vLLM

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
**Equal contribution*

Presenter: Yuankai Li, Date: 2025/10/27

UCDAVIS

Background: LLM serving

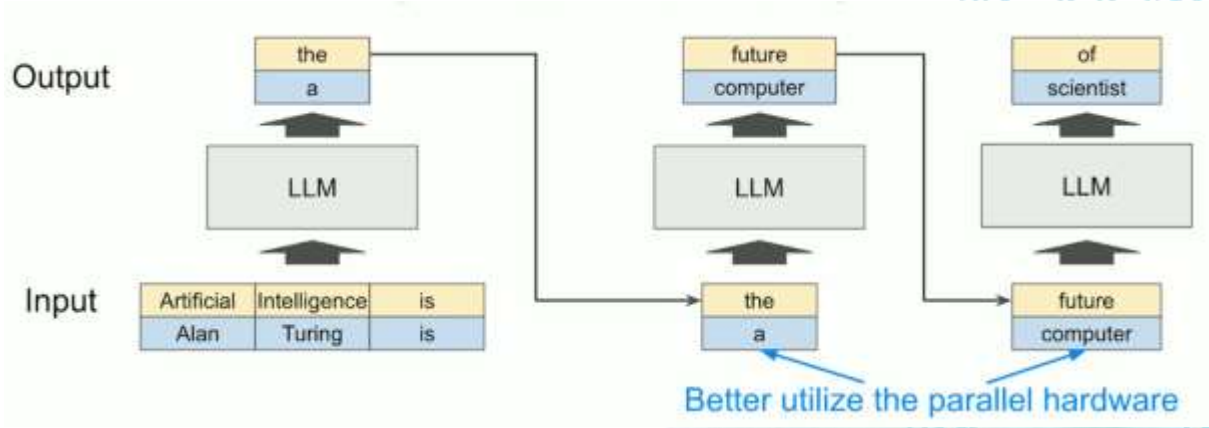
Serving LLMs is slow and expensive because

The sequential dependency make it difficult to fully utilize the parallelism

Batch multiple requests together to improve throughput

Memory management for KV Cache is inefficient,

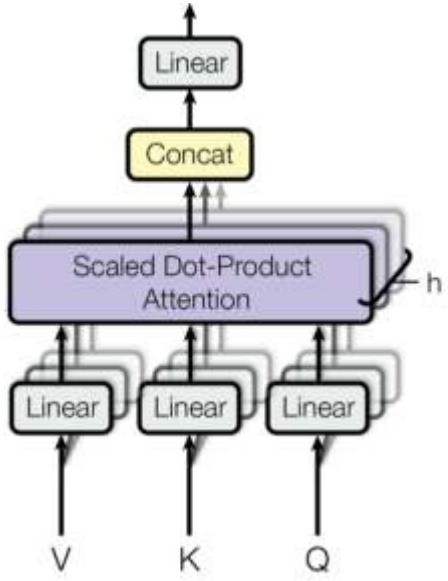
leads to limited `batch_size` (GPU OOM).



SOSP'23 Presentation | Efficient Memory Management for Large Language Model Serving with PagedAttention

Preliminary: Self-Attention

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



Vaswani et al. "Attention Is All You Need", 2017

Input: `x.shape=(batch_size, len, hidden_dim)`

`proj= nn.linear(hidden_dim, num_head*head_dim)`

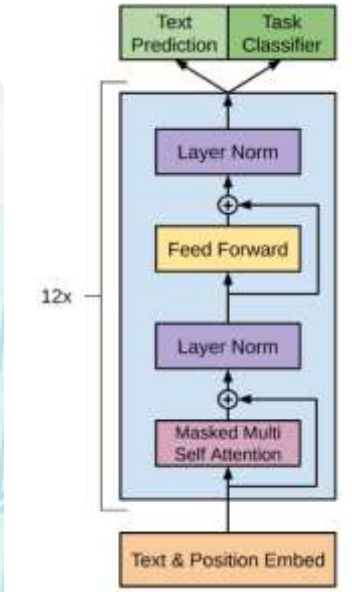
`Q = q_proj(x), V = v_proj(x), K = k_proj(x)`

View QKV as: `(batch_size, len, num_head, head_dim)`

Apply positional embedding (especially RoPE)

Scaled Dot-Prodcut Attention (Q,K,V)

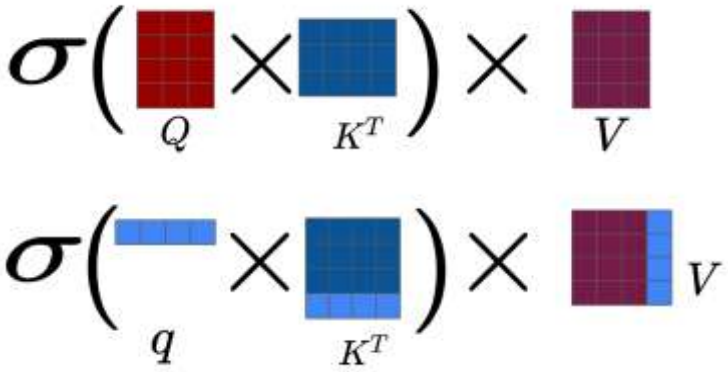
Output: `o.shape = (batch_size, len, num_head*head_dim)`



