的调度策略允许即将超时的请求抢占调度,因此相比FIFO有更好的SLO满足率,但仍不及MQLserve。总的来说,MQLserve相比FIFO可达到525%～650%的SLO满足率提升,相比Fastserve可达到36.4%～108.33%的SLO满足率提升。

此外,可以观察到聚类和SRTF模块对吞吐和SLO满足率有着不同的作用。一方面,MQLserve(w/o cluster)在这两个模型上的吞吐分别为4.33 reqs/s和1.72 reqs/s,表明聚类单独为吞吐带来了4.65%～5.64%的提升率;然而,MQLserve(w/o cluster)与MQLserve相比,可达到持平甚至略高的SLO满足率,这是因为聚类操作增加了任务的排队时间,在推理速度较快的场景会对端到端延迟有较大影响。另一方面,MQLserve(w/o SRTF)分别为4.49 reqs/s和1.78 reqs/s,说明单独使用基于预测的SRTF平均只带来了1.12%～2%的吞吐提升率,影响不大;然而,MQLserve(w/o SRTF)的SLO满足率分别仅为0.05%和0.02%,说明SRTF策略为SLO满足率带来了400%～650%的提升,显示出SRTF在提高SLO满足率方面的重要作用。

4.6 动态任务添加稳定性测试

为了测试MQLserve在动态任务添加时的稳定性,本文在一段连续的在线服务(Llama2-7b@RTX 4090、请求到达率为20 reqs/s、初始任务数量为10)过程中,进行了5次新任务的添加,每次添加均执行3.2节中的步骤。在任务添加期间,线上服务的吞吐变化如图13所示。

在第一次任务添加期间(添加1个任务),系统平均吞吐下降了13%;在第二次任务添加期间(添加3个任务),平均吞吐下降了14%;而后三次分别

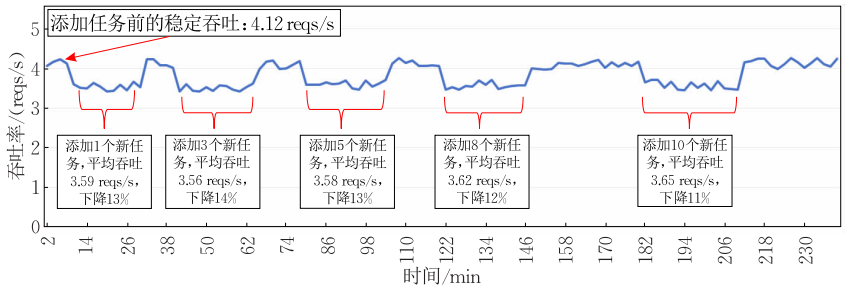

图 13　动态任务添加对在线 MQLserve 吞吐的影响(Llama2-7b@RTX 4090、请求到达率为 20 reqs/s、初始任务数量为 10)

添加 5、8、10 个任务时，平均吞吐分别下降了 13%、12%和 11%。可以观察到，尽管添加的任务数量不同，平均吞吐下降的幅度相对稳定，约在 11%到 14%之间，并未随着任务数量增加而显著扩大。这种现象表明，MQLserve 在处理任务动态添加时，吞吐的下降幅度基本不变，与添加的任务数量无关(实际上，如 3.2 节所介绍的，MQLserve 的动态任务添加过程是逐层、异步进行的，所以添加更多任务只会稍微延长任务添加的时间，而不会对线上服务性能造成更严重的影响)。这种一致性展示了 MQLserve 在应对不同任务量变化时的鲁棒性和高效性。

系统吞吐下降幅度的波动(11%～14%)与系统当时的负载状态有关，例如在负载压力大或资源较紧张时，吞吐下降可能更为明显。但总的来说，无论动态添加多少任务，MQLserve 都可以保持较稳定的系统吞吐，不会对在线服务性能有很大的影响。此外，在每次任务添加完成后，系统吞吐均恢复到正常水平，确保系统在任务添加后的性能稳定，展示了其在线服务的可靠性和可扩展性。

5　总　结

本文针对现有 LLM 服务系统在多任务场景下的不足，创新地提出了一个多 LoRA 任务服务系统 MQLserve。一方面，本系统设计了一种灵活高效的动态多任务量化算法，支持多任务的模型联合量化，显著降低了模型部署的显存需求；同时支持实时的动态任务添加，提升了线上服务的稳定性和灵活性。另一方面，本系统针对多任务场景，提出了一种新型的基于输出长度预测和聚类的调度算法，有效地解决了传统调度算法在多任务场景下存在的高昂显存开销和频繁显存切换等问题，提升了系统性能。

参　考　文　献

[1] ChatGPT：Optimizing Language Models for Dialogue. 2022. https://openai.com/blog/chatgpt

[2] OpenAI. GPT-4 technical report. CoRR abs/2303.08774, 2023

[3] Introducing meta Llama 3：The most capable openly available LLM to date. Meta，2024. https：//ai.meta.com/blog/meta-llama-3/

