KVCache优化方法文献综述

## 1.1 Mooncake

Mooncake（应用于Kimi）：核心是一个全局调度器（命名为Conductor），基于当前KVCache和工作负载的分配情况，负责调度请求。Mooncake采用Prefill、Decoding分开的架构，并且利用CPU内存和SSD构建分层缓存，并通过预测性调度策略（如热块复制、早期请求拒绝）优化过载场景的吞吐量。实验表明，Mooncake在长上下文场景下表现出色，其支持200万Token上下文，在真实负载下吞吐量提升75%。

图示

AI 生成的内容可能不正确。

Mooncake采用了一种分离架构，不仅将预填充节点与解码节点分离，还将GPU集群中的CPU、DRAM、SSD和RDMA资源分组，以实现分离的KVCache缓存。这种分离架构利用未充分利用的资源，提供充足的缓存容量和传输带宽，从而实现高效的近GPU前缀缓存，而无需额外成本。

### 1.1.1 Mooncake架构

Mooncake的架构包括以下几个主要组件：

（1）预填充节点池：负责处理请求的预填充阶段，生成KVCache。Mooncake的预填充池采用了分块管道并行（CPP）机制，以扩展单个请求的处理能力，从而减少长上下文输入的TTFT。与传统的序列并行（SP）解决方案相比，CPP减少了网络消耗，并简化了对频繁弹性扩展的依赖。此外，Mooncake还采用了分层预填充机制，使得KVCache的流传输能够重叠延迟，进一步优化了预填充阶段的性能。

（2）解码节点池：负责处理请求的解码阶段，利用KVCache生成输出。

（3）全局调度器（Conductor）：负责根据KVCache的分布和工作负载调度请求，并在必要时复制或交换KVCache块。

### 1.1.2 KVCache-centric调度算法：

Mooncake的调度算法旨在平衡实例负载和用户体验，特别是TTFT和TBT SLO。该算法包括一个基于启发式的自动热点迁移方案，该方案无需精确预测未来的KVCache使用情况即可复制热点KVCache块。实验结果表明，这种缓存感知调度可以显著降低实际场景中的TTFT。

### 1.1.3 过载场景下的调度：

与现有研究不同，Mooncake面临的是高度过载的场景。为了应对这些挑战，Mooncake开发了一种基于预测的早期拒绝策略，以减少在过载情况下的计算资源浪费。实验表明，Mooncake在长上下文场景中表现出色，与baseline相比，在某些模拟场景中可以实现高达525%的吞吐量提升，同时遵守SLO。在实际工作负载下，Mooncake的创新架构使Kimi能够处理75%以上的请求。

## 1.2 Infinite-LLM

Infinite-LLM（阿里巴巴）提出基于DistAttention的集群级KVCache调度，一种将attention和KVCache细分为规则的小子块的方法，从而允许注意层在多个实例中高效分布和计算。其核心创新在于将注意力层与非注意力层解耦，利用数学等效变换（如MicroAttention分块计算）减少跨节点通信开销，并通过贪心算法动态分配KVCache块，在一个包含32个A100 GPU的集群中，上下文长度从几个到 2000K 的token的数据集上进行评估。InfiniteLLM与最先进的方法相比，吞吐量提高了1.35-3.4倍，从而实现高效和弹性的 LLM 部署。该方法适用于多GPU集群环境，但对网络带宽要求较高。

### 1.2.1 主要技术

（1）解耦注意力和非注意力层：Infinite-LLM将注意力层和非注意力层分开，使得每一层可以独立进行资源调度，从而优化了内存利用和计算效率。

（2）DistAttention：该分布式注意力机制通过在GPU子块中局部进行注意力计算，仅共享必要的信息，避免了传统方法中由于大规模KV缓存传输而带来的通信开销。这种方法与传统的注意力机制数学上等价，但更具可扩展性。

（3）集群级资源调度：Infinite-LLM引入了一种集群级GPU内存池策略，使得任务所需的内存可以动态从其他GPU实例借用，从而提高了长上下文任务的内存利用率。

（4）吞吐量提升：评估结果显示，Infinite-LLM相比最先进的LLM服务方法在任务处理吞吐量方面有1.35-3.4倍的提升，尤其是在处理长上下文任务时表现尤为突出。

图形用户界面, 图示

AI 生成的内容可能不正确。

## 1.3 PQCache

PQCache引入适用于长文本的基于产品量化(Product Quantization)的KVCache，将KVCache划分为子空间并聚类，利用近似最近邻搜索（Maximum Inner-Product Search, MIPS）快速检索关键Token。其核心目标是在保持模型性能的前提下，显著降低内存占用和跨设备通信开销。实验显示，其仅需1/5的Token参与注意力计算即可保持模型性能。