Radford et al. "Improving Language Understanding by Generative Pre-Training"

Preliminary: Decoding & KV Cache

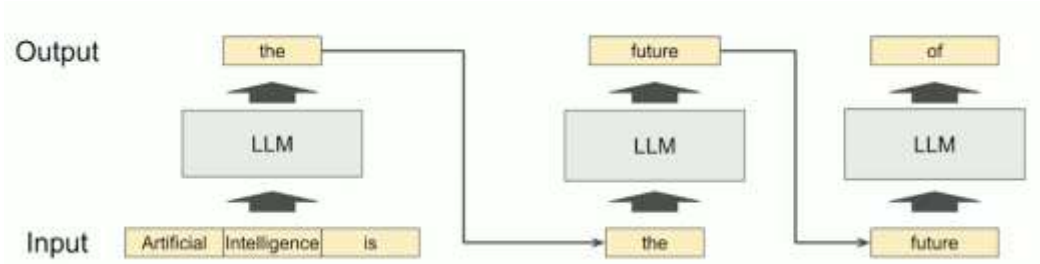
$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



https://pages.cs.wisc.edu/~shivaram/cs744-sp24-slides/cs744-vllm.pdf

During inference, `len=1`

K and V can be reused (**KV Cache**)



SOSP'23 Presentation | Efficient Memory Management for Large Language Model Serving with PagedAttention

Decoding

Auto-regressively generate until

- Reaches `max_seq_len`
- Generate certain tokens like `<eos>` or `<|end_of_sequence|>`

Motivation: Memory Fragmentation

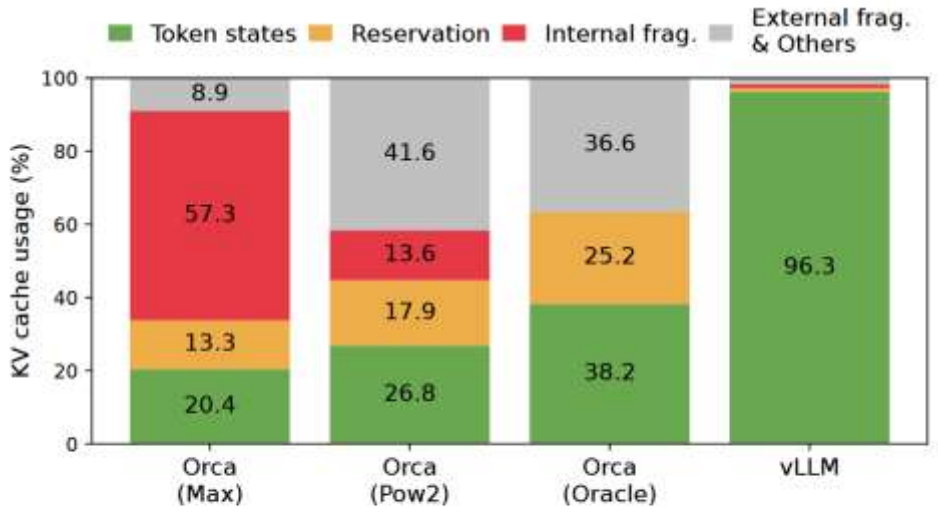


Figure 2. Average percentage of memory wastes in different LLM serving systems during the experiment in §6.2.

KV Cache can be huge!

For a 13B llama-2 model, one K cache could be (7B/13B/70B (4096 / 5120 / 8192 hidden; 32 / 40 / 80 layers)[2]

One single token would take: `num_hidden * layer * FP32` = 40 * 5120 * 4 = 819,200 bytes ~ **0.82 MB**

One full request could be several GBs: `batch_size * prefix_len * single_token` = 1 * 2048 * 819200 ~ **1.67 GB**

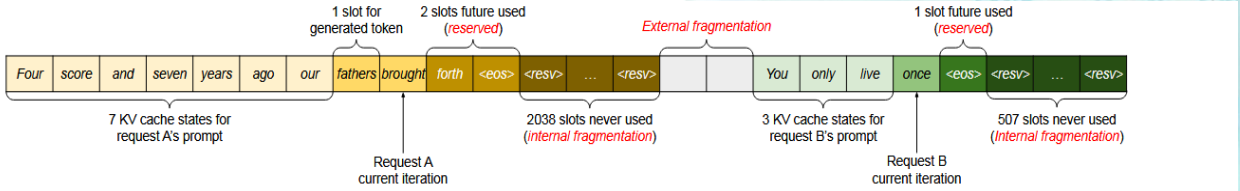


Figure 3. KV cache memory management in existing systems. Three types of memory wastes – reserved, internal fragmentation, and external fragmentation – exist that prevent other requests from fitting into the memory. The token in each memory slot represents its KV cache. Note the same tokens can have different KV cache when at different positions.

