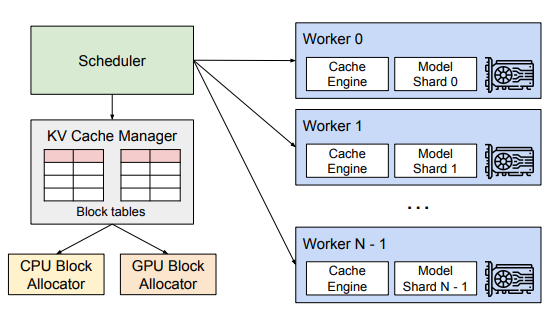
# 一、业界推理优化算法调研

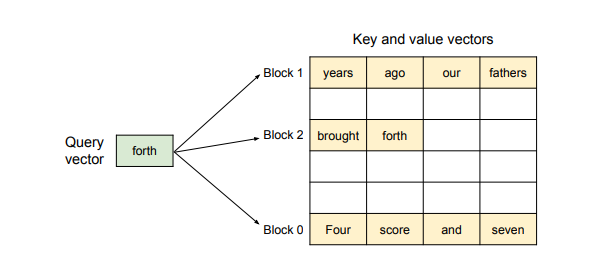
最新的高效推理系统vLLM采用paged attention、KV-Cache压缩、模型量化和连续批处理（Continues Batching）等技术。



在内存管理效率低下的情况下，内存会因为碎片化和冗余复制而被显著浪费，从而限制批处理大小。为了解决这个问题，vLLM提出了PagedAttention，这是一种受到操作系统中的经典虚拟内存和分页技术启发的注意力算法。在此基础上，vLLM实现了KV缓存内存的近乎零浪费，以及在请求内部和跨请求之间灵活共享KV缓存，从而进一步减少内存使用。其的评估显示，与最先进的系统（如FasterTransformer和Orca）相比，vLLM在相同的延迟水平下，将流行大语言模型（LLM）的吞吐量提高了2-4倍。随着序列变长、模型变大和解码算法变得更复杂，这一改进更为显著。

下面具体介绍vLLM中使用的优化推理技术：

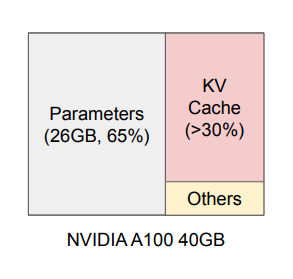
**Paged attention：**Paged attention是一种用于优化大规模神经网络（如大型语言模型）中注意力机制的方法。传统的注意力机制在处理非常长的序列时，计算和内存的需求会显著增加，paged attention旨在解决这些问题，提高计算效率和内存利用率。



它首先将输入序列划分为较小的块或页（pages）。每个块内的注意力计算是独立进行的，从而减少了需要同时处理的元素数量。然后在每个块内使用局部注意力机制，仅计算块内元素之间的相互关系，定期在整个序列范围内引入全局注意力，以捕捉不同块之间的长距离依赖关系。

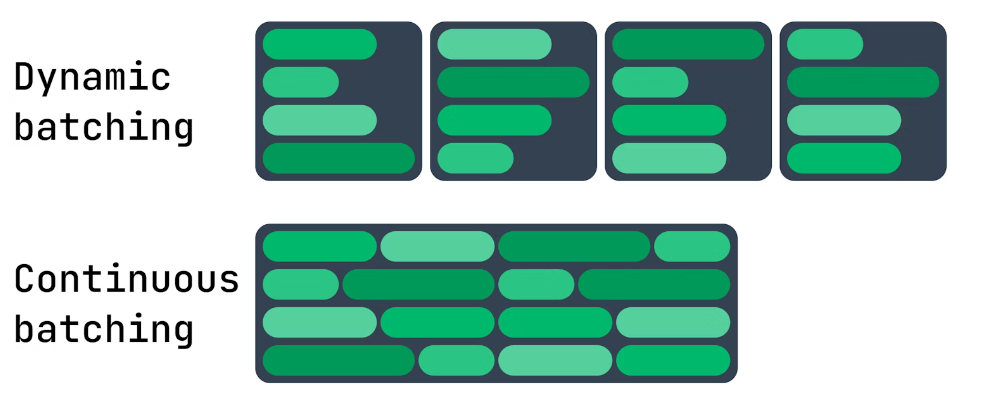
通过限制每次注意力计算的范围，Paged attention大大减少了内存的使用。对于非常长的序列，这种方法尤其有效，因为传统的全局注意力机制通常需要的内存是序列长度的平方级别。Paged attention适用于处理非常长的文本、时间序列数据或其他需要长序列建模的任务。通过分块和局部注意力，可以有效避免传统方法中的计算瓶颈。

