https://crad.ict.ac.cn
62(6): 1513−1529, 2025

Pipe-RLHF: 计算模式感知的 RLHF 并行加速框架

徐 颖[1,2] 王梦迪[1] 程 龙[3] 刘 炼[1,2] 赵世新[1,2] 张 磊[1] 王 颖[1]

[1]（中国科学院计算技术研究所 北京 100190）

[2]（中国科学院大学 北京 100190）

[3]（华北电力大学控制与计算机工程学院 北京 102206）

（xuying18z@ict.ac.cn）

Pipe-RLHF: A Computation Mode-Aware Parallel Framework for RLHF

Xu Ying[1,2], Wang Mengdi[1], Cheng Long[3], Liu Lian[1,2], Zhao Shixin[1,2], Zhang Lei[1], and Wang Ying[1]

[1]（*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190）

[2]（*University of Chinese Academy of Sciences*, *Beijing* 100190）

[3]（*School of Control and Computer Engineering*, *North China Electric Power University*, *Beijing* 102206）

Abstract Reinforcement learning with human feedback (RLHF) has been proven effective in aligning large language models (LLMs) with human preferences. The most costly part of RLHF is proximal policy optimization (PPO), which consists of three dependent steps. Different PPO steps in RLHF exhibit different computation modes, simply employing the same parallelization strategy to accelerate all steps that involve multiple model variants, as done in existing frameworks, will lead to poor performance in the PPO generation step due to insufficient utilization of computational resources. Thus, we introduce Pipe-RLHF, a parallelism framework for RLHF fine-tuning, which adaptively employs distinct parallelization strategies for different steps according to the computation mode. Specifically, we first investigate the characteristics of various computation modes to explore their best-fit parallelization approach. And then, we present a novel delayed inter-batch pipeline parallelization approach specifically designed for the PPO generation step, enabling the sufficient utilization of computational resources. Subsequently, based on the proposed inter-batch pipeline parallelization approach, we define a hierarchical parallel plan space for distributed RLHF fine-tuning. Finally, we present optimization algorithms to find the optimal parallelization plan from the defined hierarchical parallel plan space to minimize the overall time consumption. Implementation and evaluation across multiple LLMs demonstrates that the proposed Pipe-RLHF achieves 3.7 times speedup compared with existing methods while achieving near-linear scalability.

Key words reinforcement learning with human feedback (RLHF); proximal policy optimization (PPO); large language models fine-tuning; distributed systems; parallel computing

摘 要 基于人类反馈的强化学习（reinforcement learning with human feedback, RLHF）作为当前大语言模型（large language models, LLMs）对齐的主流方法，其核心优化算法——近端策略优化（proximal policy optimization, PPO）却面临着显著的效率问题. PPO 由生成、推理、训练 3 个相互关联的阶段组成，各个阶段有着不同的计算特性. 然而，现有的 RLHF 并行框架采用相同并行策略顺序执行 PPO 的所有阶段，这导致以下 2 个问题：其一，生成阶段不能充分利用计算资源，进而影响整体效率；其二，阶段间严格串行执行，

收稿日期：2025-03-01；**修回日期**：2025-04-13

基金项目：国家自然科学基金项目（92473205）；国家重点研发计划项目（2023YFB4404400）

This work was supported by the National Natural Science Foundation of China (92473205) and the National Key Research and Developmemnt Program of China (2023YFB4404400).

未能充分利用潜在并行性. 针对上述问题,提出了一个新型 RLHF 并行框架——Pipe-RLHF.该框架能够自适应地根据各阶段的计算特征确定最优并行策略,突破现有阶段串行范式,采用异步 PPO 算法发掘阶段间的并行性. 具体而言,创新性地提出了适用于 PPO 生成阶段的延迟批间流水线并行方法,显著提升了该阶段的计算资源利用率;再次,使用异步 PPO 解放阶段间的依赖关系,将阶段间并行应用到 PPO 的加速上;最后,针对 PPO 算法的整体优化,构建了分层并行策略空间,并提出了一套优化算法以实现该空间中的最优解搜索. 通过在多个大语言模型上的性能评估实验表明,相较于现有方法,Pipe-RLHF 最高可实现 3.7 倍的加速比,充分验证了该框架的有效性和优越性.

关键词 基于人类反馈的强化学习;近端策略优化;大模型微调;分布式系统;并行计算

中图法分类号 TP311.13; TP309

DOI: 10.7544/issn1000-1239.202550127 **CSTR:** 32373.14.issn1000-1239.202550127

基于人类反馈的强化学习(reinforcement learning with human feedback, RLHF)作为一种经典的强化学习算法,已在游戏操作[1-3]、机器人控制[4-6]及推荐系统[7]等多个领域取得了显著成效. 近期研究表明,RLHF 在指导大语言模型(large language models, LLMs)学习符合人类偏好和道德基准的行为方面展现出卓越性能,从而成为大语言模型价值观对齐的主流方法[8-10]. 目前,RLHF 已被广泛应用于多个先进大语言模型的微调过程中,包括 Open-AI 的 GPT-4[11]、Anthropic 的 Claud[12]、DeepMind 的 Sparrow[13]、Google 的 Gemini[14]、以及 DeepSeek[15]等. 这充分体现了 RLHF 在大语言模型对齐领域的重要地位和实际价值.

使用 RLHF 对大模型进行微调主要包含 3 个关键环节:监督微调(supervised fine-tuning, SFT)、奖励模型训练和近端策略优化(proximal policy optimization, PPO)[9,16]. PPO 作为 RLHF 的核心环节[17-18],因其高复杂度和高计算开销成为了整个 RLHF 流程的性能瓶颈[19]. PPO 的每个训练迭代过程包含 3 个紧密关联的阶段,分别对应大语言模型中生成、推理和训练 3 种不同计算模式. 首先,在生成阶段,被微调的大模型(即策略模型)根据输入的提示词(prompt)生成一批输出文本. 随后,在推理阶段,这些输出文本将被同时输入到奖励模型、SFT 模型以及策略模型本身,用于计算奖励值(reward)等关键统计量. 最后,在训练阶段,基于前 2 个阶段产生的文本和统计量,对策略模型进行参数更新和优化. 值得注意的是,PPO 的各阶段之间存在着密切的依赖关系,包括数据传递和参数同步等. 在分布式计算环境下,如何对这些阶段涉及的模型和计算任务进行切分与调度,将直接影响大语言模型微调的效率和收敛性能. 因此,本文研究致力于解决一个关键问题:如何在不破坏阶段间的数据关联的前提下,将 RLHF 这一涉及多模型协同的复杂负载高效地映射到分布式系统上,从而实现最优的执行效率.

当前,针对 RLHF 微调流程,已有多个分布式框架被提出,包括 Trl[20], Trlx[21], DeepSpeed-Chat[19]等,这些框架普遍采用三维并行(3D parallelism)方法来加速 PPO 的所有阶段,并对所有阶段实施相同的并行策略,这种统一的并行方案存在显著的优化空间. 尽管三维并行方法在大模型训练和推理阶段表现出良好的加速效果[22-23],但其在生成阶段的性能却不尽如人意,这主要源于以下原因:大模型在文本生成阶段本身的计算密度较低,而三维并行方法进一步对算子进行切分,导致计算密度进一步下降,从而造成计算资源利用率的严重不足. 因此,采用统一的三维并行策略来加速 PPO 的所有阶段是一种次优方案,其导致的生成阶段耗时增加将显著影响整个 RLHF 流程的效率. 此外,现有框架普遍采用顺序执行 PPO 各阶段的方式,因此仅能关注到单个阶段内部的并行优化机会. 这种设计思路未能从 PPO 阶段的全局视角出发,探索更丰富的并行优化可能性,例如阶段间的并行执行机会. 这一局限性进一步限制了现有框架的性能提升空间.

为了解决上述问题,本文研究提出了一个高效的 RLHF 大模型微调并行框架—Pipe-RLHF. 不同于现有工作仅关注于单个 PPO 阶段内部并行优化,Pipe-RLHF 从 PPO 的整体行为出发,统筹考虑所有 PPO 阶段的特性与关联性,充分挖掘 PPO 中的并行机会. 本文研究的主要创新和贡献有 3 点:

1) 针对 PPO 各阶段计算模式的差异性,深入分析各阶段特性. 基于分析结果,创新性地提出了适用于 PPO 生成阶段的延迟批间流水线并行方法,显著提升了生成阶段的计算资源利用率.

2) 验证了异步 PPO 算法在大模型微调中的有效

性，打破了现在大模型微调中 PPO 的 3 个阶段必须依次执行的桎梏，使得阶段间并行成为可能.

3）构建了一套基于 PPO 优化过程全局的分层并行策略空间，并设计了一套高效的优化算法，能够在该空间中快速求解最优并行策略. 这一创新为 RLHF 框架的并行优化提供了系统性的解决方案.

为验证所提出并行框架 Pipe-RLHF 的有效性，本文研究在不同规模的大语言模型上开展了系统性实验评估. 实验结果表明，与现有主流框架相比，Pipe-RLHF 在性能优化方面取得了显著提升，最高可实现 3.7 倍的加速比. 这一结果充分证明了本文所提出方法的优越性和实用价值.

1 研究背景及预备知识

1.1 RLHF

RLHF 是一种融合了强化学习与人类主观判断的训练技术. 该技术通过将人类偏好转化为奖励信号来指导模型优化，使模型能够习得那些难以通过传统奖励函数定义的复杂行为模式[13,24]. 近期研究表明，RLHF 在大语言模型微调领域展现出显著的应用价值，特别是在实现大语言模型与人类偏好对齐方面具有巨大潜力[16,25-26]. 具体而言，将 RLHF 用于大模型微调包含 3 个环节[9,16]：

1）监督微调环节. 首先由专业人员构建人工标注的示例数据集，即针对一组提示词，人工生成相应的回复，形成问答对集合. 随后，利用该数据集对预训练模型进行初步微调，使其具备基本任务执行能力. 该环节输出的模型称为 SFT 模型，作为后续优化的基础.

2）奖励模型训练环节. 我们首先收集人类偏好数据集. 具体做法为：根据给定提示词生成多个回复，并由人类进行排序，这些提示词、回复、排序共同组成了偏好数据集里的一条数据. 随后基于偏好数据集训练奖励模型，使其评分能够准确反映人类偏好特征. 一般而言，奖励模型从 SFT 初始化，添加一个线性层来获取表示分值的标量.

3）PPO 环节，又称为强化学习环节. 此环节通过强化学习算法优化大语言模型，使其生成满足人类偏好的高质量输出. 在此阶段，大模型以 SFT 模型为起点，以最大化奖励模型评分为目标，应用 PPO 算法进行模型微调. PPO 是一种强健有效的强化学习算法，在 RLHF 中起到了重要作用，是 RLHF 算法的核心，同时也是 RLHF 算法中最复杂且耗时的环节.