[1] Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention", 2023
[2] Touvron et al. "Llama-2: Open Foundation and Fine-Tuned Chat models"

Motivation: Memory Fragmentation

Internal Fragmentation

Pre-allocate contiguous space that may never be used
(unknown output length)

External Fragmentation

Application-level code (PyTorch) leave memory between KV Caches
(non-uniform per-request max lengths)

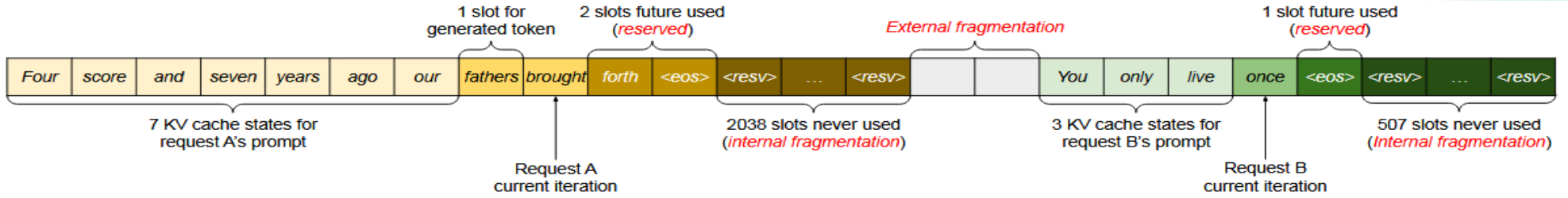


Figure 3. KV cache memory management in existing systems. Three types of memory wastes – reserved, internal fragmentation, and external fragmentation – exist that prevent other requests from fitting into the memory. The token in each memory slot represents its KV cache. Note the same tokens can have different KV cache when at different positions.

Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention", 2023

Method: Paged Attention

Paging analogy

PagedAttention partitions the KV cache of each sequence into **KV blocks**. Each block contains the **K and V vectors** for a fixed number of tokens (block size B)

The PagedAttention algorithm allows the KV blocks to be stored in **non-contiguous physical memory**, which enables more flexible paged memory management.

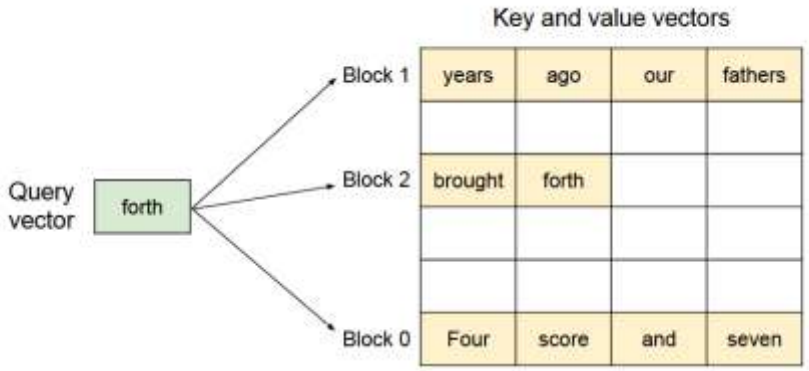


Figure 5. Illustration of the PagedAttention algorithm, where the attention key and values vectors are stored as non-contiguous blocks in the memory.

Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention", 2023

Implementation: vLLM

KV Cache Manager

Analogous to the *virtual memory* in OS

A request's KV cache is represented as **a series of logical KV blocks**, filled from left to right as new tokens and their KV cache are generated. The last KV block's unfilled positions are reserved for future generations. On GPU workers, **a block engine allocates a contiguous chunk of GPU DRAM** and divides it into **physical KV blocks**

The KV block manager also
maintains block tables—the mapping between logical and physical KV blocks of each request.

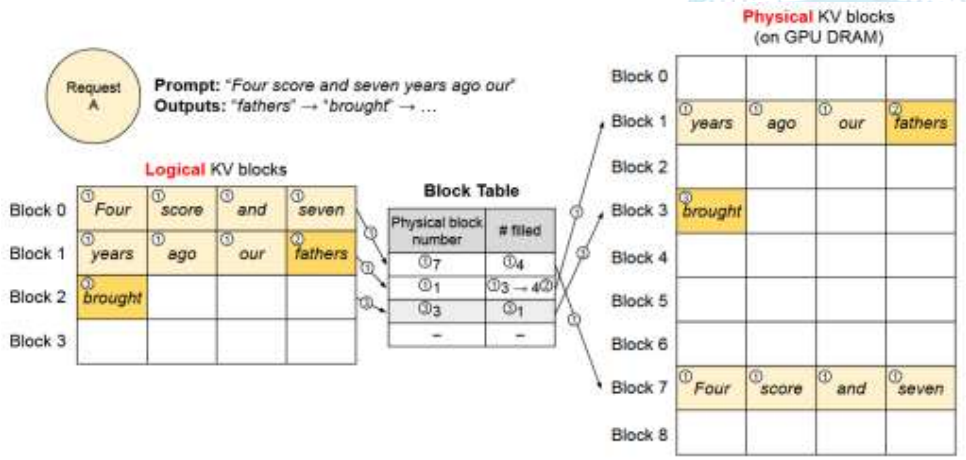


Figure 6. Block table translation in vLLM.

