

大语言模型推理中的键值缓存压缩方法综述

胡世鹏 张广艳 郑纬民

(清华大学计算机科学与技术系 北京 100084)

(hsp21@mails.tsinghua.edu.cn)

Survey on KV Cache Compression Methods for Large Language Model Inference

Hu Shipeng, Zhang Guangyan, and Zheng Weimin

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)

Abstract Large language model (LLM) inference requires buffering and reusing intermediate data, namely KV cache. However, the rapid growth in the size of KV cache imposes significant storage pressure on systems. On one hand, the large-scale KV cache of ongoing inference requests leads to GPU memory bottlenecks, thereby limiting inference parallelism. On the other hand, the long-term memory capability enabled by LLM service providers results in large-scale KV cache being stored on persistent storage devices, incurring high storage costs and loading overhead. To alleviate the storage pressure caused by KV cache, existing research proposes various KV cache compression strategies to reduce its size. Nonetheless, it remains challenging to achieve high compression ratios while minimizing the impact on model response quality and keeping the additional computational overhead at an acceptable level. We present a systematic survey of recent advances in KV cache compression strategies. First, we analyze the scale of KV cache and its impact on inference efficiency and storage cost. Based on this, we identify three major challenges in designing KV cache compression strategies: preserving model response quality, minimizing computational overhead, and ensuring generalizability. We then categorize existing KV cache compression approaches into five types: merge-based, quantization-based, token eviction-based, sharing-based, and attention head pruning-based strategies. Each category is elaborated upon with representative examples, discussing their design principles and implementation techniques. Finally, we outline potential future research directions in KV cache compression.

Key words large language model; inference service; KV cache; storage system; data compression

摘要 大语言模型的部署推理需要针对中间数据,即键值缓存,进行存储复用。然而键值缓存的空间规模迅速增长,为系统带来显著的存储压力:一方面,当前推理请求的大规模键值缓存导致显存瓶颈问题,从而限制推理并行度。另一方面,长期记忆需求导致海量键值缓存长期存储在持久存储设备内,带来高昂的存储成本和加载开销。为了缓解键值缓存带来的存储压力,现有研究针对键值缓存进行压缩,从而削减其空间规模,然而如何在提升压缩率的同时减小对模型回答质量的影响并将压缩带来的计算开销控制在较低程度仍面临着诸多挑战。系统地综述了键值缓存压缩策略的研究进展,首先介绍键值缓存的规模及其对推理效率、存储成本的影响,并总结键值缓存压缩策略面临的三大挑战:模型回答质量、计算开销以及泛化性。然后对现有的键值缓存压缩策略进行分类:基于合并的、基于量化的、基于标记驱逐的、基于共享的以及基于注意力头削减的压缩策略。对每种压缩策略进行阐述,并分析其设计原理和软件技术。最后指出了键值缓存压缩策略的未来研究方向。

收稿日期: 2025-06-09; 修回日期: 2025-11-13

基金项目: 国家自然科学基金项目(62025203)

This work was supported by the National Natural Science Foundation of China (62025203).

通信作者: 张广艳 (gyzh@tsinghua.edu.cn)

关键词 大语言模型;推理服务;键值缓存;存储系统;数据压缩

中图法分类号 TP333

DOI: 10.7544/issn1000-1239.202550461 CSTR: 32373.14.issn1000-1239.202550461

近年来,大语言模型(large language model, LLM)如 GPT 系列^[1]、DeepSeek^[2]、Gemini^[3]和 LLaMa^[4]等在语言理解^[5]、生成^[6-8]、逻辑推理^[9-11]等任务上取得了突破性进展,并被广泛部署于智能问答^[12-13]、代码生成^[14-17]、教育辅导^[18-19]等实际应用场景中。然而,这些模型在推理阶段的计算资源消耗极为庞大,特别是其核心组件:Transformer 注意力机制^[20],在处理长文本或多轮对话^[21]时显著增加了计算量。

为了缓解推理阶段重复计算的问题,主流系统普遍针对键值缓存(KV cache)数据进行存储复用^[22],即将注意力机制中历史标记所生成的键(key)和值(value)向量暂存起来,并在后续标记生成时重复利用,从而避免重新计算已有上下文的表示。这种“以存储换计算”的思路极大提升了模型的推理效率,在解码阶段的自回归生成过程中显著降低延迟并提升吞吐率。

然而,随着大语言模型规模的不断扩张与功能需求的持续增强,键值缓存数据也为系统带来了前所未有的存储压力。第一,序列长度的增长使得键值缓存的规模随之线性扩展,部分模型支持超过 20 万标记的上下文长度^[23],这加剧了存储资源消耗。第二,现有大模型常包含几十层以上的 Transformer 结构,每层均需缓存高维向量。第三,模型服务的实际部署场景往往要求高并发支持,每个用户会话都需独立维护 1 组键值缓存,这进一步拉高了总体存储占用^[21]。第四,长期上下文机制的引入^[24-25],例如跨轮对话的连续记忆,也要求长时间存储键值缓存数据。

这些因素的叠加导致键值缓存总体规模迅速增长,对系统存储资源构成多重挑战。一方面, GPU 显存的瓶颈直接限制了推理任务的并发能力,降低了硬件资源利用率;另一方面,当键值缓存长期暂存至外存设备(如 SSD)时^[26-27],又带来访问延迟与存储成本问题。

在此背景下,研究者开始关注键值缓存压缩(KV cache compression)这一新兴方向,旨在有效减小键值缓存规模以及缓解显存和外存的压力,从而提升推理系统效率、降低存储成本。近年来,业界和学术界已提出多种键值缓存压缩方法,涵盖键值缓存合

并^[28-29]、低比特量化^[30-32]、标记弃用^[33-35]、键值缓存共享^[36-37]以及注意力头削减^[38-39]等多个维度。这些方法在压缩率、生成质量和系统性能上表现出不同的权衡特性,初步展示出降低大模型键值缓存规模的潜力。

然而,将键值缓存压缩方法应用于实际推理系统仍面临诸多关键挑战。一方面,压缩操作通常是有损的,可能导致模型回答质量下降;另一方面,压缩本身的计算开销亦不可忽视,若设计不当反而会增加系统延迟。此外,压缩方法的泛化能力仍有待提升,许多方法在任务迁移、模型切换或上下文变化时效果波动明显,难以满足通用推理服务对稳定性的要求。

综上所述,键值缓存压缩作为大语言模型推理优化的关键路径之一,其研究价值日益凸显,本文从原理机制、技术挑战与方法类别等多个角度展开系统化综述,期望为相关研究人员和系统开发者提供参考,促进键值缓存压缩方法在大语言模型推理系统中的广泛与高效应用,主要贡献有 3 个方面:

1)系统梳理大语言模型推理中大语言规模键值缓存带来的存储压力,总结了大语言模型键值缓存压缩方法设计面临的三大挑战,即如何保证模型回答质量、如何降低额外引入的计算开销、如何保证泛化性。

2)对现有键值缓存压缩方法进行分类,分为基于合并、基于量化、基于标记弃用、基于共享、基于注意力头削减的压缩方法。针对每类键值缓存压缩方法,以具体例子分析其对应的设计原理与实现方式。

3)对键值缓存压缩方法的未来研究方向进行了展望,指出未来应当着重提升跨任务与跨场景的泛化能力、减少压缩带来的额外计算开销并实现压缩方法的自动化选择与参数自适应调节,才能推动键值缓存压缩方法在更广泛的大语言模型服务场景中实现高效落地。

1 背景介绍

大语言模型在推理阶段的计算复杂度普遍较高,这不仅增加了大语言模型服务的部署成本,也显著

提升了响应延迟,对实际应用造成了不小的挑战。为了提升推理效率,现有方法普遍针对键值缓存进行暂存复用^[21-22],将历史标记在推理过程中计算得到的键和值向量暂存起来,供后续生成阶段复用。这种“以空间换时间”的方法可以有效减少重复计算,从而提升推理效率。

然而,随着大模型推理应用的持续发展,键值缓存机制带来了显著的存储资源压力。第一,输入序列长度的不断增加^[40-41]意味着需要存储更多标记的键值缓存。第二,模型参数规模的不断扩大,会导致每个标记对应的键值缓存规模大幅提升。第三,多用户并发访问成为实际部署中不可回避的需求,使得系统需同时维护多个用户会话的键值缓存,占用更多存储资源。第四,为提升模型的能力边界,越来越多的大语言模型服务商引入了长期上下文记忆机制^[24-25],以实现跨会话、多轮交互的持续记忆,这进一步拉长了键值缓存的存储时间。

这些因素的叠加效应导致键值缓存的总体规模迅速膨胀,从而给 GPU 显存和外存带来巨大压力。一方面,显存资源紧张将限制可并行推理的任务数,影响系统的吞吐量和服务能力;另一方面,键值缓存存在外存中进行存储时^[26-27]可能引发访问延迟增加与带宽瓶颈等问题,拖慢整体响应速度,针对键值缓存的长期存储还会带来巨大存储开销。

1.1 大语言模型推理中的键值缓存

当前主流的大语言模型^[1-4]普遍基于 Transformer 架构构建,该架构的核心组件是注意力(attention)机制^[20],其能够有效建模序列中各个位置之间的依赖关系。在实际推理过程中,大语言模型通常分为预填充(prefill)阶段与解码(decode)2个阶段^[42]。

在预填充阶段,模型对输入序列中所有的提示标记(prompt tokens)进行并行处理,计算得到每个位置的隐藏表示与注意力所需的键和值向量(键值缓存),并据此生成第1个输出标记。在此过程中,由于输入序列已知,计算可以并行展开,效率较高。进入解码阶段后,模型采用自回归方式逐步生成后续标记。在每一轮迭代中,模型仅接收最新生成的1个标记作为输入,并基于其构建查询(query)向量,结合所有标记的键值缓存完成注意力计算并生成下一个输出标记。由于解码阶段的计算具有强依赖性,因此其只能逐步串行进行,计算效率显著下降。

解码阶段的注意力计算依赖于2个核心要素,一是当前标记生成的查询向量和键值缓存,二是所有先前标记的键值缓存,如图1所示。

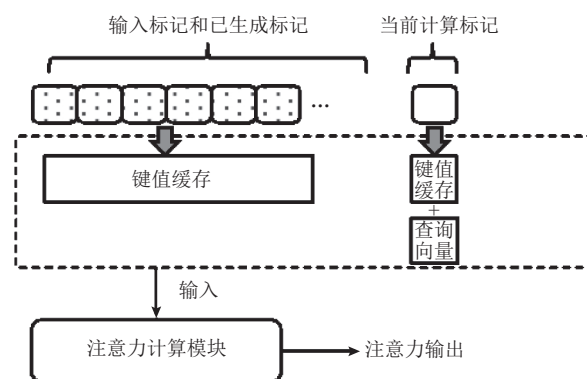


Fig. 1 The attention computing mechanism in LLM inference

图1 大语言模型推理中的注意力计算机制

1.2 键值缓存的空间开销

近年来,大语言模型推理支持的序列长度越来越长,例如 Claude 支持最长 20 万个标记的历史对话长度^[23]。由于键值缓存的空间规模和序列长度呈线性关系,键值缓存的总体规模也随之迅速上升,给显存和外存带来巨大的空间压力。