[4] Zhang J，Zhao Y，Saleh M，et al. PEGASUS：Pre-training with extracted gap-sentences for abstractive summarization//Proceedings of the 37th International Conference on Machine Learning. Online，2020：11328-11339

[5] Liu Y，Lapata M. Text summarization with pretrained encoders//Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing. Hong Kong，China，2019：3728-3738

[6] Edunov S，Ott M，Auli M，et al. Understanding back-translation at scale//Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. Brussels，Belgium，2018：489-500

[7] Vaswani A，Shazeer N，Parmar N，et al. Attention is all you need//Advances in Neural Information Processing Systems 30：Annual Conference on Neural Information Processing Systems 2017. Long Beach，USA，2017：5998-6008

[8] Zhang Y，Sun S，Galley M，et al. DialoGPT：Large-scale generative pre-training for conversational response generation //Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics：System Demonstrations. Online，2020：270-278

[9] Roller S，Dinan E，Goyal N，et al. Recipes for building an open-domain chatbot//Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics. Online，2021：300-325

[10] Hu E J，Shen Y，Wallis P，et al. LoRA：Low-rank adaptation of large language models//Proceedings of the 10th International Conference on Learning Representations. Online，2022

[11] Ding N, Qin Y, Yang G, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. Nature Machine Intelligence, 2023, 5(3): 220-235

[12] Houlsby N, Giurgiu A, Jastrzebski S, et al. Parameter-efficient transfer learning for NLP//Proceedings of the 36th International Conference on Machine Learning. Long Beach, USA, 2019: 2790-2799

[13] Liu H, Tam D, Muqeeth M, et al. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning//Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022. New Orleans, USA, 2022: 1950-1965

[14] Frantar E, Ashkboos S, Hoefler T, et al. GPTQ: Accurate post-training quantization for generative pre-trained transformers. CoRR abs/2210.17323, 2022

[15] Lin J, Tang J, Tang H, et al. AWQ: Activation-aware weight quantization for LLM compression and acceleration//Proceedings of the 7th Annual Conference on Machine Learning and Systems. Santa Clara, USA, 2024: 87-100

[16] Liu S Y, Liu Z, Huang X, et al. LLM-FP4: 4-bit floating-point quantized transformers//Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Singapore, 2023: 592-605

[17] Dettmers T, Lewis M, Belkada Y, et al. GPT3.int8(): 8-bit matrix multiplication for transformers at scale//Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022. New Orleans, USA, 2022: 30318-30332

[18] Yao Z, Yazdani A R, Zhang M, et al. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers//Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022. New Orleans, USA, 2022: 27168-27183

[19] Kwon W, Li Z, Zhuang S, et al. Efficient memory management for large language model serving with PagedAttention//Proceedings of the 29th Symposium on Operating Systems Principles. Koblenz, Germany, 2023: 611-626

[20] Sheng Y, Cao S, Li D, et al. S-LoRA: Serving thousands of concurrent LoRA adapters. CoRR abs/2311.03285, 2023

[21] Chen L, Ye Z, Wu Y, et al. Punica: Multi-tenant LoRA serving. CoRR abs/2310.18547, 2023

[22] Han S, Mao H, Dally W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. CoRR abs/1510.00149, 2015

[23] Gong Y, Liu L, Yang M, et al. Compressing deep convolutional networks using vector quantization. CoRR abs/1412.6115, 2014

[24] Zhou S, Wang J, Wang J, et al. Point to set similarity based deep feature learning for person re-identification//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Hawaii, USA, 2017: 5028-5037

[25] Park E, Ahn J, Yoo S. Weighted-entropy-based quantization for deep neural networks//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Hawaii, USA, 2017: 7197-7205

[26] Hubara I, Courbariaux M, Soudry D, et al. Quantized neural networks: Training neural networks with low precision weights and activations. Journal of Machine Learning Research, 2018, 18(1): 1-30

[27] Jacob B, Kligys S, Chen B, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Salt Lake City, USA, 2018: 2704-2713

[28] Alistarh D, Grubic D, Li J, et al. QSGD: Communication-efficient SGD via gradient quantization and encoding//Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017. Long Beach, USA, 2017: 1709-1720

[29] Zhang H, Li J, Kara K, et al. ZipML: Training linear models with end-to-end low precision, and a little bit of deep learning//Proceedings of the 34th International Conference on Machine Learning. Sydney, Australia, 2017: 4035-4043

[30] Fu F, Hu Y, He Y, et al. Don't waste your bits! Squeeze activations and gradients for deep neural networks via TinyScript//Proceedings of the 37th International Conference on Machine Learning. Online, 2020: 3304-3314

[31] Faghri F, Tabrizian I, Markov I, et al. Adaptive gradient quantization for data-parallel SGD//Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020. Online, 2020: 3174-3185

[32] Micikevicius P, Narang S, Alben J, et al. Mixed precision training. CoRR abs/1710.03740, 2017