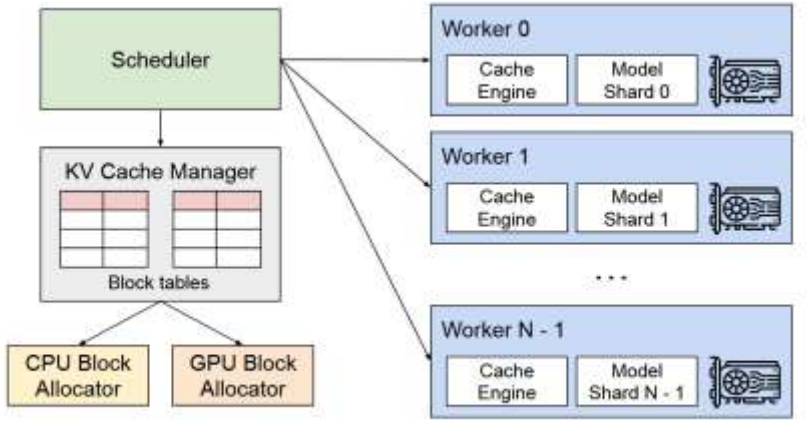


Figure 4. vLLM system overview.

Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention", 2023

Running Example: vLLM

Four score and seven years ago our [fathers] [brought] …

1. vLLM reserves the necessary KV blocks for the KV cache during the prefill step.
2. vLLM generates the new token with the PagedAttention algorithm, and the block table's `#filled` record is **updated**
3. As the last logical block is full, vLLM stores the newly generated KV cache in a new logical block; vLLM **allocates a new physical block** for it and **stores this mapping in the block table**

Once a request finishes its generation, its KV blocks can be freed to store the KV cache of other requests

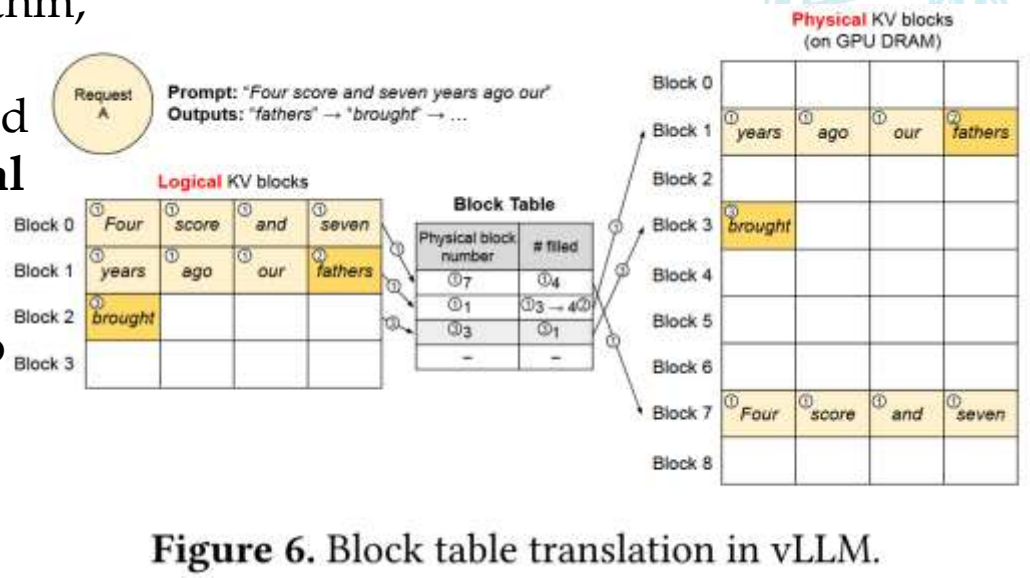


Figure 6. Block table translation in vLLM.

Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention", 2023

Details: Decoding Method

Parallel Sampling LLM generates multiple sampled outputs for a single input prompt

- *reference count* for each physical block & *copy-on-write*

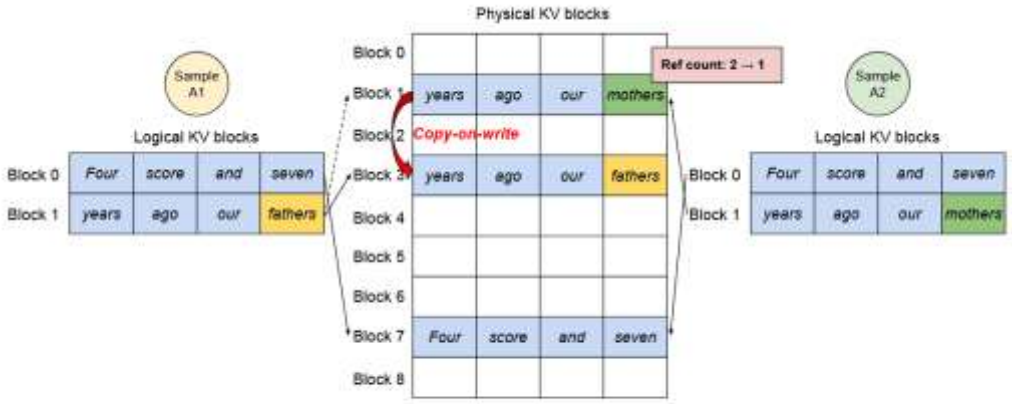


Figure 8. Parallel sampling example.

