学号：23724590

# 《高级计算机系统结构》
# 课程论文

## 计算加速技术在大语言模型中的应用


姓　　名　郎志远

学　　号　23724590


论文评分 _____

2024 年 1 月 25 日

# 目 录

# 计算加速技术在大语言模型中的应用

[摘要]本文探讨了计算加速技术在大型语言模型中的应用，旨在探索这些模型在训练和推理过程中面临的计算资源和内存需求挑战。本文介绍了大语言模型的发展背景，并讨论了几种计算加速技术，使用 Yi-6B-Chat 模型在部署了两张 A100 的服务器上进行实验，实验结果表明，KV Cache 和 Flash Attention 技术能够有效降低模型的存储和计算需求，提高训练和推理效率。特别是在使用 Flash Attentio 时，训练时间平均减少了约 26%，显示出显著的性能提升。

[关键字]大语言模型 计算加速 推理优化 Flash Attention

## The Application of Computational Acceleration Techniques in Large Language Models

**Abstract**: This paper explores the application of computational acceleration techniques in large language models (LLMs), aiming to address the challenges of computational resources and memory requirements faced by these models during training and inference processes. The paper introduces the development background of LLMs and discusses several computational acceleration techniques. Experiments were conducted using the Yi-6B-Chat model on a server equipped with two A100 GPUs. The results indicate that KV Cache and Flash Attention techniques can effectively reduce the storage and computational demands of the model, enhancing training and inference efficiency. Notably, the use of Flash Attention led to an average reduction of approximately 26% in training time, demonstrating significant performance improvements.

**Key words:**Large Language Models Computational Acceleration Inference Optimization Flash Attention

## 1、引言

在当今人工智能的快速发展浪潮中，大语言模型已经成为推动自然语言处理领域革新的核心动力。这些模型，以其庞大的参数规模和深度学习能力，能够在多种语言任务上展现出令人瞩目的表现，从而在机器翻译、文本生成、情感分析等多个领域实现了前所未有的突破。然而，随着模型规模的不断扩大，它们对计算资源的需求也呈现出指数级增长，这不仅对硬件设施提出了更高的要求，也对研究人员提出了新的挑战。特别是在训练和部署这些模型时，所需的计算力、内

存容量以及数据传输速度成为了制约其进一步发展的关键因素。

为了应对这些挑战，计算加速技术应运而生，它们通过一系列创新的方法来优化模型的训练和推理过程。这些技术包括但不限于算法层面的改进、硬件层面的优化以及并行计算策略的创新。算法层面的优化，如高效的注意力机制和低秩近似，能够有效减少模型的计算复杂度，从而在保持性能的同时降低资源消耗。硬件层面的优化则涉及到利用 GPU 的并行处理能力，以及开发专为 AI 计算设计的新型硬件，如谷歌的 TPU，这些硬件能够显著提升模型的计算效率。此外，通过数据并行、模型并行和流水线并行等策略，可以将计算任务分散到多个处理单元，从而实现大规模模型的高效训练。

本文旨在深入探讨计算加速技术在大语言模型中的应用，通过在本地部署大语言模型并使用 KV Cache、Flash Attention 等计算加速方法进行测试，分析这些技术如何帮助研究人员和开发者克服资源限制，实现更高效、更经济的模型训练和部署。本文将详细介绍这些技术的工作原理，评估它们在实际应用中的性能表现，并探讨它们在提升大语言模型训练效率和推理速度方面的潜力。同时，也将讨论在实际应用中遇到的挑战，并对未来的研究方向提出展望。

# 2、大语言模型介绍

## 2.1 大语言模型的发展

在人工智能的发展历程中，自然语言处理一直是研究的热点和挑战所在。随着深度学习技术的突破，特别是神经网络模型的出现，NLP 领域迎来了革命性的变化。这些变化的核心在于，模型能够自动从大量文本数据中学习语言的复杂结构和语义，而不再依赖于人工设计的规则和特征。这一转变标志着从基于规则的系统向基于数据驱动的系统的过渡。深度学习的核心在于其能够处理和学习数据的多层次特征。然而，传统的神经网络模型，如卷积神经网络和循环神经网络，在处理自然语言时面临挑战，因为语言的序列特性和长距离依赖关系使得这些模型难以有效捕捉上下文信息。为了克服这些限制，研究者们探索了新的模型架构，其中最引人注目的是 Transformer 模型。Transformer 最早是在 2017 年由 Google 团队在《Attention is All You Need》[1]这篇文章中提出的，Transformer 的模型架构如图 1 所示，它包含了一个编码器模块和一个解码器模块，其中编码器由 6 个相同的层组成，每层包含两个子层。如图 2 所示，第一个子层是多头自注意力机制，它允许模型在处理输入序列时，考虑序列中所有元素之间的关系。第二个子层是一个位置无关的全连接前馈神经网络，它对输入进行非线性变换。在每个子层周围都有残差连接，这意味着子层的输出会与输入相加，以保持信息的流动。每个子层之后都跟着层归一化，这有助于稳定训练过程。解码器同样由 6 个相同的层组成，但除了编码器中的两

个子层外，还增加了第三个子层，这个子层执行多头自注意力，关注编码器堆栈的输出。解码器中的每个子层也使用残差连接和层归一化。解码器中的自注意力子层被加入了遮蔽机制，确保预测只能依赖于位置 i 之前已知的输出并且输出在时间上有一个位置偏移，这与遮蔽机制相结合，确保了预测的顺序性。