[33] Peng H, Wu K, Wei Y, et al. FP8-LM: Training FP8 large language models. CoRR abs/2310.18313, 2023

[34] Sun X, Choi J, Chen C Y, et al. Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks//Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019. Vancouver, Canada, 2019: 4901-4910

[35] Fishman M, Chmiel B, Banner R, et al. Scaling FP8 training to trillion-token LLMs. CoRR abs/2409.12517, 2024

[36] Shen H, Mellempudi N, He X, et al. Efficient post-training quantization with FP8 formats//Proceedings of the Seventh Annual Conference on Machine Learning and Systems. Santa Clara, USA, 2024: 483-498

[37] Hubara I, Nahshan Y, Hanani Y, et al. Accurate post training quantization with small calibration sets//Proceedings of the 38th International Conference on Machine Learning. Online, 2021: 4466-4475

[38] Dettmers T, Svirschevski R, Egiazarian V, et al. SpQR: A sparse-quantized representation for near-lossless LLM weight

compression//Proceedings of the 12th International Conference on Learning Representations. Vienna, Austria, 2024

[39] Xiao G, Lin J, Seznec M, et al. SmoothQuant: Accurate and efficient post-training quantization for large language models// Proceedings of the International Conference on Machine Learning. Hawaii, USA, 2023: 38087-38099

[40] Yu G I, Jeong J S, Kim G W, et al. Orca: A distributed serving system for transformer-based generative models// Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA, 2022: 521-538

[41] Wu B, Zhong Y, Zhang Z, et al. Fast distributed inference serving for large language models. CoRR abs/2305.05920, 2023

[42] Zheng Z, Ren X, Xue F, et al. Response length perception and sequence scheduling: An LLM-empowered LLM inference pipeline//Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023. New Orleans, USA, 2023: 65517-65530

[43] Jin Y, Wu C F, Brooks D, et al. S^3: Increasing GPU utilization during generative inference for higher throughput// Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023. New Orleans, USA, 2023: 18015-18027

[44] Tiedemann J, Choukri K, Declerck, T, et al. Parallel data, tools and interfaces in opus//Proceedings of the 8th International Conference on Language Resources and Evaluation. Istanbul, Turkey, 2012: 2214-2218

[45] Hasan T, Bhattacharjee A, Islam M S, et al. Xlsum: A large-scale multilingual abstractive summarization dataset for 44 languages. CoRR abs/2106.13822, 2021

[46] Zhao Y, Qi Z, Nan L, et al. QTSumm: A new benchmark for query-focused table summarization. CoRR abs/2305.14303, 2023

[47] Gunasekar S, Zhang Y, Aneja J, et al. Textbooks are all you need. CoRR abs/2306.11644, 2023

[48] Keenan W. Materializing Religion: Expression, Performance and Ritual. Routledge. London, UK, 2006

[49] Lin C Y. Rouge: A package for automatic evaluation of summaries//Proceedings of the Workshop on Text Summarization Branches Out. Barcelona, Spain, 2004: 74-81

附录 A.　MLGPTQ 算法推导证明

MLGPTQ 基于 GPTQ，据输入数据的激活值来适应对应的 LoRA adapter 以及基座大模型权重矩阵，使得量化后的激活值损失最小。

对于 n 个任务，要使得量化后的激活值损失最小，有公式

$$\arg\min_{\hat{\boldsymbol{W}}} \left\| (\boldsymbol{W}+\boldsymbol{B}_i\boldsymbol{A}_i)\boldsymbol{X}-(\hat{\boldsymbol{W}}+\boldsymbol{B}_i\boldsymbol{A}_i)\boldsymbol{X} \right\|_2^2,\quad i\in\{0,1,\cdots,n-1\},$$

其中，$\boldsymbol{W}$ 表示基座大模型某一线性层的权重，$\hat{\boldsymbol{W}}$ 表示量化后的权重，$\boldsymbol{B}_i$ 和 $\boldsymbol{A}_i$ 表示第 i 个 LoRA adapter 的低秩适应矩阵，n 表示系统中服务的任务种类的数量，每种任务都有一个 LoRA adapter。MLGPTQ 的目的是通过逐层量化，为每一个线性层的权重 $\boldsymbol{W}$ 找到一个 $\hat{\boldsymbol{W}}$，使得其激活值变化最小。

对于一个已经完成预训练和微调的模型，服务于某个任务 i 时，其 $\boldsymbol{W}+\boldsymbol{B}_i\boldsymbol{A}_i$ 不会再改变。记此 $\boldsymbol{W}+\boldsymbol{B}_i\boldsymbol{A}_i$ 为 Wl_i，设模型的目标函数为 E，由于其已预训练完成收敛，会达到一个局部极小值，其泰勒展开式为

$$\nabla E=\left(\frac{\partial E}{\partial Wl_i}\right)^{\mathrm{T}}\nabla Wl_i+\frac{1}{2}\nabla \boldsymbol{Wl}_i^{\mathrm{T}}\cdot\boldsymbol{H}\cdot\nabla Wl_i+O(\left\|\nabla Wl_i\right\|^3),$$