Beam Search is widely used to decode the most probable output sequence from an LLM

- share not only the initial prompt blocks but also **other blocks across different candidates**
- Reduce memory copy!

beam width parameter k=4

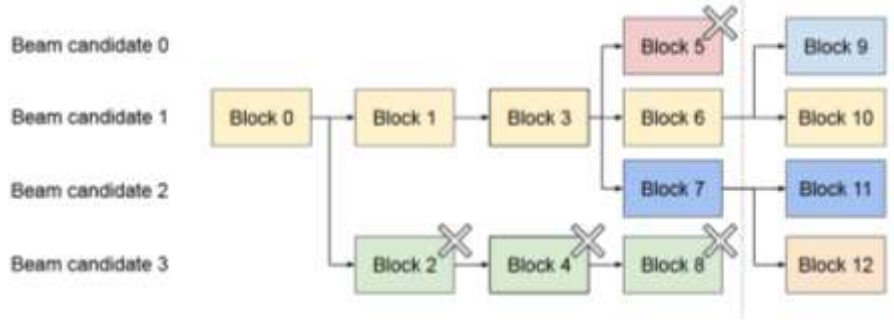


Figure 9. Beam search example.

Shared Prefix Prefix like *system prompt* is shared across tasks

- Reserve certain physical blocks

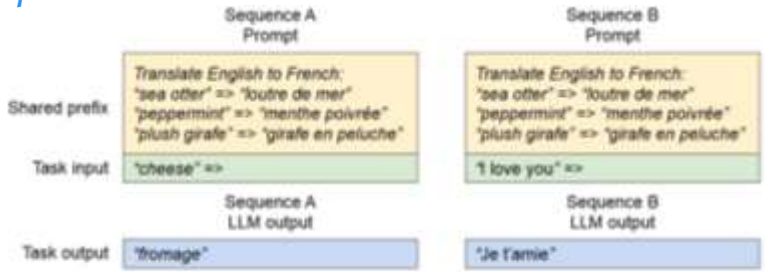


Figure 10. Shared prompt example for machine translation. The examples are adopted from [5].

Mixed decoding methods

- Requests with different sampling

Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention", 2023

Details: Scheduling, Preemption & Distributed Execution

When vLLM run out of physical blocks on GPUs

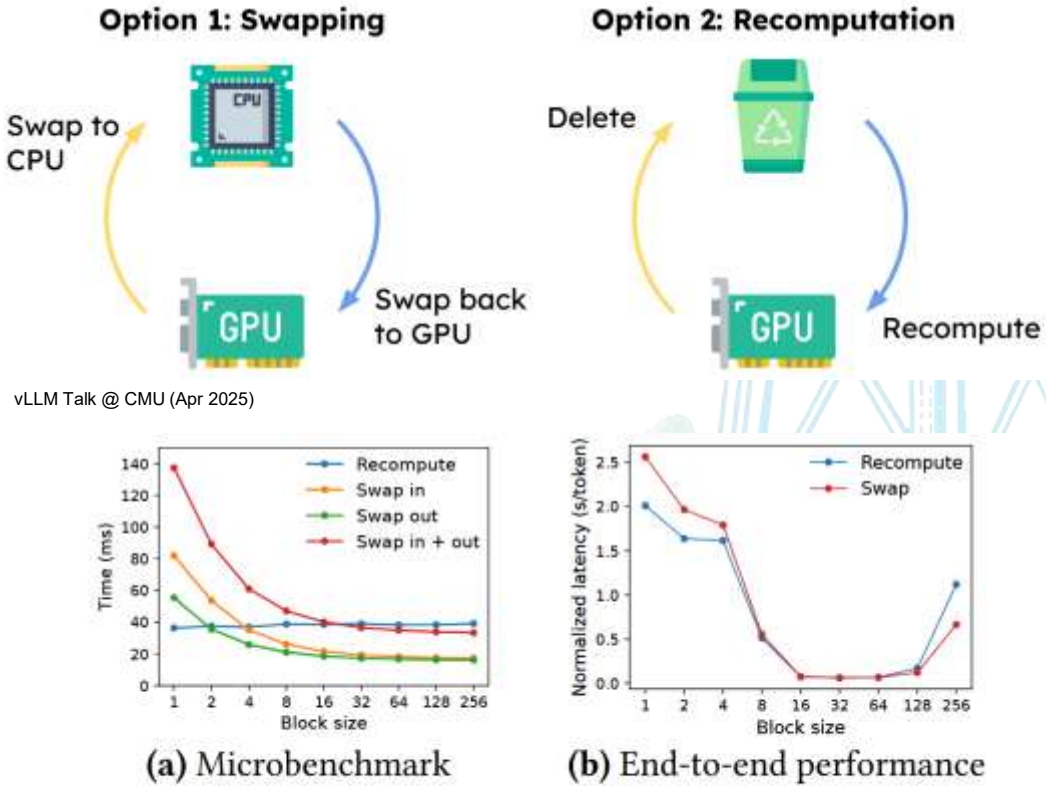
vLLM adopt the *first-come-first-serve (FCFS)* policy

