Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Woosuk Kwon[1,*] Zhuohan Li[1,*] Siyuan Zhuang[1] Ying Sheng[1,2] Lianmin Zheng[1] Cody Hao Yu[3]
Joseph E. Gonzalez[1] Hao Zhang[4] Ion Stoica[1]
[1]UC Berkeley [2]Stanford University [3]Independent Researcher [4]UC San Diego

Abstract

High throughput serving of large language models (LLMs) requires batching sufficiently many requests at a time. However, existing systems struggle because the key-value cache (KV cache) memory for each request is huge and grows and shrinks dynamically. When managed inefficiently, this memory can be significantly wasted by fragmentation and redundant duplication, limiting the batch size. To address this problem, we propose PagedAttention, an attention algorithm inspired by the classical virtual memory and paging techniques in operating systems. On top of it, we build vLLM, an LLM serving system that achieves (1) near-zero waste in KV cache memory and (2) flexible sharing of KV cache within and across requests to further reduce memory usage. Our evaluations show that vLLM improves the throughput of popular LLMs by 2-4× with the same level of latency compared to the state-of-the-art systems, such as FasterTransformer and Orca. The improvement is more pronounced with longer sequences, larger models, and more complex decoding algorithms. vLLM's source code is publicly available at https://github.com/vllm-project/vllm.

1 Introduction

The emergence of large language models (*LLMs*) like GPT [5, 37] and PaLM [9] have enabled new applications such as programming assistants [6, 18] and universal chatbots [19, 35] that are starting to profoundly impact our work and daily routines. Many cloud companies [34, 44] are racing to provide these applications as hosted services. However, running these applications is very expensive, requiring a large number of hardware accelerators such as GPUs. According to recent estimates, processing an LLM request can be 10× more expensive than a traditional keyword query [43]. Given these high costs, increasing the throughput—and hence reducing

SOSP '23, October 23–26, 2023, Koblenz, Germany

ACM ISBN 979-8-4007-0229-7/23/10.
https://doi.org/10.1145/3600006.3613165

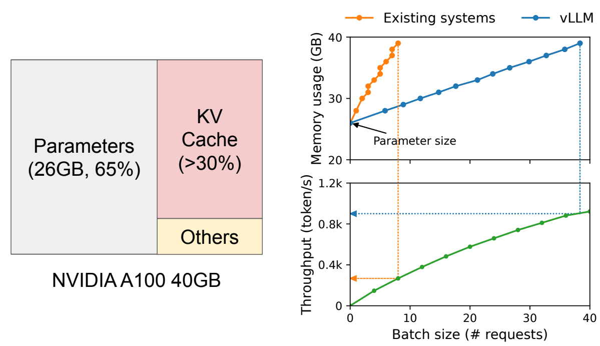


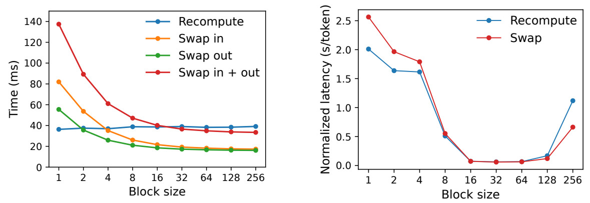
Figure 1. *Left:* Memory layout when serving an LLM with 13B parameters on NVIDIA A100. The parameters (gray) persist in GPU memory throughout serving. The memory for the KV cache (red) is (de)allocated per serving request. A small amount of memory (yellow) is used ephemerally for activation. *Right:* vLLM smooths out the rapid growth curve of KV cache memory seen in existing systems [31, 60], leading to a notable boost in serving throughput.

the cost per request—of *LLM serving* systems is becoming more important.

At the core of LLMs lies an autoregressive Transformer model [53]. This model generates words (tokens), *one at a time*, based on the input (prompt) and the previous sequence of the output's tokens it has generated so far. For each request, this expensive process is repeated until the model outputs a termination token. This sequential generation process makes the workload *memory-bound*, underutilizing the computation power of GPUs and limiting the serving throughput.