1.2 PPO

PPO 作为 OpenAI 提出的经典强化学习算法，凭借其卓越的性能和良好的易用性，在自然语言处理等多个领域得到了广泛应用[18]. 本节将详细介绍使用 PPO 对大语言模型进行微调的过程.

典型的 PPO 大语言模型微调系统通常包含多个协同工作的模型组件，主要包括：策略模型（actor model），即待优化的主体模型；参考模型（reference model），即 RLHF 第 1 阶段获得的 SFT 模型；奖励模型（reward model），基于人类偏好数据训练的打分模型；评论家模型，通常通过在策略模型主体上添加线性层实现，与策略模型共享主体参数[20-21,27]. PPO 的完整训练流程如图 1 所示，包含生成、推理、训练 3 个关键阶段.

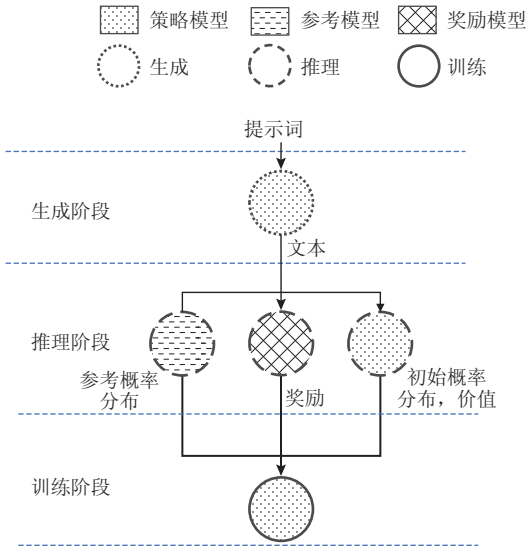


Fig. 1 Process of fine-tuning LLM with PPO

图 1 使用 PPO 微调大语言模型的流程

1）生成阶段. 策略模型采用自回归生成方式，基于给定的提示词集合逐 token 地生成相应的输出文本.

2）推理阶段. 将生成阶段的提示词及其对应的输出文本拼接成完整的文本，然后输入到参考模型、奖励模型、策略模型中，分别获得参考概率分布、评分、价值和初始概率分布等统计量.

3）训练阶段. 策略模型首先计算完整文本的当前概率分布. 随后，将当前概率分布与推理阶段获得的统计量（包括参考概率分布、奖励、价值估计等）相结合，共同计算得到损失函数. 基于该损失函数，系统执行反向传播算法获取梯度，进而对策略模型的参数进行迭代更新. 本阶段通常进行多次模型更新. 当仅执行单次更新时，当前概率分布可直接复用初始概率分布计算结果.

1.3 计算模式

如图 2 所示, 大语言模型主要存在 3 种计算模式, 每种模式具有独特的计算特征:

推理模式作为最基础的计算模式, 仅包含 1 个前向传播过程. 推理模式下的输入通常是一批文本, 张量尺寸为 [b, s, h], 其中 b 为批次大小, s 为文本长度, h 为隐藏层维度.

训练模式由前向传播与反向传播共同构成, 其输入张量尺寸为 [b, s, h]. 具体而言, 模型首先通过前向传播计算损失函数, 随后执行反向传播获取参数梯度, 最终利用这些梯度更新模型参数.

生成模式采用自回归机制, 通过多次前向传播迭代完成. 每个前向传播以一批 token 作为输入生成新的 token, 且新生成的 token 会作为下次前向传播的输入, 构成迭代处理链路. 值得注意的是, 生成模式的输入数据尺寸为 [b, 1, h], 与推理模式和训练模式存在显著差异.

1.4 并行方法

目前主流的针对单模型的并行方法主要有三维并行和 ZeRO.

三维并行作为当前大语言模型并行计算的标杆性方法, 结合了数据并行、张量并行和流水线并行 3 种策略, 显著提升了模型的训练效率[28-29]. 其中, 数据并行将输入数据均匀切分成多个小批次, 并将它们分配到不同的 GPU 上, 每个 GPU 都持有模型的全部参数, 各个 GPU 并行单独计算本地的参数梯度, 而后通过 all-reduce 聚合梯度并更新参数. 不同于数据并行, 张量并行是将模型参数进行划分, 每个 GPU 持有全部的输入数据, 但仅存储每层的部分参数, 每个 GPU 只计算每层的部分输出或中间结果, 然后通过 all-reduce 或者 all-gather 等通信原语重构完整结果, 每个 GPU 只负责本地参数的更新[30]. 流水线并行以层为单位将模型划分为多段, 每段包含多层[31-32]. 在基础流水线并行方案中, 当前段需要等待之前段结束之后才能开始执行, 这使得计算资源因等待充斥着大量的空闲时间, 我们称之为“气泡”. 为最小化气泡影响, 通常将输入数据切分为多个微批次(micro-batches), 通过流水线执行策略提高资源利用率[31], 本文将这种方法称为“批内流水线并行”.

ZeRO[33] 由微软公司的 DeepSpeed 团队提出, 是数据并行的变体. 不同于传统数据并行中每个 GPU 都存储完整模型副本的做法, ZeRO 通过切分参数、梯度、优化器状态来消除存储冗余. 根据切分范围, ZeRO 分为 3 个递进的阶段: ZeRO-1, ZeRO-2 和 ZeRO-3. ZeRO-1 是最基本的 ZeRO 阶段, 仅切分优化器状态, 即每个 GPU 存储完整的参数、梯度和部分优化器状态. 在每块 GPU 计算得到本地梯度后, 先使用 ReduceScatter 获取本地优化器状态对应的全局梯度,

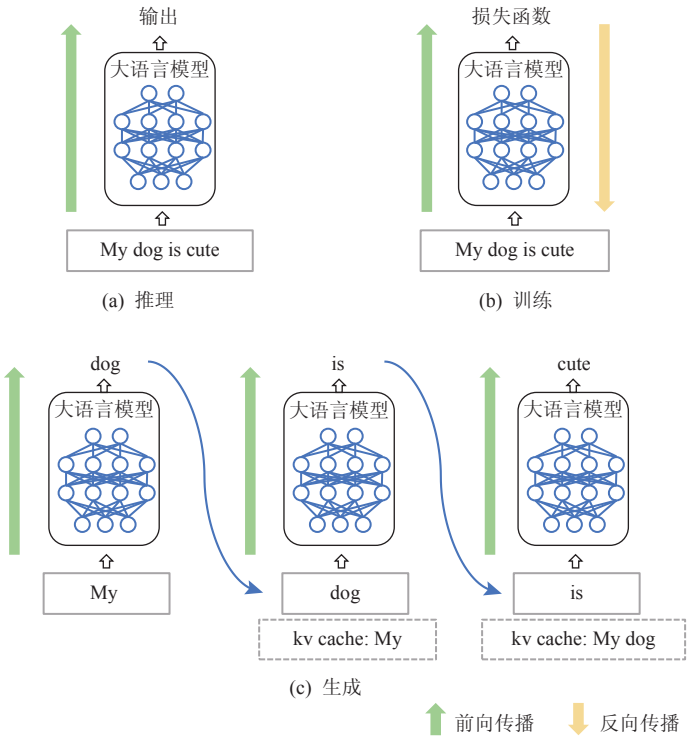


Fig. 2 Three computation modes for LLMs

图 2 大语言模型的 3 种计算模式

然后更新本地参数，最后通过 AllGather 获取全部的最新参数. ZeRO-1 的通信开销及内存节省与模型可学习参数的数量成正比. ZeRO-2 在 ZeRO-1 的基础上，进一步对梯度进行了切分，在使用 ReduceScatter 获取本地优化器状态对应的全局梯度后，释放优化器状态不在本地的梯度所占用的存储空间. ZeRO-2 的通信开销与 ZeRO-1 相同，但进一步节省了存储空间. ZeRO-3 对参数、梯度、优化器状态全部进行切分. 因其每个 GPU 仅存储部分参数，故而在每个算子进行前向传播、反向传播前，均需要进行 AllGather 从其他 GPU 获取完整参数，因此其通信开销及节省内存与模型总参数量成正比.

2 研究动机

在描述我们的研究动机之前，有必要明确以下关键术语的定义：

1)并行方法. 指特定的并行范式，通常由任务的可切分维度定义. 例如，数据并行、模型并行以及三维并行等都属于并行方法的范畴.

2)并行策略. 指在特定并行方法下的具体任务的切分方案. 例如，当我们使用三维并行在 64 个 GPU 上对大模型进行并行训练时，[2, 4, 8] 就是一个可行的三维并行策略，含义为在数据维度的切分为 2，张量维度的切分为 4，流水线维度的切分为 8.

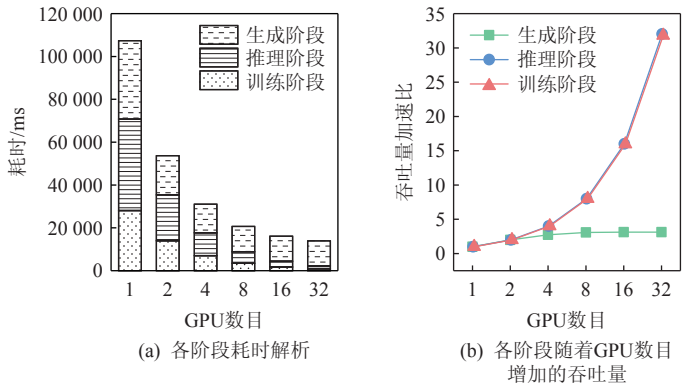
如表 1 所示，PPO 中的 3 个阶段分别对应 3 种不同的计算模式. 然而，现有 RLHF 框架忽略了这一重要特性，采用统一的三维并行策略对所有阶段进行加速. 我们对这一方法进行了复现并量化分析了该方法在各阶段的耗时分布. 复现实验中 SFT 模型采用 GPT-J-6B 架构，且所有模型均从 SFT 模型初始化. 遵循现有框架的做法，对于每一个集群配置，我们通过网格搜索获取训练阶段的最优三维并行策略，并将其应用于推理阶段和生成阶段.