1) 显存空间压力。GPU 中当前推理请求的所有标记(输入标记和已生成标记)对应的键值缓存通常都需要常驻于 GPU 显存中,以确保解码阶段每步都能迅速访问历史上下文。由于每个标记在注意力层中均会生成维度较高的键值缓存向量,随着序列长度的增加,单条请求所占用的显存空间迅速增长。以典型的模型配置为例,对于1条包含约2000个标记的输入序列,其生成的键值缓存在多层 Transformer 网络中叠加后,可能会占用高达 5 GB 的显存资源,这对显存资源有限的 GPU 设备造成了巨大的压力。即使是目前广泛应用于大语言模型推理的高端 GPU 如 NVIDIA A100,其显存容量虽达到 40 GB 或 80 GB,在实际部署中也仅能容纳数量有限的并发推理请求。显存瓶颈由此成为限制大模型服务扩展性的重要因素之一。更值得关注的是,在推理流程中,尤其是解码阶段,计算负载相对较轻,并不构成 GPU 的主要瓶颈。相比之下,显存资源的不足成为主要限制因素。由于每个请求的键值缓存占用的显存较大,导致同时可并行计算的推理请求数量下降,从而造成 GPU 整体计算资源利用率偏低。例如,大量 GPU 计算资源可能处于空闲状态,却因显存耗尽无法调度更多任务。这种显存瓶颈导致的资源闲置现象直接限制了大语言模型推理服务系统的整体吞吐。

2) 外存空间和带宽压力。外存设备在当前的大语言模型服务系统中同样扮演着关键角色^[26-27]。诸如 ChatGPT, Claude 等主流大模型服务商均已实现对

用户长期上下文的记忆,以提供更具连贯性和个性化的对话体验。这一能力要求在每次用户请求到来时,模型不仅要处理当前输入,还需参考用户的历史对话上下文,以生成符合语境的高质量响应。为了避免每次都重复计算历史对话对应的键值缓存,需要将这些数据长期存储在外存设备中。这样,当用户发起新请求时,系统可以直接加载对应的键值缓存

数据,从而显著减少冗余计算开销、加快响应速度。图2展示了键值缓存加载过程。此外,在实际应用中,不同用户的请求往往共享某些固定的提示模板或公共上下文内容,例如统一的系统提示词(system prompts)、知识库文档^[43]中的通用知识段落等。为了提升整体推理效率,这类共享型键值缓存也被长期存储于外存中供多个请求重用,进一步降低计算冗余。

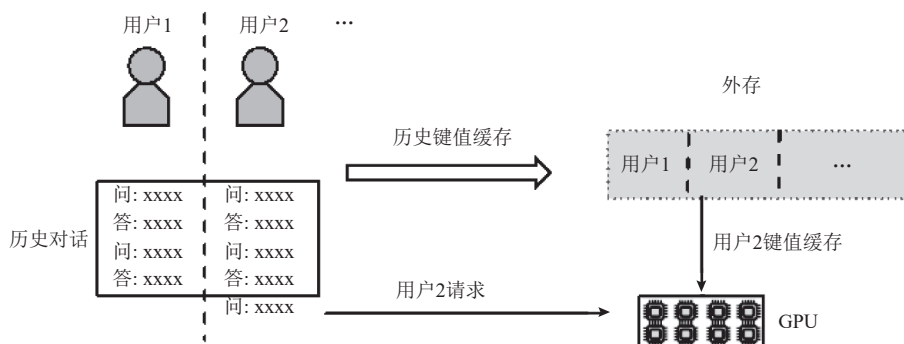


Fig. 2 The KV cache loading process when inference request arrives

图2 推理请求到来时的键值缓存加载过程

尽管目前的大模型推理服务商(如 DeepSeek, Kimi 等^[26-27])普遍采用大规模外存设备来存储海量的键值缓存数据,这种做法也带来了新的挑战:一方面,大规模键值缓存的长期保存导致服务商需投入大量的外存资源,随着用户数量增长,外存容量需求呈爆发式增长,运营成本大幅度增加。另一方面,当用户请求到来时,系统需将对应的键值缓存从外存加载至 GPU 显存,以完成后续推理任务。然而,外存的带宽较低,在加载大规模键值缓存时容易成为性能瓶颈,导致加载延迟显著提高,严重影响用户体验。

2 键值缓存压缩方法的设计挑战

为应对大规模键值缓存所带来的一系列问题,即推理并行度受限、存储成本高昂、加载延迟显著增加等问题,对键值缓存进行有效的压缩以减小其空间占用是提升大语言模型推理效率的关键途径之一。通过压缩键值缓存,可以在一定程度上缓解显存压力、降低存储成本、提高数据加载效率,从而提升大语言模型推理服务系统的总体吞吐与响应速度。

然而,设计一种高效、通用且低开销的键值缓存压缩方法依然面临一系列的技术挑战,主要体现在3个方面:1)如何在压缩后保证模型仍能生成高质量的回答;2)如何将压缩引入的计算开销控制在较低水平;3)如何保证在多种场景下的泛化性。

2.1 模型回答质量

当前已有的键值缓存压缩方法多属于有损压缩范畴,例如低比特量化^[30-31]等。这类技术在压缩过程中会不可避免地造成数值信息损失,因此无法准确还原原始的键值缓存。当这些被引入数值损失的键值缓存用于大语言模型推理的注意力计算时,由于注意力机制对输入向量的数值精度较为敏感,可能会导致注意力计算分布发生偏移,进而影响模型生成的文本质量。

这种压缩带来的语义偏移在实际应用中表现为:模型生成的回答相较于使用未压缩键值缓存时的回答存在一定差异,甚至在某些情况下可能导致语义错误、上下文脱节或事实不一致等问题。如果这种回答质量的下降超出用户可接受的范围,在对一致性和准确性要求较高的任务场景中(如医疗问答^[44]、金融分析^[45-46]、代码生成^[14-17]等)将会严重影响模型可用性。

因此,在设计键值缓存压缩方法时,不能单纯以压缩率作为唯一优化目标。除了尽可能减小压缩后数据的空间规模,还必须系统性地评估压缩对模型回答质量的影响。理想的压缩方法应在保证模型性能稳定的前提下实现有效的空间节约,从而在性能与空间开销之间取得良好平衡,保证模型的可用性。

2.2 压缩方法的计算开销

尽管针对键值缓存进行压缩在提升推理效率方

面具有显著优势,即有效减小键值缓存加载延迟、提升并行度,从而降低用户请求的响应时延,但不可忽视的是,许多压缩方法本身也会引入一定的计算开销。若压缩操作引入的计算开销过大,不仅可能抵消其所带来的性能收益,甚至可能对系统实时性造成负面影响。因此,压缩方法的计算开销是其能否实际部署的重要考量因素之一。

以量化(quantization)方法为例,此类方法通过将高精度的浮点表示(如 16 b)量化为更低比特宽度的表示(如 8 b 或 4 b),以减少数据体积。虽然这类方法在理论上可以显著节省空间,但压缩过程涉及浮点到定点的映射等操作。

另一个典型的例子是基于重要性评估的标记弃用压缩方法,即通过评估每个历史标记对当前生成标记的相对重要性,删除部分重要性较低的历史标记的键值缓存,从而减少键值缓存的总体规模。这种方法虽然在保持生成质量的同时实现了较高的压缩率,但其压缩过程需执行额外的重要性评分计算(如基于注意力得分),这一过程在高并发场景下开销较高。

2.3 压缩方法的泛化性

虽然现有键值缓存压缩方法在部分测试负载与特定任务场景中展现出良好的压缩效果与生成质量保障能力,但在更广泛的应用环境中,这些方法往往面临显著的泛化性不足问题。即,在任务类型、输入长度、模型架构或训练范式发生变化的情况下,压缩方法的性能难以稳定维持,甚至可能引发严重的模型回答质量下降,限制其在通用大模型服务系统中

的实际可用性。

Gao 等人^[47]的研究指出,当前多种主流压缩方法在长上下文场景下表现不佳。随着输入序列长度的显著增长,若压缩方法未能准确保留重要标记的键值信息,则会造成语义丢失,进而降低模型的回答质量。此外,该研究还发现,某些压缩方法在面对如智能问答这一类对精确语义定位要求较高的任务时,也容易引发模型生成内容的偏离和信息遗漏。

类似地,Liu 等人^[48]的研究进一步指出,现有压缩方法在算术推理类任务(例如 GSM8K^[49])中表现较差。压缩过程中一旦破坏关键推理路径所依赖的上下文表示,极易造成逻辑跳跃或结论错误。该研究还发现,将压缩方法应用在经过指令微调(instruction fine-tuning)^[50]后的大语言模型上时,可能导致回答质量下降最高达 25.53%,表明压缩方法对不同模型微调方法的适应能力较差。

3 典型的键值缓存压缩方法

大语言模型推理中的键值缓存压缩方法能够有效减小键值缓存规模,缓解显存和外存压力,从而降低整体存储成本并且提高推理效率。我们将键值缓存压缩方法分为基于合并的压缩方法、基于量化的压缩方法、基于标记弃用的压缩方法、基于共享的压缩方法和基于注意力头削减的压缩方法 5 类。本节将依次对这 5 类键值缓存压缩方法进行阐述和比较。表 1 对这 5 类压缩方法在压缩维度、引入的额外计算操作等方面进行了对比。

Table 1 Classification of KV Cache Compression Methods for LLM Inference

表 1 大语言模型推理中的键值缓存压缩方法分类

压缩方法	年份	压缩方法类型	压缩维度	是否改变模型架构	额外计算操作	是否支持压缩率调节	压缩率调节方式
KVMerger ^[28]	2024	基于合并	标记维度	否	相似度计算/带权合并计算	是	调节注意力分数阈值
MiniCache ^[29]	2024	基于合并	层维度	否	重参数化合并计算	是	调节参与合并的层数
KVQuant ^[30]	2024	基于量化	数据位宽	否	缩放因子计算/量化计算	是	调节量化比特数
CacheGen ^[31]	2024	基于量化	数据位宽	否	差值计算/量化计算	是	调节量化比特数
Ecco ^[32]	2025	基于量化	数据位宽	否	缩放因子计算/量化计算	是	调节量化比特数
Scissorhands ^[33]	2023	基于标记弃用	标记维度	否	注意力分数计算	是	调节弃用触发阈值
Impress ^[34]	2025	基于标记弃用	标记维度	否	注意力分数计算	是	调节标记保留比例
ClusterKV ^[35]	2025	基于标记弃用	标记维度	否	注意力分数计算	是	调节标记保留比例
KVSharer ^[36]	2024	基于共享	层维度	否	距离计算/相似度计算	是	调节共享层数
LCKV ^[37]	2024	基于共享	层维度	是	模型重新训练	是	调节共享层数
MQA ^[38]	2019	基于注意力头削减	注意力头维度	是	模型重新训练	否	
GQA ^[39]	2023	基于注意力头削减	注意力头维度	是	模型重新训练	是	调节注意力头分组数目

