Efficient Memory Management for Large Language Model Serving with PagedAttention

서울시립대학교
UNIVERSITY OF SEOUL

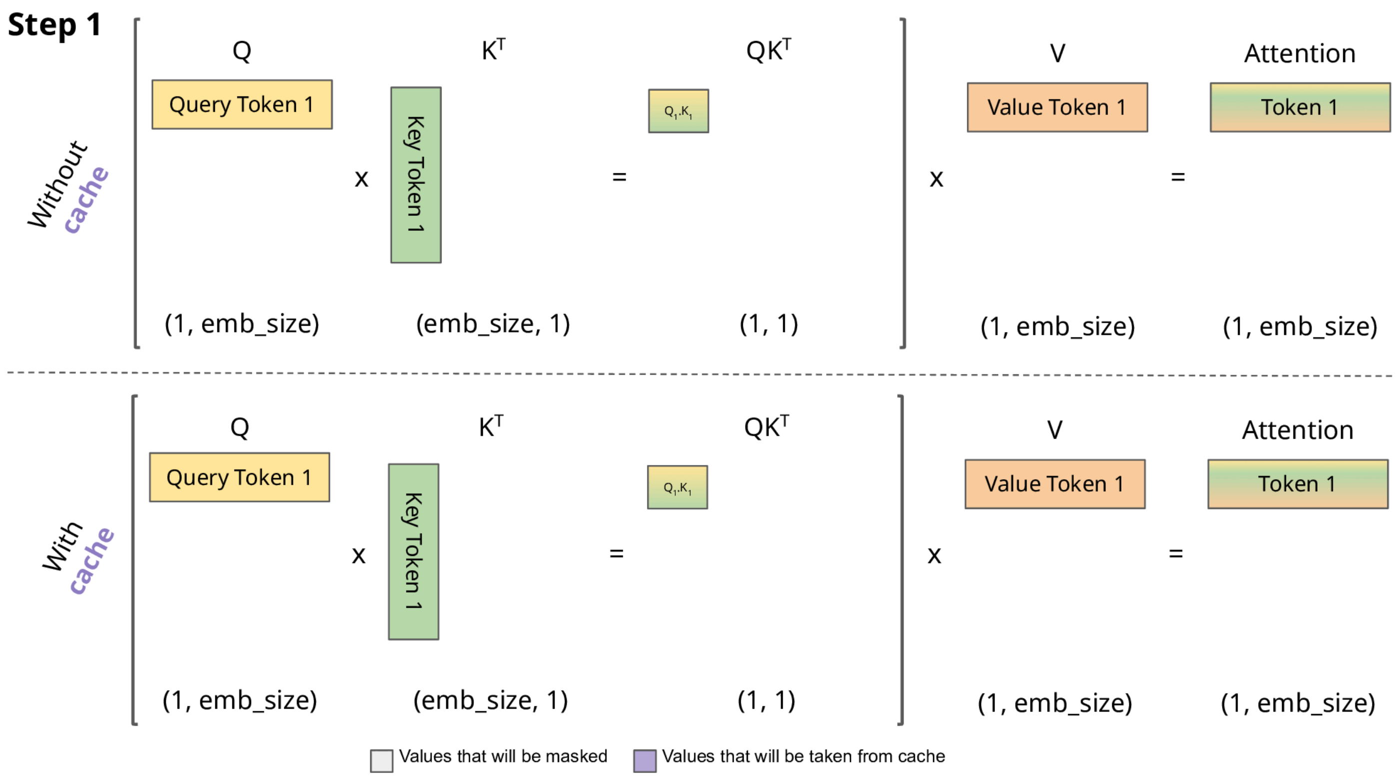
CIDA Lab.