Table 1 Computation Mode of Models in Each PPO Stage

表 1 PPO 各阶段中模型的计算模式

阶段名称	模型种类	计算模式
生成	策略模型	生成
推理	策略模型、参考模型、奖励模型	推理
训练	策略模型	训练

如图 3 所示，相同并行策略在不同计算模式下呈现出显著的性能差异. 在三维并行策略下，PPO 的训练阶段和推理阶段均能稳定获得良好的加速效果，而增加 GPU 给生成阶段带来的加速效益随 GPU 数量的增加趋于无. 换言之，在训练阶段表现优秀的三维并行策略并不适用于生成阶段的加速，现有框架对所有阶段统一采用相同三维并行策略的做法，最终导致生成阶段成为 PPO 算法的性能瓶颈，严重制约了整体效率的提升. 已知训练阶段使用的三维并行策略不适用于生成阶段，我们进一步探讨了为生成阶段单独优化三维并行策略的可行性. 图 4 展示了通过网格搜索为生成阶段专门求取三维并行策略的性能表现. 实验结果表明，现有的由数据并行、张量并行、批内流水线并行组成的三维并行方法在生成阶段的加速效果随 GPU 数量的增加而显著降低，甚至在 GPU 数量超过一定数值后彻底失效. 这种现象的根源在于，生成阶段本身具有较低的计算密度，而现有的三维并行方法对算子进行切分，导致计算密度进一步降低，最终导致计算负载不足以充分利用 GPU 的计算资源，三维并行失效. 因此，现有的三维并行方法并不适用于加速 GPU 数目较多时的生成阶段，其大规模 GPU 集群的生成阶段加速场景中存在根本性局限.



(a) 各阶段耗时解析

(b) 各阶段随着GPU数目增加的吞吐量

Fig. 3 Time-consuming analysis of speeding up all PPO steps using 3D parallelism strategy

图 3 使用三维并行策略加速所有 PPO 阶段的耗时分析

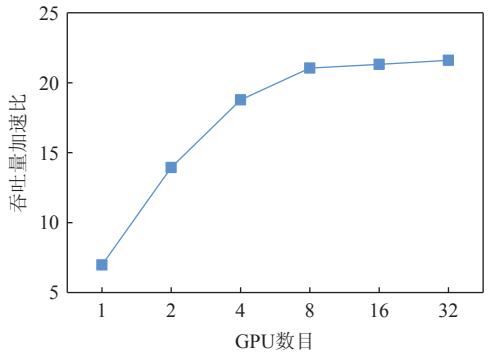


Fig. 4 Acceleration generation step using 3D parallelism strategy

图 4 使用三维并行策略加速生成阶段

综上所述, 现有 RLHF 并行框架和方法在生成阶段面临计算密度过小和计算资源利用率不足的关键挑战. 为了解决这个问题, 我们提出以下 2 个优化方向: 其一为优化资源配置, 通过减少参与生成阶段的 GPU 数量, 降低算子切分程度, 从而提高计算密度; 其二为开发新型并行方法, 避免或最小化算子层面的切分. 以上述 2 点为出发点, 我们开始设计新的 RLHF 并行框架.

3 Pipe-RLHF:高效灵活的 RLHF 并行框架

本节将详细介绍 Pipe-RLHF 的框架设计, 主要包括阶段间并行、延迟批间流水线并行、策略空间形式化和并行策略优化算法.

3.1 阶段间并行

我们根据第 2 节测量的 PPO 各个阶段的并行特征, 将 PPO 的 3 个阶段划分为 2 个任务组:

1) 高计算密度任务组, 包括推理阶段和训练阶段.

2)低计算密度任务组, 包括生成阶段.

我们的目标是减少低计算密度任务组的 GPU 数目, 在 GPU 总数目不变的前提下, 一个可行的思路是将总 GPU 资源划分为 2 个小的资源组, 分别分配给高计算密度任务组和低计算密度任务组. 然而在现有的 PPO 算法中, 各个阶段之间都存在依赖关系导致 2 个任务组之间无法同时计算, 必须相互等待, 这会造成资源的浪费, 如图 5(a)所示.

为了解决这个问题, 我们使用异步 PPO 算法, 该算法的核心思想是允许生成文本的策略模型和当前训练的策略模型不同, 即我们可以利用先前策略模型进行文本生成. 异步强化学习已被广泛应用于各种分布式系统中, 其有效性已得到充分证明[32-33].

异步 PPO 算法的阶段间依赖关系如图 5(b)所示, 该算法通过允许使用历史参数版本的策略模型生成的数据来训练当前策略模型, 从而缓解了生成阶段与训练阶段间的参数版本依赖, 使得阶段间并行成为可能. 为简化表示, 本文研究将 3 个阶段重新组织为 2 个任务组, 任务组间的依赖关系如图 5(c)所示, 可以看到, 参数版本相同的任务组间不存在依赖关系, 可以并行执行.

因此, 基于异步 PPO 算法, 本文将总 GPU 资源划分为 2 组, 分别分配给高计算密度任务组和低计算密度任务组, 实现 2 组任务并行处理. 如图 5(d)所示, 因为任务组间的参数版本依赖关系的约束(图 5(c)中虚线箭头), 系统整体的执行呈现流水线模式.

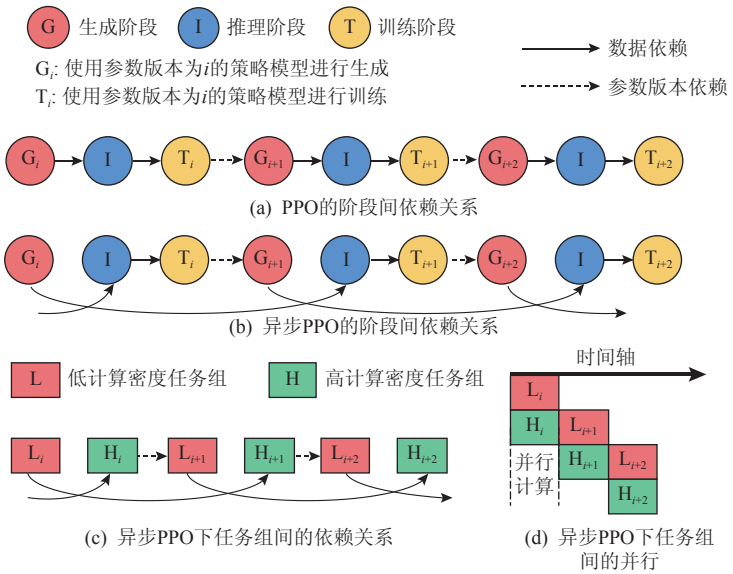


Fig. 5 Inter-stage dependent relationship analysis in different PPO algorithms

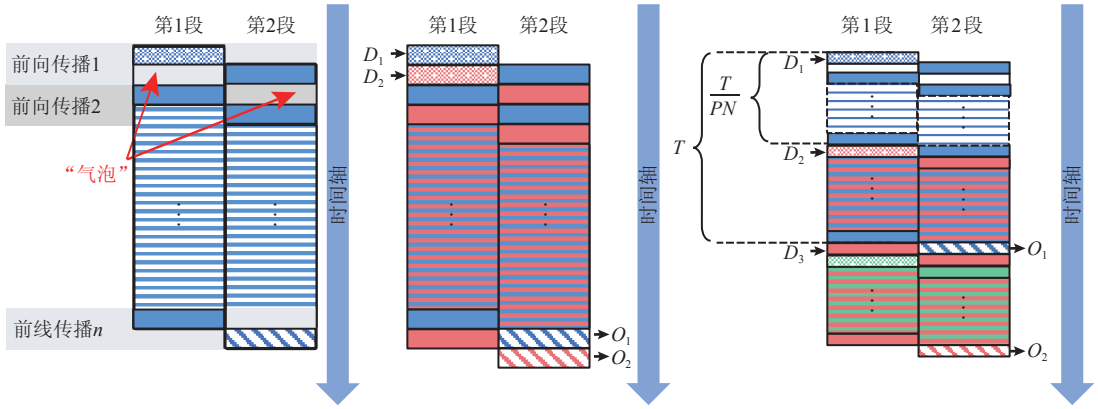
图 5 不同 PPO 算法中阶段间依赖关系分析

3.2 延迟批间流水线并行

本节旨在探索一种适用于 PPO 生成阶段的并行计算方法. 基于第 2 节的分析结果, 生成阶段应避免对算子内部进行切分, 以防止计算密度的进一步降低. 现有的数据并行和张量并行方法均涉及算子层面的切分, 因此无法满足需求; 而原始的流水线并行方法以层为单位对模型进行切分, 未涉及算子内部的切分, 符合本文研究的优化目标.

以流水线并行为基础, 本文进一步探讨生成阶段的并行优化问题. 生成阶段是一个自回归的过程, 由多个首尾相接的前向传播组成, 简单地进行流水线并行会导致大量的“气泡”. 如图 6(a)所示, 当模型以层为单位被切分为 2 段时, 在执行单个前向传播过程中, 第 2 段计算必须等待第 1 段计算完成后才能启动. 此外, 由于自回归特性, 下一个前向传播必须等待前一个前向传播完成后才能开始. 这种相互等待机制使得“气泡”比例高达 50%, 且随着切分段数的增加, “气泡”比例会进一步上升. 为解决这一问题, 必须引入有效的“气泡”填充机制.

为了解决“气泡”填充问题, 本文使用其他批数据的计算来进行“气泡”填充, 而非现在常见的将输入数据切分为多个微批的填充方式. 因为该做法对输入数据进行了切分, 不符合我们避免对算子内部进行切分的需求. 如图 6(b)所示, 本文研究采用的方法是在第一批输入数据 D_1 进行第 2 段计算的同时, 启动第 2 批输入数据 D_2 在第 1 段的计算. 通过这种设计, 2 批数据的计算能够相互填充, 从而有效消除“气泡”.



(a) 流水线并行下, 单批次数据在生成阶段的运算时序图　(b) 批间流水线并行, 在第i批输入数据开始运算后立即插入第i+1批输入数据会导致有效批大小增大　(c) 延迟批间流水线并行

Fig. 6 Timing diagram of operation of generation stage under diverse bubble filling approaches

图 6 不同气泡填充方法下生成阶段的运算时序图

生成阶段会执行多次前向传播, 即输入数据 D_1 会经历很多次第 2 段计算, 我们要考虑在 D_1 的哪一个第 2 段计算时插入 D_2. 如图 6(b)所示, 最直观的做法是在 D_1 的首个第 2 段计算就插入 D_2, 但这种做法破坏了生成阶段的有效批大小(batch size), 进而损坏模型的训练质量. 例如, 当 D_1 的有效批大小为 B, 切分段数为 PN 时, 图 6(b)中的办法会突发式地集中生成 $B\times PN$ 个文本, 而这 $B\times PN$ 个文本会被应用到接下来的训练阶段, 造成 PPO 有效批大小变动为 $B\times PN$. 有效批大小是一个高度敏感的超参数, 贸然改动可能会导致不可预料的后果, 在有些情况下会严重影响模型训练质量. 例如, 在情感分析任务中, 将有效批大小从 32 增加到 128, 会导致微调后的策略模型的评分严重下降(从 2.4 降到 2.126).