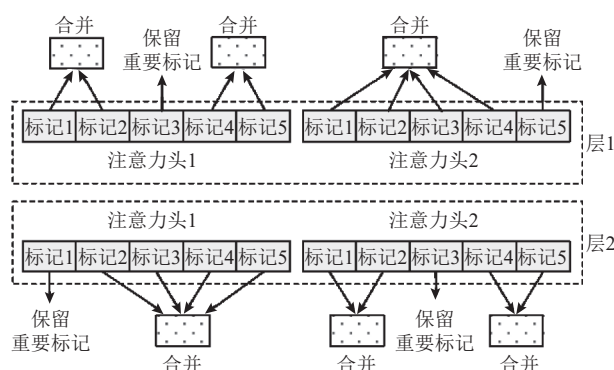
3.1 基于合并的压缩方法

基于合并的压缩方法选择若干键值缓存向量并设计合并算法对其进行合并,从而减小键值缓存的整体规模。合并对象的选取方法是该类压缩方法是否高效的关键,在选取时应当尽可能保证原始语义信息的完整,同时尽量扩大合并范围。

KVMerger^[28]由佐治亚理工大学提出,其基于标记之间的相似性进行键值缓存的合并。KVMerger基于LLaMa2-7b-chat大语言模型使用SynthWiki数据集(平均序列长度为4000)进行测试,针对同一个序列中不同标记对应的键值缓存中的键向量计算其余弦相似度,发现距离越近的标记其键向量之间的余弦相似度越高,其中某些标记的键向量之间的余弦相似度超过了90%,而距离越远的标记其键向量之间的余弦相似度越低。因此KVMerger得出结论:键向量的相似性存在显著的局部性,即相邻的标记具有更高的键向量相似性。

KVMerger进一步发现上述键向量之间的相似度和模型架构内部的不同层与注意力头紧密相关,而与数据集的选取相关性较弱。KVMerger针对每个标记计算该标记和相邻标记的键向量的余弦相似度,并按照注意力头和层进行分组,即每个组内都是同一层内同一个注意力头的键向量。经过测试发现,不同组之间的余弦相似度差异较大,最大相差50%。与此相对,针对不同的数据集TREC, TriviaQA, Lcc, NarrativeQA进行测试,发现对于同一层,键向量的相似度在不同数据集上基本保持一致。因此KVMerger得出结论:在选取合并对象时,应当按照层和注意力头分组后以组为粒度进行选取。具体选取算法:KVMerger先计算各个标记的键向量,然后按照注意力头和层进行分组。在每组内,依次计算相邻标记的余弦相似度,若该值高于设定的阈值(在实现时设置为0.75),即可归类为同一个合并批次,否则就不属于同一个合并批次。该做法利用了标记的局部性,即只有连续的标记才可以进行合并。同时该方法会选择重要标记,即包含重要语义信息且不适合进行合并的标记,这一选择过程是基于注意力分数实现的,即针对所有标记计算其注意力分数,注意力分数最高的若干个标记设置为重要标记,重要标记不参与合并。图3展示了1个合并对象选取的例子,该例子中模型的层数为2,并且含有2个注意力头。

当完成合并对象的选取后,应当针对同一组内的键值缓存进行合并,KVMerger设计了带权合并算法,通过为每个标记的键值缓存引入1个权值,然后



注:以“层”和“注意力头”为粒度选择合并对象。

Fig. 3 Merging objects selection method in KVMerger

图3 KVMerger的合并对象选取方法

进行加权合并,避免简单的平均合并算法导致的非重要标记引入噪声的问题。

具体而言,KVMerger在同一个合并批次内的所有标记中选取1个中枢标记,中枢标记的选取通过计算聚合注意力分数实现,然后利用高斯核计算其余标记的权重,键向量的分布离中枢标记越近的标记具有越高的权重,从而尽可能地减小距离较远的标记引入的噪声。

最终,KVMerger通过优化合并对象的选取策略和重要标记的识别,在提升压缩率的同时避免原始语义的损失,实现高效的压缩。

由浙江大学与莫纳什大学联合提出的MiniCache^[29],也是一种基于合并的键值缓存压缩方法。该方法通过挖掘模型不同层级之间的表示冗余性,设计出一种高效的键值缓存合并方法,在保证模型回答质量的前提下有效降低键值缓存的空间开销。

MiniCache的设计核心建立在以下关键观察之上:大语言模型不同层之间的键值缓存存在显著的相似性,且该相似性在中层至深层更为显著。为验证这个假设,MiniCache基于LLaMa3-70B模型,分别在多个自然语言任务数据集COQA, GSM8K, TruthfulQA上进行实证分析,评估推理过程中序列同一个位置在相邻层之间的键值缓存的余弦相似度。实验结果表明,在浅层(第1~10层),键值缓存相似度相对较低,其中COQA数据集上相似度普遍低于0.8, GSM8K数据集上低于0.65, TruthfulQA数据集上低于0.55;而在中层至深层(第20层以后),3个数据集的相似度普遍提升至0.8~0.9,呈现出高度一致的表示趋势。该实验结果表明,在推理阶段,大语言模型后续层之间的键值缓存表示高度冗余。这个观察为进行跨层合并压缩提供了理论支持。

在Transformer模型中,标记(token)是文本序列

中的最小输入单元,通常对应一个词或子词。在推理阶段,模型会为每个标记生成独立的查询、键和值向量,用于注意力计算。不同标记所承载的语义作用并不均等,例如,实体词、动词、条件词或逻辑连接词往往决定句子结构与语义流向。

尽管层级之间键值缓存整体表现出较强的相似性,MiniCache 进一步指出,并非所有的标记都适合参与合并操作。部分标记由于携带丰富语义信息,或位于语言结构关键位置,其在不同层间的键值缓存的差异较为显著,若强行合并可能造成严重的语义信息损失,进而影响模型的回答质量。为分析此现象,Reddy 等人^[51]基于 LLaMa2-7B 模型在 COQA 数据集上对键值缓存进行了标记粒度的相邻层相似性分析。结果表明,大部分标记在相邻层之间的键值缓存余弦相似度高于 0.6。但存在少数标记,其相似度低至 $-0.6\sim 0.2$,表现出典型的离群者行为(outlier behavior)。这类标记通常与长距离依赖关系、实体引用、条件控制等语义关键因素相关。根据这个发现,MiniCache 提出这些离群标记(outlier tokens)的键值缓存应跳过合并操作,需要通过对其进行单独保留以维护语义完整性。

MiniCache 的具体压缩方法包括 3 个技术模块。

1)合并起始层的选择。考虑到模型浅层键值缓存的跨层相似性较低,MiniCache 仅在中层至深层进行键值缓存合并。具体策略为:以模型总层数的一半作为压缩起始层,从该层开始至输出层的所有键值缓存均参与合并。

2)重参数化(reparameterization)合并机制。为了压缩存储空间,MiniCache 对每个合并的键值缓存向量仅保留其方向向量(directional vector),并通过重参数化^[52]方式补充 2 个轻量级标量信息,一个表示合并组的平均幅度(magnitude),另一个表示相邻层间的夹角角度(angle)。通过上述三元组表示方式,在保持精度的同时,有效减小了键值缓存的总体规模。

3)基于角度距离的异常标记识别与保留策略。MiniCache 引入角度距离(angular distance)作为标记间表示差异的衡量指标,从而判定某标记是否具备合并条件。当标记在相邻层之间的键值缓存的角度距离超过设定阈值时,即视为表示发生剧烈变化,该标记的键值缓存将被完整保留且跳过压缩流程,以避免语义退化。总体合并方法如图 4 所示。

在多个主流数据集和大语言模型上,MiniCache 在保证模型回答质量基本没有下降的前提下,针对键值缓存实现了 30%~50% 的压缩率。MiniCache 展

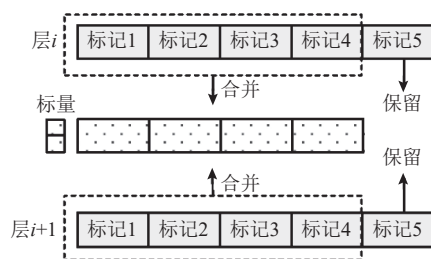


Fig. 4 KV cache merging method in MiniCache

图 4 MiniCache 的键值缓存合并方法

示了跨层冗余压缩的巨大潜力,为未来低成本大模型部署提供了切实可行的技术路径。

3.2 基于量化的压缩方法

基于量化的压缩方法使用更少的比特数(例如 4 b)来存储原本使用较多比特数(16 b)表示的键值缓存,从而缩小键值缓存的总体规模。

Hooper 等人^[30]提出的 KVQuant 是一种典型的基于量化的键值缓存压缩方法,旨在通过将键值缓存压缩为不超过 4 b 的数据来显著降低其空间开销。KVQuant 的核心创新在于其量化粒度的设计,具体而言,它采用了基于通道粒度的量化方法,而非传统的按标记粒度进行量化。这个方法的成功依赖于对键向量特性深刻的观察和分析。

在 Transformer 的注意力机制中,每个键向量通常是 1 个高维向量,例如维度为 5120。这些维度可被视为多个通道(channels),每个通道代表一种特征维度,用于编码输入序列中不同的语义或上下文关系。换言之,通道可理解为键值向量在特征空间上的独立维度,它们在不同位置的标记间共享语义空间,但数值分布与尺度可能存在差异。

在传统的量化方法中,量化操作通常是按照每个标记的粒度来计算量化时的缩放因子,而 KVQuant 则为每个通道单独计算量化缩放因子。这个设计的动机来源于以下观察:键向量的不同通道之间,幅度存在显著差异,并且部分通道的键向量呈现出离群者现象。而在同一个通道内,不同标记的键向量的幅度变化较小,表现出较强的相似性。因此, KVQuant 将幅度相似的键向量归为一组,统一计算缩放因子;而对于幅度差异较大的键向量,分别计算其缩放因子,从而提高量化的精度与效率。

需要注意的是,针对值向量,上述观察并不成立。由于值向量的表现与键向量不同, KVQuant 仍然采用传统的按标记粒度量化的方式进行处理。

KVQuant 的重要创新策略是在执行旋转位置编码^[53]之前进行键值缓存量化。旋转位置编码是当前

大多数主流大语言模型中常见的处理方式, 它将每个标记的位置信息与其对应的查询/键向量结合。然而, KVQuant 发现, 在应用旋转位置编码后, 同一通道的键向量幅度差异会显著增大。这是因为不同标记在序列中的位置不同, 对其进行旋转位置编码会产生较大的幅度变化。为了解决这一问题, KVQuant 选择在键向量执行旋转位置编码之前先进行量化, 从而避免不必要的幅度波动。