- *all-or-nothing eviction policy*, since all blocks of a sequence are accessed together
- To recover the evicted block, we either do *swapping* or *recomputation*
  - Swapping: copy to CPU memory and bring back
  - Recomputation: recompute KV Cache
    - Can be generated in *one prompt iteration*

Distributed Execution

vLLM implement **Megatron-LM** style tensor model parallelism with *SPMD*

- Linear layer partitioned
- Attention head partitioned – Each GPU stores the corresponding KV Cache



vLLM Talk @ CMU (Apr 2025)

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.

Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention", 2023

Implementation: vLLM

vLLM's goal is to build the **fastest** and **easiest-to-use** open-source LLM inference & serving engine

Kernel-level Optimization vLLM contains custom CUDA kernels for key operations like PagedAttention

- Fused reshape and block write
- Fused block read and attention
- Fused block copy

Various Decoding Algorithms vLLM creates `fork`, `append` and `free` to run algorithms like parallel sampling, beam search and prefix sharing

Experiment

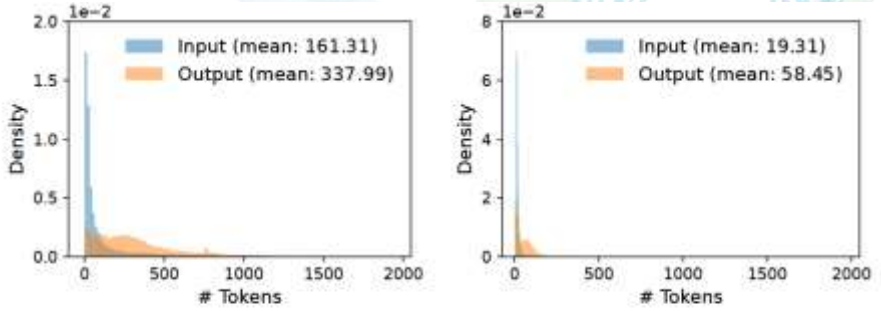
Metric: Serving throughput

Input/Output Length Distribution:
- Alpaca dataset (instruction-following)
- ShareGPT datasetx (conversation)

Baselines:
- NVIDIA FastTransformer (FT)
- Orca (assume using *buddy algorithm*) (iteration-level scheduling)
  - Oracle: Know exact output length
  - Pow2: Over-reserve by at most x2
  - Max: Over-reserve to maximum possible

Table 1. Model sizes and server configurations.

Model size	13B	66B	175B
GPUs	A100	4×A100	8×A100-80GB
Total GPU memory	40 GB	160 GB	640 GB
Parameter size	26 GB	132 GB	346 GB
Memory for KV cache	12 GB	21 GB	264 GB
Max. # KV cache slots	15.7K	9.7K	60.1K



(a) ShareGPT **(b)** Alpaca

Figure 11. Input and output length distributions of the (a) ShareGPT and (b) Alpaca datasets.

Kwon et al. “Efficient Memory Management for Large Language Model Serving with PagedAttention”, 2023

Experiment

Basic sampling

On the ShareGPT dataset, vLLM can sustain **1.7×–2.7× higher** request rates compared to Orca (Oracle) and **2.7×–8×** compared to Orca (Max), while maintaining similar latencies. Alpaca dataset follows a similar trend to the ShareGPT dataset.

vLLM enables **batching more requests**

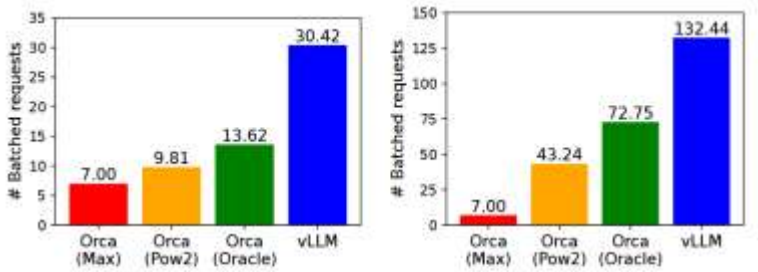


Figure 13. Average number of batched requests when serving OPT-13B for the ShareGPT (2 reqs/s) and Alpaca (30 reqs/s) traces.

