SOSP 2023

Efficient Memory Management for Large Language Model Serving with PagedAttention

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, Ion Stoica

Presenter: Jun-Chen Hung
2024/12/04

https://arxiv.org/pdf/2309.06180

Slide here!

Outline

- Introduction
- Backgrounds
- Problem
- Method
- Experiments
- Conclusion

Introduction

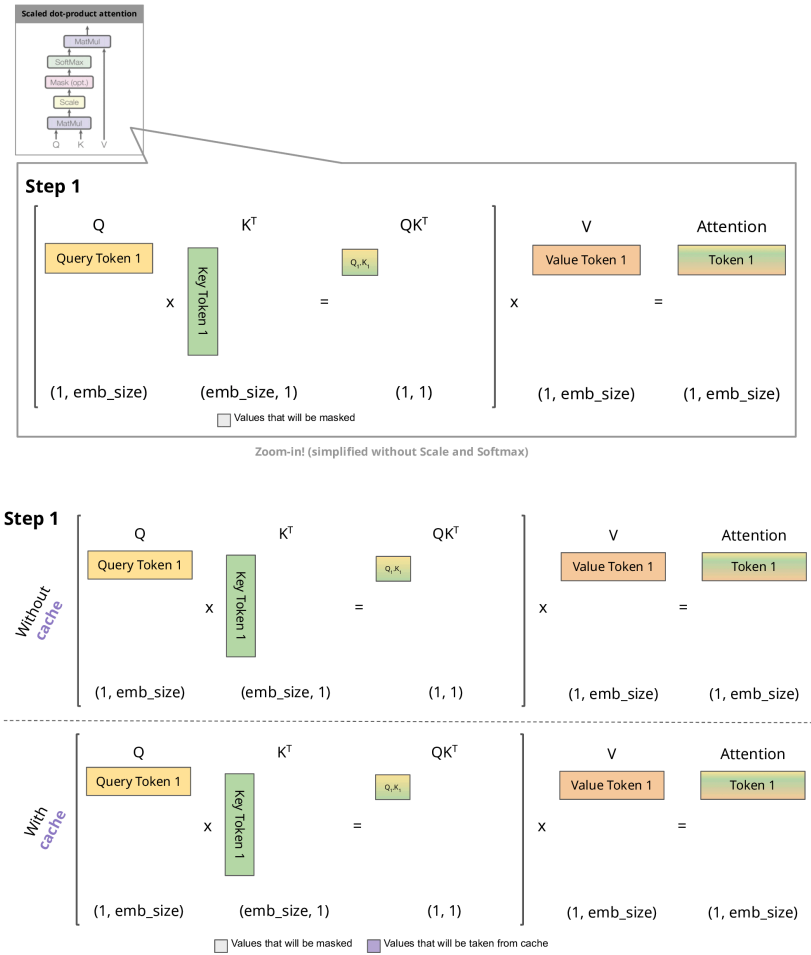
Introduction

- Identify the challenges in memory allocation in serving LLMs and quantify their impact on serving performance.
- Propose PagedAttention, an attention algorithm that operates on KV cache stored in non-contiguous paged memory, which is inspired by the virtual memory and paging in OS.
- Design and implement vLLM, a distributed LLM serving engine built on top of PagedAttention.
- Evaluate vLLM on various scenarios and demonstrate that it substantially outperforms the previous state-of-the art solutions such as FasterTransformer and Orca

Backgrounds

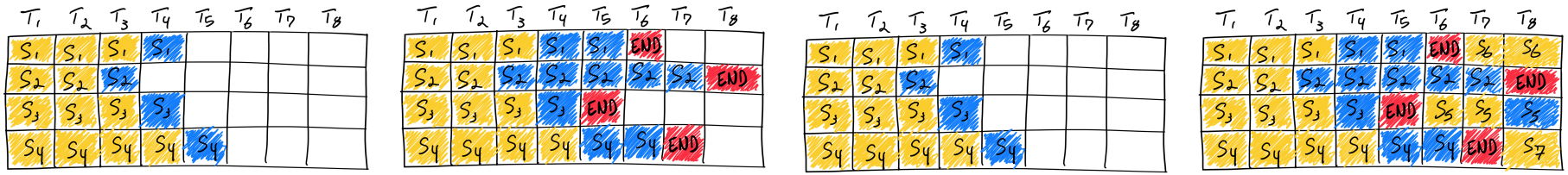
Backgrounds

- Transformer
  - $q_i = W_q x_i, k_i = W_k x_i, v_i = W_v x_i$
  - $a_{ij} = \frac{exp(q_j^\intercal k_t / \sqrt{d})}{\sum_{t=1}^{i} exp(q_i^\intercal k_t / \sqrt{d})}$
  - $o_i = \sum_{t=1}^{i} a_{ij} v_i$
- Autoregressive Generation
  - Depends on all previous tokens.
  - 2 phases: prompt phase & autoregressive generation phase
- KV cache
  - Reduce compute count of previous tokens.