由于大部分键值缓存的数值分布在较为狭窄的区间内, 量化操作带来的精度损失较小, 但对于少数离群点(outliers), 即数值分布在较宽区间的键值缓存, 量化可能会带来显著的精度损失。为此, KVQuant 提出了离群点检测机制。具体而言, KVQuant 动态设置了一个离群点阈值, 对于超出此阈值的键值缓存不执行量化操作, 从而避免精度损失。键向量基于通道粒度进行离群点判定, 而值向量基于标记粒度进行离群点判定, 从而确保每个向量在其相应粒度上进行最合适的量化操作。

KVQuant 进一步提出了一个针对序列中第 1 个标记的特殊处理机制。通常情况下即使序列中的第 1 个标记在语义上的重要性较低, 大模型仍会为该标记分配较高的注意力分数^[54]。为避免对模型回答质量造成不必要的影响, KVQuant 特意保留第 1 个标记的原始键值缓存, 不进行量化处理, 以防止模型对该标记的高敏感度影响后续推理的准确性。

在 KVQuant 中, 针对键向量和值向量的量化采用了不同的缩放因子计算与更新策略。特别地, 由于键向量的缩放因子是基于通道粒度进行计算的, KVQuant 发现在推理中每当生成新的标记后, 所有通道上的键向量分布会发生变化, 因此所有通道的缩放因子都需要重新计算。如果每次推理都重新计算这些缩放因子, 势必会导致计算效率严重下降。为了解决这个问题, KVQuant 针对键向量采用了离线计算的方式, 即在初始阶段通过固定的校准数据集来预先计算键向量的缩放因子。相对地, 由于新标记的引入并不会改变已经处理过的标记的值向量分布, KVQuant 采用在线的方式计算值向量的缩放因子, 并且能够保证相对较低的缩放因子更新频率。

KVQuant 通过创新性的量化策略实现了极高的压缩比, 在多个任务和数据集上显著提高了大语言模型的推理效率, 同时保持了高质量的推理结果, 为大规模语言模型的高效部署提供了有力的技术支持。

Liu 等人^[31]的 CacheGen 是一种将差分编码(delta encoding)与非均匀量化(non-uniform quantization)策

略相结合的键值缓存压缩方法。该方法通过对不同标记之间键值缓存的数值差值进行压缩, 并在此基础上引入层感知(layer-aware)的比特分配与熵编码优化方法, 在保持模型回答质量的同时显著提升压缩率。

CacheGen 的设计核心基于以下关键观察: 大语言模型中, 相邻标记的键值缓存在数值上高度相似。为验证该假设, Liu 等人^[31]基于 LLaMa-7B 与 LLaMa-13B 模型, 在 LongChat 数据集上进行了广泛实验, 对比分析了所有键值缓存的原始数值分布和在同一层与同一个通道中相邻标记之间键值缓存差值的分布。实验表明, 键值缓存的差值的值域更窄、波动更小。因此, 若对键值缓存差值进行量化, 而非直接量化原始值, 在同等压缩率下可以获得更高的数值精度, 从而提升模型性能。

为进一步提高解压效率, CacheGen 并未针对相邻标记逐个计算差值, 而是将序列中每连续 10 个标记划分为 1 组, 选取组内第 1 个标记作为锚点标记(anchor token), 并对其余标记相对于该锚点标记的键值缓存差值进行量化。这种设计允许对 1 组标记进行并行解压, 从而显著提升解码速度。

CacheGen 的创新还在于根据模型不同层对量化误差的容忍度差异采用层感知的非均匀比特分配策略。实验发现, 大语言模型对不同层的键值缓存精度有不同的敏感性。在浅层执行量化时, 模型的回答准确率下降至 0.4 以下; 在中层(第 4~7 层)量化后, 模型的回答准确率约为 0.7; 而在深层量化时, 模型的回答准确率仍保持在 0.9 以上。这是因为浅层的键值缓存编码了模型对输入信息的低层语义特征, 误差的积累会被层层放大, 从而对最终输出造成严重影响; 而深层则更多提取抽象语义与高阶知识, 对数值精度的依赖相对较低。基于这个观察, CacheGen 将模型分层如下: 浅层(前 1/3 层)采用 8 b 精度量化压缩; 中层(中间 1/3 层)采用 4 b 量化压缩; 深层(后 1/3 层)采用 3 b 量化压缩。该策略在显著提高整体压缩率的同时, 有效避免了模型回答质量下降。图 5 展示了 CacheGen 的整体压缩流程。此外, 为保证差分编码过程中锚点标记的精度, CacheGen 对所有锚点标记的键值缓存统一采用 8 b 量化。由于锚点标记需参与多次差值运算, 其误差会被放大传播, 因此使用更高比特精度可以显著缓解累积误差带来的精度损失。

在完成差分化量化后, CacheGen 进一步引入熵编码(entropy coding)技术以压缩量化后的数值。特别

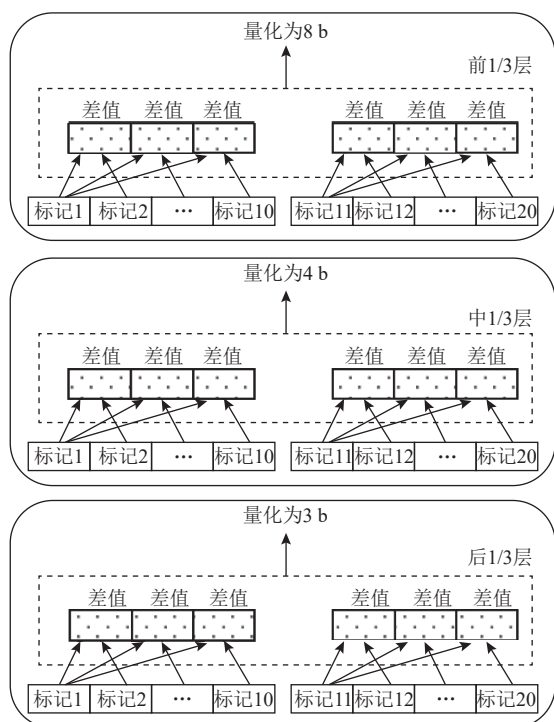


Fig. 5 Compression method of CacheGen based on quantization of delta value

图5 CacheGen中基于差值量化的压缩方法

地, CacheGen 对量化值按照层与通道 2 个维度进行分组, 利用同层同通道数据的统计规律进行算术编码(arithmetic coding)^[55]。CacheGen 实证分析了不同分组粒度下的键值缓存信息熵变化, 以 LLaMa-13B 为例, 不分组时, 信息熵约为 5.3; 按标记分组时, 信息熵基本无明显变化; 按通道分组时, 信息熵降至约 5.2; 进一步按层分组后, 信息熵可降至约 5.1。这表明, 通道与层在键值缓存的数值分布结构上存在显著的统计相关性, 利用这些结构进行组内熵编码可有效提升编码效率, 从而进一步减少存储开销。

Cheng 等人^[32]提出的 Ecco 基于动态量化进行高效键值缓存压缩。其核心思想是通过键向量与值向量进行混合粒度的动态量化, 在显著降低存储与传输开销的同时, 最大限度地保持推理精度。

Ecco 主要设计的出发点是观察到键向量和值向量在不同层和不同注意力头之间存在显著的数值分布差异, 而在同一个语义上下文内, 其数值动态范围通常有限。基于这个特性, Ecco 引入了分层感知与通道自适应的量化策略。具体而言, 系统首先对每一层的键向量和值向量进行统计分析, 获得其动态范围分布。随后, 依据不同层与注意力头的特征, 对键向量和值向量采用混合精度量化(如 8 b, 4 b 混合)进行编码, 以在压缩率与精度之间取得平衡。

为进一步降低量化误差, Ecco 还引入了可学习的量化参数优化机制, 在离线阶段学习每层的最优缩放系数与偏移量, 从而减少低比特量化带来的数值偏差。

总体而言, Ecco 通过将传统静态量化方法扩展为结构感知、动态自适应的压缩机制, 在保证模型生成质量的同时, 显著降低了键值缓存的空间开销。

3.3 基于标记弃用的压缩方法

基于标记弃用的键值缓存压缩方法通过识别并弃用重要性较低的标记对应的键值缓存, 有效地减少了键值缓存的整体空间规模。该方法的核心思想是对模型在推理过程中不同标记的被关注度或重要性进行动态评估, 从而判断哪些标记的键值缓存对于最终模型输出的贡献较小。这些低重要性的标记对应的键值缓存可以被排除, 进而大幅减轻存储负担。

Liu 等人^[33]提出的 Scissorhands 是一种典型的基于标记弃用的压缩方法, 旨在通过识别和弃用对模型推理贡献较小的标记的键值缓存来有效减小键值缓存的存储需求, 并且保证高质量的模型回答。其核心思想是利用标记间的注意力分数来评估其重要性, 并通过对标记间重要性的持续预测优化键值缓存管理。

Scissorhands 首先提出了一个重要的观察: 某些特定标记对不同位置的多个后续标记具有较高的重要性。这种现象可以通过注意力机制中的注意力分数来度量。Scissorhands 使用 OPT-6B 模型在 C4 数据集上进行测试, 通过计算每个标记之前所有标记对该标记的注意力分数, 发现某些特定标记对多个后续标记的重要性较为显著。特别地, 模型中位置分别为第 22, 63, 98, 121, 152, 177 的标记, 均与 3 个随机挑选的标记具有较高的注意力分数。这个现象在模型的不同层和不同数据集上均有出现。由此, Scissorhands 得出结论: 尽管每个标记关注的标记略有不同, 但整体上一些特定标记在多个后续标记中始终保持较高的重要性。

基于上述观察, Scissorhands 提出了一种创新的策略: 通过分析各标记对已生成标记的重要性来预测其对未来标记的重要性。尽管大语言模型遵循自回归生成的范式, 即每次只生成 1 个标记, 且无法在生成过程中直接计算未来标记的注意力分数, Scissorhands 仍能通过已生成标记的注意力信息推测哪些标记对未来标记具有较高的重要性。Scissorhands 进一步通过一系列实验验证了这个重要的假设: 对于现有标记而言, 重要性较高的标记往往对未来的标

记也具有较高的重要性。为此, Scissorhands 在多个数据集上 (OpenBookQA 和 WikiText) 对不同规模的模型 (OPT-6B, 13B, 30B, 66B) 进行了测试。实验表明, 通过把序列前半部分的标记与后半部分的标记设置为 2 组, 计算它们之间的注意力分数, 可以有效地识别出哪些标记分别在这 2 组扮演关键角色。具体而言, 第 1 组中的关键标记能够较为准确地覆盖第 2 组中的关键标记。因此, Scissorhands 提出基于当前标记的重要性来预测未来标记重要性, 从而弃用低重要性标记。

在实现过程中, Scissorhands 设定了一个存储开销上限, 当键值缓存的规模超过该上限时, 会启动弃用过程。弃用的核心步骤包括: 首先计算每个标记对已生成标记的注意力分数, 筛选出注意力分数高于设定阈值的标记, 并将这些标记视为关键标记。随后, 弃用非关键标记的键值缓存。为了避免弃用过程中可能带来的语义信息丢失, Scissorhands 特别保留了最近生成的 10 个标记的缓存, 这是因为计算这些标记的重要性时样本较少, 弃用它们可能会导致模型语义信息的丢失, 从而影响推理的质量。