为了在填充气泡的同时保持模型的训练质量, 本文研究引入了延迟调度策略. 具体方法如图 6(c)所示, 假设一批数据的首个第 1 段计算开始到最后 1 个第 1 段计算结束的时间间隔为 T, 模型切分段数为 PN, 则我们在 T/PN 时插入下一批数据. 为了更加清晰地说明所使用的延迟调度方法, 我们给出如下实例:

假设模型被切分为 2 段, 每段的计算耗时为 1 秒, 生成阶段新生成的 token 数目为 128, 即每批数据要经历 128 次前向传播, 则延迟调度下生成阶段的执行情况如图 7 所示. D_1 分别在 1, 3, 5, …, 255 秒时执行第 1 段计算, 在 2, 4, …, 256 秒时执行第 2 段计算, 在 256 秒时输出由 D_1 生成的文本. 则 T=255, PN=2, 255/2=127.5, 最接近该时刻的下一段空闲时间为第 128 秒, 我们在该空闲插入 D_2, 则 D_2 在 128, 130, …, 382 秒执行第 1 段计算, 在 129, 131, …, 383 秒执行第 2 段计算, 最终在第 383 秒时输出文本. 同理, D_3 将在 257 秒

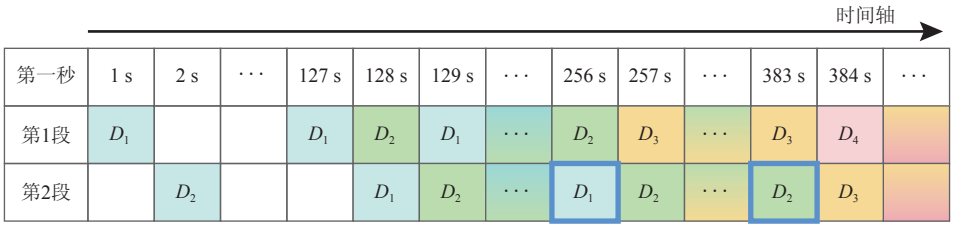


Fig. 7 An example of delayed inter-batch pipeline

图 7 延迟批间流水线实例

时插入, 在 512 秒时输出; D_4 将在 384 秒时插入, 在 639 秒时输出等.

通过上述实例可以看到延迟调度可以有效解决“气泡”问题, 除了 D_2 插入之前的启动阶段存在“气泡”, 当 D_2 插入后“气泡”被完全填充.

接下来延迟调度是否会破坏 PPO 的有效批大小. 从整体来看, 每批数据的总计算时长为 T, 且每过 T/NP 的时间开启一批新的数据计算, 其执行时间如图 8 中的红色方块所示. 我们可以将其等效为一系列单批处理时常为 T/PN 的任务块, 该任务块每过 T/PN 的时间产生一个输出. 为使整体 PPO 框架性能最大化, 与该任务块进行流水线并行的高计算密度度任务组的单批耗时也应约等于 P/PN. 系统整体的执行时间如图 8 所示, 与图 5(d)相比, 唯一的差别为 2 个任务组间的参数版本延迟由 1 变成了 PN, 即第 i 版模型进行训练时, 使用的是 i-PN 版模型生成的数据, 其他方面与异步 PPO 完全相同. 为了进一步验证异步调度的有效性, 本文选取文本总结、情感分析 2 个任务场景, 分别使用原始 PPO 算法和延迟版本数目为 4 的异步 PPO 算法对 GPT-1.3B 进行微调, 其结果如图 9 所示, 经过异步 PPO 微调的大模型质量与原始 PPO 相差无几, 甚至在文本总结任务上优于原始 PPO 的微调结果, 该实验进一步验证了异步 PPO 不会伤害模型的微调质量.

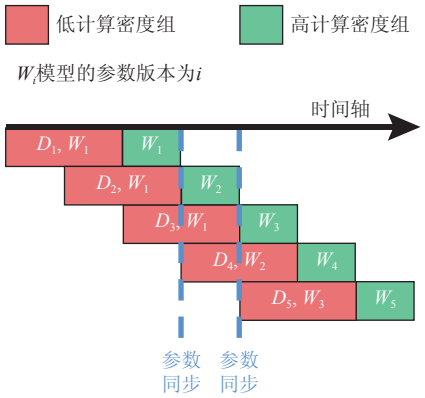


Fig. 8 Version delay of delayed inter-batch pipeline parallelism

图 8 延迟批间流水线并行的版本延迟

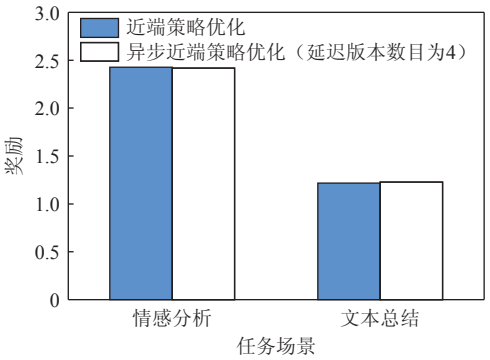


Fig. 9 Comparison of the model evaluation reward after using PPO and asynchronous PPO for fine-tuning

图 9 使用 PPO 和异步 PPO 进行微调后的模型评估奖励对比

我们将本节中提出的批间流水线与延迟调度结合的并行方法称为延迟批间流水线并行. 该方法专为 PPO 生成阶段设计, 可以有效避免对算子内部进行切分, 同时通过延迟调度, 在保证模型微调质量的前提下加速生成阶段.

3.3 并行策略优化

Pipe-RLHF 并行框架采用 2 级并行设计: 第 1 级并行为阶段间并行, 即低计算密度任务组和高计算密度任务组以流水线的形式并行计算; 第 2 级为 2 个任务组内的并行. 具体来说, 对于高计算密度任务组, 即推理阶段和训练阶段, 我们使用传统的由数据并行、张量并行、批内流水线并行组成的三维并行方法, 而对于低计算密度任务组, 即生成阶段, 我们使用由数据并行、张量并行和批间流水线并行组成的并行方法, 因为该并行方法除了引入数据切分并行、张量切分并行、层间切分并行之外, 还引入了批间并行, 我们将其称之为 3.5 维并行方法.

相应地, 两级并行的策略空间可以分层形式化为: 对于阶段间并行, 其对应的并行策略空间可以形式化为(N_l, N_h), 其中 N_l 为分配给低密度组的 GPU 数目, N_h 为分配给高密度组的 GPU 数目. 对于任务组内并行, 其策略空间可以形式化为(dp, tp, pp),其中, pp

在表示高计算密度任务组的并行策略时，其含义为批内流水线并行度，而表示低计算密度任务组的并行策略时，其含义为延迟批间流水线并行度.

对上述策略空间求解可以形式化为如下问题：

$$\text{obj}: \min_{(N_\text{l},N_\text{h},dp_\text{l},tp_\text{l},pp_\text{l},dp_\text{h},tp_\text{h},pp_\text{h})} T_\text{PPO}, \tag{1}$$

$$\text{sbj}: T_\text{PPO} = \max(T_\text{l}(N_\text{l}), T_\text{h}(N_\text{h})) + T_\text{comm} \tag{2}$$

$$T_\text{l}(N_\text{l}) = T^{3.5\text{D}}(\pi^\text{a}, dp_\text{l}, tp_\text{l}, pp_\text{l}), \tag{3}$$

$$T_\text{h}(N_\text{h}) = T^{3\text{D}}(\pi^\text{a}, \pi^\text{r}, \pi^\text{ref}, dp_\text{h}, tp_\text{h}, pp_\text{h}), \tag{4}$$

$$N_\text{l} + N_\text{h} \leqslant N, \tag{5}$$

$$dp_\text{l} \times tp_\text{l} \times pp_\text{l} \leqslant N_\text{l}, \tag{6}$$

$$dp_\text{h} \times tp_\text{h} \times pp_\text{h} \leqslant N_\text{h}, \tag{7}$$

其中，π^a, π^r, π^ref分别为策略模型、奖励模型和参考模型. T_comm 为 2 个任务组间进行参数同步的耗时，由 2 个任务组内策略模型的参数分布决定.

本文设计了一套启发式的算法来求解上述问题，如算法 1 所示. 遍历所有的合法配置(N_l, N_h)，根据式(2)求取各个合法配置下的T_PPO，取结果最小的配置. S_l 和S_h 分别表示 2 个任务组的内部并行策略，其格式为$S = (dp, tp, pp)$. 本文研究采用了一种算法优化技巧：按照N_l从小到大的顺序进行遍历，当出现局部极小点，则停止遍历，该局部极小点就为所求最优点，该优化技巧的有效性证明如下：

已知低密度任务组耗时$T_\text{l}(N_\text{l})$ 随着N_l的增大单调递减，高密度任务组耗时$T'_\text{h}(N_\text{l}) = T_\text{h}(N - N_\text{l})$，其随着$N_\text{l}$的增大单调递增，则所求目标函数$f(N_\text{l}) = \max(T_\text{l}(N_\text{l}), T'_\text{h}(N_\text{l}))$在$T_\text{l}$和$T'_\text{h}$的交点 k 处取得最小值. 具体分析为：

当$N_\text{l} = k$时，$T_\text{l}(k) = T'_\text{h}(k) = f(k)$，

当$N_\text{l} < k$时，$f(N_\text{l}) = T_\text{l}(N_\text{l}) > T_\text{l}(k) = f(k)$，

当$N_\text{l} > k$时，$f(N_\text{l}) = T'_\text{h}(N_\text{l}) > T'_\text{h}(k) = f(k)$.

因此k是目标函数$f(T_\text{l}) = \max(T_\text{l}, T_\text{h})$的局部极小点，同时也是全局最小点.

算法 1. PPO 的并行优化.

① procedure *AutoParallelPPO*(N, π^a, π^r, π^ref);
② $Buff = \varnothing$;
③ for $N_\text{l} \in \{1, 2, \cdots, N-1\}$ do
④ $N_\text{h} = N - N_\text{l}$;
⑤ $T_\text{l}(N_\text{l})$, S_l = *AutoParallel*3.5D(π^a, N_l);
⑥ $T_\text{h}(N_\text{h})$, S_h = *AutoParallel*3D(π^a, π^r, π^ref, N_h);
⑦ 使用通信模拟器获取 $T_\text{comm}(S_\text{l}, S_\text{h})$;
⑧ get T_PPO according to 式(2);
⑨ if 出现局部极小值
⑩ return N_l, N_h, S_l, S_h;
⑪ else
⑫ add $\{T_\text{PPO}, N_\text{l}, N_\text{h}, S_\text{l}, S_\text{h}\}$ to *Buff*;
⑬ end if
⑭ end for
⑮ end procedure

此外，T_comm 相对$\max(T_\text{l}, T_\text{h})$来说往往较小，式(2)中由$\max(T_\text{l}, T_\text{h})$占主导地位，因此$\max(T_\text{l}, T_\text{h})$的最小值即为$T_\text{PPO}$的最小值.