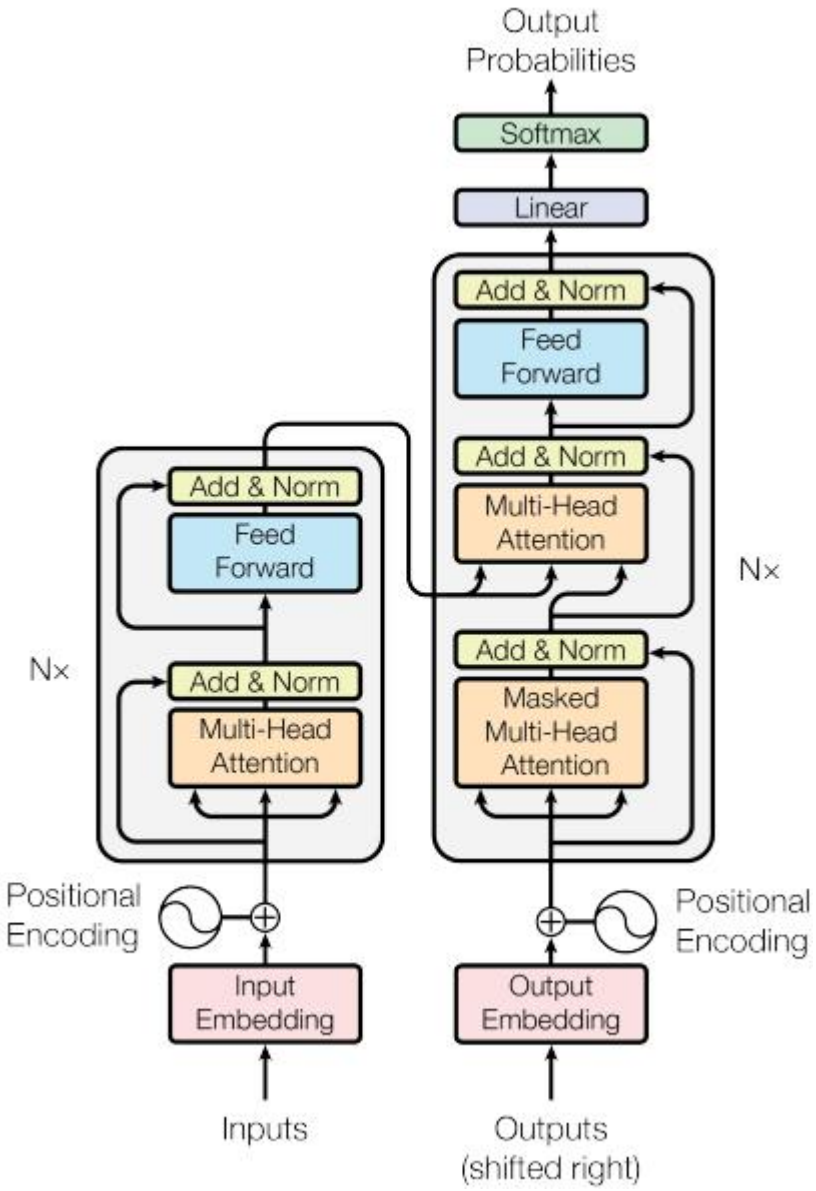


图 1 Transformer 的模型架构

随着研究的深入，Transformer 展示了其作为通用模型架构的潜力。尤其是它通过自注意力机制，允许模型在处理序列数据时对每个元素进行全局关注，从而更好地理解上下文。这一创新为大语言模型的发展奠定了基础。

大语言模型在 NLP 领域的应用已经触及到生活的方方面面，通过在大规模文本数据集上进行预训练，它们学习到了丰富的语言知识，在文本生成领域，

它们能够创作文章、撰写邮件、甚至生成诗歌，其生成的文本质量越来越接近人类水平。在文本理解方面，大语言模型在情感分析、文本分类、问答系统等任务中的表现，已经能够满足商业应用的需求。此外，大语言模型还被用于构建更智能的搜索引擎，通过理解用户的查询意图，提供更加精准和相关的搜索结果。大语言模型不仅在文本生成、理解等 NLP 任务上展现出了卓越的性能，其应用范围也在迅速扩大，从学术研究到商业应用，它们在提高语言处理任务的效率和准确性方面发挥了重要作用[2]。例如，OpenAI 的 GPT 系列模型能够生成连贯的文本、回答问题，甚至编写代码，而 BERT[3]则在理解语言的深层次语义上取得了显著成就。此外大语言模型的关键特性不仅体现在其庞大的参数规模，更在于其强大的上下文理解能力和迁移学习能力。这些模型能够通过微调快速适应新的 NLP 任务，而无需从头开始训练，这极大地提高了模型的实用性和灵活性。并且大语言模型在处理多语言任务时也展现出了跨语言的泛化能力，这对于构建全球化的智能系统具有重要意义。这些模型的成功应用不仅推动了 NLP 技术的发展，也为其他领域的 AI 应用提供了新的思路。随着计算能力的提升和算法的不断优化，大语言模型将继续在 AI 的未来扮演重要角色，引领我们进入一个更加智能、更加自然的人机交互时代。然而，大语言模型的这些成就并非没有代价，随着模型规模的不断扩大，它们对计算资源的需求也日益增长，这不仅对硬件设施提出了更高的要求，也对研究人员提出了新的挑战[4]。如何在有限的资源下训练和部署这些模型，以及如何确保模型的泛化能力和对数据的敏感性，成为了当前研究的热点。

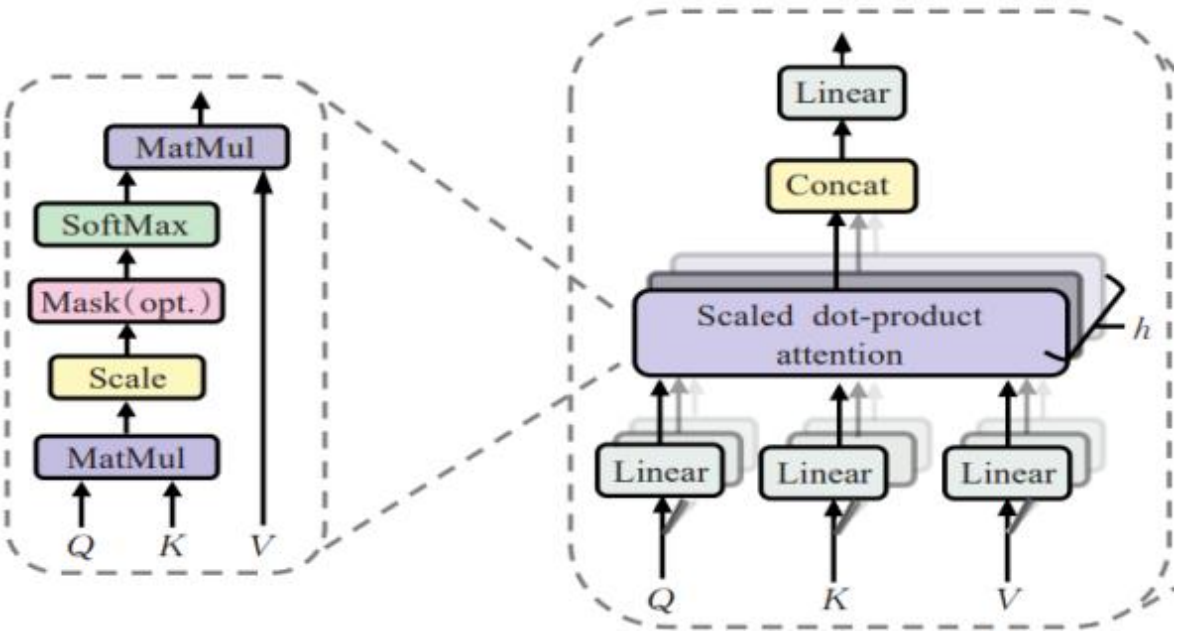


图 2 自注意力（左）和多头自注意力（右）

## 2.2 大语言模型面临的计算挑战

大语言模型的规模和复杂性带来了显著的计算挑战，特别是在推理阶段，这直接影响了模型的实时性和可扩展性，只有提高推理效率，降低资源消耗，才能使这些强大的模型能够在更广泛的实际应用中发挥作用。大语言模型的参数量通常达到数十亿甚至数百亿，这导致在实际应用中，尤其是在资源受限的环境中，部署和运行这些模型变得异常困难。此外，随着模型规模的不断扩大，它们也面临着日益严峻的计算挑战，这些挑战主要体现在以下几个方面：

- 计算资源需求。大语言模型的参数数量通常达到数十亿甚至数千亿，这要求巨大的计算资源来支持模型的训练和推理。这种资源密集型的训练过程需要大量的 GPU 和内存。此外，模型的存储和传输也需要大量的带宽和存储空间，这在资源有限的环境中尤其具有挑战性。
- 训练时间与效率。由于模型的复杂性，大语言模型的训练时间通常以月甚至年计。这种长时间的训练周期限制了模型迭代的速度，使得研究人员难以快速适应新的数据集或任务需求。并且长时间的训练过程也可能导致资源的浪费，尤其是在模型需要频繁调整和优化的情况下。
- 数据传输与同步问题。在分布式训练环境中，大语言模型需要在多个计算节点之间同步数据和模型状态。这不仅增加了通信开销，还可能导致同步延迟，影响训练效率。特别是在模型并行和流水线并行策略中，如何高效地管理数据流和模型状态成为了一个技术难题。
- 模型部署与推理效率。大语言模型在实际应用中的部署需要考虑到推理效率。在资源受限的设备上，如移动设备和边缘计算节点，如何优化模型以降低计算需求，同时保持高性能，是一个重要的研究方向。