其中，$\boldsymbol{H}=\frac{\partial^2 E}{\partial W^2}$ 为海森矩阵。由于其已经收敛，其一阶偏导数可近似为 0，且忽略高阶项 $O(\left\|\nabla Wl_i\right\|^3)$，得到误差为

$$\nabla E\approx\frac{1}{2}\nabla \boldsymbol{Wl}_i^{\mathrm{T}}\cdot\boldsymbol{H}\cdot\nabla Wl_i。$$

量化过程将 Wl_i 量化为 $Q(Wl_i)$，于是有 $\nabla Wl_{iq}=Q(Wl_{iq})-Wl_{iq}$，其中 Wl_{iq} 表示第 i 个任务的权重矩阵的第 q 个元素。于是得到优化的目标函数：

$$\arg\min_{q,\nabla Wl_i}\frac{1}{2}\nabla \boldsymbol{Wl}_i^{\mathrm{T}}\cdot\boldsymbol{H}\cdot\nabla Wl_i,$$
$$\text{s.t. } \boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}=Q(Wl_{iq}),$$

上述公式中，$\boldsymbol{e}_q^{\mathrm{T}}$ 代表一个单位向量，其位置 q 处为 1，其余位置为 0。这是一个有约束的凸优化问题，利用拉格朗日乘数法，我们引入拉格朗日函数：

$$L=\frac{1}{2}\nabla \boldsymbol{Wl}_i^{\mathrm{T}}\cdot\boldsymbol{H}\cdot\nabla Wl_i+\lambda(\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}-Q(Wl_{iq}))。$$

接下来对 ∇Wl_i 和 λ 求偏导，因为此时是在求稳定状态下的解，所以令偏导数为 0。

首先，对 ∇Wl_i 求偏导数：

$$\frac{\partial L}{\partial(\nabla Wl_i)}=\boldsymbol{H}\nabla Wl_i+\lambda\boldsymbol{e}_q=0,$$

其中，$\frac{1}{2}\nabla \boldsymbol{Wl}_i^{\mathrm{T}}\boldsymbol{H}\nabla Wl_i$ 的偏导是 $\boldsymbol{H}\nabla Wl_i$。$\lambda(\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}-Q(Wl_{iq}))$ 中 $\lambda\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i$ 的偏导是 $\lambda\boldsymbol{e}_q$。

令偏导为零，得到

$$\boldsymbol{H}\nabla Wl_i+\lambda\boldsymbol{e}_q=0。$$

其次，对 λ 求偏导数：

$$\frac{\partial L}{\partial\lambda}=\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}-Q(Wl_{iq})=0,$$

其中，$\lambda(\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}-Q(Wl_{iq}))$ 的偏导是 $\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}-Q(Wl_{iq})$。

令偏导为零，得到

$$\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}-Q(Wl_{iq})=0。$$

接下来求解 λ 和 ∇Wl_i，从 $\boldsymbol{H}\nabla Wl_i+\lambda\boldsymbol{e}_q=0$，可以解出 ∇Wl_i：

$$\nabla Wl_i=-\lambda \boldsymbol{H}^{-1}\boldsymbol{e}_q$$

将 ∇Wl_i 代入 $\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}-Q(Wl_{iq})=0$：

$$\boldsymbol{e}_q^{\mathrm{T}}(-\lambda \boldsymbol{H}^{-1}\boldsymbol{e}_q)+Wl_{iq}-Q(Wl_{iq})=0,$$

$$-\lambda \boldsymbol{e}_q^{\mathrm{T}}\boldsymbol{H}^{-1}\boldsymbol{e}_q+Wl_{iq}-Q(Wl_{iq})=0。$$

因为 $\boldsymbol{e}_q$ 是单位向量，$\boldsymbol{e}_q^{\mathrm{T}}\boldsymbol{H}^{-1}\boldsymbol{e}_q$ 是 $\boldsymbol{H}^{-1}$ 在位置 (q,q) 的元素，即 $(\boldsymbol{H}^{-1})_{qq}$，所以

$$\lambda=\frac{Wl_{iq}-Q(Wl_{iq})}{(\boldsymbol{H}^{-1})_{qq}}。$$

将 λ 代入 $\nabla Wl_i=-\lambda \boldsymbol{H}^{-1}\boldsymbol{e}_q$：

$$\nabla Wl_i=-\frac{Wl_{iq}-Q(Wl_{iq})}{(\boldsymbol{H}^{-1})_{qq}}\boldsymbol{H}^{-1}\boldsymbol{e}_q。$$

利用上述结果计算 ∇E：

$$\nabla E=\frac{\partial}{\partial Wl_{iq}}L$$

$$=\frac{\partial}{\partial Wl_{iq}}\left(\frac{1}{2}\nabla \boldsymbol{Wl}_i^{\mathrm{T}}\boldsymbol{H}\nabla Wl_i+\lambda(\boldsymbol{e}_q^{\mathrm{T}}\nabla Wl_i+Wl_{iq}-Q(Wl_{iq}))\right)。$$