图示

AI 生成的内容可能不正确。

### 1.3.1 预填充阶段：PQ结构构建

（1）键向量分块与聚类：

对每个注意力头的键向量（维度为dh）进行分块，划分为m个子空间（如m=2m=2），每个子空间维度为dm=dh/m。

对每个子空间的键向量进行K-Means聚类，生成2b个中心点（centroids）。例如，b=6时每个子空间聚类数为64。

存储每个键的PQ码（子空间聚类索引，总内存为s×m×b bits）和中心点矩阵（内存为m×2b×dm）。

（2）与计算重叠：

在GPU执行注意力计算的同时，异步在CPU进行聚类，避免增加预填充延迟。

自适应调整聚类迭代次数，确保K-Means时间不超过单层Transformer计算时间。

### 1.3.2 解码阶段：近似注意力计算

（1）查询-中心点内积计算：

对当前查询向量（Query）同样分块为m个子空间，分别与对应子空间的中心点计算内积，得到m×2b的分数矩阵。

根据PQ码快速聚合各子空间分数，生成近似注意力分数（Softmax前）。

（2）Top-k检索与数据获取：

基于近似分数选择Top-k重要Token，从CPU内存中获取对应的完整键值对。

引入GPU块级缓存：将高频访问的Token块（如局部窗口内的Token）保留在GPU，采用LRU/LFU策略管理，减少PCIe传输。

通信与计算重叠：

预取PQ码和中心点，与GPU解码计算并行执行。异步加载检索到的键值对，隐藏数据传输延迟。

## 1.4 H2O

H2O(Heavy-Hitter Oracle)，基于注意力分数动态丢弃低权重token，保留得分高的token(称为Heavy Hittters)，从而减少KVCache的内存占用。但可能因信息丢失导致模型性能下降。

### 1.4.1 主要贡献

（1）重击词（H2）识别与缓存优化：论文发现，LLM 的注意力矩阵具有高度稀疏性，大多数注意力分数的贡献来自少数几项，称为“重击词”。这些重击词显著影响生成效果，因此对其进行保留至关重要。

（2）H2O驱逐策略：提出了一种基于H2的简单、低成本的缓存驱逐策略，称为H2O。该策略在每一步推理过程中根据局部统计信息动态选择保留的KV对，从而大幅减少了KV缓存的内存占用。

图示

AI 生成的内容可能不正确。

## 1.5 PowerInfer

PowerInfer的设计理念基于LLM推理过程中神经元激活的局部性特征。通过观察到神经元激活遵循幂律分布，少数“热激活神经元”在多个输入中持续被激活，而大部分“冷激活神经元”根据特定输入进行激活，PowerInfer通过预加载“热激活神经元”到GPU中，将“冷激活神经元”留给CPU计算，成功减轻了GPU内存压力并提高了推理速度。

### 1.5.1 主要贡献

（1）GPU-CPU混合推理引擎：PowerInfer采用GPU-CPU混合推理的方法，将“热激活神经元”加载到GPU中进行快速计算，而将“冷激活神经元”转移到CPU上处理。这种方式不仅减轻了GPU内存压力，还提高了数据传输效率，降低了GPU和CPU之间的传输延迟。

（2）神经元激活预预测技术：PowerInfer引入了在线预测器，在推理过程中根据每层的激活模式预测哪些神经元会被激活。仅计算那些被预测为激活的神经元，避免了对不活跃神经元的计算，从而进一步加速了推理过程。

（3）优化的神经元分配策略：PowerInfer设计了一种基于整数线性规划（ILP）的神经元分配策略，旨在将活跃度较高的神经元优先分配到GPU上进行处理，从而最大化GPU的计算效能，同时避免过多的数据传输。

（4）高效的稀疏矩阵计算：通过引入针对神经元的稀疏矩阵计算操作，PowerInfer 能够高效处理稀疏激活的神经元，减少了传统矩阵操作中的冗余计算，提升了性能。

### 1.5.2 系统设计与挑战

PowerInfer的核心设计思想是利用神经元激活的稀疏性和局部性来优化LLM推理。每个推理步骤只计算那些预测为激活的神经元，尤其是在小批量请求和低延迟场景中，GPU和CPU的资源使用更加高效。然而，实现这一目标也面临了一些挑战：

（1）在线预测器的内存占用：预测器用于识别每个层的活跃神经元，但它们会占用大量GPU内存。PowerInfer通过自适应调整预测器大小，并根据层的稀疏性调整其复杂度，有效解决了这一问题。

（2）稀疏操作的优化：传统的稀疏矩阵计算库往往不能很好地适应动态稀疏性，PowerInfer通过设计“神经元感知稀疏操作”直接与单个神经元交互，避免了不必要的矩阵格式转换，提高了计算效率。

## 参考文献：

SONG Y, MI Z, XIE H, et al. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU \*[J].

FENG Y, LV J, CAO Y, et al., 2024. Ada-KV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference[J].

XIAO C, ZHANG P, HAN X, et al., 2024. InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory[J].

ZHANG Z, SHENG Y, ZHOU T, et al., 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models[J].

JIN Y, WANG T, LIN H, et al. P/D-Serve: Serving Disaggregated Large Language Model at Scale[J].

ZHANG H, JI X, CHEN Y, et al. PQCache: Product Quantization-based KVCache for Long Context LLM Inference[J].

Anon. Infinite-LLM: Efficient LLM Service for Long Context with DistAttention and Distributed KVCache[J].

QIN R, LI Z, HE W, et al. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving[J].

SONG Y, MI Z, XIE H, et al. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU \*[J].