# 3、相关工作

在大模型计算加速领域，许多研究人员和团队通过创新的方法和理论贡献，推动了大语言模型（大语言模型）训练和推理效率的提升。Alexey Radul 和 Yann LeCun 等人提出了 LoRA（Low-Rank Adaptation）[5]，这是一种通过在 Transformer 模型中引入低秩矩阵来实现参数高效更新的方法，LoRA 显著减少了模型更新时所需的参数数量，同时保持了模型性能。R Child，S Gray，A Radford，I Sutskever 等人提出了了 Sparse Transformer[6]，它是一种通过稀疏化注意力机制来减少计算需求的方法。Sparse Transformer 在保持模型性能的同时，显著降低了计算和内存消耗。Samyam Rajbhandari、Jeff Rasley、Olatunji Ruwase 和 Yuxiong He 等人共同开发了 ZeRO（Zero Redundancy Optimizer）[7]，这是一种内存优化技术，通过消除数据并行训练中的冗余内存消耗，显著提高了训练大型模型的效率。ZeRO 使得在有限的硬件资源下训练万亿参数模型成为可能。Y Zhang，

P Zhang，Y Yan 提出了 FlashAttention[8]，这是一种通过优化内存访问模式来加速 Transformer 模型推理的技术，FlashAttention 显著提高了模型的推理速度，降低了延迟。Nathan Srebro 和 Ido Carmon 等人提出了一种名为 ALiBi（Adaptive Linear Blockwise Inference）[9]的方法，该技术通过动态调整模型的计算粒度，实现了在保持模型性能的同时，显著提高了推理效率。这些研究人员和团队的工作不仅推动了计算加速技术的发展，也为大语言模型在实际应用中的部署和使用提供了可能。他们的研究成果在学术界和工业界都产生了深远的影响，为未来的 AI 研究和应用奠定了坚实的基础。

# 4、计算加速技术介绍

计算加速优化一般有两个目标，一个是更低的延迟，一个是更高的吞吐量。其中延迟是指单个请求返回的时间，而吞吐量是一定时间内处理请求的总量。这两者有时是不可兼得的。任何计算的本质都是 CPU/GPU 执行一系列的指令，针对上面两个优化目标，在模型结构固定的情况下，我们能做的优化可以分为两类：计算侧优化和内存 IO 优化。计算侧优化致力于减少需要执行的指令数量，即减少不必要的或重复的运算或者是充分利用硬件的并发度，要么是让单条指令可以一次处理多条数据，要么是利用 CPU 和 GPU 的多核心机制，同时执行多条指令。而内存 IO 优化则是利用缓存局部性加速内存读取指令的执行速度，或者减少不必要的内存读写以加速内存 IO 速度。

## 4.1 计算侧优化

### 4.1.1 KV Cache

KV Cache[10]是一种在大型语言模型推理过程中常用的优化技术，特别是在基于 Transformer 架构的模型中。它的核心思想是利用自注意力机制中的重复计算来提高推理效率。在自注意力机制中，模型在处理序列数据时，会多次计算相同的键（Key）和值（Value）对，这些计算在标准的自注意力过程中是重复进行的。

如图 3 所示，针对计算重复，KV Cache 使用了缓存机制：在模型的自注意力层中，KV Cache 会将已经计算过的键值对（K,V）存储在一个缓存中。这些键值对代表了模型在处理输入序列时的上下文信息。当模型在后续的推理步骤中再次需要相同的键值对时，KV Cache 可以直接从缓存中检索这些值，而无需重新计算。这样可以显著减少计算量，因为自注意力机制中的计算是模型推理过程中最耗时的部分之一。

KV Cache 通过牺牲一定的内存空间来换取推理时间的减少。在资源有限的环境中，这种权衡可能是必要的，因为它可以显著提高模型的响应速度，尤其是在需要快速处理大量请求的场景中。KV Cache 适用于那些具有自注意力机制

的模型，如 GPT、BERT 等。这些模型在处理长序列时，KV Cache 可以显著提高效率。尽管 KV Cache 在提高推理速度方面表现出色，但它也有一些局限性。例如，如果输入序列的长度变化很大，或者模型需要处理动态长度的输入，KV Cache 可能需要更复杂的管理策略来确保缓存的有效性。此外，缓存的维护和更新也需要额外的计算资源。在实际应用中，KV Cache 通常作为推理框架的一部分被集成，如 Hugging Face 的 Transformers 库就提供了 KV Cache 的支持。通过这种方式，开发者可以轻松地在他们的模型中启用 KV Cache，以实现推理性能的提升。随着模型和硬件的不断进步，KV Cache 将继续在优化大型语言模型的推理效率方面发挥重要作用。