由于 $\nabla Wl_i=-\frac{Wl_{iq}-Q(Wl_{iq})}{(\boldsymbol{H}^{-1})_{qq}}\boldsymbol{H}^{-1}\boldsymbol{e}_q$ 和 $\lambda=\frac{Wl_{iq}-Q(Wl_{iq})}{(\boldsymbol{H}^{-1})_{qq}}$，得到

$$\nabla E=\frac{Wl_{iq}-Q(Wl_{iq})}{2(\boldsymbol{H}^{-1})_{qq}},$$

最终结果为

$$\lambda=\frac{Wl_{iq}-Q(Wl_{iq})}{(\boldsymbol{H}^{-1})_{qq}},$$

$$\nabla Wl_i=-\frac{Wl_{iq}-Q(Wl_{iq})}{(\boldsymbol{H}^{-1})_{qq}}\boldsymbol{H}^{-1}\boldsymbol{e}_q,$$

$$\nabla E=\frac{Wl_{iq}-Q(Wl_{iq})}{2(\boldsymbol{H}^{-1})_{qq}}。$$

在实现中，每次获得一个批次的任务的少量数据，然后加载这个任务对应的 LoRA adapter，而后计算其海森矩阵并更新 Wl_i。由于 LoRA adapter 权重 $\boldsymbol{B}_i$ 和 $\boldsymbol{A}_i$ 的参数量相比基座大模型权重 $\boldsymbol{W}$ 来说可以忽略不计，所以仅仅量化基座大模型权重 $\boldsymbol{W}$，即使用下式：

$$\nabla \boldsymbol{W}=-\frac{Wl_{iq}-(Q(\boldsymbol{W})+(\boldsymbol{B}_i\boldsymbol{A}_i)_q)}{(\boldsymbol{H}^{-1})_{qq}}\boldsymbol{H}^{-1}\boldsymbol{e}_q,$$

来更新基座大模型参数权重 $\boldsymbol{W}$。同时，由于每一次更新都会改变 $\boldsymbol{W}$，导致 ∇E 也会改变，所以也要迭代更新 E，从而更好地进行下一批次的更新。

附录 B. 实验数据集与 LoRA 微调模型说明

数据集。本文主要选取了 9 个主要数据集进行测试，涵盖了机器翻译、文本总结、图表总结和代码生成等任务；对于 70b 模型扩展测试，本文另外选取了两个问答任务和一个数学推理任务，总结见表 5。

表 5 数据集汇总

数据集名称	简称	平均输入长度	平均输出长度	任务类型
OPUS-法语-英语	trans-fr	121	105	机器翻译
OPUS-捷克语-英语	trans-cs	47	47	机器翻译
OPUS-印度尼西亚语-英语	trans-id	47	38	机器翻译
OPUS-越南语-英语	trans-nl	72	65	机器翻译
OPUS-丹麦语-英语	trans-da	72	71	机器翻译
OPUS-瑞典语-英语	trans-sw	64	65	机器翻译
XLSum	Xlsum	2595	125	文本总结
QTSUMM	QTsum	1350	339	表格总结
tiny-codes	tiny-codes	328	1890	代码生成
alpaca-cleaned	alpaca	87	679	问答
dolphin	dolphin	1096	626	问答
GSM8k	GSM8k	240	293	数学推理

代码生成任务本文选取了经典双语翻译数据集 OPUS，选取了其中的法语-英语（French-to-English）、捷克语-英语（Czech-to-English）、印度尼西亚语-英语（Indonesian-to-English）、越南语-英语（Vietnamese-to-English）、丹麦语-英语（Danish-to-English）以及瑞典语-英语（Swedish-to-English）共 6 种翻译任务，充分考虑了语种的多样性；文本翻译任务，选取了 XLSum 数据集，一个全面而多样化的数据集，其中包括了从 BBC 中提取，并经过专业人士标注的 135 万个专业文本-总结对。对于表格总结任务，选取了 QTSUMM 数据集，QTSUMM 数据集是一个大规模数据集，用于在表格数据上以查询为重点的摘要的任务，它包含 7111 张人类标注的查询-总结对，包含涵盖各种主题的 2934 张表格，代码生成任务选取了 tiny-codes 数据集，该合成数据集是 1600 万短而清晰的代码片段的集合，可以帮助 LLM 模型学习如何使用天然和编程语言来推理。该数据集涵盖了各种各样的编程语言，例如 Python、TypeScript、JavaScript、Ruby、Julia、Rust、C++、Bash、Java、C# 和 Go。问答任务选取了 alpaca-cleaned 和 dolphin 数据集，alpaca-cleaned 数据集是基于原始 alpaca 数据集修订而成，原始数据集包含 52000 条由 OpenAI 的 text-davinci-003 生成的指令和示例，旨在通过指令调优提升语言模型的指令跟随能力，修订后的版本修复了幻觉内容、空输出、不一致输入格式、错误答案及不清晰指令等问题，使数据质量更高；dolphin 数据集是一个开源、未经过滤且可商用的数据集，包含约 100 万条通过 GPT-4 扩展