Backgrounds

- Batching
  - Naive: Make earlier requests wait for later ones or delay the incoming requests until earlier ones finish.
  - Fine-grained: New requests can be processed after waiting for a single iteration, not waiting for the entire batch to complete. eg. Cellular batching, Iteration-level scheduling

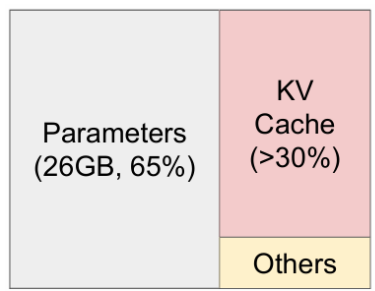


https://www.anyscale.com/blog/continuous-batching-llm-inference

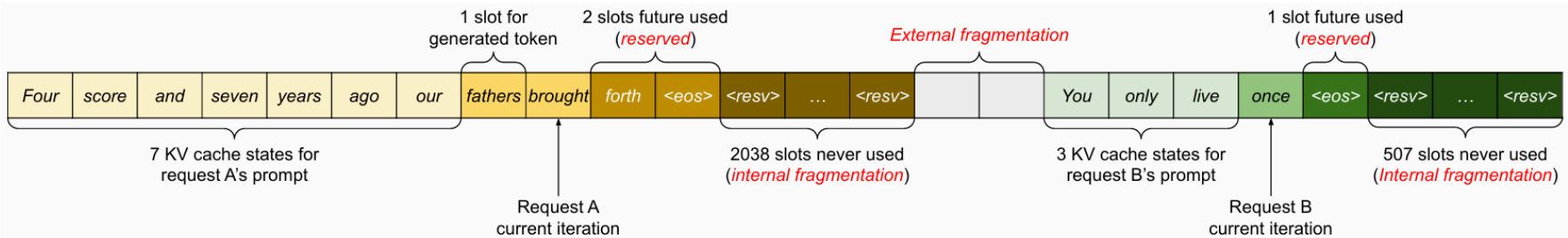
Problem

Problem

- **Large KV cache**
  - Single token KV cache of 13B OPT model: 800KB
    - 2 (k&v vectors) × 5120 (hidden state size) × 40 (number of layers) × 2 (bytes per FP16).
  - 2048 tokens costs 1.6GB KV cache size
  - Memory will become an increasingly significant bottleneck
- **Complex decoding algorithms**
  - LLM services offer a range of decoding algorithms for users to select from, each with varying implications for memory management complexity.
- **Scheduling for unknown input & output lengths.**
  - Requires the memory management system to accommodate a wide range of prompt lengths.



Example

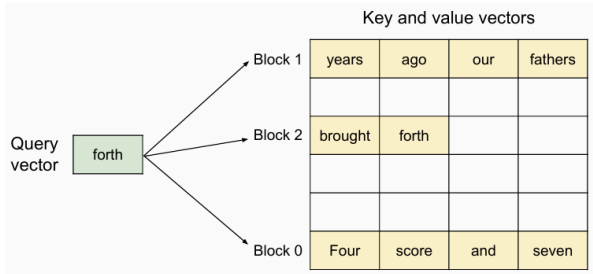


- Reserved: For future tokens.
- Internal fragmentation: for potential maximum sequence lengths.
- External fragmentation: From the memory allocator like the buddy allocator.

Method

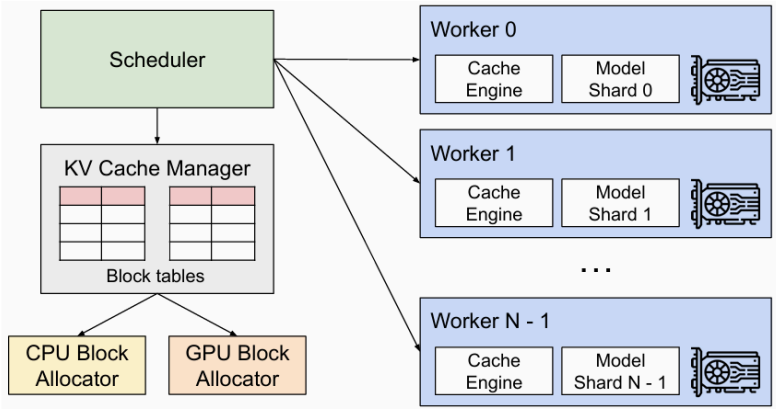
PagedAttention

- Inspired by the classic idea of paging in operating systems.
- KV Block (B): key and value vectors for a fixed number of tokens.
  - Key block: $K_j = (k_{(j-1)B+1}, ..., k_{jB})$
  - Value block: $V_j = (v_{(j-1)B+1}, ..., v_{jB})$
  - $A_{ij} = \frac{exp(q_i^\intercal K_j / \sqrt{d})}{\sum_{t=1}^{[i/B]} exp(q_i^\intercal K_t 1/\sqrt{d})}, A_i j = (a_{i,(j-1)B+1}, ..., a_{i,jB})$
  - $o_i = \sum_{t=1}^{[i/B]} V_j A_{ij}^\intercal$
- Allows the KV blocks to be stored in non-contiguous physical memory
- Enables more flexible paged memory management in vLLM.