阶段内并行策略的求取采用网格搜索法，详见算法 2 和算法 3. 本文使用多进程并行评估网格上所有的合法点，以加速算法的执行速度. 对于生成阶段，为了保证异步 PPO 算法的有效性，参数版本延迟不宜过大，故而对延迟批间流水线并行度进行限制，规定其不可大于阈值P_max(算法 3 行④).

算法 2. 训练阶段和推理阶段的并行优化.

① procedure *AutoParallel*3D(N_h, π^a, π^r, π^ref);
② $Buff = \varnothing$;
③ for $tp \in \{1, 2, \cdots, N_\text{h}\}$ do
④ for $pp \in \left\{1, 2, \cdots, \frac{N_\text{h}}{tp}\right\}$ do
⑤ $dp \leftarrow \frac{N_\text{h}}{tp \times pp}$;
⑥ if *simu_mem*_3D($\pi^\text{a}, \pi^\text{r}, \pi^\text{ref}, dp, tp, pp$) $\leqslant$ *memory_limit* then
⑦ $t \leftarrow$ *simu_time*_3D($\pi^\text{a}, \pi^\text{r}, \pi^\text{ref}, dp, tp, pp$);
⑧ add $\{t, dp, tp, pp\}$ to *Buff*;
⑨ end if
⑩ end for
⑪ end for
⑫ $t_\text{opt} = \min_t(Buff)$;
⑬ $dp_\text{opt}, tp_\text{opt}, pp_\text{opt} = \arg\min_t(Buff)$;
⑭ return t_opt, $(dp_\text{opt}, tp_\text{opt}, pp_\text{opt})$;
⑮ end procedure

算法 3. 生成阶段的并行优化.

① procedure *AutoParallel*3.5D(N_l, π^a);
② $Buff = \varnothing$;
③ for $tp \in \{1, 2, \cdots, N_\text{l}\}$ do
④ for $pp \in \left\{1, 2, \cdots, \min\left(\frac{N_\text{h}}{tp}, P_\text{max}\right)\right\}$ do
⑤ $dp \leftarrow \frac{N_\text{l}}{tp \times pp}$;
⑥ if *simu_mem*_3.5D(π^a, dp, tp, pp) $\leqslant$ *memory_limit* then
⑦ $t \leftarrow$ *simu_time*_3.5D(π^a, dp, tp, pp);
⑧ add $\{t, dp, tp, pp\}$ to *Buff*;
⑨ end if

⑩ end for
⑪ end for
⑫ $t_{opt} = \min_{t}(Buff)$;
⑬ $dp_{opt}, tp_{opt}, pp_{opt} = \arg\min_{t}(Buff)$;
⑭ return t_{opt}, (dp_{opt}, tp_{opt}, pp_{opt});
⑮ end procedure

我们针对高计算密度任务组和低计算密度任务组分别设计了耗时开销模拟器和存储开销模拟器, 以求取$T^{3.5D}(\pi^{a},S_1)$,$T^{3D}(\pi^{a},\pi^{r},\pi^{ref},S_h)$. 对于$T_{comm}$, 我们使用 multi2multi 的传输方式, 各个节点的通信耗时通过WS/BW进行估算, 其中WS为其收发的数据量, BW为其节点传输带宽, 然后求取所有节点的最大通信时长来作为T_{comm}.

4 实验数据与分析

4.1 实验方法

1)集群配置. 所有实验均在一个 A100 集群上进行. 该集群的每个节点配备 4 块 A100(40 GB)GPU, 同一节点内的 GPU 通过 NVLink v3 进行互联, 节点内通信带宽高达 600 GB/s. 不同的节点通过 InfiniBand 网络相连, 节点间带宽为 400 Gb/s.

2)大语言模型. 在 GPT 系列模型上测试 Pipe-RLHF 的性能, 参与实验的模型包括 GPT-6B, GPT-20B, GPT-66B 和 GPT-175B. 这些模型结构参数如表 2 所示, 这 4 个模型的隐藏层维度分别为 4 096, 6 144, 9 216 和 12 288; 注意力头数目分别为 32, 64, 72 和 96; 分别由 32 层、44 层、64 层和 96 Transformer block 组成.

Table 2 Model Architectures in Experiments

表 2 实验模型结构

模型	隐藏层维度	注意力头数目	模型层数
GPT-6B	4 096	32	32
GPT-20B	6 144	64	44
GPT-66B	9 216	72	64
GPT-175B	12 288	96	96

3)微调细节. 我们使用包括 Anthropic's Helpful Harmless dataset[10] 在内的多个据集对上述模型进行微调, 并使用文献 [18] 中提出的数据预处理方法来统一不同数据集的格式. 对于所有的模型, 我们将有效批大小设置为 256, PPO 每个迭代中训练阶段的参数更新次数设置为 1, 输入的提示词长度和最大新生成文本长度均设置为 256. 此外, 由于全参数微调的内存开销过大, 我们使用 LoRA 算法来冻结大部分参数来降低内存占用, 以便测试 Pipe-RLHF 在更大尺寸模型上的表现[34]. 对于上述 4 个模型, 参数版本延迟阈值P_{max}设置为 16.

4)对比方法. 在吞吐量实验中, 我们设置了如下方法作为对比组.

①全局三维并行策略. PPO 中所有阶段都使用相同的三维并行策略, 现有 RLHF 微调框架 Trl, Trlx, DeepSpeed 均采用此种方法.

②阶段独立三维并行策略. 对 PPO 中每个阶段单独求取其最优三维并行策略, 且 3 个阶段顺序执行.

③延迟批间流水线并行. 该方法中, PPO 各个阶段顺序执行, 但是 PPO 的生成阶段使用包含批间流水线并行的 3.5 维并行.

④OpenRLHF[35]. OpenRLHF 将计算资源划分为若干份, 分别分配给奖励模型、参考模型、策略模型的生成阶段和策略模型的训练阶段. 对于策略模型的生成阶段使用数据并行和张量并行, 对于其他模型则使用数据并行和 ZeRO-3.

⑤HybridFlow[36]. HybridFlow 首先将 RLHF 中的模型进行分组, 每组中有 1 个或多个模型, 之后以组为单位进行计算资源分配. HybridFlow 对所有模型都使用三维并行.

4.2 吞吐量加速比

我们使用全局三维并行策略、阶段独立三维并行策略、延迟批间流水线并行、OpenRLHF、HybridFlow 及 Pipe-RLHF 在 8 个节点(32 块 GPU)上对表 2 中 4 个模型进行微调, 测量其每秒处理的文本数目, 实验结果如图 10 所示. Pipe-RLHF 在所有情况下均优于现有并行框架使用的方法. 在 4 个模型的微调中, Pipe-RLHF 对于全局三维并行策略和阶段独立三维并行策略分别取得了 3.34 倍、2.17 倍、2.31 倍、1.12 倍的加速和 3.3 倍、2.18 倍、1.66 倍、1.1 倍的加速. 阶段独立三维并行策略较优于全局三维并行策略, 但是因为三维并行方法本身不适用于生成阶段, 因此对生成阶段单独求取三维并行策略效果并不明显. OpenRLHF 在本文实验中吞吐量很低, 多数情况下低于全局三维并行, 导致这种现象的一个原因是 OpenRLHF 的资源分配方法在同步 PPO 下会导致大量的资源空闲, 比如分配给奖励模型、参考模型和策论模型生成阶段的 GPU 在生成阶段会空闲等待. 另一个原因则是它使用 ZeRO-3 加速单个大模型, 导致大量的全局数据传输, 当节点间带宽受限时, 通信开销会非常庞大. OpenRLHF 在微调 GPT-66B 和 GPT-175B 时出现了内存不足的问题. 在本实验中, HybridFlow

往往选择将所有模型分为一组来避免资源等待空闲，在这种情况下，HybridFlow 等效于阶段独立三维并行策略. 通过对实验结果进行进一步分析可以发现，在计算资源固定的情况下，Pipe-RLHF 的相对加速效果随着模型的增大而减小. 导致这种现象的一个原因是模型越大，生成阶段的计算密度越高，计算资源利用率不足的程度越轻微. 另一个原因是越大尺寸的模型部署所需的最小 GPU 数目越多，有效策略集 (N_l,N_h) 较小，难以找到可以平衡 2 个任务组耗时的 GPU 分配. 相对于延迟批间流水线并行，Pipe-RLHF 执行 4 个模型微调的加速比分别为 1.26、1.15、1.01、0.76，意味着随着模型的增大，阶段间流水线的收益逐渐降低，甚至产生负收益. 这一方面是因为随着计算密度增大，单独使用延迟批间流水线就可以解决计算资源利用率不足的问题；另一方面是因为阶段间流水线并行效策略集 (N_l,N_h) 的缩小会导致阶段间耗时差异过大.

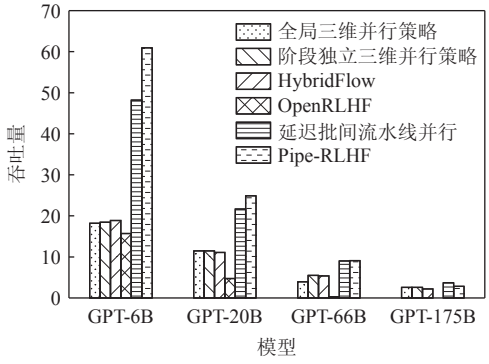


Fig. 10 Comparision of throughput

图 10 吞吐量对比

在之后的实验中，我们将单独使用延迟批间流水线并行看作 Pipe-RLHF 禁用阶段间并行的一个特殊情况，不再对其单独列出. 此外，因为全局三维并行策略、阶段独立三维并行策略和 HybridFlow 这 3 种方法的性能相似，我们之后仅对阶段独立三维并行策略进行探讨.

4.3 可扩展性

为了分析 Pipe-RLHF 的可扩展性，我们在 8~32 个节点上分别使用 Pipe-RLHF、OpenRLHF 和阶段独立三维并行策略对 4 个大模型进行微调，分别测量各种情况下的吞吐量及耗时组成. 测量结果如图 11 所示，Pipe-RLHF 的可扩展性显著优于阶段独立三维并行策略和 OpenRLHF. 阶段独立三维并行策略在模型较小时的可扩展性接近于无，随着模型的增大可扩展性有略微提升，但依旧很差. 这是因为在三维并行方法下，随着 GPU 的增加，生成阶段成为了总体微调耗时的瓶颈，且单纯扩展 GPU 数目无法进一步减小生成阶段的耗时. OpenRLHF 将不同的模型放到不同的 GPU 上，在同步 PPO 下，这种放置策略导致了巨大的资源空闲，即图中 Idle 的部分；其次，这种分开放置的方法下，各模型的共享参数要被存储多份，增大了存储压力，减小了可以微调的模型规模的上限，比如在微调 GPT-175B 时，OpenRLHF 出现了内存不足的问题. 此外，当模型超出单个 GPU 显存时，OpenRLHF 使用 ZeRO-3 并行方法切分模型，这会带来巨大的通信开销，显著提高总体耗时. 与之相反，Pipe-RLHF 展现出接近线性的可扩展性，这是因为阶段间并行和延迟批间流水线并行可以有效保持生成阶段的计算密度，且异步 PPO 使得不同的阶段可以同时执行，避免了资源等待空闲.

