Paper Review: Efficient Memory Management for Large Language Model Serving with PagedAttention

Summary: This paper tackles poor GPU memory use when serving LLMs by rethinking how the model's key-value (KV) cache is stored. Prior systems (FasterTransformer for low latency and Orca for high throughput with iteration-level scheduling) keep each request's KV cache in one big contiguous chunk, causing internal fragmentation (waste inside a reserved chunk because you over-allocated) and external fragmentation (scattered holes between chunks you cannot pack a new request into), which limits batch size and throughput. The paper proposes PagedAttention, which splits the KV cache into small fixed-size blocks (like OS paging) so blocks can live non-contiguously, enabling flexible growth, near-zero waste, and sharing. The authors build vLLM, a serving system that supports parallel sampling (multiple outputs from the same prompt reuse the prompt's KV blocks), shared prefix caching (provider-preloaded instruction prefixes reused across users), and beam search (candidates share blocks and diverge safely using reference counting and copy-on-write, meaning only the modified block is duplicated). When GPU memory runs low, vLLM can swap KV blocks to CPU RAM (temporarily offload them) or recompute them later (rebuild KV instead of transferring large data). It also supports model parallelism across GPUs (Megatron-style tensor parallelism with a centralized KV block map) and speeds up scattered memory access with custom fused CUDA kernels that combine block read and attention operations in one kernel to reduce overhead. Experiments on OPT-13B, OPT-66B, OPT-175B, and LLaMA-13B with ShareGPT and Alpaca workloads show that vLLM achieves 2x-4x higher throughput at similar latency compared to FasterTransformer and Orca, with 96%+ memory utilization, especially for longer sequences and advanced decoding methods like beam search and parallel sampling.

Evaluation: The paper identifies a clear and impactful bottleneck in large language model serving inefficient GPU memory usage caused by key-value (KV) cache fragmentation. The authors present a well-structured and technically sound solution with PagedAttention, which adapts the concept of virtual memory and paging from operating systems to GPU-based inference. The problem is significant because memory limits directly affect the number of concurrent requests and overall throughput in real-world systems. The proposed approach is both novel and practical, combining memory paging, block-level sharing, and copy-on-write techniques to achieve efficient utilization. The evaluation setup is fair and diverse, testing across multiple model sizes, workloads, and decoding methods. Key metrics such as throughput (requests per second), latency per token, and memory efficiency are clearly reported. The results consistently validate the claimed improvements. Comprehensive ablation studies further analyze block size, swapping versus recomputation, and kernel optimizations, highlighting careful system design and thoughtful trade-off analysis. Overall, the paper is well structured and clearly explains both the core system concepts and their practical application in large language model serving.

Main takeaways

1. Memory, not compute, is the main bottleneck in large language model (LLM) serving, most GPU memory is consumed by the dynamic key-value (KV) cache, not by the model weights
2. PagedAttention enables sharing of memory blocks, cutting memory use by up to 55% for complex decoding methods like beam search, parallel sampling, and shared prefixes.

Strengths

1. PagedAttention introduces a highly novel and effective mechanism for LLM serving by applying OS-like paging to the KV cache, resolving significant memory fragmentation issues in existing systems
2. This serving system demonstrates impressive practical results, achieving a significant 2-4x throughput boost over prior state-of-the-art implementations, making it an essential optimization for serving large models.

Weaknesses

1. The swapping method is slow when blocks are very small because it has to move many tiny pieces of data between the CPU and GPU, which wastes PCIe bandwidth. In such cases, recomputing the data is usually faster
2. While throughput gains are clear, the paper offers less discussion of latency variability and tail-latency behavior under real request spikes.

Discussion

1. The paper focuses on transformer-based inference. How could the idea of paged, shared memory be adapted in other contexts like training, diffusion models, or vision transformers?