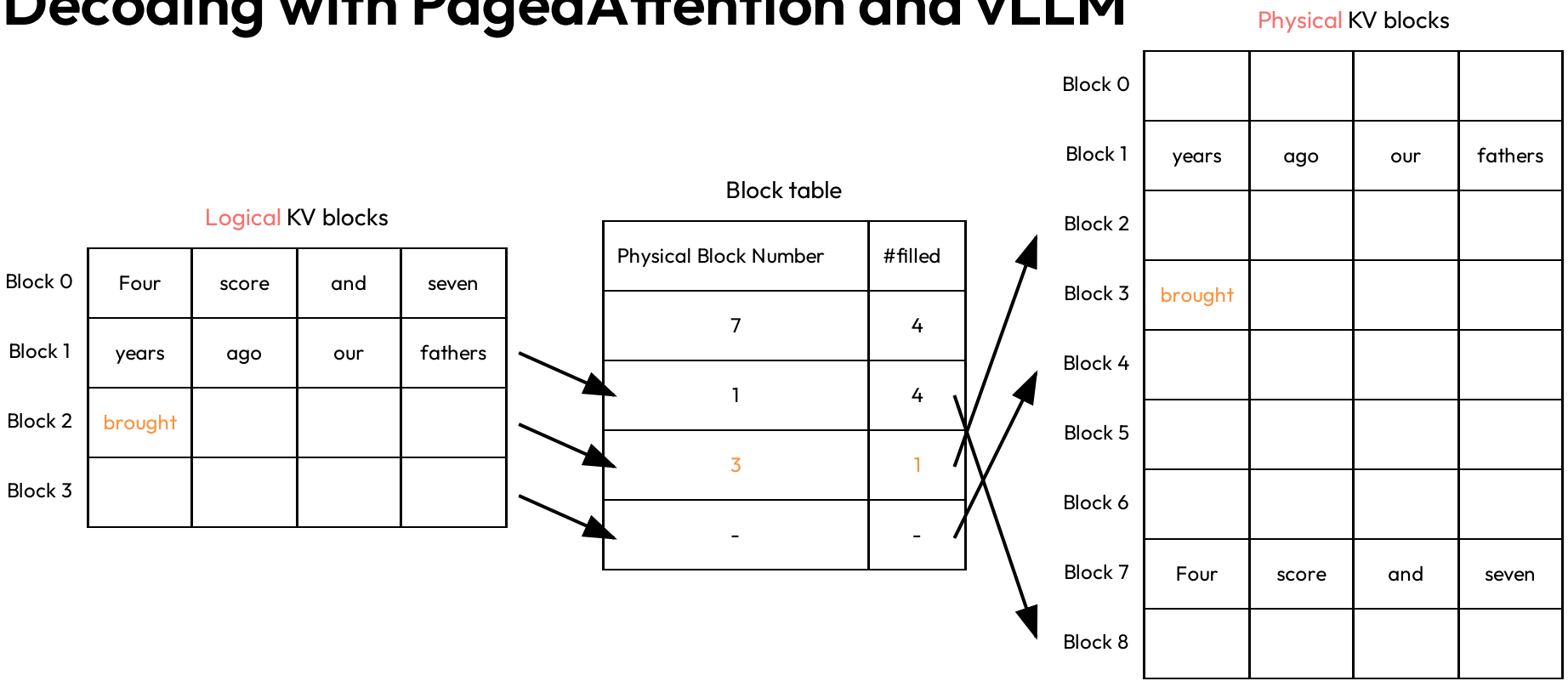


KV Cache Manager

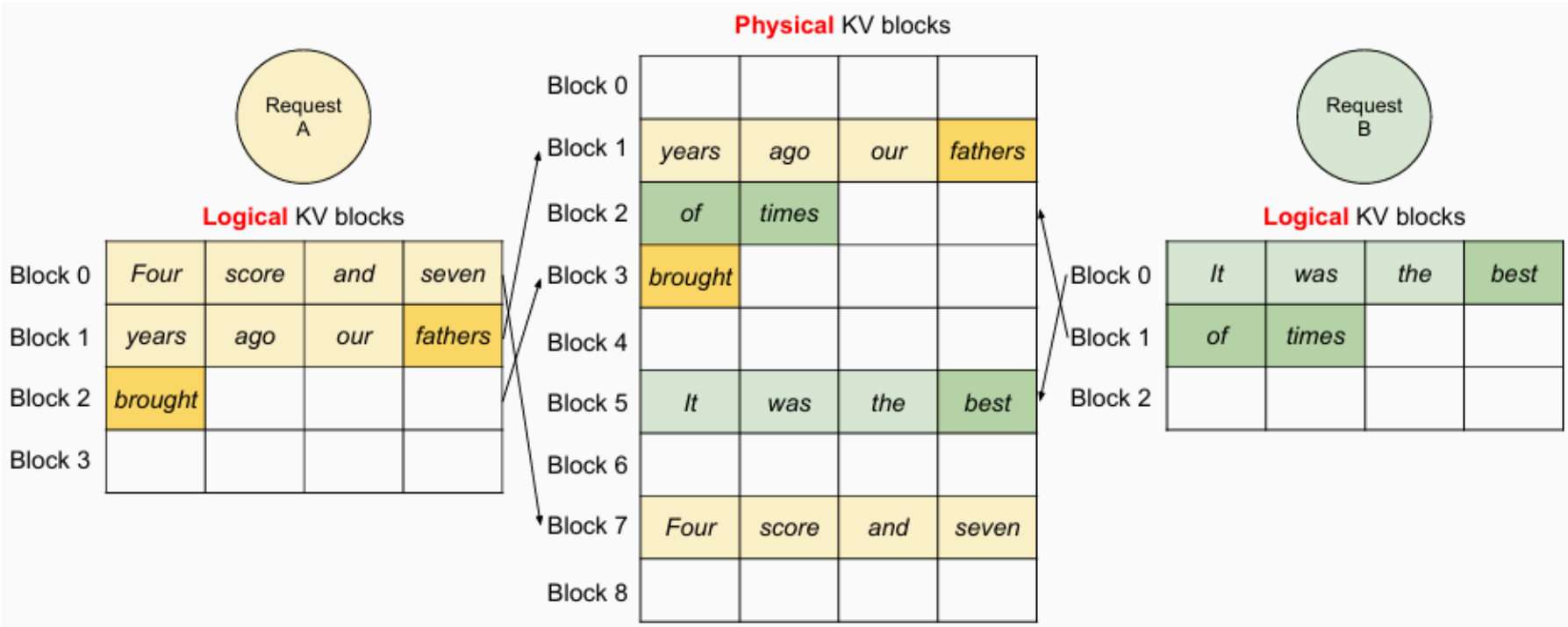
- Analogous to the virtual memory
- Contiguous logical pages can correspond to non-contiguous physical memory pages.
- Enabled by PagedAttention, we organize the KV cache as fixed-size KV blocks, like pages in virtual memory.
- Can serve multiple requests at the same time