4.4 延迟批间流水线并行

本节探讨延迟批间流水线对生成阶段的加速效果. 我们分别测试了在使用三维并行和包含批间流水线并行的 3.5 维并行时，GPT-6B 在 1~32 个 GPU 上生成文本的速度. 结果如图 12 所示，当 GPU 数目为 4 时，三维并行开始出现计算资源利用率不足的问题，加速效果降低. 而 3.5 维并行依靠延迟批间流水线并行完美规避了这个问题，呈现出近乎线性的可扩展性.

4.5 计算资源利用率

为了更直观地说明 Pipe-RLHF 的加速原理，我们使用 NVDIA Nsight Compute 统计了使用 Pipe-RLHF 和三维并行微调大模型时的全局计算资源利用率. 全局计算资源利用率数值上等于每一个 GPU 的资源利用率的平均值，而每一个 GPU 的资源利用率可以通过对算子利用率加权求和获得，权值为该算子的执行时长占比. 全局资源利用率的统计结果如表 3 所示. 不出意外我们观察到模型越小计算资源利用率越低，这不仅是因为算子切片太小不足以完全利用所有计算单元，还因为算子切片的 GPU 的执行时间太短，不能覆盖 CPU 对算子的调度时间，造成了 GPU 的空闲等待. 而 Pipe-RLHF 一直保持着较高的全局计算资源利用率，这得益于它避免了对算子的过度切分，保证了每个算子切片的计算负载，从而阻止了上述问题的发生.

4.6 并行策略分析

在本节中，我们基于一个实例对 4.3 节中的并行策略求取算法的有效性进行分析. 当可用 GPU 数目为 128，微调模型为 GPT-6B 时，通过算法 1 求取的最优阶段间 GPU 分配为 N_l=32，N_h=96. 图 13 展示了该

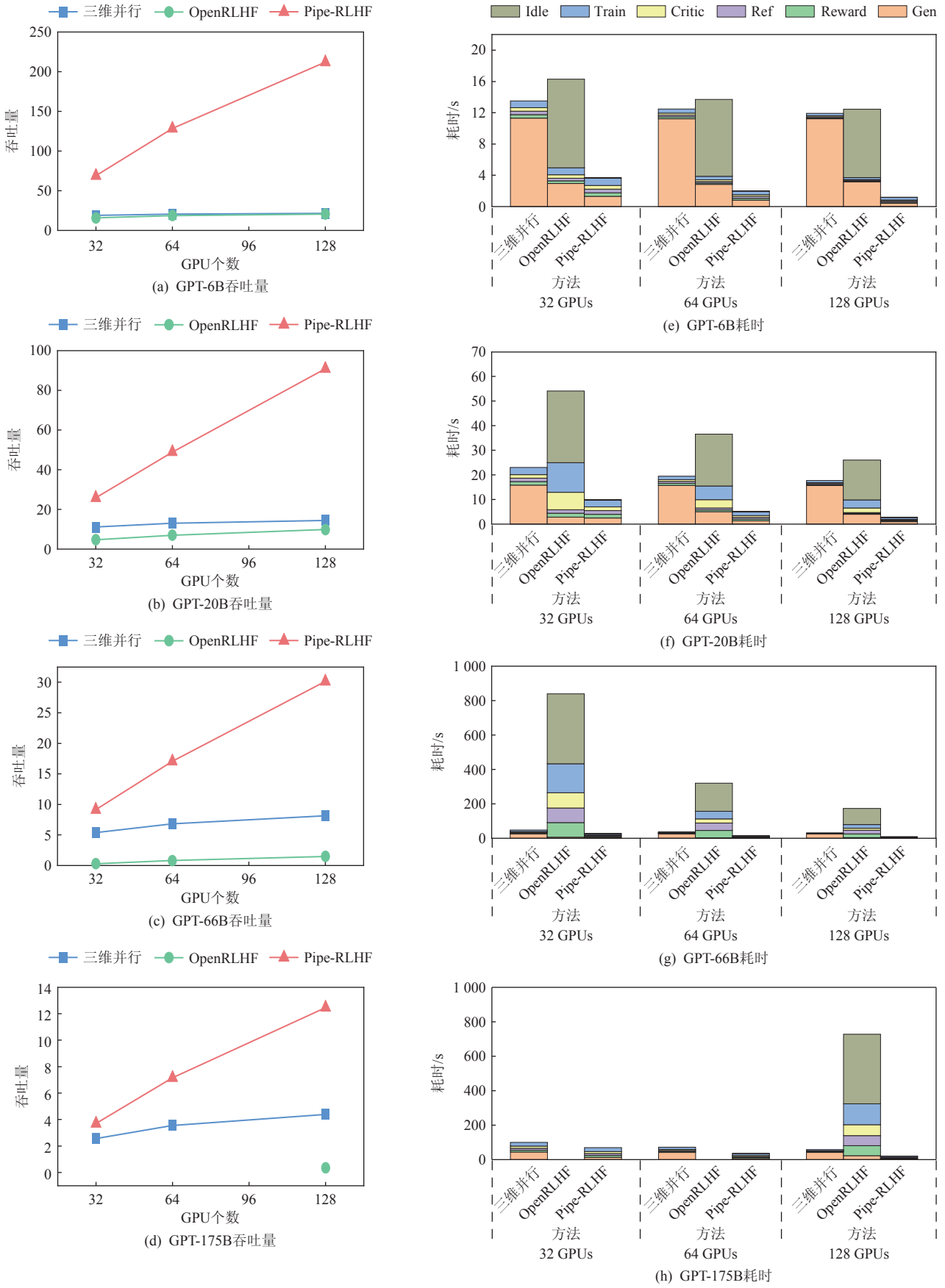


Fig. 11 Comparison of scalability and time cost analysis

图 11 可扩展性对比和耗时分析

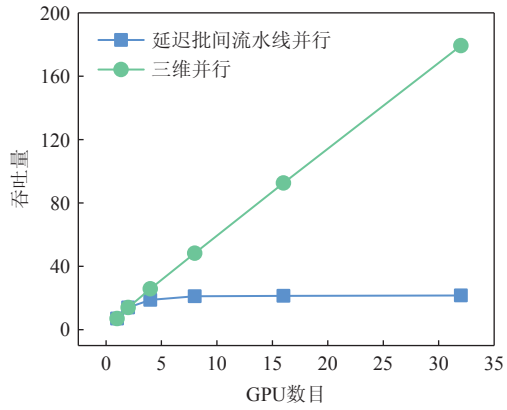


Fig. 12 Comparison between 3D parallelism and delayed inter-batch pipeline parallelism in generation efficiency

图 12 三维并行与延迟批间流水线并行的生成效率对比

Table 3 Comparison of Computing Resource Utilization

表 3 计算资源利用率对比 %

模型	三维并行 计算资源利用率	Pipe-RLHF 计算资源利用率
GPT-6B	23.7	68.7
GPT-20B	38.3	74.4
GPT-66B	55.6	76.3
GPT-175B	64.2	77.5

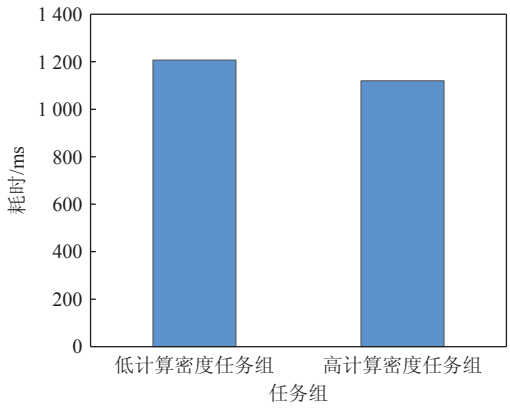


Fig. 13 Time-consuming of GPT-6B for the low computational density task group and the high computational density task group

图 13 GPT-6B 的低计算密度任务组耗时和高计算密度任务组耗时

分配策略下低计算密度任务组和高计算密度任务组的平均单批耗时，由图 13 可得 2 个阶段的耗时非常接近，说明算法 1 求得了符合需求的分配策略. 对于生成阶段，算法 3 求得的并行策略为 dp=2, tp=1, pp=16，最大化地减少了对算子的切分.

4.7 Pipe-RLHF 与现有推理引擎的结合

目前，针对生成阶段资源利用率低下这一问题，有工作设计了专用于生成阶段的推理引擎. 推理引擎优化了小尺寸算子在单个 GPU 上的运算，与本文方法是正交的关系，推理引擎和本文方法可以结合使用，同时从系统层面和任务切分层面对生成阶段进行优化.

DeepSpeed-Chat 是当前顶级的 RLHF 微调框架之一，它的核心技术是专为生成阶段设计的推理引擎，而在并行方法上则简单使用 ZeRO(训练阶段、推理阶段)和张量并行(生成阶段). 我们将 DeepSpeed-Chat 的推理引擎应用到 Pipe-RLHF 中，探索 Pipe-RLHF 和生成专用推理引擎结合后的效果. 作为对比，我们还测量了原始 DeepSpeed-Chat 的性能以及 DeepSpeed-Chat 卸除专用引擎后的性能，测量结果如图 14 所示. 可以看到，不论是否使用专用推理引擎，Pipe-RLHF 的性能都显著优于使用简单并行方法的 DeepSpeed-Chat，虽然 DeepSpeed-Chat 的专用引擎一定程度上减缓了资源利用率不足的问题，但效果有限. 此外，相对于原始 Pipe-RLHF，装备了专用引擎的 Pipe-RLHF 取得了 1.18 倍、1.15 倍、1.06 倍的加速比，这说明 Pipe-RLHF 可以有效与生成专用的推理引擎相结合，进一步提升性能，并且提升效果随着模型的减小而增加.