Improving the throughput is possible by batching multiple requests together. However, to process many requests in a batch, the memory space for each request should be efficiently managed. For example, Fig. 1 (left) illustrates the memory distribution for a 13B-parameter LLM on an NVIDIA A100 GPU with 40GB RAM. Approximately 65% of the memory is allocated for the model weights, which remain static during serving. Close to 30% of the memory is used to store the dynamic states of the requests. For Transformers, these states consist of the key and value tensors associated with the attention mechanism, commonly referred to as *KV cache* [41], which represent the context from earlier tokens to generate new output tokens in sequence. The remaining small

[*]Equal contribution.



(a) Microbenchmark **(b)** End-to-end performance

Figure 19. (a) Overhead of recomputation and swapping for different block sizes. (b) Performance when serving OPT-13B with the ShareGPT traces at the same request rate.

7.3 Comparing Recomputation and Swapping

vLLM supports both recomputation and swapping as its recovery mechanisms. To understand the tradeoffs between the two methods, we evaluate their end-to-end performance and microbenchmark their overheads, as presented in Fig. 19. Our results reveal that swapping incurs excessive overhead with small block sizes. This is because small block sizes often result in numerous small data transfers between CPU and GPU, which limits the effective PCIe bandwidth. In contrast, the overhead of recomputation remains constant across different block sizes, as recomputation does not utilize the KV blocks. Thus, recomputation is more efficient when the block size is small, while swapping is more efficient when the block size is large, though recomputation overhead is never higher than 20% of swapping's latency. For medium block sizes from 16 to 64, the two methods exhibit comparable end-to-end performance.

8 Discussion

Applying the virtual memory and paging technique to other GPU workloads. The idea of virtual memory and paging is effective for managing the KV cache in LLM serving because the workload requires dynamic memory allocation (since the output length is not known a priori) and its performance is bound by the GPU memory capacity. However, this does not generally hold for every GPU workload. For example, in DNN training, the tensor shapes are typically static, and thus memory allocation can be optimized ahead of time. For another example, in serving DNNs that are not LLMs, an increase in memory efficiency may not result in any performance improvement since the performance is primarily compute-bound. In such scenarios, introducing the vLLM's techniques may rather degrade the performance due to the extra overhead of memory indirection and non-contiguous block memory. However, we would be excited to see vLLM's techniques being applied to other workloads with similar properties to LLM serving.

LLM-specific optimizations in applying virtual memory and paging. vLLM re-interprets and augments the idea of virtual memory and paging by leveraging the application-specific semantics. One example is vLLM's all-or-nothing swap-out policy, which exploits the fact that processing a request requires all of its corresponding token states to be stored in GPU memory. Another example is the recomputation method to recover the evicted blocks, which is not feasible in OS. Besides, vLLM mitigates the overhead of memory indirection in paging by fusing the GPU kernels for memory access operations with those for other operations such as attention.

9 Related Work

General model serving systems. Model serving has been an active area of research in recent years, with numerous systems proposed to tackle diverse aspects of deep learning model deployment. Clipper [11], TensorFlow Serving [33], Nexus [45], InferLine [10], and Clockwork [20] are some earlier general model serving systems. They study batching, caching, placement, and scheduling for serving single or multiple models. More recently, DVABatch [12] introduces multi-entry multi-exit batching. REEF [21] and Shepherd [61] propose preemption for serving. AlpaServe [28] utilizes model parallelism for statistical multiplexing. However, these general systems fail to take into account the autoregressive property and token state of LLM inference, resulting in missed opportunities for optimization.

Specialized serving systems for transformers. Due to the significance of the transformer architecture, numerous specialized serving systems for it have been developed. These systems utilize GPU kernel optimizations [1, 29, 31, 56], advanced batching mechanisms [14, 60], model parallelism [1, 41, 60], and parameter sharing [64] for efficient serving. Among them, Orca [60] is most relevant to our approach.