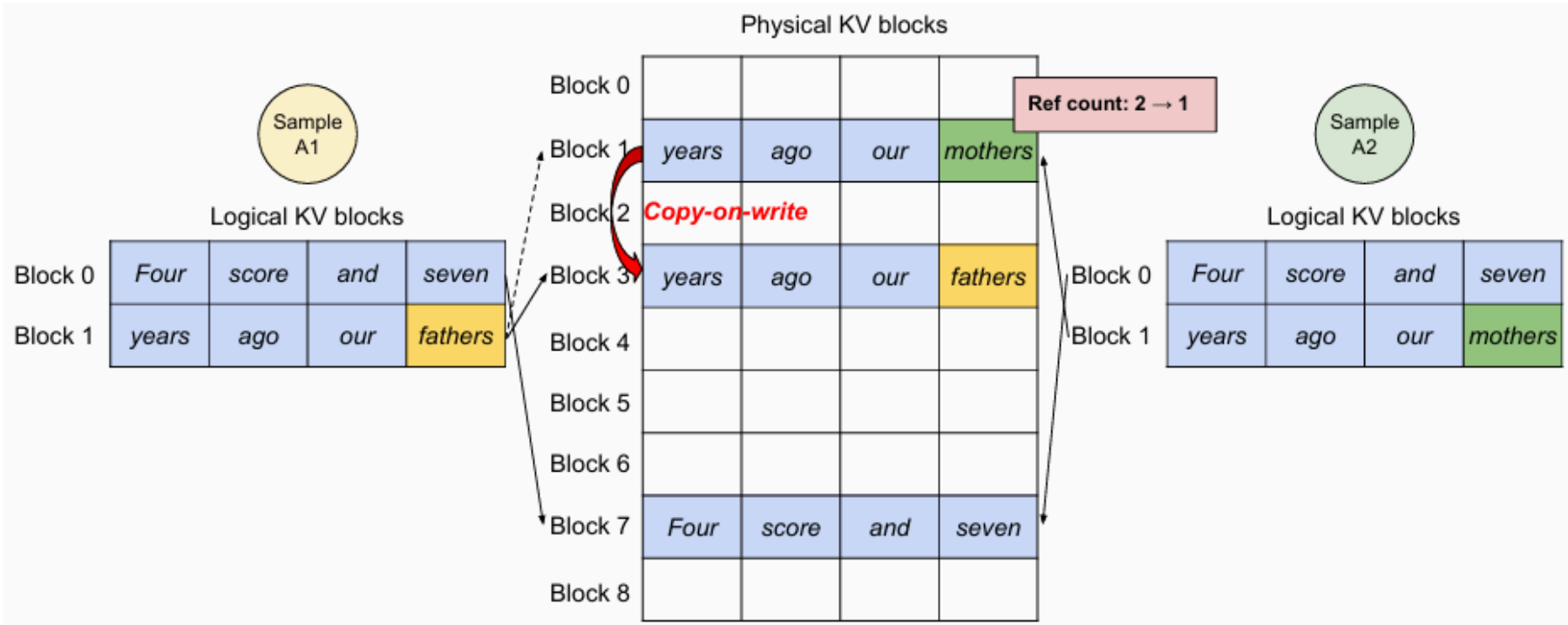
Decoding with PagedAttention and vLLM



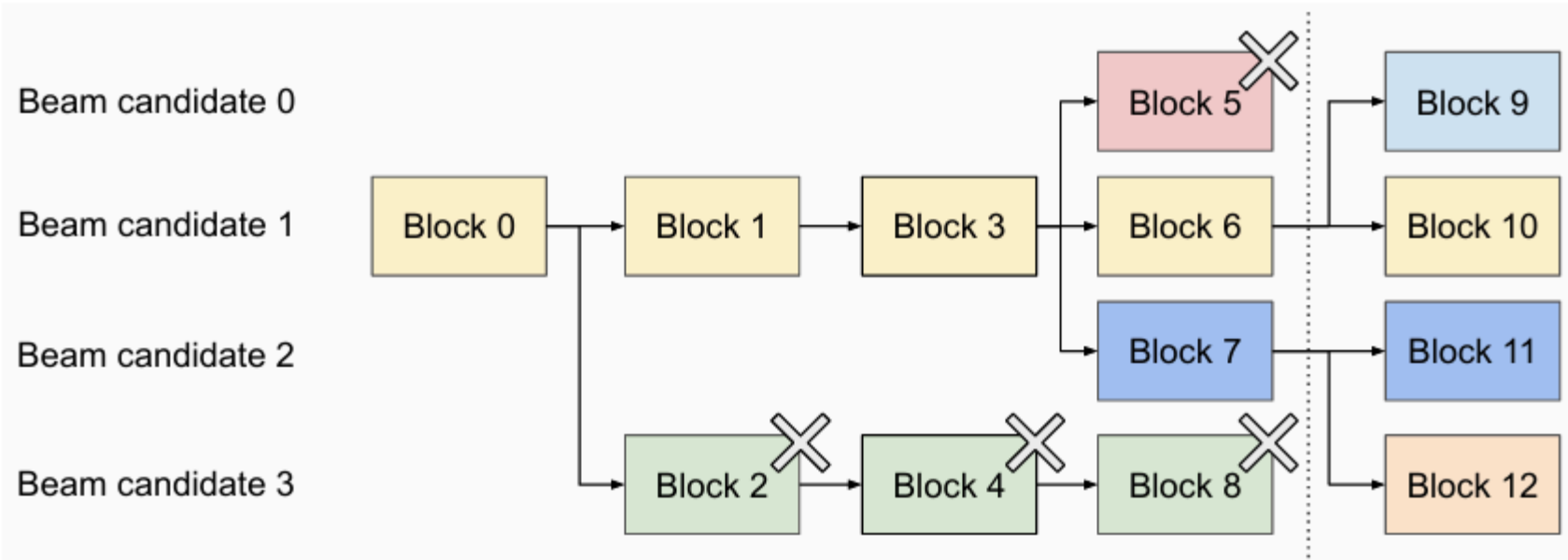
Multiple Decode



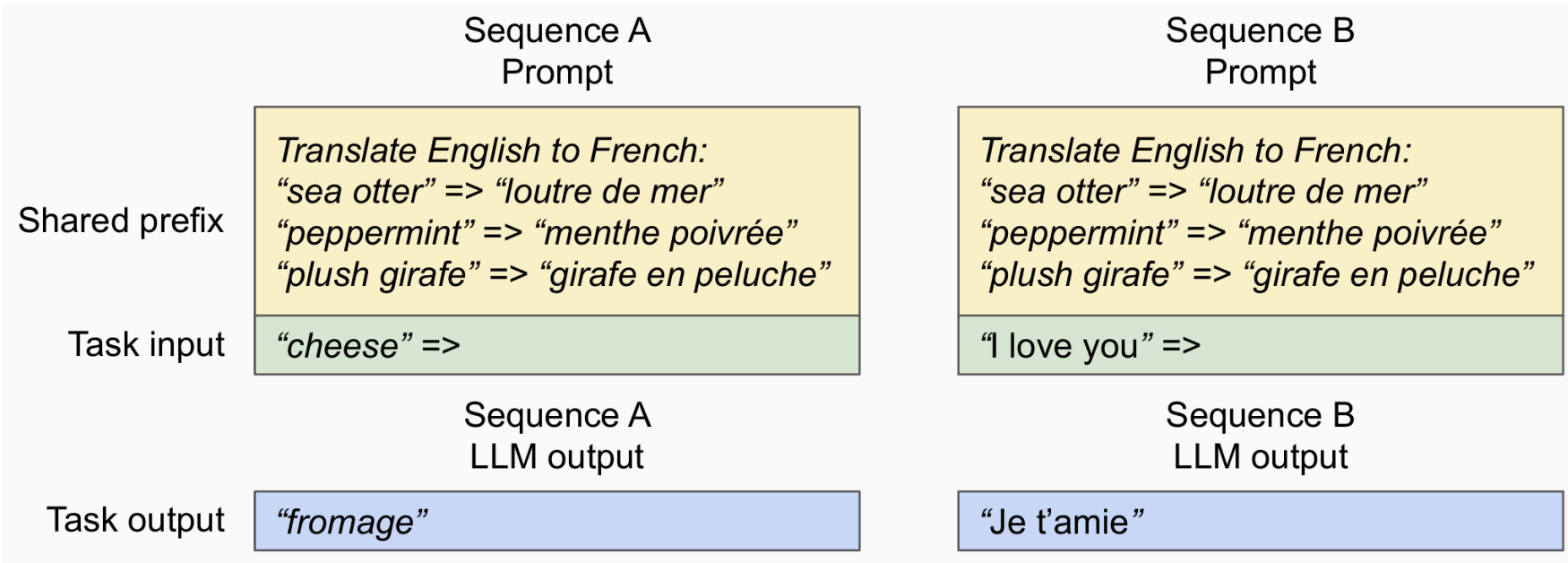
Parallel sampling



Beam Search


Beam candidate 0
Beam candidate 1
Beam candidate 2
Beam candidate 3
Block 0
Block 1
Block 3
Block 5
Block 6
Block 7
Block 2
Block 4
Block 8
Block 9
Block 10
Block 11
Block 12

Shared Prefix



Scheduling and Preemption

- Scheduling policy: first-come-first-serve
- Question: The input prompts for an LLM can vary significantly in length, and the resulting output lengths are not known a priori
  1. Which blocks should it evict?
  2. How to recover evicted blocks if needed again?

Scheduling and Preemption

1. Which blocks should it evict?
   - Apply **all-or-nothing** eviction policy
     - Multiple sequences within one request are gang-scheduled as a **Sequence Group**.