的 FLANv2 指令和约 350 万条通过 GPT-3.5 扩展的 FLANv2 指令，旨在支持多种基础模型的训练，并以 Orca 论文为参考设计，去除了重复数据和偏向性内容。数学推理采用了 GSM8k 数据集，它是一个包含 8500 个高质量、语言多样的小学数学应用题的数据集，这些题目由人类出题者创建。这些问题的解决步骤在 2 到 8 步之间，解决方案主要涉及使用基本的算术运算(+−×÷)进行一系列初等计算，以得出最终答案。

LoRA 微调模型。所有任务的 LoRA adapter 均为 huggingface 开源模型，或利用对用数据集的训练集进行微调，并在测试集上进行测试。更具体的总结见表 6。

表 6　LoRA 微调模型汇总

任务	秩	微调层	来源
trans-fr	16	mlp	https://huggingface.co/kaitchup/Llama−2−7b-mt-French-to-English
trans-cs	16	mlp	https://huggingface.co/kaitchup/Llama-2-7b-mt-Czech-to-English
trans-id	16	mlp	https://huggingface.co/kaitchup/Llama-2-7b-mt-Indonesian-to-English
trans-nl	16	mlp	https://huggingface.co/kaitchup/Llama-2-7b-mt-Vietnamese-to-English
trans-da	16	mlp	https://huggingface.co/kaitchup/Llama-2-7b-mt-Danish-to-English
trans-sw	16	mlp	https://huggingface.co/kaitchup/Llama-2-7b-mt-Swedish-to-English
Xlsum	16	self-attn	https://huggingface.co/Alexyfxia/xlsum_llama2
QTsum	16	self-attn	https://huggingface.co/Alexyfxia/QTsum_llama2
tiny-codes	16	self-attn	https://huggingface.co/Alexyfxia/tiny_codes_llama2
alpaca	64	self-attn	https://huggingface.co/iamshnoo/alpaca-2-70b-english
dolphin	64	self-attn	https://huggingface.co/dfurman/Llama-2-70B-Instruct-v0.1
GSM8k	64	self-attn	https://huggingface.co/haritzpuerto/LLaMA2-70B-dcot

附录 C.　实验评价指标说明

本文使用了多种评价指标以充分验证 MLGPTQ 量化算法的准确性和有效性，具体的评价指标如下：

(1) sacrebleu，在图中表示为 BLEU，是一种经典的机器翻译评测标准，它通过比较机器翻译的输出和一个或多个参考翻译之间的 n-gram 重叠来评分，同时考虑了短句惩罚以防止偏好过短的翻译输出；

(2) rouge1，在图中表示为 ROUGE1，计算的是机器生成文本和参考文本之间单词的重叠比例，即一元组(单个单词)的匹配度；

(3) rouge2，表示为 ROUGE2，类似于 rouge1，表示的是二元组匹配度；

(4) nist_mt，表示为 NIST_MT，也是基于 bleu 的改进，给予不常见词更高的权重，鼓励了翻译的多样性和准确性；

(5) meteor，记作 METEOR，评分计算基于查准率和查全率的调和平均，还引入了一个惩罚因子，用以惩罚过多不连续的匹配；

(6) google_bleu，记作 G_BLEU，是对 bleu 的改进，基于平滑方法、n-gram 权重和短句惩罚等进行了调整以优化其性能。

附录 D.　MQLserve 调度算法伪代码

MQLserve 的调度算法是基于聚类和 SRTF 策略的，相比于其他调度算法，它很好地结合了多任务的场景进行改进，达到了极好的效果，其伪代码如算法 2 所示，辅助函数 *generate_new_batch* 和 *schedule_new_batch* 见算法 3。具体算法如下：维护四个队列，分别是 *prefill_reqs*、*decoding_reqs*、*hungry_prefill_reqs* 以及 *hungry_decoding_reqs*，分别维护了还未进行 prefill 阶段的请求(即新来的还未进行服务的请求)，正在 decode 阶段的请求，处于饥饿状态的未进行服务的请求以及处于饥饿状态的在 decode 阶段的请求。

算法 2.　MQLserve 多任务调度算法

输入：Four queues: *prefill_reqs*, *decoding_reqs*, *hungry_prefill_reqs*, *hungry_decoding_reqs*