**KV-Cache压缩：**KV-Cache压缩是一种优化大型语言模型（如Transformer）在推理过程中内存使用和计算效率的方法。



在自回归生成任务中，模型需要在每个步骤都能访问前面的上下文信息，以计算注意力得分。为此，模型会缓存前面生成步骤的键和值，这些缓存被称为KV-Cache。当序列变长时，KV-Cache的大小会迅速增加，占用大量内存，如上图所示。KV-Cache压缩通过减少存储的键和值的数量来缓解这一问题。常用的技术包括聚合和截断。聚合将相邻的多个键和值合并，形成新的聚合键和值，从而减少存储需求。截断则直接移除不太重要的键和值，只保留最相关的信息。尽管这些方法可能会丢失一些细节信息，但它们在保持模型性能的同时显著降低了内存消耗，提高了计算效率。通过这种压缩技术，KV-Cache能够更有效地在内存受限的环境中运行，如移动设备或边缘计算场景。压缩后的KV-Cache减少了计算注意力得分时需要处理的数据量，从而加速了推理过程。在自然语言处理任务中，如文本生成和翻译，KV-Cache压缩使得模型可以处理更长的文本，同时减少内存占用。

**连续批处理：**Continues Batching压缩是一种优化大规模神经网络处理长序列数据时的内存和计算效率的方法。在传统的序列处理过程中，模型需要逐步处理每个时间步的数据，这会导致内存和计算资源的高消耗。



Continues Batching通过将长序列数据分批次处理，将连续的多个时间步的数据批量化，从而减少模型在每个时间步所需的计算和内存资源。这种方法利用了数据的时序连续性，通过合并多个时间步的数据来减少计算频率和内存访问次数，从而提升处理效率。

具体而言，Continues Batching将长序列划分为若干较小的批次，每个批次包含多个连续的时间步。然后，模型同时处理这些批次中的所有时间步，而不是逐步处理每个时间步。这样可以减少模型的计算频率和内存访问次数，提高整体处理效率。同时，通过对连续时间步的数据进行压缩和合并，可以进一步减少内存使用量。这种方法特别适用于处理长序列数据的任务，如自然语言处理、时间序列分析等，使得模型在内存受限的环境中仍能高效运行。

**模型量化：**

模型量化是一种用于减少深度学习模型的计算和内存需求的技术，通过将模型权重和激活值从高精度（如32位浮点数）降低到低精度（如8位整数）。这种方法不仅减少了存储需求，还加速了推理过程，从而使模型更适合部署在资源受限的设备上，如移动设备和嵌入式系统。

模型量化包括动态量化和静态量化。动态量化在推理过程中动态地将权重和激活值从高精度转换为低精度，适用于推理阶段，因为它不需要对模型进行重新训练，可以显著减少计算和内存需求，特别适用于CPU推理。静态量化在模型训练后进行量化，通过校准数据来确定量化参数（如缩放因子和零点），需要在推理前进行一次离线校准，适用于需要高性能的推理，因为它在推理过程中无需额外的转换开销。