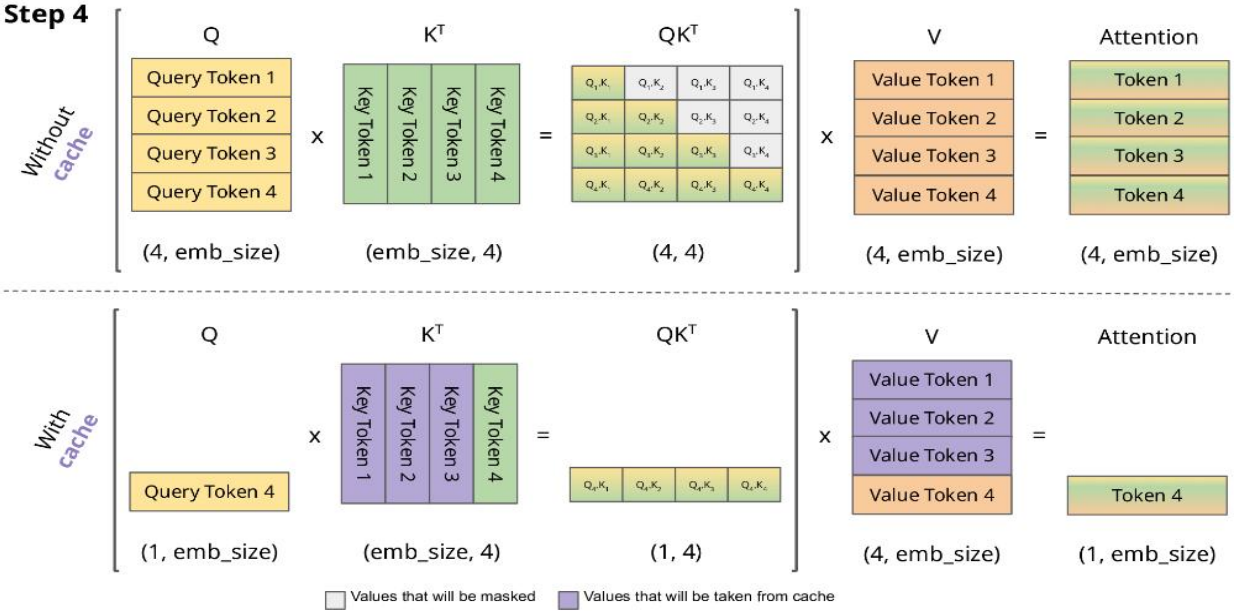


图 3 KV Cache

### 4.1.2 Kernel 优化和算子融合

在大型语言模型的推理和训练过程中，Kernel 优化和算子融合是两种重要的技术，它们对于提高模型的计算效率和性能至关重要。

Kernel 优化是指针对特定的硬件架构（如 CPU、GPU、TPU 等）对模型的计算密集型操作进行优化。这些操作通常包括矩阵乘法、卷积、激活函数等。通过编写高效的 Kernel，可以充分利用硬件的并行处理能力，减少计算时间，提高模型的运行速度。Kernel 优化通常需要针对特定的硬件平台进行，因为不同的硬件有不同的指令集和并行处理能力。例如，GPU 拥有大量的并行处理单元，适合执行大规模并行计算。通过优化 Kernel，可以显著减少模型执行时间，尤其是在执行大规模矩阵运算时。这不仅提高了模型的推理速度，也使得模型能够处理更大规模的数据集。Kernel 优化有助于更高效地利用硬件资源，减少能源消耗，这对于数据中心和移动设备等资源受限的环境尤为重要。

算子融合是指将多个连续的计算操作合并为一个单一的计算步骤。在深度学习框架中，这通常涉及到将多个神经网络层或操作（如卷积、激活函数、批量归一化等）合并在一起，以减少中间数据的传输和存储。

算子融合可以减少模型执行过程中的内存访问次数，因为合并后的算子可以直接在输入数据上进行操作，无需额外的数据移动。通过减少计算步骤，算子融合可以降低模型的总体计算复杂度，从而提高执行效率。在某些情况下，算子融合还可以简化模型的结构，使得模型更加紧凑，便于部署和维护。

在大语言模型中，Kernel 优化和算子融合的应用可以显著提升模型的训练和推理性能。这些技术对于实现实时或近实时的 NLP 应用至关重要，尤其是在需要处理大量数据和复杂任务的场景中。随着硬件技术的不断发展和深度学习框架的持续优化，Kernel 优化和算子融合将继续在提高大型语言模型的计算效率方面发挥重要作用。未来的研究可能会集中在如何自动地为不同的硬件平台和模型结构生成最优的 Kernel，以及如何更智能地进行算子融合，以实现更高的性能和更低的资源消耗。

#### 4.1.3 分布式推理

在大型模型的训练和推理过程中，为了克服单个计算资源的限制，研究人员采用了多种分布式并行策略来提高效率和扩展性[11]：

第一种是数据并行方式，如图 4 所示，在数据并行中，模型被完整地复制到多个 GPU 上。每个 GPU 都拥有模型的一个副本，并独立地处理数据集的一个子集。这种方法允许模型在多个 GPU 上同时进行推理，从而显著提高了处理速度。数据并行的主要优势在于其简单性和易于实现，但它并不改变模型的计算顺序，因此对于模型本身的计算速度提升有限。


图 4 数据并行

第二种方式是流水线并行，如图 5 所示，流水线并行是一种将模型拆分为多个部分，并将这些部分分布在不同 GPU 上的策略。每个 GPU 负责执行模型的一部分，数据在经过一个 GPU 的处理后，会传递给下一个 GPU 继续处理。这种方法类似于工业生产线，每个 GPU 专注于模型的一个特定阶段。尽管流水线并行可以提高模型的并行度，但由于各层之间的依赖关系，它并不能实现真正的并行计算，因此对于加速模型推理的效果有限。

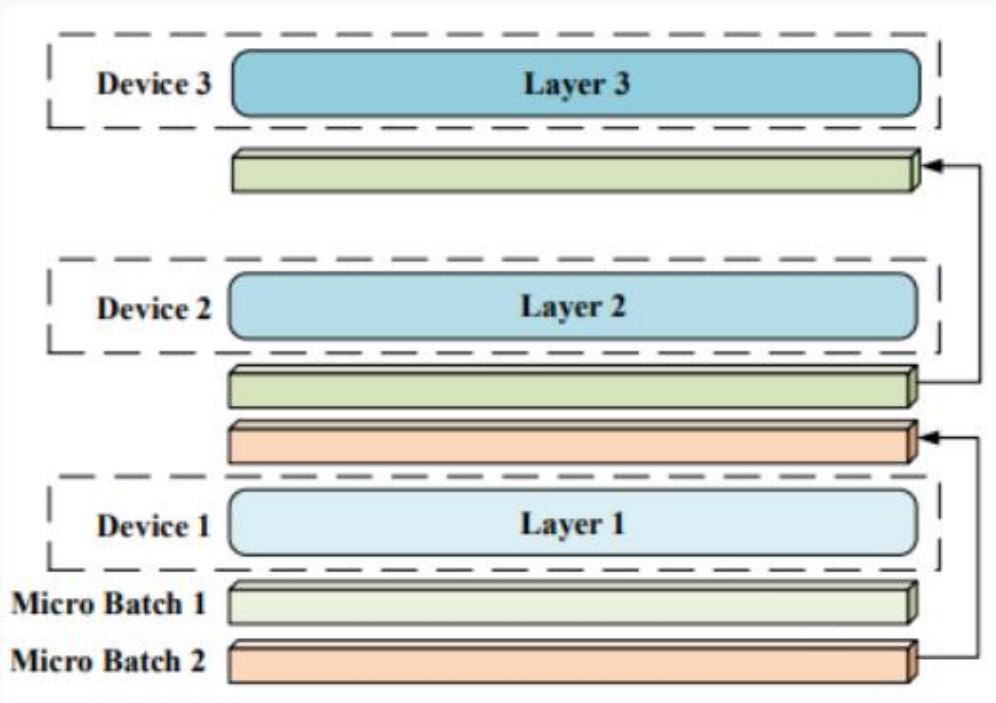


图 5 流水线并行

第三种方式是张量并行，如图 6 所示，张量并行是一种将模型的权重矩阵在多个 GPU 之间分割的策略。在这种并行方式下，模型的每一层都被拆分成多个部分，每个部分由不同的 GPU 负责计算。这种方法允许模型的每个层同时在多个 GPU 上进行计算，从而显著加速了模型的推理过程。张量并行特别适合于那些单个 GPU 内存无法容纳整个模型的情况，它通过并行化模型的计算，有效提高了推理速度。