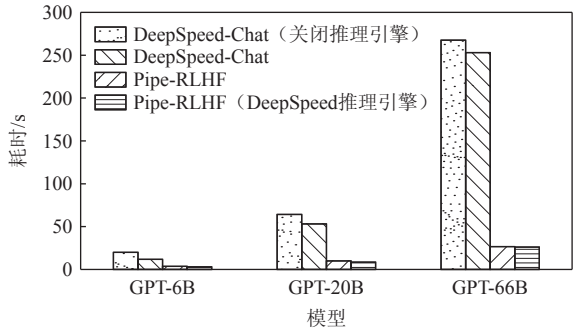


Fig. 14 Fine-tuning time cost of three models obtained by combining Pipe-RLHF with DeepSpeed inference engine

图 14 对 3 种模型结合 Pipe-RLHF 和 DeepSpeed 推理引擎取得的微调耗时

4.8 延迟版本的影响

本节探索了 Pipe-RLHF 中，超参数 P_{max} 对微调的影响. P_{max} 表示 Pipe-RLHF 的延迟版本阈值，控制着生成阶段中延迟批间流水线并行的最大并行度，也控制着生成阶段和训练阶段间的参数延迟版本上限. 我们分别探索了 P_{max} 对收敛曲线和吞吐量的影响.

图 15 中展示了不同延迟版本下 GPT-2 在情感分析任务上的微调曲线. 由图可得，延迟版本数的增加不会影响模型最终的收敛质量，但是会影响模型的收敛速率，即延迟版本数越大，模型收敛所需要的训

练步数越多. 此外, 收敛速率下降的速度随着延迟版本数的增大而增大, 比如延迟版本数从 0 增加到 16, 收敛步数从 90 增加到 130, 增加了 40; 而延迟版本数从 16 增加到 32, 收敛步数从 130 增加到了 210, 增加了 80. 图 16 中展示了 P_{max} 下, Pipe-RLHF 在 128 块 GPU 上进行的微调吞吐量. 总体来说, P_{max} 与微调吞吐量成正比, P_{max} 越大, 吞吐量越大. 对于较小的模型, P_{max} 的影响比较大, 这是因为模型越小, 计算密度越低, 所需的延迟批间流水线并行度越大, 以减小算子切分. 此时若设置较小的 P_{max}, 则会导致算子切分过多, 最终影响吞吐量. 此外, 在资源固定的情况下, 随着 P_{max} 越大其影响力越小, 这是因为当算子内切分小于一定程度时, 计算密度已经可以充分利用 GPU 计算资源, 故而进一步减小算子内切程序不会带来更显著的收益.

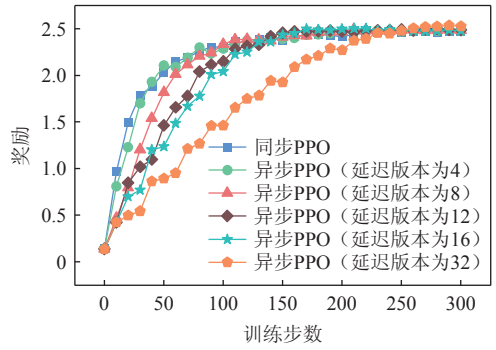


Fig. 15 Training curves of synchronous PPO and asynchronous PPO

图 15 同步 PPO 和异步 PPO 的训练曲线

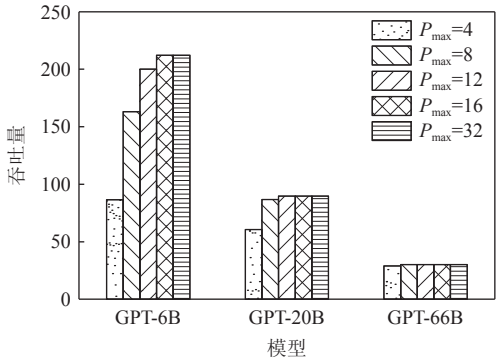


Fig. 16 Throughput of models for fine-tuning using Pipe-RLHF with different P_{max}

图 16 不同 P_{max} 下使用 Pipe-RLHF 进行模型微调的吞吐量

综上, P_{max} 太大会显著增加模型的收敛步数, 而 P_{max} 太小又会影响微调的吞吐量, 我们需要选择合适的 P_{max} 来减小模型的整体耗时.

5 相关工作

5.1 RLHF 框架

目前已经有多个工作进行了 RLHF 框架设计方面的创新和努力, 包括 Trl[20], Trlx[21], ColossalChat[37], DeepSpeed-Chat[19], HybridFlow[36], OpenRLHF[35] 和 LLaMA-Factory[38] 等.

其中, Trl 是由 Huggingface 推出的全栈库, 它提供了一组使用强化学习对大语言模型进行微调的工具, 涵盖了从监督微调(SFT)步骤、奖励建模(RM)步骤到 PPO 步骤的全流程. Trlx 是由 CarperAI 提出的一个分布式大模型微调框架, 除了奖励模以外, 还支持用户使用自定义的奖励函数对大模型进行微调. Trl 和 Trlx 均采用 Hugging Face 的 Accelerate 库来减少 RLHF 的时间消耗. ColossalChat 是一个开源项目, 它提供了完整的 RLHF 流水线, 包括数据集收集、有监督微调、奖励模型训练、强化学习微调和模型部署.

DeepSpeed-Chat 是一个端到端的 RLHF 框架, 它利用 DeepSpeed 推理引擎加速生成步骤, 该引擎具备量化、推理定制内核等功能, 可高效实现大语言模型的生成. 上述框架均使用同一并行策略来加速所有的 PPO 阶段.

HybridFlow 是字节跳动公司与香港大学联合开发的 RLHF 微调框架, 作为开源大模型强化学习引擎 veRL[39] 的底层加速框架, 它采用混合控制器架构和分层 API 设计, 同时通过 3D-HybridEngine 优化参数分配. OpenRLHF 是由 OpenLLMAI 等联合开发的开源 RLHF 微调框架, 利用 Ray, DeepSpeed 和 vLLM 优化资源利用率和训练性能, 此外该框架还将 PPO 中的多个模型部署至不同 GPU 上, 显著提升内存利用率与训练吞吐量. HybridFlow 和 OpenLLMAI 均关注于 PPO 单阶段内的优化, 没有探索 PPO 阶段间的并行性.

LLaMA-Factory 是一个集成了多种高效微调方法的大模型训练与微调平台, 支持 DPO, PPO 等多种微调技术, 并通过图形界面简化了大模型微调的复杂性. 框架旨在降低大模型微调的门槛, 为大众提供一个简单易用的 RLHF 框架, 并未对 RLHF 的高效分布式实现进行探索.

5.2 大语言模型的生成加速

除了本文提到的并行以外, 还有许多技术被应用于优化大模型的生成开销, 它们大致可以分

为 4 类：

1）低精度量化. 这类方法通过对权重、激活值进行低精度量化来缓解大语言模型生成时的访存瓶颈. 在存储、传输时，数据被量化为低精度，减少数据存储和加载开销；加载完成后，数据被反量化到正常精度进行计算，以保持计算准确性[40].

2）计算核心优化. 这类方法通过优化大模型生成的注意力计算等算子的底层 CUDA 核心代码来加速大模型的生成. 比如 FlashAttention[41] 利用算子平铺和重计算等技术，将注意力计算在批大小和查询长度上切分成多个可以放到 SRAM 中的小块分别计算，有效减少了对 GPU 显存的访问. Flash-Decoding[42] 在 FlashAttention 的基础上添加了 key/value 序列长度这一切分维度，从而提高了算子在 GPU 内核上的并行度，进而加速了注意力算子的计算.

3）kv-cache 优化. 大模型生成过程中往往需要存储大量的历史 key/value，花费了大量的显存[43]. ZeRO-Inference[44]、FlexGen[45] 将 key/value 卸载到 SSD 等其他资源上，并制定相应策略预先加载需要的数据，在减少 kv-cache 显存占用的基础上保证吞吐量. Paged-Attention[46] 不再一次性分配所有的 kv-cache，而是以块为单位动态分配和回收 kv-cache.

4）批处理技术. 通常用在大模型服务部署中，将多个用户的请求合并为一个批次统一处理，可以有效提高 GPU 利用率和吞吐量. 常用的批处理方法有静态批处理、动态批处理和连续批处理. 静态批处理使用固定的批次大小，在批次内所有请求完成计算后统一返回响应. 而动态批处理则根据当前的请求量实时调整批次的大小，但依然是在一个批次内所有计算完成后统一返回响应[47]. 作为改进，连续批处理不再等待批次内所有请求都完成，而是当某一请求完成时立刻返回该请求结果，并插入新的请求填充其位置[48].

上述方法中的低精度量化、计算核心优化、kv-cache 优化等方法均属于本地 GPU 上的优化，旨在加速单块 GPU 上的任务执行，可以与本文提出的 Pipe-RLHF 结合使用. 而批处理多适用于大模型服务部署等动态输入的场景，RLHF 的生成阶段的输入是静态且规律的，不存在使用批处理的动态、不可预测的先决条件，故而批处理技术无法用于加速 RLHF 的生成阶段.

6 总　　结

本文提出了一个灵活高效的 RLHF 分布式的微调框架 Pipe-RLHF. 具体而言，我们提出了阶段间并行和延迟批间流水线并行性方法，该方法有效提升了生成阶段的计算资源利用率，显著提高了 PPO 的执行效率. 另外，我们构建了面向 PPO 算法全流程的分层并行策略空间，并设计了一套优化算法用于获得最优并行策略. 我们对 Pipe-RLHF 进行了系统实现和全面评估. 实验结果表明，Pipe-RLHF 相比现有方案表现出显著的性能提升.

作者贡献声明：徐颖负责方案整体设计和实验分析并撰写论文；王梦迪、程龙、王颖提出论文思路并负责理论指导和论文修改；刘炼、赵世新负责论文的修改和润色；张磊负责论文的审核.