Scheduling and Preemption

1. How to recover evicted blocks if needed again?
   - **Swapping**
     - Classic technique used by most virtual memory implementations
     - Selects a set of sequences to evict and transfer their KV cache to the CPU, and stop accepting new requests.
     - Bring back and continue the processing of that sequence.
   - **Recomputation**
     - Recompute the KV cache when the preempted sequences are rescheduled.
     - Recomputation latency can be significantly lower than the original latency because they are in the Sequence Group.

Distributed Execution

- Support Megatron-LM style tensor model parallelism strategy on Transformers
- Scheduler: Calculate block table of evry worker(shard), broadcast block table when start computing
- GPU workers do not need to synchronize on memory management.

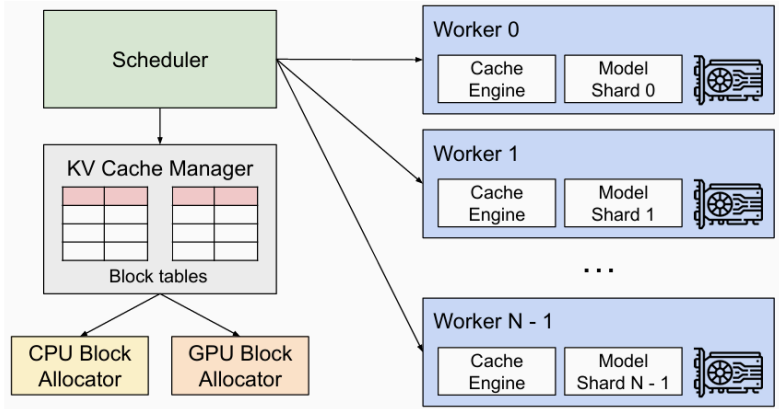


Figure 4. vLLM system overview.