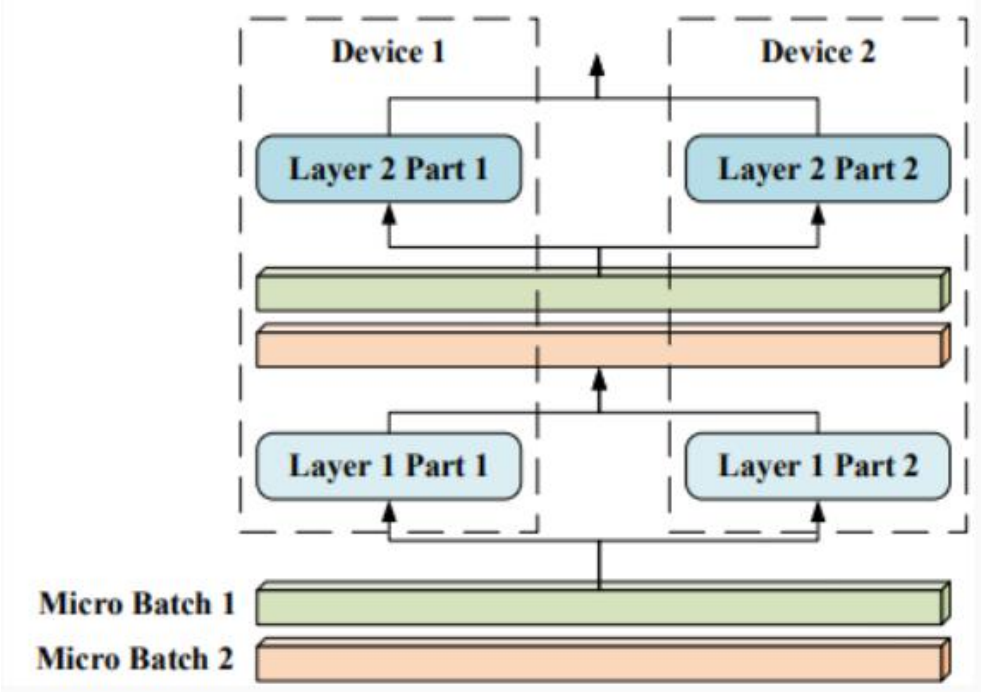


图 6 张量并行

总结来说，数据并行和张量并行都能够在一定程度上加速模型的推理过程，而流水线并行则更多地关注于提高模型的并行度，但受限于模型的计算顺序，其加速效果可能不如前两者显著。在实际应用中，选择合适的并行策略需要根据模型的大小、计算资源的可用性以及应用场景的具体需求来决定。随着硬件和软件技术的不断进步，这些分布式并行策略将继续发展，以支持更大规模、更复杂的模型在各种应用中的高效运行。

## 4.2 内存 IO 优化

在进行性能优化时，内存 I/O 效率是一个不可忽视的重要因素。在大语言模型的推理过程中，内存访问速度对于整体性能有着直接的影响。

### 4.2.1 Flash Attention

在大语言模型的自注意力计算中，Q、K、V 矩阵的矩阵乘法操作是核心步骤，但由于其规模巨大，直接进行全矩阵乘法会导致缓存不友好的问题。为了解决这一挑战，研究者们提出了多种优化策略，其中 FlashAttention 是一种特别有效的方法。FlashAttention 通过将传统的自注意力计算过程分解为更小的计算块，以适应 GPU 的缓存结构。这种方法的核心在于，它能够在不牺牲性能的情况下，减少对高带宽内存（HBM）的读写次数，从而提高整体的计算效率。FlashAttention 的主要思想如下：

IO 感知：如图 7 所示现代 GPU 的计算速度远超过内存访问速度，IO 成为了性能瓶颈。FlashAttention 关注 GPU 高带宽内存（HBM）和 GPU 片上 SRAM 之间的数据 IO 操作。它通过减少在 HBM 上的操作来提高效率。

分块：FlashAttention 通过将输入矩阵（Q, K, V）分割成小块，然后逐块进行处理，这样可以避免在 GPU 的慢速 HBM 中存储和读取整个注意力矩阵，从而减少 IO 操作。

重计算：在反向传播过程中，FlashAttention 不存储中间的注意力矩阵，而是通过存储 softmax 归一化因子来重计算注意力矩阵，这在内存效率和计算速度上都有所提升。

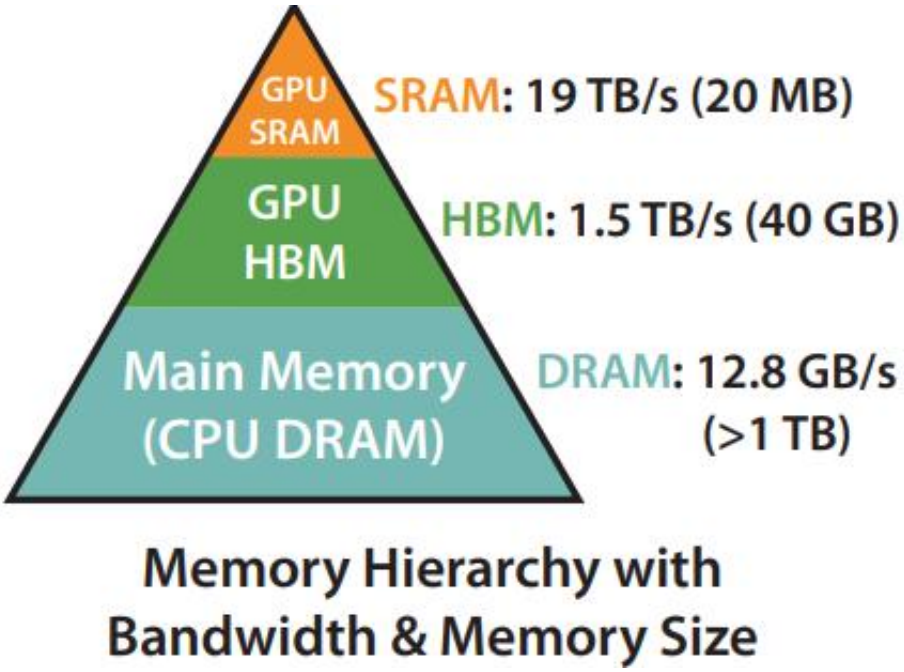


图 7 GPU 的分级缓存

FlashAttention 分为前向传播和反向传播两部分。如图 8 所示，在前向传播过程中，输入矩阵 Q,K,V 被分割成小块并加载到 GPU 的快速片上 SRAM 中。对于每个小块，计算 Q 和 K 的点积，然后应用 softmax 函数。使用 softmax 的归一化因子来调整输出，得到最终的注意力输出。在反向传播反向传播过程中使用前向传播中保存的 softmax 归一化因子和 dropout 掩码来重计算注意力矩阵。根据重计算的注意力矩阵，计算输入矩阵的梯度。