参　考　文　献

[1] Christiano P F, Leike J, Brown T B, et al. Deep reinforcement learning from human preferences[C]//Proc of the 31st Int Conf on Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc., 2017: 4302–4310

[2] Macglashan J, Ho M K, Loftin R, et al. Interactive learning from policy-dependent human feedback[C]//Proce of the 34th Int Conf on Machine Learning - Volume 70. New York: ACM, 2017: 2285–2294

[3] Warnell G, Waytowich N, Lawhern V, et al. Deep TAMER: Interactive agent shaping in high-dimensional state spaces[C]//Proc of the 32nd AAAI Conf on Artificial Intelligence and 30th Innovative Applications of Artificial Intelligence Conf and 8th AAAI Symp on Educational Advances in Artificial Intelligence. Palo Alto, CA: AAAI, 2018: 1545−1554

[4] Brown D, Goo W, Nagarajan P, et al. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations[C]//Proc of Int Conf on Machine Learning. Cambridge, MA: JMLR, 2019: 783−792

[5] Shin D, Dragan A D, Brown D S. Benchmarks and algorithms for offline preference-based reward learning[J]. arXiv preprint, arXiv: 2301.01392, 2023

[6] Sadigh D, Dragan A, Sastry S, et al. Active preference-based learning of reward functions[C]//Proc of Robotics: Science and Systems. Cambridge, Massachusetts: Escholarship. https://www.roboticsproceedings.org/rss13/p53.html

[7] Maghakian J, Mineiro P, Panaganti K, et al. Personalized reward learning with interaction-grounded learning (IGL)[C]//Proc of the 11th Int Conf on Learning Representations. Washington, DC: OpenReview. net, 2022

[8] Yi Xiaoyuan, Xie Xing. Unpacking the ethical value alignment in big models[J]. Journal of Computer Research and Development, 2023, 60(9): 1926−1945 (in Chinese)
(矣晓沅, 谢幸. 大模型道德价值观对齐问题剖析[J]. 计算机研究与发展, 2023, 60(9): 1926−1945)

[9] Ouyang Long, Wu J, Jiang Xu, et al. Training language models to

follow instructions with human feedback[C]//Proc of Conf on Neural Information Processing Systems 2022, Red Hook, NY: Curran Associates Inc., 2022: 27730−27744

[10] Bai Y, Jones A, Ndousse K, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback[J]. arXiv preprint, arXiv: 2204.05862, 2022

[11] Achiam J, Adler S, Agarwal S, et al. GPT−4 technical report[R/OL]. OpenAI, 2023[2025-04-08]. https://arxiv.org/pdf/2303.08774

[12] Anthropic. Introducing claude[EB/OL]. 2023[2025-04-09]. https://www.anthropic.com/news/introducing-claude

[13] Glaese A, Mcaleese N, Trębacz M, et al. Improving alignment of dialogue agents via targeted human judgements[J]. arXiv preprint, arXiv: 2209.14375, 2022

[14] Pichai S, Hassabis D. Introducing Gemini: Our largest and most capable AI model[EB/OL]. 2023[2025-04-09]. https://blog.google/technology/ai/google-gemini-ai/

[15] Bi Xiao, Chen Deli, Chen Guanting, et al. DeepSeek LLM: Scaling open-source language models with longtermism[J]. arXiv preprint, arXiv: 2401.02954, 2024

[16] Stiennon N, Ouyang Long, Wu J, et al. Learning to summarize with human feedback[C]//Proc of Conf on Neural Information Processing Systems 2020. Red Hook, NY: Curran Associates Inc., 2020: 3008−3021

[17] Schulman J, Wolski F, Dhariwal P, et al. 2017. Proximal policy optimization algorithms[J]. arXiv preprint, arXiv: 1707.06347, 2017

[18] Zheng Rui, Dou Shihan, Gao Songyang, et al. Secrets of RLHF in large language models part I: PPO[J]. arXiv preprint, arXiv: 2307.04964, 2023

[19] Yao Zhewei, Aminabadi R Y, Ruwase O, et al. 2023. DeepSpeed-Chat: Easy, fast and affordable RLHF training of ChatGPT-like models at all scales[J]. arXiv preprint, arXiv: 2308.01320, 2023

[20] Werra L, Belkada Y, Tunstall L, et al. trl: Transformer reinforcement learning[EB/OL]. 2020[2025-04-09]. https://github.com/hugging-face/trl

[21] Havrilla A, Zhuravinskyi M, Phung D, et al. Trlx: A framework for large scale reinforcement learning from human feedback[C]//Proc of the 2023 Conf on Empirical Methods in Natural Language Processing. Stroudsburg, PA: ACL, 2023: 8578−8595

[22] Lai Z, Li S, Tang X, et al. Merak: An efficient distributed DNN training framework with automated 3D parallelism for giant foundation models[J]. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(5): 1466−1478

[23] Song J, Yim J, Jung J, et al. Optimus-CC: Efficient large NLP model training with 3D parallelism aware communication compression[C]// Proc of the 28th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2023: 560–573

[24] Knox W B, Stone P. TAMER: Training an agent manually via evaluative reinforcement[C]//Proc of the 7th IEEE Int Conf on Development and Learning. Piscataway, NJ: IEEE, 2008: 292−297

[25] Ziegler D M, Stiennon N, Wu J, et al. 2019. Fine-tuning language models from human preferences[J]. arXiv preprint, arXiv: 1909.08593, 2019

[26] Wu J, Ouyang Long, Ziegler D M, et al. 2021. Recursively summarizing books with human feedback[J]. arXiv preprint, arXiv: 2109.10862, 2021

[27] Santacroce M, Lu Yadong, Yu Han, et al. 2023. Efficient RLHF: Reducing the memory usage of PPO[J]. arXiv preprint, arXiv: 2309.00754, 2023

[28] Zheng Lianmin, Li Zhouhan, Zhang Hao, et al. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning[C]//Proc of the 16th USENIX Symp on Operating Systems Design and Implementation (OSDI 22). Berkeley, CA: USENIX Association, 2022: 559−578

[29] Xiao W, Bhardwaj R, Ramjee R, et al. Gandiva: Introspective cluster scheduling for deep learning[C]//Proc of the 13th USENIX Conf on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 595–610

[30] Shoeybi M, Patwary M, Puri R, et al. 2019. Metatron-LM: Training multi-billion parameter language models using model parallelism[J]. arXiv preprint, arXiv: 1909.08053, 2019

[31] Huang Yanping, Cheng Youlong, Bapna A, et al. GPipe: Efficient training of giant neural networks using pipeline parallelism[C]//Proc of the 33rd Int Conf on Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc., 2019: 103−112

[32] Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: Generalized pipeline parallelism for DNN training[C]//Proc of the 27th ACM Symposium on Operating Systems Principles. New York: ACM, 2019: 1–15

[33] Rajbhandari S, Rasley J, Ruwase O, et al. ZeRO: Memory optimizations toward training trillion parameter models[C]//Proc of SC20: Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2020: 1−16

[34] Hu E J, Shen Y, Wallis P, et al. Lora: Low-rank adaptation of large language models[C]//Proc of Int Conf on Learning Representations. openreview. net, 2022

[35] Hu Jian, Wu Xibin, Zhu Zilin, et al. OpenRLHF: An easy-to-use, scalable and high-performance RLHF framework[J]. arXiv preprint, arXiv: 2405.11143, 2024

[36] Sheng Guangming, Zhang Chi, Ye Zilingfeng, et al. HybridFlow: A flexible and efficient RLHF framework[J]. arXiv preprint, arXiv: 2409.19256, 2024

[37] Li Shenggui, Liu Hongxin, Bian Zhengda, et al. Colossal-AI: A unified deep learning system for large-scale parallel training[C]//Proc of the 52nd Int Conf on Parallel Processing. New York: ACM, 2023: 766–775

[38] Zheng Yaowei, Zhang Richong, Zhang Junhao, et al. LLaMA-Factory: Unified efficient fine-tuning of 100+ language models[C]//Proc of the 62nd Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA: ACL, 2024: 400−410

[39] Volcengine. veRL: Volcano engine reinforcement learning for LLMs [EB/OL]. 2025[2025-04-08]. https://github.com/vocengine/verl

[40] Qin Haotong, Ma Xudong, Zheng Xingyu, et al. Accurate LoRA-finetuning quantization of LLMs via information retention[J]. arXiv preprint, arXiv: 2402.05445, 2024

[41] Dao T, Fu D Y, Ermon S, et al. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness[C]//Proc of the 36th Int Conf on Neural Information Processing Systems. New Orleans, LA: Curran Associates Inc. 1189: 16344−16359

[42] Dao T, Haziza D, Massa F, et al. Flash-Decoding for long-context

inference[EB/OL]. 2024[2025-03-01]. https://crfm.stanford.edu/2023/10/12/flashdecoding.html

[43] Ge Xuran, Ou Yang, Wang Bo, et al. Survey of storage optimization techniques in large language model inference[J]. Journal of Computer Research and Development, 2025, 62(3): 545−562 (in Chinese)
(葛旭冉, 欧洋, 王博, 等. 大语言模型推理中的存储优化技术综述[J]. 计算机研究与发展, 2025, 62(3): 545−562)

[44] DeepSpeed. ZeRO-Inference: Democratizing massive model inference[EB/OL]. 2022[2025-04-08]. https://www.deepspeed.ai/2022/09/09/zero-inference.html

[45] Sheng Ying, Zheng Lianmin, Yuan Binhang, et al. FlexGen: High-throughput generative inference of large language models with a single GPU[C]//Proc of Int Conf on Machine Learning. Cambridge, MA: PMLR, 2023: 31094−31116

[46] Kwon W, Li Zhouhan, Zhuang Siyan, et al. Efficient memory management for large language model serving with pagedattention[C]//Proc of the 29th Symp on Operating Systems Principles. New York: ACM, 2023: 611−626

[47] Nvidia. Triton inference server[EB/OL]. 2025[2025-04-09]. https://github.com/triton-inference-server/server

[48] Yu G I, Jeong J S, Kim G W, et al. Orca: A distributed serving system for Transformer-Based generative models[C]//Proc of the 16th USENIX Symp on Operating Systems Design and Implementation (OSDI 22). Berkeley, CA: USENIX Association, 2022: 521−538



Xu Ying, born in 1996. PhD candidate. Her main research interests include machine learning system and parallel deep learning framework.

徐　颖, 1996 年生. 博士研究生. 主要研究方向为机器学习系统、深度学习并行框架.



Wang Mengdi, born in 1995. Assistant professor. Her main research interests include accelerator design and multi-core system.

王梦迪, 1995 年生. 助理研究员. 主要研究方向为加速器设计、多核系统.



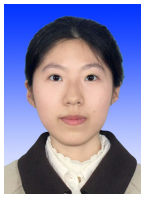
Cheng Long, born in 1985. PhD, professor, PhD supervisor. His main research interest includes distributed computing and high-performance systems.

程　龙, 1985 年生. 博士, 教授, 博士生导师. 主要研究方向为分布式计算、高性能系统.



Liu Lian, born in 1999. PhD candidate. His main research interest includes LLM infrastructure、hardware-software co-design.

刘　炼, 1999 年生. 博士研究生. 主要研究方向为大语言模型系统架构设计、软硬件协同设计.



Zhao Shixin, born in 1999. PhD candidate. Her main research interest includes computer architecture.

赵世新, 1999 年生. 博士研究生. 主要研究方向为计算机体系结构.



Zhang Lei, born in 1981. PhD, professor. Member of CCF. His main research interests include chip and computing system.

张　磊, 1981 年生. 博士, 正高级工程师. CCF 会员. 主要研究方向为芯片、计算系统.



Wang Ying, born in 1985. PhD, professor. Senior member of CCF. His main research interests include novel EDA, and processor and memory system architecture.

王　颖, 1985 年生. 博士, 研究员. CCF 高级会员. 主要研究方向为新型 EDA、处理器与存储系统体系结构.