Chen 等人^[34]提出的 Impress 是一种高效的基于标记弃用的键值缓存压缩方法, 其主要贡献在于显著降低标记重要性评估所需的加载和计算开销。Impress 的核心思想在于: 利用注意力头之间对标记重要性判断结果的高度相似性, 使用少量注意力头的键值缓存即可近似评估每个标记的重要性, 从而避免全量加载和计算。

在大语言模型推理中, 尤其是面向多请求场景的预填充阶段, 共享前缀序列的键值缓存会带来巨大的存储与访问压力。Impress 提出一个关键观察: 不同请求之间即使共享相同前缀, 这些请求对同一个前缀中标记重要性评估的结果仍存在显著差异。这个结论与 Scissorhands 所主张的标记重要性一致的假设存在明显分歧。为验证该结论, Impress 基于 OPT-6B 和 OPT-30B 模型在 RTE 和 SQuAD 这 2 个数据集上进行了大规模实证测试。实验显示, 若使用静态方式预设一组重要标记并弃用其余标记, 在不同请求上会导致平均 5.0% 与 3.3% 的模型回答准确率下降。这个结果表明, 仅依赖静态的重要标记选择方式会严重影响模型回答质量, 因此每个请求都应进行针对性的标记重要性评估。

然而, 这种按需计算策略也带来了新的挑战: 必须为每个请求重新加载前缀序列中所有标记的键向量并执行注意力计算, 从而显著增加了加载和计算

开销。为此, Impress 提出了一种高效替代方案, 该方案基于一个关键观察: 在同一层中不同注意力头评估出的标记重要性高度一致。该观察为“稀疏采样”提供了理论依据。为了验证这个观察, Impress 使用 OPT-6B 模型开展实验, 分别使用各注意力头对前缀标记进行注意力打分, 并提取其中得分最高的一部分标记作为重要标记集合。随后, 使用雅卡尔相似 (Jaccard similarity) 系数对注意力头之间的标记集合进行两两比较。结果显示, 在大多数情况下, 这些集合之间的平均相似度超过 0.95, 说明使用少量注意力头 (即探测头) 可以很好地近似估计整体注意力趋势。

进一步地, Impress 验证了这个结论对不同模型规模与重要标记采样率的鲁棒性。在 OPT-6B, OPT-13B, OPT-30B 这 3 种模型上, Impress 分别提取注意力分数最高的前 10% 与前 40% 的标记作为重要标记集合进行相似度评估。结果显示, 采样率越高, 头间一致性越强, 表明稀疏采样策略在高采样率设定下尤为有效。

基于上述分析, Impress 提出以下高效的标记重要性估计流程: 在每一层选择少量 (例如 3 个) 具有代表性的注意力头, 称为探测头 (probe heads), 用于进行重要性评估。仅加载这些探测头所需的键向量至 GPU 中并进行注意力分数计算, 从而极大减少数据加载量与计算量。然后根据注意力分数选出小部分重要标记, 最终仅加载这些重要标记的完整键值缓存用于推理, 从而避免无效数据的读取与计算, 如图 6 所示。

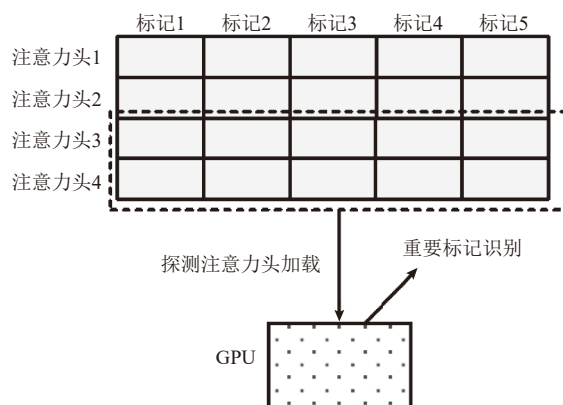


Fig. 6 Mechanism of loading small-scale probe attention heads in Impress

图6 Impress 的小规模探测注意力头加载机制

这个流程显著缓解了传统策略中的带宽瓶颈与加载延迟问题, 同时保证了推理精度。在实践中, 键

值缓存通常以固定大小的子块为单位进行加载,若1个子块同时包含了重要与非重要标记的键值缓存,即使只需其中一部分,也必须整体加载,从而导致严重的读放大(read amplification)问题。

为此, Impress 引入了一种基于重要性驱动的标记重分块机制。具体而言, Impress 动态调整每个子块的边界,使得重要标记尽可能被聚集在同一个或少数几个子块内,这样可大幅降低无用数据的加载比例,缓解读放大问题。

Liu 等人^[35]提出的 ClusterKV 是一种面向长文本推理任务的高效键值缓存压缩方案,其核心思想同样基于标记弃用机制,但在重要性评估粒度与计算方式上进行了创新设计。与传统弃用标记时计算注意力分数的方法不同, ClusterKV 将标记划分为多个语义集群,并以集群为单位评估整体重要性,从而显著降低了缓存压缩过程中标记重要性计算开销。

ClusterKV 的设计基于一个关键观察:在语义空间中位置相近的标记,其对应的注意力分数往往也较为相似。基于这个观察, ClusterKV 首先提取每个标记的键向量,并利用 K-means 聚类算法对这些键向量进行聚类,以此形成若干个具有语义相似性的标记集群。在聚类过程中,系统采用余弦相似度作为向量间的距离度量,以更准确地刻画标记在语义空间中的相对关系。

在完成聚类后, ClusterKV 在注意力分数计算阶段不再针对每个标记逐个计算,而是仅选取每个语义集群的质心代表整个集群进行计算。随后,系统依据质心的注意力得分判断该集群中所有标记的整体重要性,并决定是否将该集群对应的键值缓存内容进行保留或弃用。通过这种方式, ClusterKV 将传统弃用标记的计算过程转化为集群级别的近似评估,有效降低了计算复杂度。

总体而言, ClusterKV 的核心优势在于:充分利用标记间的语义相关性,将计算粒度从单个标记优化至语义集群,在保证精度的同时显著提升键值缓存压缩效率。

3.4 基于共享的压缩方法

基于共享的键值缓存压缩方法通过多个键值缓存向量的共享实现总体键值缓存规模的削减,其核心思想类似基于合并的压缩方法。不同点在于对于多个键值缓存向量不再计算合并得到的新的键值缓存向量,而是选取其中某个键值缓存向量进行共享,丢弃其余的键值缓存向量。

Yang 等人^[36]提出的 KVSharer 是一种典型的基

于跨层共享机制的键值缓存压缩方法。该方法的核心思想是利用大语言模型内部层与层之间的结构冗余设计层间键值缓存共享机制,从而显著减少键值缓存的空间开销,并且保证回答质量。

传统的大语言模型推理流程需要为每一层独立维护其对应的键值缓存,这在模型深度较大或输入序列较长时会带来巨大的空间开销。KVSharer 的设计基于一个关键的、经验驱动的反直觉观察:2个层之间的键值缓存差异较大时,将其中一层的键值缓存共享至另一层使用,并不会显著影响模型的回答质量。这个发现颠覆了此前研究中常见的假设,即应尽量在“相似”层之间进行键值缓存共享。KVSharer 实验证明:层间键值缓存差异越大,反而越可能具有共享潜力。这为跨层共享带来了新的设计范式,即针对键值缓存差异大的层进行共享,如图7所示。

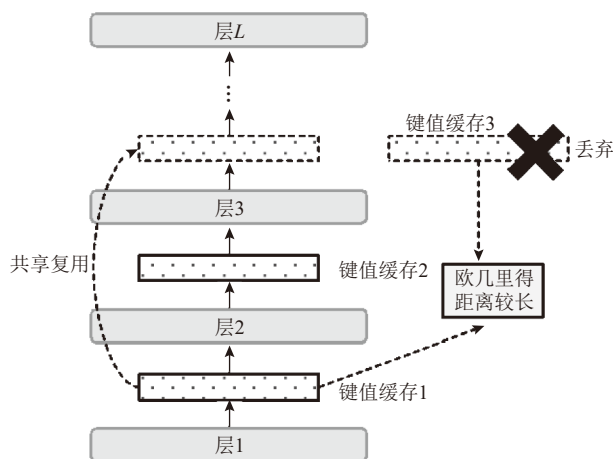


Fig. 7 KV cache sharing mechanism among layers in KVSharer

图7 KVSharer 的层间键值缓存共享机制

为系统性地确定模型中哪些层可进行键值缓存共享, KVSharer 设计了一套基于统计分析与输出保真性校验的层选择流程,具体分为以下步骤:

KVSharer 首先在一个校准数据集 (calibration dataset) 上对大语言模型执行一次完整推理过程,并在每个输入序列上保存模型各层生成的键值缓存向量。随后,对于所有样本, KVSharer 对每层的键值缓存在各序列上取平均,构造出每一层的“平均键值缓存表示”,作为该层键值缓存在语义空间中的典型分布。

进一步, KVSharer 对模型中的所有层进行两两组合得到各个层对 (layer pairs), 并计算每一层对的平均键值缓存向量之间的欧几里得距离 (Euclidean distance), 以量化其“差异性”。所有层对按距离从大到小进行排序,排在前列的层对组合具有最大的向量差异度,理论上共享潜力更高。

KVSharer 依序遍历上述排序后的层对列表,并尝试在每一层对中进行键值缓存共享。为了在最大限度压缩存储开销的同时减小模型性能损失, KVSharer 遵循以下设计原则: 优先将浅层键值缓存共享给深层使用。这是因为浅层更靠近输入, 具有更强的语义保真度, 对模型输出的影响更大; 而深层则更具抽象性, 冗余性更高, 对键值缓存精度的依赖相对较低。共享后验证输出一致性: 对于每一个层对, 在执行缓存共享后, KVSharer 会重新在校准数据集上进行一次推理, 并将共享键值缓存后模型的输出向量与未共享情况下的原始输出向量计算余弦相似度。如果相似度高于预设阈值, 说明在该层对之间共享键值缓存未对模型回答造成显著扰动, 保留该共享方法; 否则, 判定为不安全共享, 回退该层对的键值缓存共享设置。

KVSharer 会持续尝试共享操作, 直到共享的层数达到其预设上限(例如设定最多共享 6 个层对), 此时停止进一步的层对共享尝试, 并输出最终的层间键值缓存共享映射表作为压缩方法的结果。该映射可用于后续批量推理中共享键值缓存数据, 显著减少存储开销和加载开销。

Wu 等人^[37]提出的提出了一种更加激进的跨层键值缓存共享方法 LCKV, 其核心思想在于统一使用模型最深层生成的键值缓存替代其余所有层的键值缓存数据, 以最大限度减少键值缓存的存储开销。