전자전기컴퓨터공학부
2022440119 전형준

2025.04.29

Table of Contents

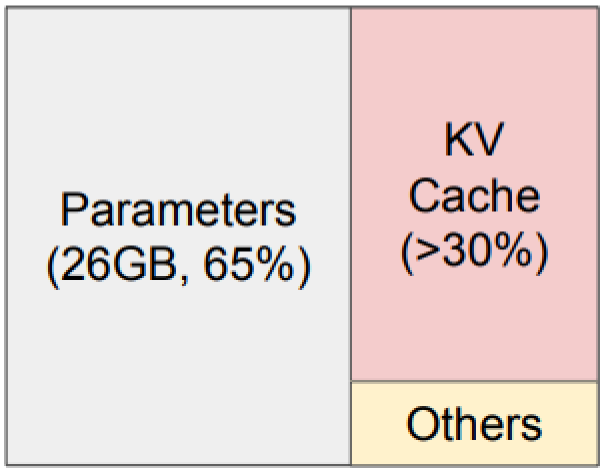
Key Value Cache



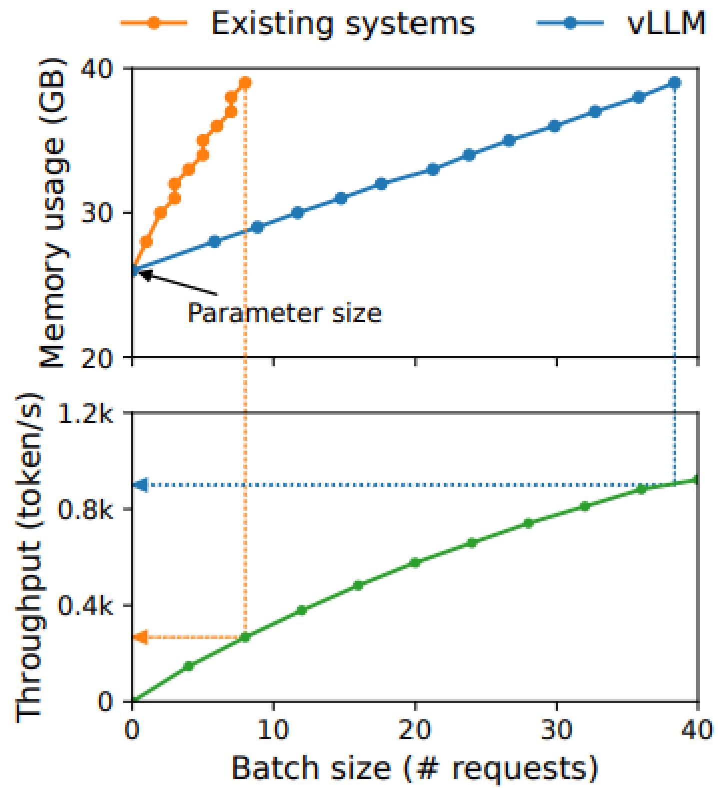
Auto-regressive decoding 과정에서 이전 token들의 key와 value값을 저장해두면, 이를 다시 계산할 필요가 없어 효율적으로 연산이 가능하다.

https://medium.com/@joaolages/kv-caching-explained-276520203249

Key Value Cache



NVIDIA A100 40GB



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.

Limitation of previous systems

In this paper, we observe that existing LLM serving systems fall short of managing the KV cache memory efficiently.

This is mainly because they store the KV cache of a request in contiguous memory space, as most deep learning frameworks require tensors to be stored in contiguous memory.

It dynamically grows and shrinks over time as the model generates new tokens, and its lifetime and length are not known a priori.

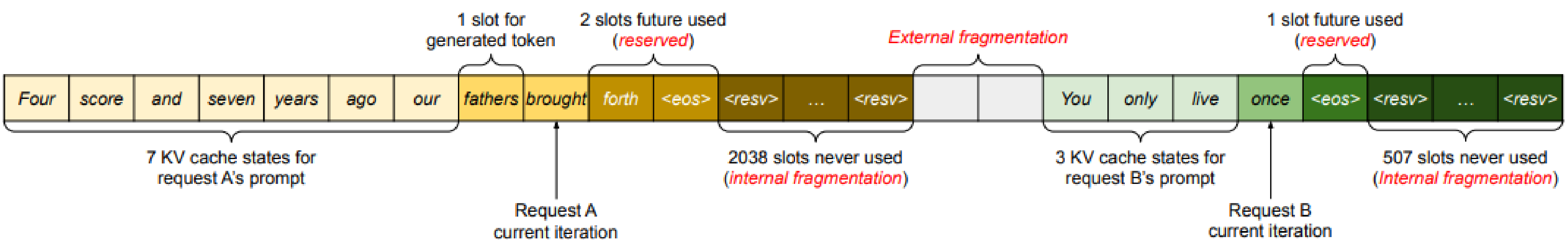
Limitation of previous systems

First, the existing systems suffer from internal and external memory fragmentation.