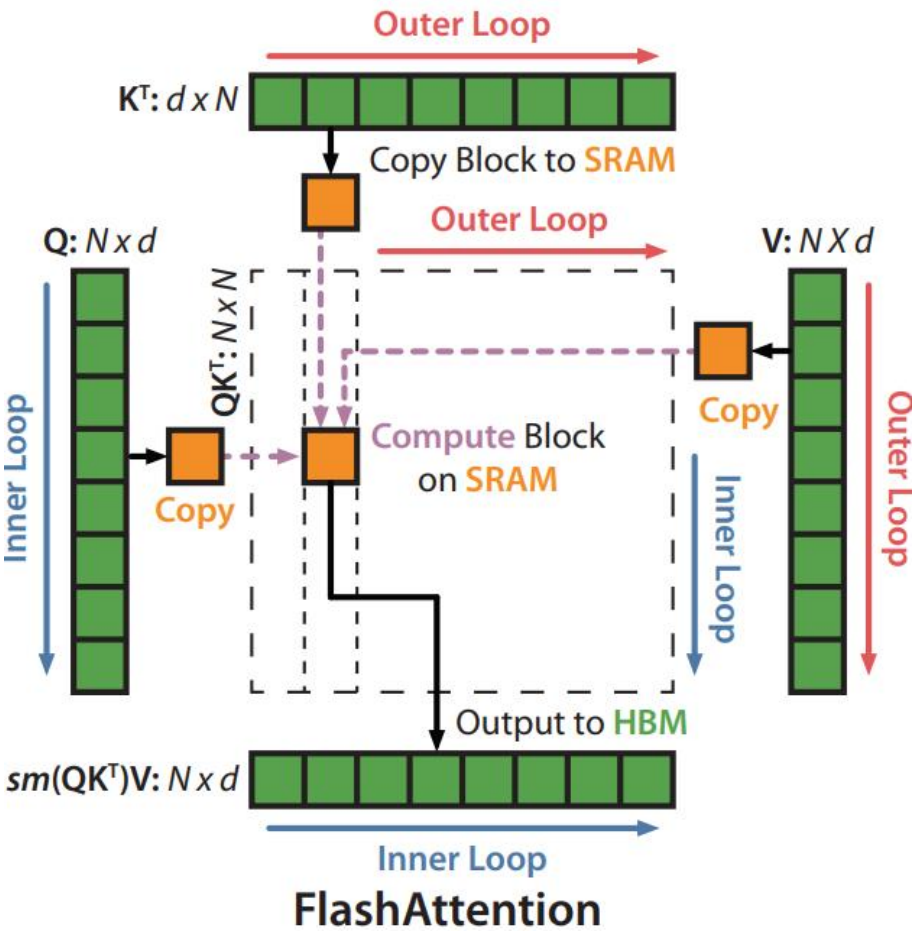


图 8 FlashAttention

尽管 FlashAttention 在减少 HBM 访问方面表现出色，但它仍然受到 GPU 的 SRAM 大小的限制。对于非常大的序列，可能需要更大的 SRAM 来支持分块操作并且 FlashAttention 需要在 CUDA 中实现，并且 FlashAttention 是针对特定 GPU 架构优化的，它可能在不同的 GPU 上表现不一致，这限制了其在多样化硬件环境中的可移植性。但 FlashAttention 作为一种高效的自注意力计算优化方法，对于提升大型语言模型的推理速度和降低资源消耗具有重要意义。随着深度学习框架和硬件的不断进步，FlashAttention 将继续在推动大型模型的广泛应用中发挥关键作用。

### 4.2.2 Flash Decoding

Flash Decoding 是一种针对在线推理场景中的自注意力计算优化技术，它特别适用于处理单个样本（batch size 为 1）且序列长度较长的情况。在这种情况下，传统的 Flash Attention 方法可能无法充分发挥 GPU 的并发计算能力，因为输入的 Q 矩阵实际上是一个向量，而不是一个矩阵。Flash Decoding 通过以下方式解决了这个问题：Flash Decoding 通过在序列长度的维度上进行并发计算，有效地利用了 GPU 的并行处理能力。具体来说，它将 K 和 V 矩阵分割成

多个小块，然后这些小块并行地与 Q 向量进行乘法运算。这种方法允许模型在处理单个样本时，仍然能够利用 GPU 的多核优势，从而提高了在线推理的效率。

Flash Decoding 的计算步骤如下包裹矩阵分割、并行乘法、结果聚合三个部分。首先，将 K 和 V 矩阵按照序列长度的维度分割成多个小块。这样，每个小块都对应于输入序列的一个固定长度的片段。然后，这些小块并行地与 Q 向量进行矩阵乘法运算。由于 Q 向量是单个样本，所以每个小块的计算都是独立的，可以同时在 GPU 的不同核心上执行。最后，将所有小块的乘法结果聚合起来，得到最终的输出。这个过程同样可以并行进行，以进一步提高效率。

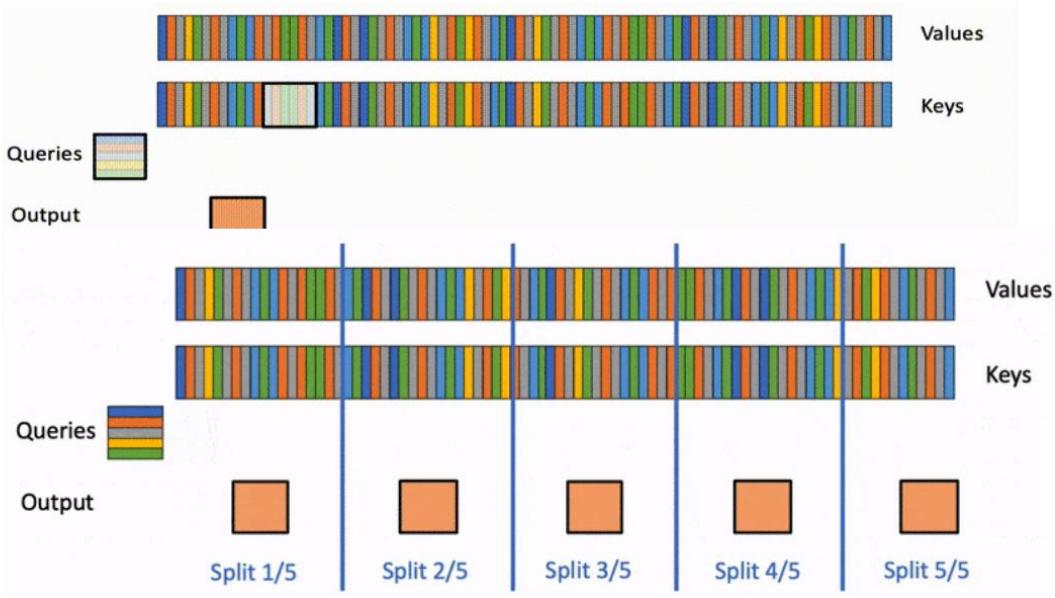


图 9 正常计算与使用 Flash Decoding 计算对比

Flash Decoding 通过在序列长度维度上的并发计算，充分利用了 GPU 的并行处理能力，即使在处理单个样本时也能保持高效。这种方法特别适用于在线推理场景，尤其是当上下文长度较长时，Flash Decoding 能够显著提高推理速度。总而言之，Flash Decoding 为在线推理场景提供了一种有效的性能优化方法，特别是在处理长序列数据时。随着在线推理需求的增长，Flash Decoding 等优化技术将在提升用户体验和降低延迟方面发挥重要作用。未来的研究可能会进一步探索如何将 Flash Decoding 与其他优化技术结合，以实现更全面的性能提升。

### 4.2.3 Continuous Batching

在批量推理过程中我们一般使用固定的 Batch Size 将多个请求 Batch 起来一起推理，这种固定的 Batch 策略叫做 Static Batching。在分配 KVCache 时，我们需要分配两个 shape 为[batch_size, seq_len, inner_dim]的 tensor，但是不同的请求可能有不同输入和输出长度，而且我们无法预知最终的输出长度，无法固定 seq_len，因此我们通常分配[batch_size, max_seq_len, inner_dim]

这样的 shape，保证所有请求的 cache 都放得下。但是这样的分配策略有两个问题：一方面不是每个请求都可以达到 max_seq_len，因此 KVCache 中很多的内存都被浪费掉了另一方面即使一些请求输出长度很短，它们仍然需要等待输出较长的请求结束后才能返回。