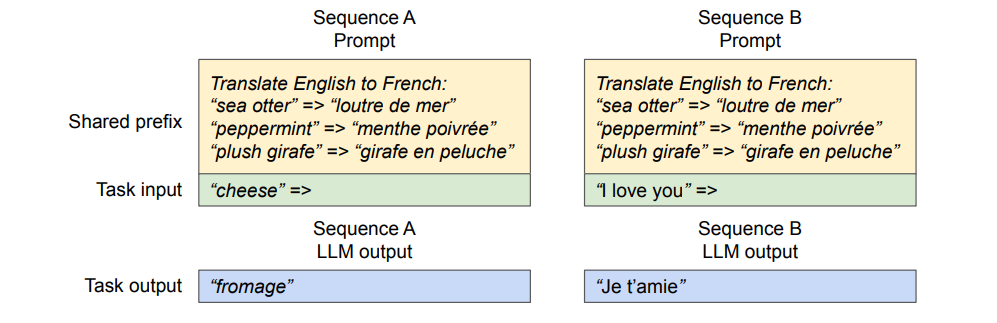
量化将浮点数映射到固定范围的整数，通过缩放因子实现精度调整。例如，将32位浮点数转换为8位整数，使用缩放因子和零点进行表示。在量化时需要确定每层神经网络的缩放因子和零点，以最小化量化误差，通常通过一小部分校准数据来估计这些参数。

此外，量化还可以分为对称量化和非对称量化。对称量化将零点固定为0，仅调整缩放因子。非对称量化的零点可变，以更精确地表示数据范围。

模型量化可能导致模型性能下降，特别是在低精度（如4位或更低）时。为此，可以使用QAT进行训练，或选择混合精度量化（部分层使用高精度，部分层使用低精度）。

模型量化在各类实际应用中广泛使用，例如在手机、平板等设备上部署深度学习模型，以实现实时图像处理、语音识别等功能。在物联网设备、智能家居设备等资源受限的环境中部署高效的模型。在数据中心中使用量化模型来降低成本，提高服务的吞吐量和响应速度。通过模型量化，可以显著降低模型的计算和存储需求，使得复杂的深度学习模型能够在资源受限的环境中高效运行，同时保持较高的精度和性能。

**Prefix共享：**通常，LLM 用户会提供任务的（长）描述，包括说明和示例输入和输出，也称为系统提示。描述与实际任务输入连接在一起，形成请求的提示，如下图所示。



对于此类应用程序，许多用户提示共享一个前缀，因此可以预先存储前缀的 KV 缓存，以减少在前缀上花费的冗余计算。具有共享前缀的用户输入提示可以简单地将其逻辑块映射到缓存的物理块（最后一个块标记为写时复制）。提示阶段计算只需要在用户的任务输入上执行。

# 二、本作品使用的推理优化算法介绍

1. Pd分离

将预填充（Prefill）和解码（Decoding）阶段分离，以优化计算效率和内存使用。这个技术特别在推理过程中有效，通过分别处理和优化这两个阶段，提升了模型生成长文本的性能。

预填充阶段涉及模型输入序列的预处理和缓存。通常包括将输入文本转换为内部表示，并计算初始的注意力键值对（KV-Pairs），以便后续生成步骤使用。

解码阶段是模型实际生成文本的过程。每一步生成一个新词，并使用先前生成的词和预填充阶段的缓存结果进行计算。

预填充阶段的缓存结果可以重复使用，避免了在解码过程中不断重新计算大规模的注意力矩阵。

此外，预填充阶段由于输入序列更长，其显存占用要大于解码阶段，可以减小一些预填充阶段的batch size。

1. Continues batching

连续批处理，当一个batch中较短的序列生成结束后可以接入一个新的请求。

1. 模型量化：采用fp16精度
2. 调整batch size，尽量提高显卡利用率（在所需计算量相同的情况下，显卡利用率越高，推理速度越快）

# 三、超参配置介绍

对于llama本身的超参，我们未作改动；对于llm-serving的超参，我们设置其perfill\_batch\_size为16，decode\_batch\_size为64，model\_dtype为DataType.FLOAT16。其余超参我们未作改动，整体见：配置文件.yaml

对于performance\_serving中test\_serving\_performance.py的超参，我们设置X=5,T=300。

# 四、优化后的推理总时长

总时长为823.84s。

# 五、运行环境说明

与文档中相同，未作改动。