1. *running_batch* ← *None*　　▷初始化队列及元数据
2. *max_cont_decode* ← Threshold value for decoding
3. *max_cont_decode_one_batch* ← Threshold value for decoding one batch
4. *decode_count* ← 0
5. *decode_count_one_batch* ← 0
6. while not terminated do
7.　if *running_batch* is empty then　　▷系统冷启动，调度第一个批次
8.　　*new_batch* ← *generate_new_batch*(*prefill_reqs*)
9.　　if *new_batch* ≠ ∅ then
10.　　Perform *prefill*(*new_batch*)
11.　　*decoding_reqs* ← *decoding_reqs* + *new_batch*
12.　　*decode_count* ← 0
13.　　*running_batch* ← *new_batch*
14.　　else
15.　　Keep idle
16.　else　　▷进行 *iteration-level* 的调度
17.　　if *decode_count* ≥ *max_cont_decode* then　　▷选择新的批次进行调度

```
18.      new_batch ← generate_new_batch(prefill_reqs, hungry_prefill_reqs)
19.      Perform prefill(new_batch)
20.      decoding_reqs ← decoding_reqs + new_batch
21.      decode_count ← 0
22.      running_batch ← new_batch
23.    else                                  ▷调度之前已经产生的批次
24.      if decode_count_one_batch ≥ max_cont_decode_one_batch then
25.        new_batch ← schedule_new_batch(running_batch, decoding_reqs, hungry_decoding_reqs)
26.        Perform decode(new_batch)
27.        decode_count_one_batch ← 0
28.        running_batch ← new_batch
29.      else
30.        Perform decode(running__batch)
31.        decode_count_one_batch ← decode_count_one_batch + 1
32.      decode_count ← decode_count + 1 ▷更新元数据
33.      decode_count ← decode_count + new_batch
```

算法 3. 调度算法所使用的辅助函数

```
1. function generate_new_batch(prefill_reqs, hungry_prefill_reqs)
2.   Sort prefill_reqs by len(prompt) + predict_output_len ascending   ▷根据剩余时间最短优先进行排序
3.   Sort hungry_prefill_reqs by waiting_time descending, then len(prompt) + predict_output_len ascending
4.   new_batch ← ∅
5.   for each req in hungry_prefill_reqs do                            ▷优先调度饥饿队列
6.     if can_add_req(req, new_batch) and not_met_max_lora(req, new_batch) then
7.       new_batch.append(req); hungry_prefill_reqs.remove(req)
8.     else
9.       break
10.  for each req in prefill__reqs do                                  ▷遍历调度正常的 prefill 队列
11.    if can_add_req(req, new_batch) and not_met_max_lora(req, new_batch) then   ▷优先调度聚类的任务
12.      new_batch.append(req); prefill_reqs.remove(req)
13.    else
14.      break
15.  if new_batch not full then                                        ▷若队列未满,则继续调度其他的任务
16.    for each req in hungry_prefill_reqs + prefill_reqs do
17.      if can_add_req(req, new_batch) then
18.        new_batch.append(req); prefill_reqs.remove(req)
19.  for each req in prefill_reqs and hungry_prefill_reqs do            ▷更新等待时间
20.    req.waiting_time ← req.waiting_time + 1
21.  for each req in prefill_reqs do                                   ▷更新饥饿队列
22.    if req.waiting_time ≥ Threshold then
23.      prefill_reqs.remove(req); hungry_prefill_reqs.append(req)
24.  return new_batch
25. function schedule_new_batch(running_batch, decoding_reqs, hungry_decoding_reqs)
26.   Sort decoding_reqs by predict_output_len − len(output) ascending   ▷根据剩余时间最短优先进行排序
27.   Sort hungry_decoding_reqs by waiting_time descending, then predict_output_len − len(output) ascending
28.   new_batch ← ∅
29.   for each req in hungry_decoding_reqs do                          ▷优先调度饥饿队列
30.     if can_add_req(req, new_batch) and req.lora ∈ running_batch then
31.       new_batch.append(req); hungry_decoding_reqs.remove(req)
32.   for each req in decoding_reqs do                                 ▷调度正常队列
33.     if can_add_req(req, new_batch) and req.lora ∈ running_batch then
34.       new_batch.append(req); decoding_reqs.remove(req)
35.   if new_batch not full then                                       ▷调度其他任务
36.     for each req in hungry_decoding_reqs + decoding_reqs do
37.       if can_add_req(req, new_batch) then
38.         new_batch.append(req); decoding_reqs.remove(req)
39.   for each req in decoding_reqs do                                 ▷更新等待时间
40.     req.waiting_time ← req.waiting_time + 1
41.   return new_batch
```

首先,判断 *running_batch* 是否为空,以此来判断系统是否处于冷启动阶段,也即上一次调度并没有进行服务(行 7)。如果 *running_batch* 为空,则进行冷启动,调用 *generate_new_batch* 来从 *prefill_reqs* 中获取新的要调度的 *new_batch*(行 8),如果获取到 *new_batch* 非空,便对 *new_batch* 进行 prefill(行 9~13),否则,调度系统继续空闲(行 14~15)。

如果 *running_batch* 非空,判断之前连续进行的 decode 阶段次数是否到达阈值 *max_cont_decode*,以检验是否需要切换至 prefill 阶段(行 17)。如果达到了,调用 *generate_new_batch* 来从 *prefill_reqs* 和 *hungry_prefill_reqs* 中调度新的