Continuous Batching 是一种动态调整批量推理策略，旨在解决 Static Batching 中的效率问题。在 Static Batching 中，为了确保所有请求的 KV Cache 能够容纳，分配固定的最大序列长度（max_seq_len）导致了内存浪费和推理延迟。Continuous Batching 通过以下方式优化这一过程：Continuous Batching 策略允许在推理过程中动态地调整批处理的大小，使得每个请求都能在完成自己的推理后立即释放 KVCache 空间。这样，新的请求可以立即使用这些空间，而不需要等待整个批次的请求都完成。这种方法提高了资源利用率，并减少了等待时间。

Continuous Batching 的原理

- 动态分配：在推理开始时，Continuous Batching 会根据当前请求的序列长度动态分配 KV Cache 空间。这意味着每个请求的 KV Cache 大小会根据其实际需要进行调整。
- 即时释放：一旦某个请求完成推理，其占用的 KV Cache 空间会被立即释放。这样，即使在处理长序列请求时，短序列请求也可以快速完成并释放资源。
- 新请求接替：释放的空间可以被新的请求立即占用，从而实现连续的推理过程。这减少了空闲等待时间，提高了整体的推理效率。

Continuous Batching 的通过动态分配 KVCache，Continuous Batching 减少了不必要的内存浪费，提高了资源的利用率。并且请求可以在完成推理后立即释放资源，后续请求无需等待，从而降低了用户的等待时间。而且 Continuous Batching 策略适用于各种长度的请求，使得推理过程更加灵活和高效。图 10 为使用 Static Batching 和 Continuous Batching 的处理方式。

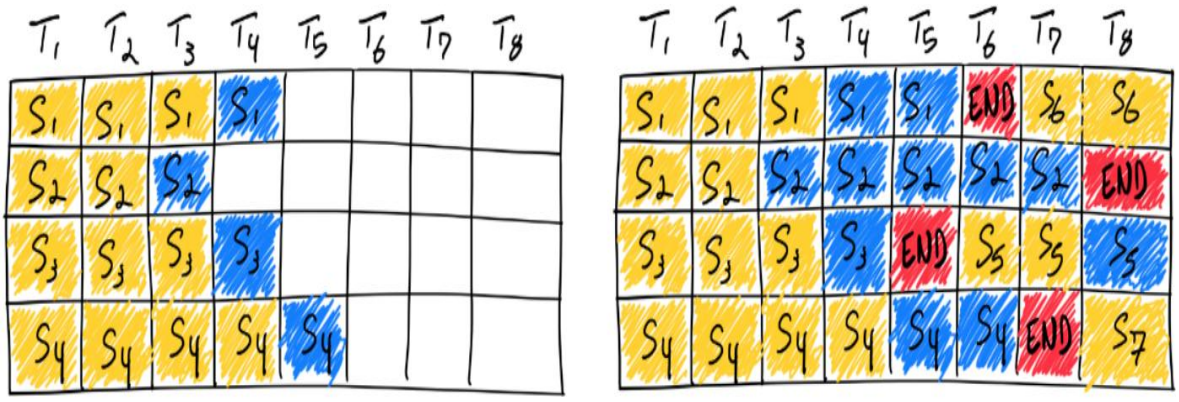

图 10 Continuous Batching 示例，黄色为 prompt，蓝色为生成的 token

在批量推理场景中，Continuous Batching 可以将模型的吞吐量提升两到三倍。当前主流的推理框架比如 Huggingface TGI, Ray serve, vllm, TensorRT-LLM 等都支持 Continuous Batching 策略。通过 Continuous Batching，我们可以在保持推理效率的同时，更好地适应不同长度的请求，这对于提高用户体验和系统的整体性能具有重要意义。

在做性能优化时，我们除了要考虑单纯计算的速度，也要考虑内存访问的速度。一方面，和 CPU 缓存一样，GPU 也有类似 L1、L2 这样的分级缓存（SRAM 和 HBM），级数越低的缓存大小越小，访问速度越快，因此我们在优化模型推理时也要考虑内存访问的局部性（Cache Locality）。另一方面，KVCache 随着 batch size 和 seq len 的增加而扩张，在推理过程中会占据较多的内存，可能会出现因为内存不够用而限制最大并发度的问题。在并发度较高或者输入输出长度较大时，内存访问反而可能成为计算的瓶颈，而非 CPU/GPU 的计算量。

# 5、实验

## 5.1 实验设计

### 5.1.1 实验目标

本实验旨在评估 KV Cache、Flash Attention 等计算加速技术在提升大语言模型）训练效率和推理速度方面的实际效果。通过比较使用相关技术前后的训练时间判断来评测这些技术在减少内存占用、降低训练时间以及提高模型并行性方面的贡献。

### 5.1.2 实验环境

硬件配置：实验在配备两块 NVIDIA A100 GPU 的高性能计算集群上进行。每个 GPU 拥有 40GB 显存。

软件环境：操作系统为 Red Hat Enterprise Linux3.10.0-693 使用 PyTorch 2.0.1 作为深度学习框架，DeepSpeed 0.12.6 作为模型训练的分布式训练框架。

### 5.1.3 实验方法

KV Cache：考虑到大模型的训练资源以及服务器节点的不稳定性，在 Transformer 模型的解码阶段，使用 KV Cache 来存储和重用已经计算过的 K 和 V 矩阵以加快训练速度，尽量避免训练过程中由服务器不稳定性引起的重新训练。KV Cache 的设置如图 11 所示


图 11 kv Cache 设置

FlashAttention：DeepSpeed 框架提供了 FlashAttention 支持，如图 12 所示只需要在使用时进行设置即可选择是否使用 FlashAttention。

```
model = model_class.from_pretrained(
    model_name_or_path,
    from_tf=bool(".ckpt" in model_name_or_path),
    config=model_config,
    trust_remote_code=True,
    use_flash_attention_2=True,
)
```

图 12 FlashAttention 设置

## 5.2 实验过程

### 5.2.1 数据及模型准备

数据集：本文在微调过程中使用的数据集来自 Chinese Open Instruction Generalist（COIG）相比其他的中文指令数据集，COIG 数据集在领域适应性、

多样性、数据质量等方面具有一定的优势。目前 COIG 数据集主要包括：通用翻译指令数据集、考试指令数据集、价值对其数据集、反事实校正数据集、代码指令数据集。该数据集的数据格式如图 11 所示

模型选择：为了保证所有实验环节能够正常执行，本文选择了对硬件要求较低的零一万物 Yi-6B-Chat 模型，它具有接近 60 亿个参数。

### 5.2.2 实施步骤

首先，初始化模型参数，使用 KV Cache，分别设置模型使用 FlashAttention 和不使用 FlashAttention 然后通过数据并行在多个 GPU 上进行微调。在微调过程中，记录了训练时间、内存使用情况以及模型的损失值。微调完成后，通过分别向初始模型与微调后的模型进行提问，比较微调后是否起到作用。

## 5.3 结果分析

通过对比实验，发现使用在都使用 KV Cache 的情况下如果不使用 FlashAttention 一次微调所需要的时间平均在 38 分钟，如果使用 FlashAttention，一次微调所需要的时间平均在 28 分钟，训练速度提升了约 26%。

表 1 对 Yi-6B-Chat 微调所用时间

| | 实验 1 | 实验 2 | 实验 3 | 实验 4 |
|---|---|---|---|---|
| 使用 FlashAttention | 28 | 30 | 28 | 27 |
| 未使用 FlashAttention | 38 | 38 | 40 | 37 |