Comparison to Orca. The iteration-level scheduling in Orca [60] and PagedAttention in vLLM are complementary techniques: While both systems aim to increase the GPU utilization and hence the throughput of LLM serving, Orca achieves it by scheduling and interleaving the requests so that more requests can be processed in parallel, while vLLM is doing so by increasing memory utilization so that the working sets of more requests fit into memory. By reducing memory fragmentation and enabling sharing, vLLM runs more requests in a batch in parallel and achieves a 2-4× speedup compared to Orca. Indeed, the fine-grained scheduling and interleaving of the requests like in Orca makes memory management more challenging, making the techniques proposed in vLLM even more crucial.

Memory optimizations. The widening gap between the compute capability and memory capacity of accelerators has caused memory to become a bottleneck for both training and inference. Swapping [23, 42, 55], recomputation [7, 24] and their combination [40] have been utilized to reduce the peak memory of training. Notably, FlexGen [46] studies how to swap weights and token states for LLM inference with limited GPU memory, but it does not target the online serving settings. OLLA [48] optimizes the lifetime and location of tensors to reduce fragmentation, but it does not do fine-grained block-level management or online serving. FlashAttention [13] applies tiling and kernel optimizations to reduce the peak memory of attention computation and reduce I/O costs. This paper introduces a new idea of block-level memory management in the context of online serving.

10 Conclusion

This paper proposes PagedAttention, a new attention algorithm that allows attention keys and values to be stored in non-contiguous paged memory, and presents vLLM, a high-throughput LLM serving system with efficient memory management enabled by PagedAttention. Inspired by operating systems, we demonstrate how established techniques, such as virtual memory and copy-on-write, can be adapted to efficiently manage KV cache and handle various decoding algorithms in LLM serving. Our experiments show that vLLM achieves 2-4× throughput improvements over the state-of-the-art systems.

Acknowledgement

We would like to thank Xiaoxuan Liu, Zhifeng Chen, Yanping Huang, anonymous SOSP reviewers, and our shepherd, Lidong Zhou, for their insightful feedback. This research is partly supported by gifts from Andreessen Horowitz, Anyscale, Astronomer, Google, IBM, Intel, Lacework, Microsoft, Mohamed Bin Zayed University of Artificial Intelligence, Samsung SDS, Uber, and VMware.

References

[1] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, et al. 2022. DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. *arXiv preprint arXiv:2207.00032* (2022).

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[3] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. *Advances in neural information processing systems* 13 (2000).

[4] Ond rej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 Conference on Machine Translation. In *Proceedings of the First Conference on Machine Translation*. Association for Computational Linguistics, Berlin, Germany, 131–198. http://www.aclweb.org/anthology/W/W16/W16-2301

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[7] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174* (2016).

[8] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

[10] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. 2020. InferLine: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 477–491.

[11] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 613–627.

[12] Weihao Cui, Han Zhao, Quan Chen, Hao Wei, Zirui Li, Deze Zeng, Chao Li, and Minyi Guo. 2022. DVABatch: Diversity-aware Multi-Entry Multi-Exit Batching for Efficient Processing of DNN Services on GPUs. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 183–198.

[13] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.

[14] Jiarui Fang, Yang Yu, Chengduo Zhao, and Jie Zhou. 2021. TurboTransformers: an efficient GPU serving system for transformer models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 389–402.

[15] FastAPI. 2023. FastAPI. https://github.com/tiangolo/fastapi.

[16] Pin Gao, Lingfan Yu, Yongwei Wu, and Jinyang Li. 2018. Low latency rnn inference with cellular batching. In *Proceedings of the Thirteenth EuroSys Conference*. 1–15.

[17] Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W Mahoney, and Kurt Keutzer. 2021. Ai and memory wall. *RiseLab Medium Post* 1 (2021), 6.

[18] Github. 2022. https://github.com/features/copilot

[19] Google. 2023. https://bard.google.com/

[20] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving {DNNs} like Clockwork: Performance Predictability from the Bottom Up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 443–462.

[21] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale Preemption for Concurrent {GPU-accelerated} {DNN} Inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 539–558.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[23] Chien-Chin Huang, Gu Jin, and Jinyang Li. 2020. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1341–1355.

[24] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. 2020. Checkmate: Breaking the memory wall with optimal tensor rematerialization.