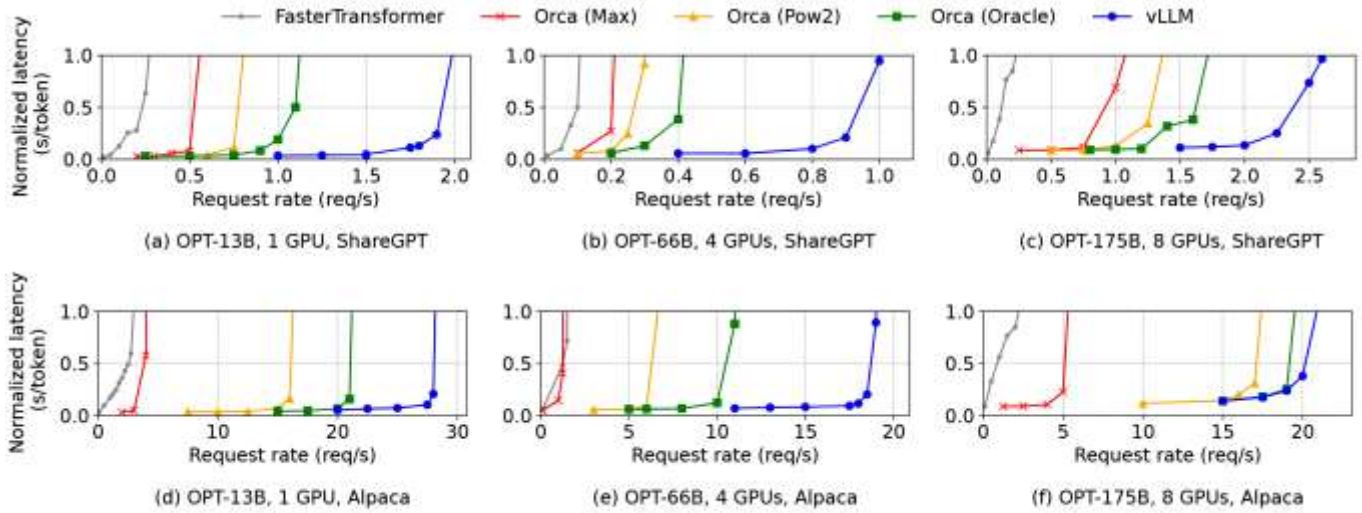


Figure 12. Single sequence generation with OPT models on the ShareGPT and Alpaca dataset

Kwon et al. “Efficient Memory Management for Large Language Model Serving with PagedAttention”, 2023

Experiment

Parallel generation, Beam search & Prefix sharing

The improvement of vLLM over Orca (Oracle) on OPT-13B and the Alpaca dataset goes from **1.3×** in basic sampling to **2.3×** in beam search with a width of 6. (Beam search allows for more sharing)

vLLM achieves 1.67× higher throughput than Orca (Oracle) when the one-shot prefix is shared.
vLLM achieves 3.58× higher throughput than Orca (Oracle).

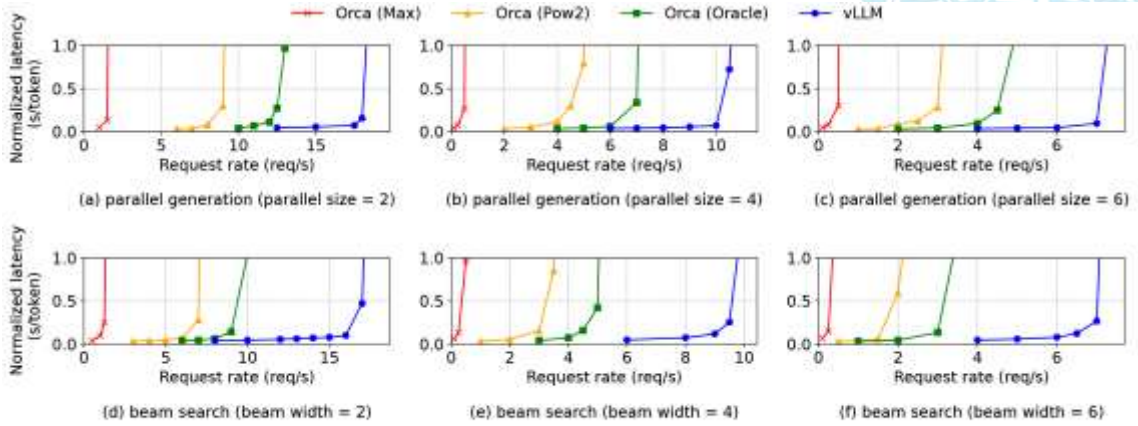
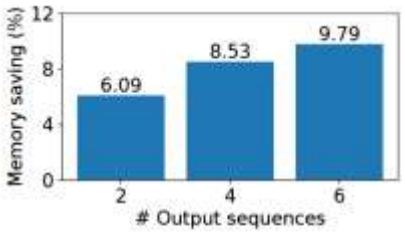
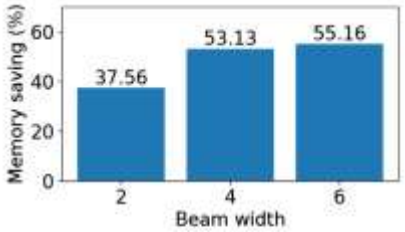


Figure 14. Parallel generation and beam search with OPT-13B on the Alpaca dataset.