Implementation

- Kernel-level Optimization
  - Optimize memory access patterns that are not efficiently supported by existing systems PagedAttention introduces.
- Supporting Various Decoding Algorithms
  - `fork` : Creates a new sequence from an existing one
  - `append` : Appends a new token to the sequence
  - `delete` : Deletes the sequence

Experiments

Experimental Setup

- **Model**: OPT 13B, 66B, 175B, Llama 13B
- **Server**: A2 instance of NVIDIA A100 GPUs on Google Cloud
- **Workload**: ShareGPT, Alpaca
- **Baseline**:
  - FasterTransformer: Distributed inference engine highly optimized for latency.
  - Orca: SOTA LLM serving system optimized for throughput.
    - Orca(Oracle): reserved space == real space
    - Orca(Pow2): reserved space <= 2x real space
    - Orca(Oracle): reserved space == `MAX_LENGTH`
- **Key metrics**: *normalized latency* of the system

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

Baseline sampling

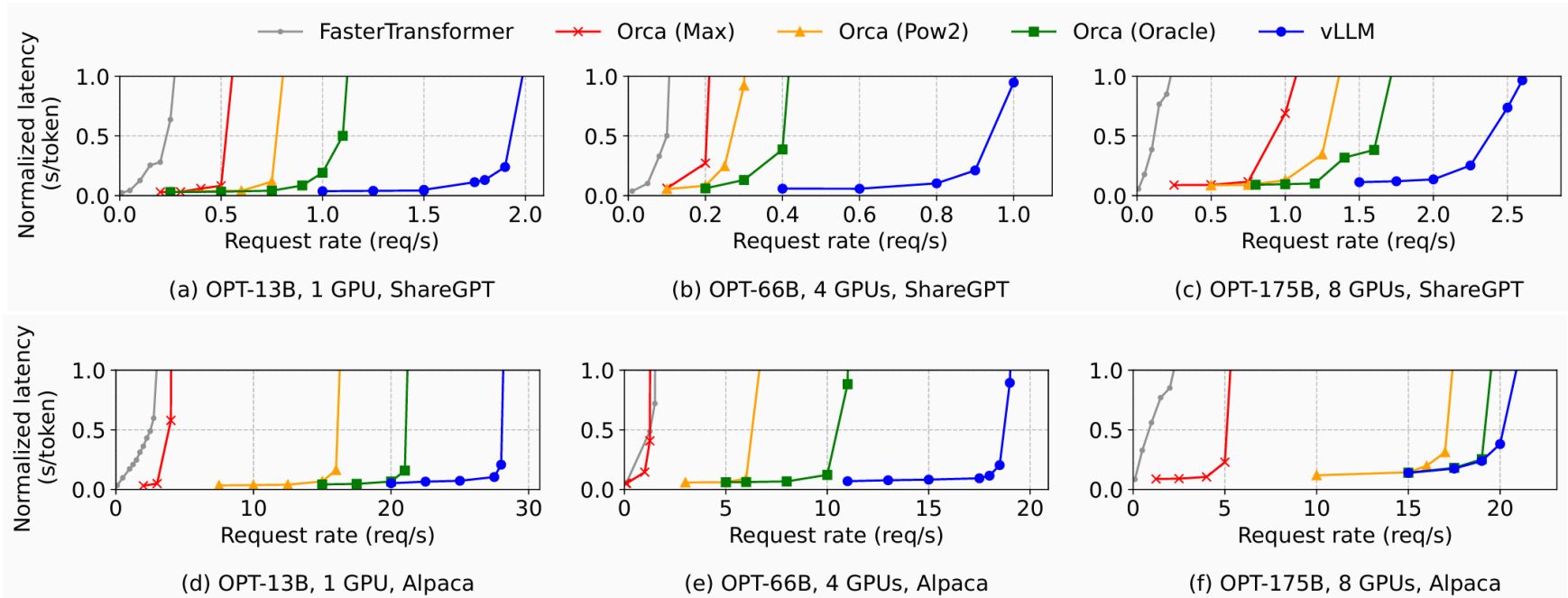


Figure 12. Single sequence generation with OPT models on the ShareGPT and Alpaca dataset