LCKV 的方法建立在一个关键假设之上: 大语言模型最深层的键值缓存承载的信息更为浓缩、抽象和全局, 相较浅层或中间层具有更强的代表性。基于这个观察, LCKV 提出在整个推理过程中, 仅保留最深层(即最后 1 层)生成的键值缓存, 并将其共享用于中间层与浅层的注意力计算。具体地, 模型在执行每一层的注意力操作时, 使用该层生成的查询向量与最深层的键向量和值向量进行计算, 从而避免存储浅层和中间层的键值缓存并减少推理阶段的计算操作(省去了这些层的键值缓存计算)。

尽管该共享方法在存储效率上具有显著优势, 但也引入了一个重要的计算依赖问题: 在每轮生成过程中, 浅层注意力计算依赖于当前标记在最深层的键值缓存, 而该键值缓存尚未被计算完成, 导致计算时序上出现“前层等待后层”的死锁问题。为应对此问题, LCKV 引入了一种掩码规避机制: 在注意力操作中显式地屏蔽每个标记的查询向量与其自身位置上的最深层键值向量之间的注意力计算, 从而消除对该标记最深层键值缓存的直接依赖。实验发现,

即使跳过上述操作, 模型的语义理解与回答能力依然保持稳定, 回答质量并未出现显著下降。

尽管上述策略显著提高了键值缓存数据的压缩率, 在一些复杂任务上仍存在回答质量下降的问题。为进一步缓解这个问题, LCKV 引入了一种结构性增强方法——“三明治式”架构。该结构将整个模型的层级划分为 2 部分: 预热层(warm-up layers)包括最前面几层(如前 2~3 层)和最后几层(如后 2~3 层), 针对预热层保留传统的全量键值缓存并执行标准注意力计算。共享层位于模型中间层的若干层, 完全丢弃其自身的键值缓存。图 8 展示了上述的“三明治式”架构。这种“三明治式”结构在保证压缩率的同时, 显著提升了推理过程的稳定性与最终回答的准确性。

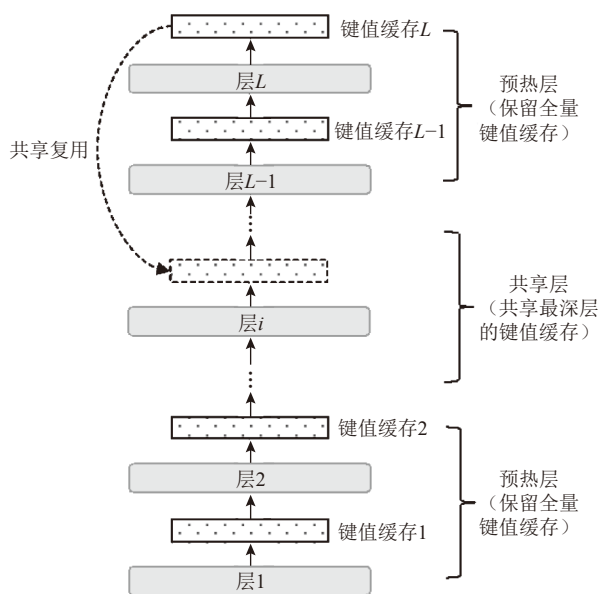


Fig. 8 Sandwich architecture of KV cache sharing mechanism among layers in LCKV

图 8 LCKV 层间键值缓存共享机制的三明治结构

3.5 基于注意力头削减的压缩方法

基于注意力头削减的键值缓存压缩方法是一类在注意力头维度上进行显式裁剪的技术, 旨在通过减少参与推理过程的注意力头数量来有效降低键值缓存的整体存储需求。该类方法与基于合并的压缩方法在压缩机制上具有一定相似性, 然而两者之间存在一个关键区别: 基于注意力头削减的方法需要对模型结构进行改动, 并依赖重新训练以保持模型性能稳定; 而基于合并的方法往往在保持模型结构不变的前提下进行键值缓存的融合。

为有效缓解大语言模型在推理过程中因多头注意力机制所导致的键值缓存规模膨胀问题, 谷歌公司提出了多查询注意力(multi-query attention, MQA)

机制^[38]。该机制通过结构性地合并键向量和值向量,实现了对键值缓存的大幅压缩,同时显著降低了模型在推理阶段的计算与存储成本,成为键值缓存压缩技术中的重要代表。

传统的大语言模型普遍采用多头注意力(multi-head attention, MHA)机制,即每个注意力头独立计算查询、键和值向量。虽然这种机制增强了模型的表达能力,提升了对复杂语义关系的建模效果,但在推理阶段也带来了巨大的计算与存储负担。每个标记都需为每个注意力头保存其对应的键值缓存,这使得键值缓存的空间开销随注意力头数量增长而线性增长。

为解决上述问题,MQA提出一种结构化的注意力头压缩方法。该机制在保留多个独立查询向量的同时仅保留1组共享的键向量和值向量。也就是虽然模型仍以多头方式生成查询向量以保持表示的多样性,但所有查询均统一与同一组键向量和值向量进行注意力计算。图9展示了上述MQA架构。这种设计将注意力头与键值缓存之间的强绑定关系解耦,使键值缓存的规模不再随着注意力头数量的增加而增加,从而实现了键值缓存空间的大幅压缩。

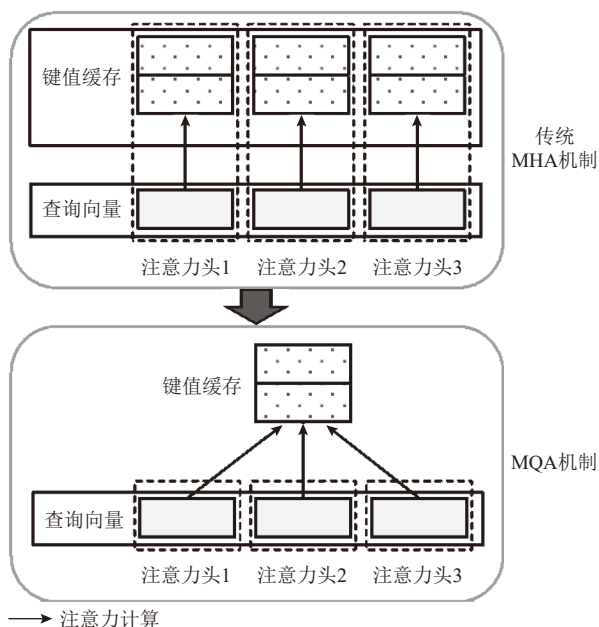


Fig. 9 MQA mechanism reduces a large number of key value caches in the attention head dimension compared with the traditional MHA mechanism

图9 MQA机制相比于传统MHA机制在注意力头维度削减大量的键值缓存

由于MQA合并了键向量和值向量,注意力模块的参数总量有所减少。为了避免模型参数规模下降

带来的性能损失,MQA在模型的前馈神经网络模块中引入了参数补偿策略,即通过扩展中间维度来提升模型的表示能力:对于一个典型的包含4096维的前馈网络层,将其升至5440维,从而增强模型回答质量。这个调整能够在压缩键值缓存的同时减小模型回答质量的下降。

实验表明,在机器翻译等语言生成任务中,采取MQA机制的模型既能够大幅压缩键值缓存,又基本保持了模型的输出质量,例如在BLEU指标上的下降幅度极小,几乎可以忽略不计。更重要的是,MQA带来的键值缓存压缩率远高于一般共享方法,其结构简单、实现直接,适用于高效推理部署。

MQA的优势不仅体现在存储压缩上,更体现在推理效率的提升与工程部署的可行性上。与传统的注意力压缩方法(如Scissorhands或Impress)不同,MQA不依赖于复杂的标记重要性分析机制,而是在模型架构层面直接减少键值缓存的维度开销,这种方式对存储空间和带宽压力的缓解更为彻底。

为在键值缓存压缩率与模型回答质量之间实现更灵活、精细的权衡,谷歌公司进一步提出了分组查询注意力(grouped-query attention, GQA)机制^[39]。该机制是在多查询注意力机制基础上的结构性改进,通过对注意力头进行分组并在组内共享键值缓存,实现压缩效果与回答质量之间的更优平衡。

传统的多头注意力机制虽然具备强大的表达能力,但每个注意力头都需独立计算并存储其对应的键向量和值向量,导致键值缓存随模型注意力头数目的增加迅速膨胀。而MQA通过仅保留1组共享键值缓存实现了极高的压缩率,但也不可避免地牺牲了部分模型表达能力和回答精度。为进一步在效率与性能之间寻找中间解,GQA机制应运而生。

GQA的核心思想是:将原有的多个查询注意力头划分为若干组,每组内的查询注意力头共享1组键向量和值向量。图10展示了GQA架构。通过控制分组数量,可以在MHA与MQA之间构建一个连续的压缩-性能谱系。当分组数量为1时,GQA完全等价于MQA,达到最大压缩率;当分组数量等于原始注意力头数量时,GQA退化为MHA,保留最高表示能力;介于两者之间的设置则提供了权衡压缩率与回答质量的弹性空间。

该机制从架构层面对注意力模块进行了重构,因此需要重新训练模型以适配新的权重分布与信息流结构。考虑到从头训练一个采用GQA结构的大语言模型成本高昂,GQA设计中引入了一种高效的

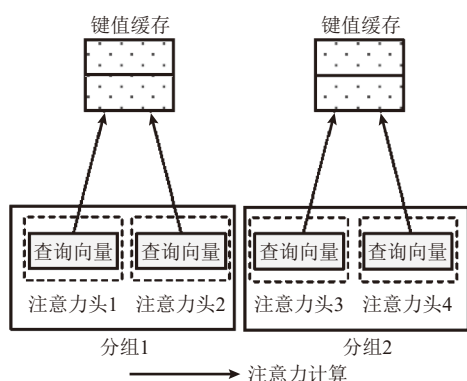


Fig. 10 GQA mechanism divides query attention heads into several groups and sharing KV cache within a group

图 10 GQA 机制针对查询注意力头进行分组并在组内进行键值缓存共享

迁移学习策略。具体做法是:在已有的 MHA 模型权重基础上,针对每组查询头所对应的键向量和值向量的模型权重进行平均聚合,从而构造出每组共享的键向量和值向量权重。以此生成的新权重构成了 GQA 结构下的模型初始检查点。为了进一步优化模型性能,研究人员仅使用原始训练数据中的一个较小子集(实践中约为原始的 5%)对新模型进行训练,从而在显著降低训练成本的同时快速恢复并接近原始模型的回答质量。这种策略充分展现了 GQA 在工程实践中的可行性和高效性。

实验表明, GQA 在大幅度压缩键值缓存规模的同时,几乎不影响模型在实际任务中的表现。例如,在 CNN/Daily Mail 数据集上,采用 GQA 架构的模型在保持可观压缩率的同时,仅在回答质量上出现不到 1 个百分点的轻微下降(性能损失约为 0.68%)。这种性能表现展现了 GQA 在精度可控压缩场景中的独特优势。