(a) Parallel sampling



(b) Beam search

Figure 15. Average amount of memory saving from sharing KV blocks, when serving OPT-13B for the Alpaca trace.

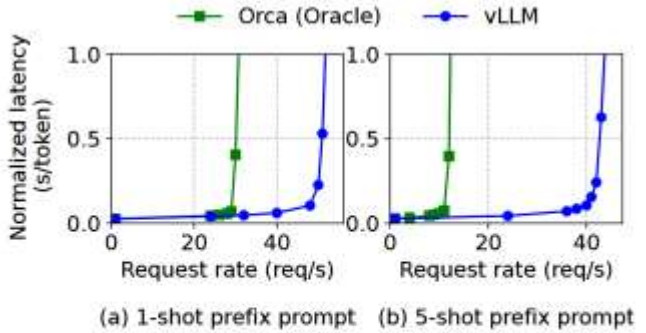


Figure 16. Translation workload where the input prompts share a common prefix. The prefix includes (a) 1 example with 80 tokens or (b) 5 examples with 341 tokens.

Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention", 2023

Ablation

Kernel Microbenchmark

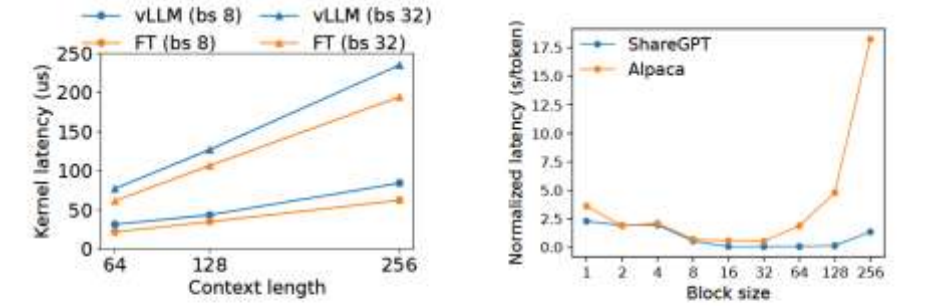
Custom kernel introduce **20-26%** higher attention kernel latency

Block Size

Block size 16 is large enough to **efficiently utilize the GPU** and small enough to **avoid significant internal fragmentation** in most workloads. vLLM sets its default block size as 16

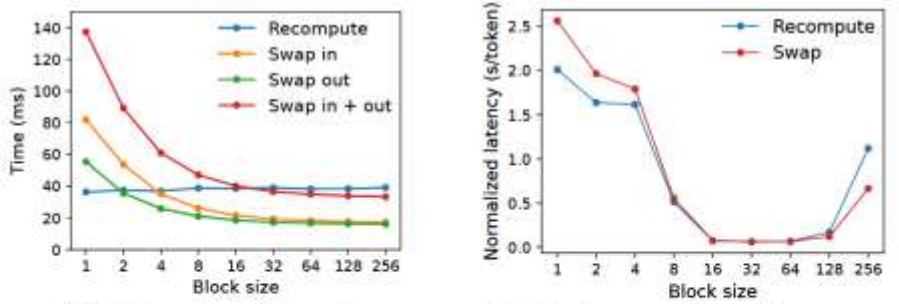
Swapping Vs Recomputation

Recomputation is more efficient when the block size is small, while swapping is more efficient when the block size is large



(a) Latency of attention kernels. **(b)** End-to-end latency with different block sizes.

Figure 18. Ablation experiments.



(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.

Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention", 2023

UCDAVIS

Overview

Problem: Inefficient KV Cache management limits LLM serving throughput

Motivation: Current approach fails at internal fragmentation and external fragmentation

Method: Apply OS paging techniques to KV Cache, introduce PagedAttention which break KV Cache into fix-sized blocks that can be stored in memory more flexibly

Experiment: vLLM improves throughput by 2-4x over SOTA, especially benefiting from complex decoding

UCDAVIS

Credits & Resources

Some content is borrowed from:

[1] SOSP '23 Presentation| Efficient Memory Management for Large Language Model Serving with PagedAttention (https://www.youtube.com/watch?v=UdNocRPQS3Y)

[2] https://pages.cs.wisc.edu/~shivaram/cs744-sp24-slides/cs744-vllm.pdf

[3]vLLM Talk @ CMU (Apr 2025)(https://llmsystem.github.io/llmsystem2025spring/assets/files/llmsys-22-vLLM_woosuk_kwon-1f34697dbb1a1fb5b798daf6eff14b67.pdf)

To better understand KV Cache and PagedAttention:

[4] nano-vllm(https://github.com/GeeeekExplorer/nano-vllm)

UCDAVIS

Q & A

UCDAVIS