Parallel Sampling and Beam Search

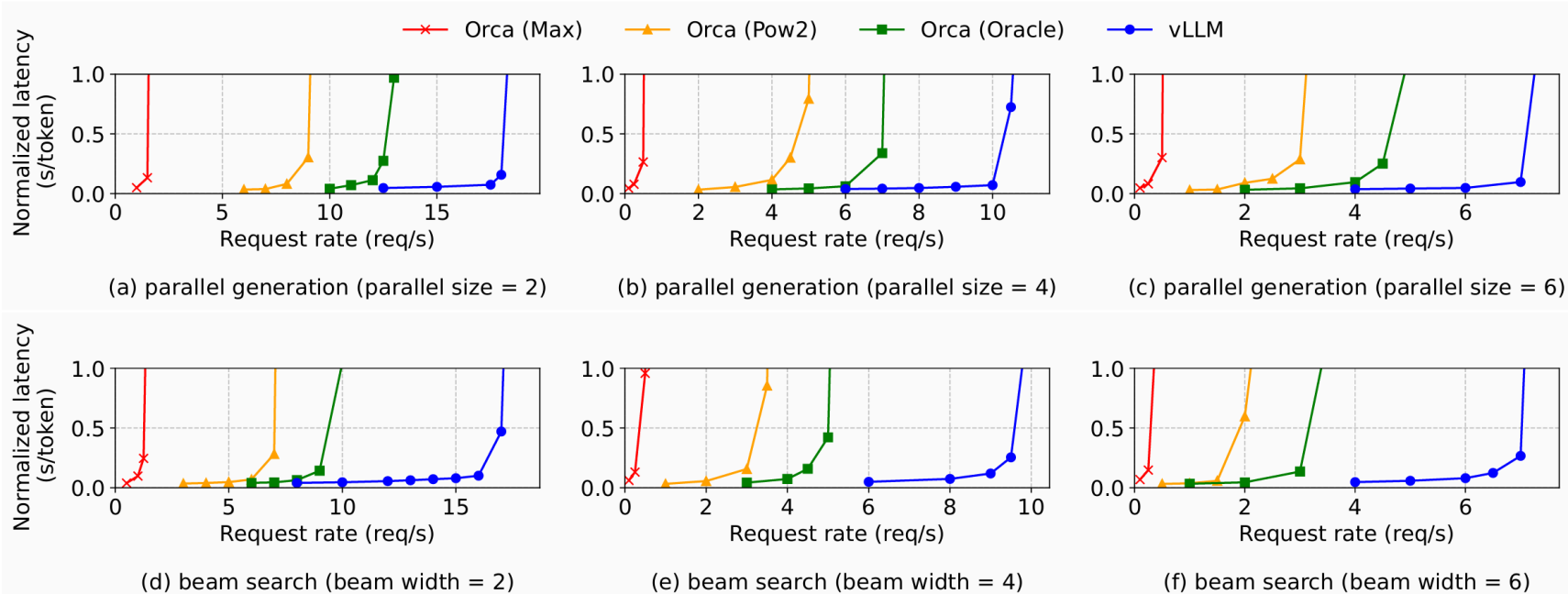
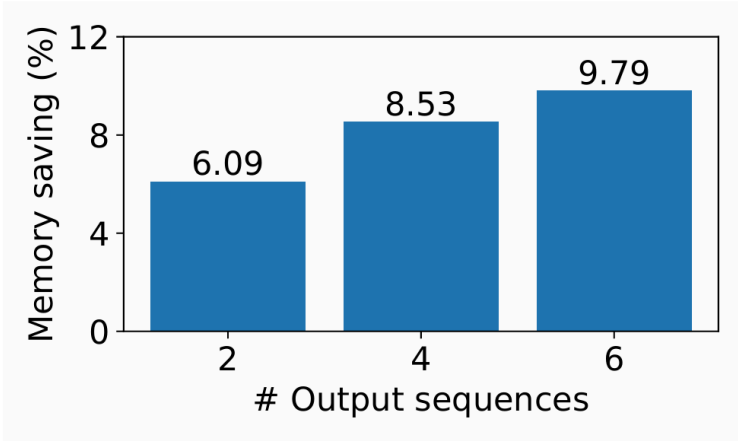
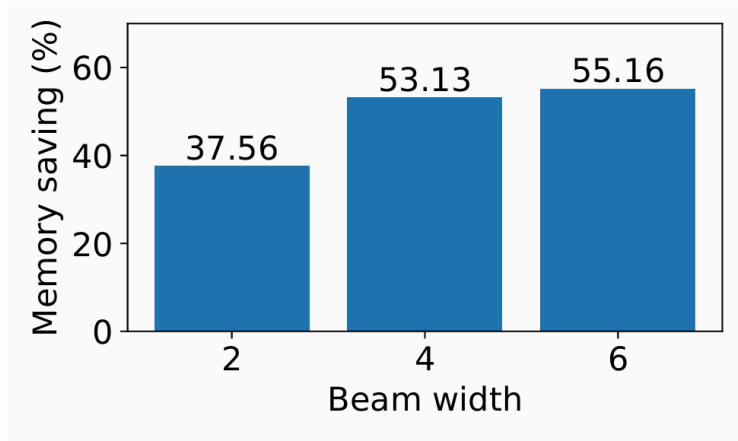


Figure 14. Parallel generation and beam search with OPT-13B on the Alpaca dataset.

Parallel Sampling and Beam Search



(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.

Shared prefix

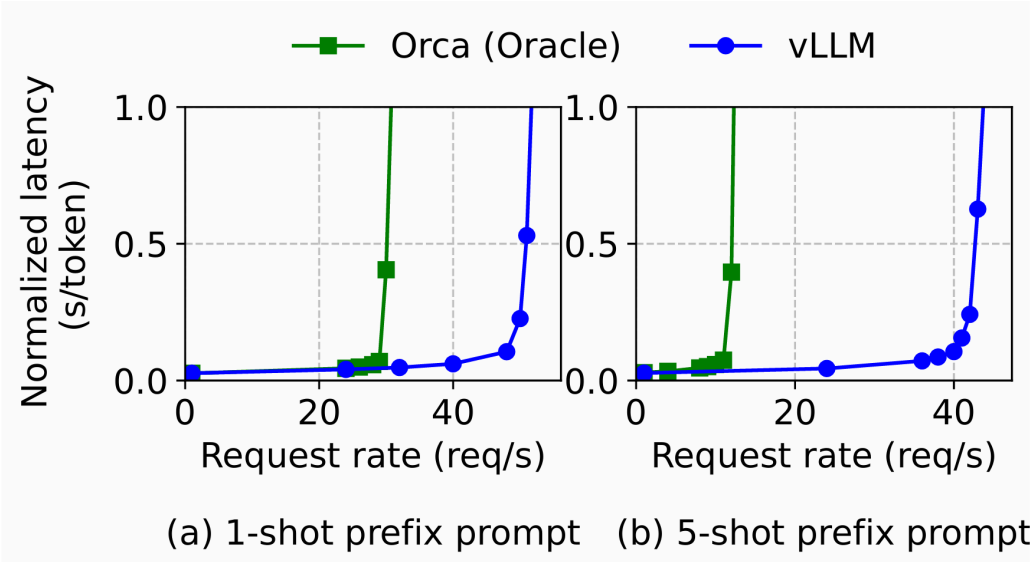


Figure 16. Translation workload where the input prompts share a common prefix. The prefix includes (a) 1 example with 80 tokens or (b) 5 examples with 341 tokens.