在实验中每一个 epoch 中计算是每块 A100 所需要的显存在 26GB 左右，这已经超出了消费级显卡的最大显存容量，可以看出针对大语言模型的训练几微调仍需要继续优化。

微调后，针对模型提问效果如图 13、14 所示，可以看出基础模型虽能回答出关于节日信息的部分内容，但相比微调后的模型，他缺少的具体的节日时间等信息，并且输出格式，由此可见微调后的模型相比基础模型有所改善。

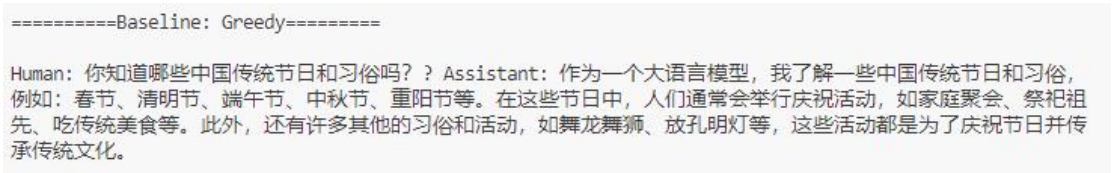


图 13 基础模型回答结果

```
==========finetune: Greedy==========

Human: 你知道哪些中国传统节日和习俗吗？？Assistant: 中国的传统节日和习俗众多，以下是一些典型的例子：

1. 春节（农历正月初一）：这是中国最重要的节日，象征着新的一年的开始，人们会举行各种庆祝活动，如贴春联、放鞭炮、吃饺子、穿新衣、拜年等。

2. 清明节（农历三月三日）：又称寒食节，是中国传统的扫墓节日，人们会去墓地祭扫，缅怀先人。

3. 端午节（农历五月初五）：端午节的主要习俗是赛龙舟和吃粽子，相传是为了纪念战国时期楚国大夫屈原。

4. 中秋节（农历八月十五）：中秋节的主要习俗是赏月、吃月饼、品酒、玩花灯等，象征着阖家团圆。

5. 重阳节（农历九月初九）：重阳节的主要习俗是登高
```

图 14 微调后模型输出结果

本实验表明，KV Cache、Flash Attention 等计算加速技术能够有效提升大语言模型的训练和推理效率。这些技术不仅提高了模型的运行速度，还降低了对硬件资源的需求。尽管在实施过程中遇到了一些挑战，但通过不断的和调整，我们成功地解决了这些问题。未来的工作将集中在进一步优化这些技术，以及探索新的加速策略，以实现更高效的 AI 模型训练和部署。

## 6、总结与展望

经过实验研究和数据分析，本得出了一系列关于“零一万物”大语言模型在计算加速技术应用方面的结论。首先，KV Cache 和 FlashAttention，能够在不牺牲模型性能的前提下，显著降低模型的存储和计算需求。这一发现对于资源受限的环境尤为重要，使得“零一万物”模型能够更广泛地部署在各种硬件平台上。其次，推理优化技术，展现了在提高模型推理效率方面的潜力。这些技术通过减少重复计算和动态调整计算资源，使得模型在处理复杂任务时更加灵活和高效。特别是在 GPU 等专用硬件上的应用，在推理速度的提升非常显著，这对于需要快速响应的应用场景具有重要意义。在分布式推理方面，我们的实验结果表明，通过在多个计算节点上并行处理任务，可以显著提高系统的吞吐量，同时保持模型性能。这一策略对于处理大规模数据集和高并发请求的 NLP 应用尤为关键，为构建可扩展的智能系统提供了有效途径。

尽管我们的研究取得了积极成果，但仍存在一些挑战和未来研究方向。首先，模型压缩技术在进一步减少模型大小的同时，如何确保模型在特定任务上的准确性和泛化能力，是一个值得深入研究的问题。其次，随着硬件技术的不断进步，如何设计更高效的硬件架构来充分利用这些硬件优势，以及如何实现硬件与软件的深度协同，是未来研究的重要方向。

此外，分布式推理在实际部署中的稳定性和可扩展性也需要进一步优化。在多节点环境中，如何确保任务分配的公平性和节点间的高效通信，以及如何处理节点故障，都是需要解决的问题。最后，随着模型规模的不断扩大，如何平衡模型的泛化能力和特定任务的性能，以及如何确保模型的可解释性和安全性，也是

未来研究的关键点。

未来的工作将集中在以下几个方面：使用新的优化算法，继续探索新的模型优化算法在训练大模型时的加速效果，以实现更高效的资源利用和更优的性能平衡。增加系统稳定性与可扩展性在分布式推理环境中，研究如何提高系统的鲁棒性和容错能力，以及如何优化任务分配和节点通信。模型泛化与可解释性：探索模型泛化能力的增强方法，同时提高模型的可解释性，以满足实际应用中对透明度的需求。另外，我会尝试使用更大的数据集对模型进行微调，是模型能够达到较好的交互效果。

随着人工智能技术的不断发展，我们期待大语言模型模型能够在计算加速技术的推动下，实现更广泛的应用，为人类社会带来更多智能化的服务和解决方案。同时，我们也期待通过不断的研究和创新，克服现有挑战，推动大型语言模型技术向更高效、更智能、更安全的方向发展。

# 参考文献


[1] Vaswani A , Shazeer N , Parmar N ,et al.Attention Is All You Need[J].arXiv, 2017.DOI:10.48550/arXiv.1706.03762.

[2] Dong L , Yang N , Wang W ,et al.Unified Language Model Pre-training for Natural Language Understanding and Generation[J]. 2019.DOI:10.48550/arXiv.1905.03197.

[3] Devlin J , Chang M W , Lee K ,et al.BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[J]. 2018.

[4] 柯沛,雷文强,黄民烈.以 ChatGPT 为代表的大型语言模型研究进展[J].中国科学基金, 2023, 37(5):714-723.

[5] Hu E J , Shen Y , Wallis P ,et al.LoRA: Low-Rank Adaptation of Large Language Models[J]. 2021.DOI:10.48550/arXiv.2106.09685.

[6] Child R , Gray S , Radford A ,et al.Generating Long Sequences with Sparse Transformers[J]. 2019.DOI:10.48550/arXiv.1904.10509.

[7] Rajbhandari S , Rasley J , Ruwase O ,et al.ZeRO: Memory Optimization Towards Training A Trillion Parameter Models[J]. 2019.DOI:10.48550/arXiv.1910.02054.

[8] Yu Z , Pengyuan Z , Yonghong Y .Long short-term memory with attention and multitask learning for distant speech recognition[J].Qinghua Daxue Xuebao/Journal of Tsinghua University, 2018, 58(3):249-253.DOI:10.16511/j.cnki.qhdxxb.2018.25.016.

[9] Press O , Smith N A , Lewis M .Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation[J]. 2021.DOI:10.48550/arXiv.2108.12409.

[10] Waddington D , Colmenares J , Kuang J ,et al.KV-Cache: A Scalable High-Performance Web-Object Cache for Manycore[C]//Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing.ACM, 2013.DOI:10.1109/UCC.2013.34.

[11] 李天健，林达华．一种针对超大模型的分布式推理部署系统.CN202211373207.4[2024-01-25].