4 各种压缩方法比较

基于合并、量化、标记弃用、共享以及注意力头削减的键值缓存压缩方法从不同维度降低键值缓存的规模,形成了较为系统的技术体系。这些方法在压缩原理、实现复杂度与适用场景方面各具特点,这 5 类方法的主要优势与潜在局限总结如下:

1) 基于合并的压缩方法的优势在于能够挖掘标记或层间的冗余性,无需修改模型结构,可通过相似度或角度阈值灵活调节压缩率。缺陷在于需要计算相似度,存在额外计算开销,并且合并对象或阈值设置不当可能引入较大误差。

2) 基于量化的压缩方法的优势在于可直接减少每个键值向量的位宽,压缩率较高,并可灵活调整量化比特数以平衡精度和压缩效果。缺陷在于不同层或通道需要独立计算缩放因子,并且有些方法需要进行动态量化,实现复杂度较高。

3) 基于标记弃用的压缩方法的优势在于可动态适应不同上下文长度的任务,并且探测头采样和聚类机制的引入能够显著降低注意力分数计算开销。缺陷在于若弃用策略不准确,可能造成语义断裂、回答质量下降,并且在高并发场景中,实时评估开销仍然较大。

4) 基于共享的压缩方法的优势在于不需要对每层独立缓存,显著减少层间存储规模,随模型规模的扩展性较好,引入的“三明治式”架构能够兼顾压缩率和稳定性。缺陷在于层间共享过多可能引发语义信息丢失,并且部分方法需要调整模型结构重新训练,复杂度较高。

5) 基于注意力头削减的压缩方法的优势在于能够从模型结构层面减少键值缓存量,压缩率较高,在推理阶段不涉及额外计算开销,能够显著提升推理速度。缺陷在于需要改变模型结构重新进行训练,灵活性较差,尤其在语义复杂任务中可能出现性能下降。

在实际应用中,这 5 类键值缓存压缩方法可通过多维协同整合实现更高效的压缩。由于各方法在压缩维度上具有互补性(例如量化作用于数值精度层面,合并与共享减少结构冗余,标记弃用动态控制上下文长度,注意力头削减从架构层面降低缓存需求),研究者可将其有机结合,构建分层、分阶段的混合压缩框架。例如,首先通过标记弃用或重要性评估筛选低贡献标记,减少序列长度;随后对剩余标记的键值缓存执行基于相似度的合并或层间共享,以进一步消除冗余;最后在数值层面施加低比特量化或混合精度压缩,从而在保证模型回答质量的同时显著提升整体压缩率。此外,还可在模型结构层采用注意力头削减(如 MQA 或 GQA)以进一步降低键值缓存的存储需求。通过这种“弃用—合并/共享—量化—结构优化”的多层协同方式,可有效实现高压压缩率与低精度损失的综合平衡,为大语言模型的高效部署提供可行路径。

当前主流大语言模型在推理过程中面临海量键值缓存的存储压力,不同的压缩算法对模型的影响存在差异。由于 LLaMa 系列中较新的模型采取 GQA 架构,因此该系列模型在注意力头维度已经进

行了压缩,可结合基于标记弃用的策略等进行进一步压缩。同时,其标准化的 Transformer 结构与较高的层间冗余使基于合并和量化的压缩效果尤为突出,在保证回答质量的同时可实现显著的空间节约。与之相对,DeepSeek 模型采取了另一种注意力头架构——MLA,因此和现有的基于注意力头削减的压缩方法如 GQA 和 MQA 不兼容。针对指令微调模型的键值缓存进行压缩时,在压缩比例低于 30% 时性能下降非常显著^[46]。这表明指令微调虽然提升了模型的整体表现,但可能增加模型在激进压缩下的脆弱性。因此,不同模型需结合自身层结构、任务类型和推理需求选择合适的压缩策略,从而在降低键值缓存的存储压力的同时尽量保持回答质量。

5 未来研究方向展望

键值缓存压缩作为大语言模型推理的重要优化技术,其核心目标在于在保证模型回答质量的前提下最大限度降低缓存开销,以支持长上下文推理与高并发部署。其总体思路是通过在时序维度、结构维度与数值维度上挖掘冗余性,对键值缓存进行压缩,从而实现空间与计算的平衡。目前主流方法可分为五大类:基于合并的压缩方法通过跨层或跨标记的相似性合并来减少冗余存储;基于量化的方法以低比特精度表示键值向量来实现高压缩率;基于标记弃用的方法根据重要性动态保留或删除部分缓存以控制上下文长度;基于共享的方法复用缓存数据以降低存储需求;而基于注意力头削减的方法则通过结构性调整在模型层面根本减少键值缓存生成量。这些方法从不同角度平衡了压缩率、精度与计算成本,已在主流模型中展现出显著的性能提升。

未来针对键值缓存压缩方法的研究仍面临三大关键挑战,进一步探索和突破这些挑战方能推动键值缓存压缩方法在更广泛的大模型服务场景中实现高效落地。

1)提升跨任务与跨场景的泛化能力。现有压缩方法通常在特定的测试任务或模型上能够保障较好的模型回答质量,但该能力在更复杂的场景下可能显著退化。实际部署中,大模型服务需要适配不同类型的生成任务、上下文长度分布以及模型微调方式(如指令微调),例如 Singh 等人^[46]的工作指出,针对经过指令微调的模型应用现有压缩方法时,模型回答质量往往出现明显下降。因此,未来的压缩方法设计需要摆脱针对特定负载优化的局限性,构建具

备跨任务稳健性、跨输入模式自适应能力的压缩机制,确保在复杂实际场景中的可用性和可靠性。

2)减少压缩带来的额外计算开销。许多压缩方法在部署过程中引入了额外的计算操作,例如注意力分数计算等,这将带来显著的时间成本和算力资源开销。在推理延迟高度敏感的实际应用中,如搜索问答、智能客服、代码补全等场景,压缩方法若导致推理延迟增加,将直接影响用户体验与服务稳定性。在实现键值缓存的高效压缩的同时,未来研究应当力求将额外计算开销控制在可接受范围内。

3)实现压缩方法自动化选择与参数自适应调节。随着研究进展,键值缓存压缩方法的种类和组合方式日益丰富,不同方法之间具有良好的互补性,且均支持多级压缩率调节。这一灵活性虽带来更大的压缩潜力,但也显著增加了用户配置的复杂度。若将压缩方法的选择、组合与参数调节完全交由用户完成,不仅使用门槛较高,还可能导致次优选择甚至功能失效。因此,一个亟待解决的关键问题是:如何基于用户的业务目标(如实时性优先、质量优先、成本约束等)和任务特征,自动选择最合适的压缩方法组合,并动态调节各压缩方法的内部参数,实现在压缩率与回答质量之间的自适应权衡。

从全局视角来看,键值缓存压缩正逐步成为大语言模型推理系统中的关键优化方向,其未来发展将呈现出多维融合、智能调控与系统协同的总体趋势。一方面,随着模型规模和上下文长度的持续增长,研究将从单一压缩策略转向跨层、跨通道联合压缩,实现多粒度的动态平衡;另一方面,压缩方法将与模型训练、推理调度、显存管理深度融合,形成“算法—系统—硬件”协同优化的新范式。同时,自动化和智能化将成为该领域的重要方向,系统能够依据任务类型、上下文复杂度和硬件资源状况自适应选择压缩策略与参数,以实现性能、精度与效率的整体最优。此外,随着多模态模型的兴起,键值缓存压缩也将从单模态优化扩展到视频、图文等多模态场景,强调跨模态一致性和语义连续性;而标准化接口与评测体系的建立,将推动该领域形成开放、可迁移、可复现的技术生态。

具体而言,未来研究可从以下 5 个方向进行优化与突破:1)在压缩机制上应探索融合式多阶段压缩框架,通过“标记筛选—层间共享—混合量化”多层协同实现更高压缩率与稳定性能。2)可引入自适应决策与学习优化机制,利用强化学习等方式动态调节压缩率。3)应加强软硬件协同设计,在 GPU/NPU

层实现原生压缩算子功能,减少数据搬移与传输延迟。4)在多模态与长上下文任务中,可研究语义保持与模态对齐机制,防止压缩后的语义漂移。5)应推动建立统一的压缩评测与对比标准,从压缩率、计算开销和精度损失等维度系统评估不同算法在各个模型上的表现。通过对上述方向的持续探索,键值缓存压缩技术有望成为支撑下一代大语言模型高效推理与可扩展部署的核心基础。

6 总 结

大语言模型推理需要存储海量键值缓存以实现“以存换算”,但由此带来的存储资源瓶颈严重制约了模型推理的效率。针对这个挑战,研究者提出了多种键值缓存压缩方法,以期在尽量保持模型回答质量的前提下,有效降低键值缓存规模。当前主流压缩方法包括基于合并、量化、标记弃用、共享和注意力头削减5类方法,这些方法在压缩率与性能鲁棒性方面均表现出良好平衡,初步验证了其工程应用潜力。同时,不同方法在压缩原理、实现复杂度及适用场景等方面各具特点。

未来,键值缓存压缩技术的研究可从以下几个方向进一步探索:融合式多阶段压缩策略、自适应压缩率决策机制、软硬件协同优化设计、语义保持与多模态对齐方法、压缩评测标准体系构建。