Chatbot

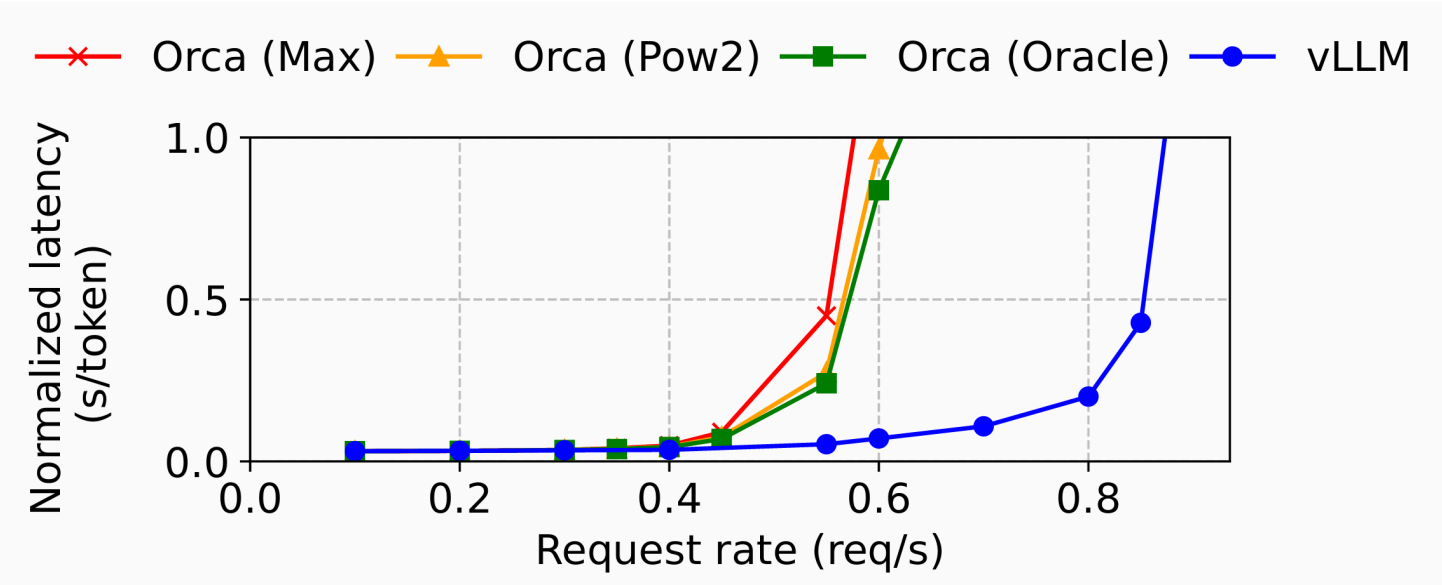
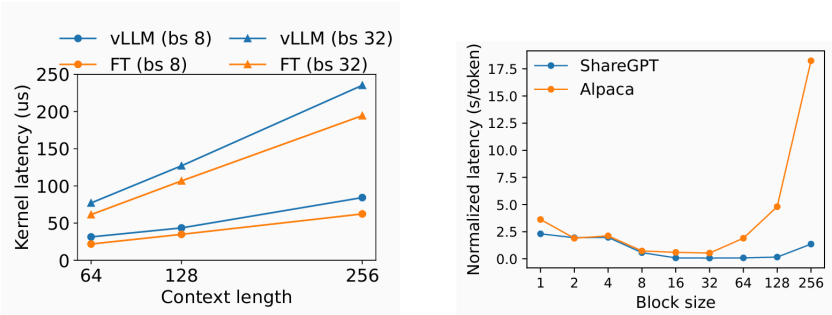


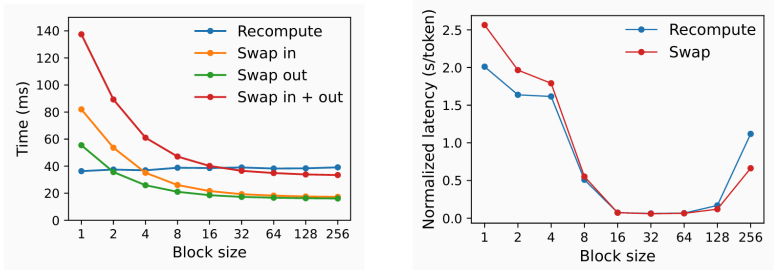
Figure 17. Performance on chatbot workload.

Ablation Study



(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.

Conclusion

Conclusion

- **Problem**
  - Large KV cache
  - Complex decoding algorithms
  - Scheduling for unknown input & output lengths.
- **Solution**
  - Proposes PagedAttention, a new attention algorithm that allows attention keys and values to be stored in non-contiguous paged memory.
  - Presents vLLM, a high-throughput LLM serving system with efficient memory management enabled by PagedAttention.
- **Experiment**
  - 2-4× throughput improvements over the SOTA systems.

Pros

- Key feature is inovative
- Support multi-node inference
- Opensource with Apache 2.0 licences

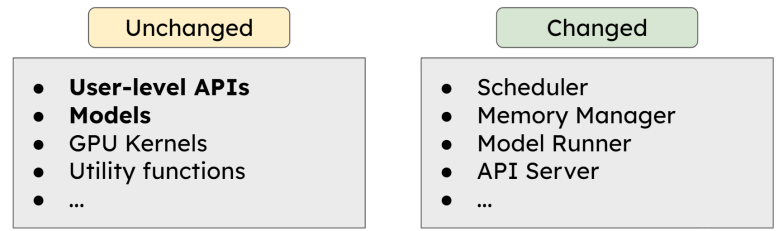
Cons

- Quite unstable

What is vLLM V1?

Re-architect the "core" of vLLM

based on the lessons from V0 (current version)



Q&A