请求(行 18～22)；否则，继续判断对一个 batch 的连续调度是否达到阈值，以检验是否需要更新当前服务批次数据(行 24)。如果达到，调用 *schedule_new_batch* 从 *decoding_reqs* 和 *hungry_decoding_reqs* 中按照 SRTF 和聚类策略调度新的处于 decode 阶段的请求(行 25～28)；否则，继续处理当前的 *running_batch*(行 30～31)。这种二级阈值判断的策略可以很好地减少因为频繁的 batch 切换而造成的频繁的 LoRA adapter 和 KV cache 的换入换出，同时也可以通过调整阈值的大小来保证调度的即时性和灵活性。

值得说明的是，现有的大语言模型推理服务的调度策略，大部分都属于启发式的调度策略，如 FIFO、Fastserve 以及 MQLserve 的调度策略等。随着机器学习和深度学习的发展，部分研究工作也尝试将基于群智能的方法(如蚁群算法和粒子群算法等)或基于深度强化学习的方法(如 DQN 和 PPO 等)应用于实时调度中。本文出于资源需求和时效性要求的原因，没有采用类似的算法。一方面，这类算法往往需要占用较多的显存资源，不适合本文所关注的资源受限场景。

另一方面，这类算法通常需要较高的调度时间，而大语言模型的在线服务对实时性要求较高，过高的调度速度会对系统性能带来负面影响；此外，相对于单任务服务场景，在多任务服务场景中还需要额外考虑任务种类等属性，这无疑对调度算法的时效性有着更高的要求。尽管本文出于以上原因没有采用基于群智能或深度强化学习的方法，这仍然是一个非常好的角度，并希望在未来工作中验证其效果。

FU Fang-Cheng, Ph. D., postdoctoral researcher. His research interests include machine/deep learning systems, and big data analytics and processing.

XIA Yi-Fei, Ph. D. candidate. His research interests include deep learning systems.

CUI Bin, Ph. D., professor, Ph. D. supervisor. His research interests include database system, big data management and analytics, and ML/DL systems.

Background

As the deployment of large language models (LLMs) in downstream tasks such as text summarization, machine translation, and dialogue systems increases, techniques for efficient model fine-tuning and deployment have become critical due to the growing size of these models and the well-known GPU shortage problem. Two main approaches have emerged: parameter-efficient fine-tuning (PEFT) and quantization-then-deployment. PEFT, exemplified by techniques like LoRA, involves training small-scale adapters to align the base model to specific tasks, thereby reducing fine-tuning costs. Meanwhile, low-bit quantization methods such as GPTQ and AWQ decrease memory demands and improve inference efficiency while preserving model quality.

Despite these advancements, existing LLM serving systems are primarily designed for single-task scenarios. The increasing need for multi-task capabilities has spurred the development of systems like S-LoRA and Punica, which utilize a shared base model and activate different task-specific LoRA adapters based on incoming requests, allowing for the concurrent processing of multiple tasks. However, several challenges persist in multi-task scenarios.

Firstly, integrating mainstream model quantization methods with multi-task serving systems is problematic. These quantization techniques require task-specific calibration, making it difficult to maintain a unified quantized model across multiple tasks due to the need for different LoRA adapters during the quantization process. This results in performance issues or limitations in resource-constrained environments.

Secondly, existing systems cannot dynamically add new tasks once a model is deployed, lacking flexibility in multi-task management. In these systems, adding tasks typically necessitates halting and restarting the serving process, which compromises stability and robustness.

Thirdly, existing systems fail to address the variations in workloads across different tasks, such as sequence length and processing time. Thus, they need to load multiple adapters and frequently switch between them during scheduling, resulting in significant inefficiencies.

To overcome these issues, this paper proposes a new system called MQLserve.

MQLserve introduces an innovative multi-task quantization algorithm, MLGPTQ, which allows for joint quantization using multi-task data, enabling a shared quantized base model across tasks. This algorithm also supports incremental quanti-

zation for new tasks without disrupting existing services.

In addition, MQLserve employs a novel multi-task scheduling strategy that predicts output lengths and groups tasks, which minimizes memory usage and swapping, thereby enhancing system performance. MQLserve integrates these features to provide a flexible and high-performance multi-task serving platform suitable for resource-limited settings.

Experimental evaluations of MQLserve show that, compared to existing systems, MQLserve can handle models 3.5 times larger, increases average throughput by 30% under high load, reduces average latency by 27%, speeds up response time by 4.61 times, and improves SLO compliance by 5.6 times under normal load compared to existing systems.

This work was supported by the National Science and Technology Major Project (2022ZD0116315), the National Natural Science Foundation of China (62402011), the China National Postdoctoral Program for Innovative Talents (BX20230012), the China Postdoctoral Science Foundation (2024M750103), the Beijing Natural Science Foundation (4244080), and the Research grant No. IPT-2024JK29.