作者贡献声明:胡世鹏负责文献调研和论文撰写;张广艳负责设计论文框架并修改论文;郑邦民负责确定论文思路并给出论文初步架构。

参 考 文 献

- [1] Achiam J, Adler S, Agarwal S, et al. GPT-4 technical report[J]. arXiv preprint, arXiv: 2303.08774, 2023
- [2] Liu Aixin, Feng Bei, Xue Bing, et al. DeepSeek-V3 technical report[J]. arXiv preprint, arXiv: 2412.19437, 2024
- [3] Team G, Anil R, Borgeaud S, et al. Gemini: A family of highly capable multimodal models[J]. arXiv preprint, arXiv: 2312.11805, 2023
- [4] Touvron H, Lavril T, Izacard G, et al. LLaMA: Open and efficient foundation language models[J]. arXiv preprint, arXiv: 2302.13971, 2023
- [5] Chen Xuanning, Ye Junjie, Zu Can, et al. Robustness of GPT large language models on natural language processing tasks[J]. *Journal of Computer Research and Development*, 2024, 61(5): 1128–1142 (in Chinese)
- (陈炫婷, 叶俊杰, 祖璨, 等. GPT 系列大语言模型在自然语言处理任务中的鲁棒性[J]. *计算机研究与发展*, 2024, 61(5): 1128–1142)
- [6] Lin Meng, Dai Chengwei, Guo Tao. A method for generating explanations of offensive memes based on multimodal large language models[J]. *Journal of Computer Research and Development*, 2024, 61(5): 1206–1217 (in Chinese)
- (林萌, 戴程威, 郭涛. 基于多模态大语言模型的攻击性模因解释生成方法[J]. *计算机研究与发展*, 2024, 61(5): 1206–1217)
- [7] Li Yuhong, Huang Yingbing, Yang Bowen, et al. SnapKV: LLM knows what you are looking for before generation[C]//Proc of the 38th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2024: 22947–22970
- [8] Li Minghan, Chen Xilun, Holtzman A, et al. Nearest neighbor speculative decoding for LLM generation and attribution[C]//Proc of the 38th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2024: 80987–81015
- [9] Chen Huimin, Liu Zhiyuan, Sun Maosong. The social opportunities and challenges in the era of large language models[J]. *Journal of Computer Research and Development*, 2024, 61(5): 1094–1103 (in Chinese)
- (陈慧敏, 刘知远, 孙茂松. 大语言模型时代的社会机遇与挑战[J]. *计算机研究与发展*, 2024, 61(5): 1094–1103)
- [10] Zhang Jinghan, Wang Xiting, Ren Weijie, et al. RATT: A thought structure for coherent and correct LLM reasoning[C]//Proc of the 39th AAAI Conf on Artificial Intelligence. Palo Alto, CA: AAAI, 2025: 26733–26741
- [11] Xiao Ziyang, Zhang Dongxiang, Han Xiongwei, et al. Enhancing LLM reasoning via vision-augmented prompting[C]//Proc of the 38th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2024: 28772–28797
- [12] Li Zhenyu, Fan Sunqi, Gu Yu, et al. FlexKBQA: A flexible LLM-powered framework for few-shot knowledge base question answering[C]//Proc of the 39th AAAI Conf on Artificial Intelligence. Palo Alto, CA: AAAI, 2024: 18608–18616
- [13] Keat K, Venkatesh R, Huang Yidi, et al. PGxQA: A resource for evaluating LLM performance for pharmacogenomic QA tasks[C]//Proc of the 30th Pacific Symp on Biocomputing. Singapore: World Scientific, 2024: 229–246
- [14] Gu Qiuhan. LLM-based code generation method for golang compiler testing[C]//Proc of the 31st ACM Joint European Software Engineering Conf and Symp on the Foundations of Software Engineering. New York: ACM, 2023: 2201–2203
- [15] Wang Jianxun, Chen Yixiang. A review on code generation with LLMs: Application and evaluation[C]//Proc of the 1st IEEE Int Conf on Medical Artificial Intelligence. Piscataway, NJ: IEEE, 2023: 284–289
- [16] Fakhoury S, Naik A, Sakkas G, et al. Exploring the effectiveness of LLM based test-driven interactive code generation: User study and empirical evaluation[C]//Proc of the 46th Int Conf on Software Engineering. New York: ACM, 2024: 390–391
- [17] Mu Fangwen, Shi Lin, Wang Song, et al. ClarifyGPT: A framework for enhancing LLM-based code generation via requirements clarification[J]. *Proceedings of the ACM on Software Engineering*, 2024, 1:

- 2332–2354
- [18] Stamper J, Xiao Ruiwei, Hou Xinying. Enhancing LLM-based feedback: Insights from intelligent tutoring systems and the learning sciences[C]//Proc of the 25th Int Conf on Artificial Intelligence in Education. Berlin: Springer, 2024: 32–43
- [19] Molina I V, Montalvo A, Ochoa B, et al. Leveraging LLM tutoring systems for non-native English speakers in introductory CS courses[J]. arXiv preprint, arXiv: 2411.02725, 2024
- [20] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Proc of the 31st Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2017: 5998–6008
- [21] Gao Bin, He Zhuomin, Sharma P, et al. Cost-Efficient large language model serving for multi-turn conversations with CachedAttention [C]//Proc of the 30th USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2024: 111–126
- [22] Kwon W, Li Zhuohan, Zhuang Siyuan, 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
- [23] Gao Shiwei, Chen Youmin, Shu Jiwei. Fast state restoration in LLM serving with HCache[C]//Proc of the 20th European Conf on Computer Systems. New York: ACM, 2025: 128–143
- [24] Hatalis K, Christou D, Myers J, et al. Memory matters: The need to improve long-term memory in LLM-agents[C]//Proc of AAAI Fall Symp Series. Palo Alto, CA: AAAI, 2023: 277–280
- [25] Wang Weizhi, Dong Li, Cheng Hao, et al. Augmenting language models with long-term memory[C]//Proc of the 37th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2023: 74530–74543
- [26] An Wei, Bi Xiao, Chen Guanting, et al. Fire-Flyer AI-HPC: A cost-effective software-hardware co-design for deep learning[C]//Proc of the 24th Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2024: 1–23
- [27] Qin Ruoyu, Li Zheming, He Weiran, et al. Mooncake: Trading more storage for less computation—A KV cache-centric architecture for serving LLM Chatbot[C]//Proc of the 23rd USENIX Conf on File and Storage Technologies. Berkeley CA: USENIX Association, 2025: 155–170
- [28] Wang Zheng, Jin Boxiao, Yu Zhongzhi, et al. Model tells you where to merge: Adaptive KV cache merging for LLMs on long-context tasks[J]. arXiv preprint, arXiv: 2407.08454, 2024
- [29] Akide L, Liu Jing, Pan Zizheng, et al. MiniCache: KV cache compression in depth dimension for large language models[C]//Proc of the 38th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2024: 139997–140031
- [30] Hooper C, Kim S, Mohammadzadeh H, et al. KVQuant: Towards 10 million context length LLM inference with KV cache quantization [C]//Proc of the 38th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2024: 1270–1303
- [31] Liu Yuhao, Li Hanchen, Cheng Yihua, et al. CacheGen: KV cache compression and streaming for fast large language model serving[C]//Proc of the 38th Conf of the ACM Special Interest Group on Data Communication. New York: ACM, 2024: 38–56
- [32] Cheng Feng, Guo Cong, Wei Chiyue, et al. Ecco: Improving memory bandwidth and capacity for LLMs via entropy-aware cache compression[C]//Proc of the 52nd Annual Int Symp on Computer Architecture. New York: ACM, 2025: 793–807
- [33] Liu Zichang, Desai A, Liao Fangshuo, et al. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time[C]//Proc of the 37th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2023: 52342–52364
- [34] Chen Weijian, He Shuibing, Qu Haoyang, et al. IMPRESS: An importance-informed multi-tier prefix KV storage system for large language model inference[C]//Proc of the 23rd USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2025: 187–201
- [35] Liu Guangda, Li Chengwei, Zhao Jieru, et al. ClusterKV: Manipulating LLM KV cache in semantic space for recallable compression[C]//Proc of the 62nd ACM/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2025: 1–7
- [36] Yang Yifei, Cao Zouying, Chen Qiguang, et al. KVSharer: Efficient inference via layer-wise dissimilar KV cache sharing[J]. arXiv preprint, arXiv: 2410.18517, 2024
- [37] Wu Haoyi, Tu Kewei. Layer-condensed KV cache for efficient inference of large language models[C]//Proc of the 62nd Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA: ACL, 2024: 11175–11188
- [38] Shazeer N. Fast transformer decoding: One write-head is all you need[J]. arXiv preprint, arXiv: 1911.02150, 2019
- [39] Ainslie J, Lee-Thorp J, De Jong M, et al. GQA: Training generalized multi-query transformer models from multi-head checkpoints[J]. arXiv preprint, arXiv: 2305.13245, 2023
- [40] Ding Yiran, Zhang Li, Zhang Chengruidong, et al. LongRoPE: Extending LLM context window beyond 2 million tokens[J]. arXiv preprint, arXiv: 2402.13753, 2024
- [41] Ma Xuezhe, Yang Xiaomeng, Xiong Wenhan, et al. Megalodon: Efficient LLM pretraining and inference with unlimited context length[C]//Proc of the 38th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2024: 71831–71854
- [42] Zhong Yinmin, Liu Shengyu, Chen Junda, et al. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving[C]//Proc of the 18th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2024: 193–210
- [43] Jeong C. Generative AI service implementation using LLM application architecture based on RAG model and LangChain framework[J]. Journal of Intelligence and Information Systems, 2023, 29(4): 129–164
- [44] Qiu Jianing, Lam K, Li Guohao, et al. LLM-based agentic systems in

- medicine and healthcare[J]. *Nature Machine Intelligence*, 2024, 6(12): 1418–1420
- [45] Yu Yangyang, Yao Zhiyuan, Li Haoqiang, et al. Fincon: A synthesized LLM multi-agent system with conceptual verbal reinforcement for enhanced financial decision making[C]//Proc of the 38th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2024: 137010–137045
- [46] Singh K, Kaur S, Smiley C. FinQAPT: Empowering financial decisions with end-to-end LLM-driven question answering pipeline [C]//Proc of the 5th ACM Int Conf on AI in Finance. New York: ACM, 2024: 266–273
- [47] Gao Wei, Zhou Xinyu, Sun Peng, et al. Rethinking key-value cache compression techniques for large language model serving[J]. arXiv preprint, arXiv: 2503.24000, 2025
- [48] Liu Xiang, Tang Zhenheng, Chen Hong, et al. Can LLMs maintain fundamental abilities under KV cache compression?[J]. arXiv preprint, arXiv: 2502.01941, 2025
- [49] Cobbe K, Kosaraju V, Bavarian M, et al. Training verifiers to solve math word problems[J]. arXiv preprint, arXiv: 2110.14168, 2021
- [50] Liu Haotian, Li Chunyuan, Wu Qingyang, et al. Visual instruction tuning[C]//Proc of the 37th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2023: 34892–34916
- [51] Reddy S, Chen D, Manning C D. CoQA: A conversational question answering challenge[J]. *Transactions of the Association for Computational Linguistics*, 2019, 7: 249–266
- [52] Salimans T, Kingma D P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks [C]//Proc of the 30th Conf on Advances in Neural Information Processing Systems. Red Hook, NY: Curran Associates Inc, 2016: 901–909
- [53] Su Jianlin, Lu Yu, Pan Shengfeng et al. RoFormer: Enhanced transformer with rotary position embedding[J]. *Neurocomputing*, 2024, 568: 127063
- [54] Xiao Guangxuan, Tian Yuandong, Chen Beidi, et al. Efficient streaming language models with attention sinks[J]. arXiv preprint, arXiv: 2309.17453, 2023
- [55] Rissanen J, Langdon G G. Arithmetic coding[J]. *IBM Journal of Research and Development*, 1979, 23(2): 149–162



Hu Shipeng, born in 1999. PhD candidate. His main research interests include storage systems and large language models.

胡世鹏, 1999 年生。博士研究生。主要研究方向为存储系统、大语言模型。



Zhang Guangyan, born in 1976. PhD, associate professor, PhD supervisor. Distinguished member of CCF. His main research interests include big data computing, storage systems, and distributed processing.

张广艳, 1976 年生。博士, 副教授, 博士生导师。CCF 杰出会员。主要研究方向为大数据计算、存储系统、分布式处理。



Zheng Weimin, born in 1946. Master, professor, PhD supervisor, academician of Chinese Academy of Engineering. Fellow of CCF. His main research interests include high performance computing, network storage, and parallel compiler.

郑伟民, 1946 年生。硕士, 教授, 博士生导师, 中国工程院院士。CCF 会士。主要研究方向为高性能计算、网络存储、并行编译。