Internal fragmentation

To store the KV cache of a request in contiguous space, they pre-allocate a contiguous chunk of memory with the request's maximum length (e.g., 2048 tokens).

Besides, external memory fragmentation can also be significant, since the pre-allocated size can be different for each request.



Limitation of previous systems

Second, the existing systems cannot exploit
the opportunities for memory sharing.

In parallel sampling and beam search scenarios,
the request consists of multiple sequences
that can partially share their KV cache.

However, memory sharing is not possible in the
existing systems because the KV cache of
the sequences is stored in separate contiguous spaces.

Limitation of previous systems

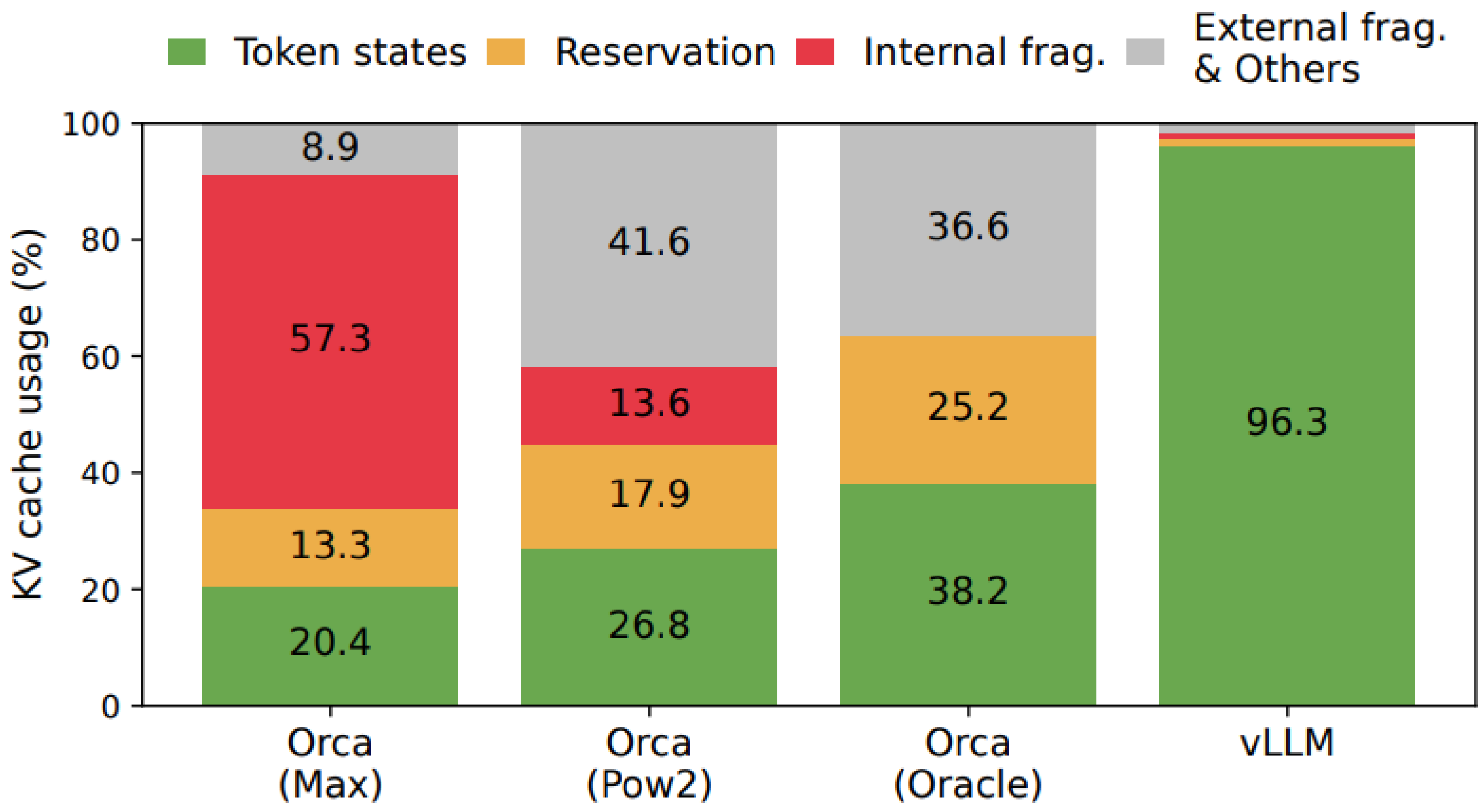


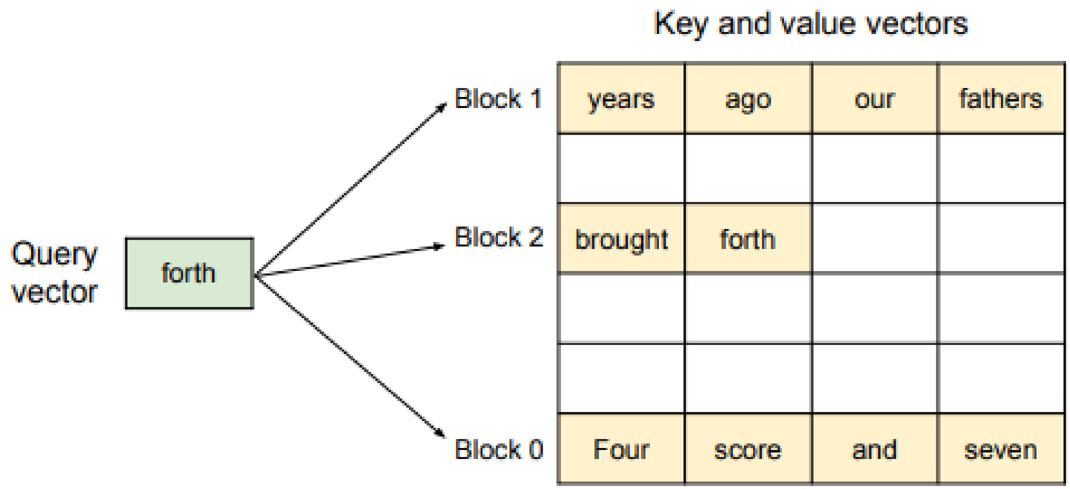
Fig. 2 show that only 20.4% - 38.2% of the KV cache memory is used to store the actual token states in the existing systems.

PagedAttention

PagedAttention divides the request's KV cache into blocks,
each of which can contain the attention keys and
values of a fixed number of tokens.

The blocks for the KV cache are not necessarily
stored in contiguous space.

PagedAttention



This design alleviates internal fragmentation by using relatively small blocks and allocating them on demand.

Moreover, it eliminates external fragmentation as all blocks have the same size.

Finally, it enables memory sharing at the granularity of a block, across the different sequences associated with the same request or even across the different requests.

PagedAttention

$$a_{ij} = \frac{\exp(q_i^\top k_j/\sqrt{d})}{\sum_{t=1}^{i}\exp(q_i^\top k_t/\sqrt{d})}, \quad o_i = \sum_{j=1}^{i} a_{ij} v_j.$$

$$A_{ij} = \frac{\exp(q_i^\top K_j/\sqrt{d})}{\sum_{t=1}^{\lceil i/B \rceil}\exp(q_i^\top K_t \mathbf{1}/\sqrt{d})}, \quad o_i = \sum_{j=1}^{\lceil i/B \rceil} V_j A_{ij}^\top,$$

$$K_j = (k_{(j-1)B+1}, \ldots, k_{jB}) \in \mathbb{R}^{d\times B} \quad q_i \in \mathbb{R}^d$$

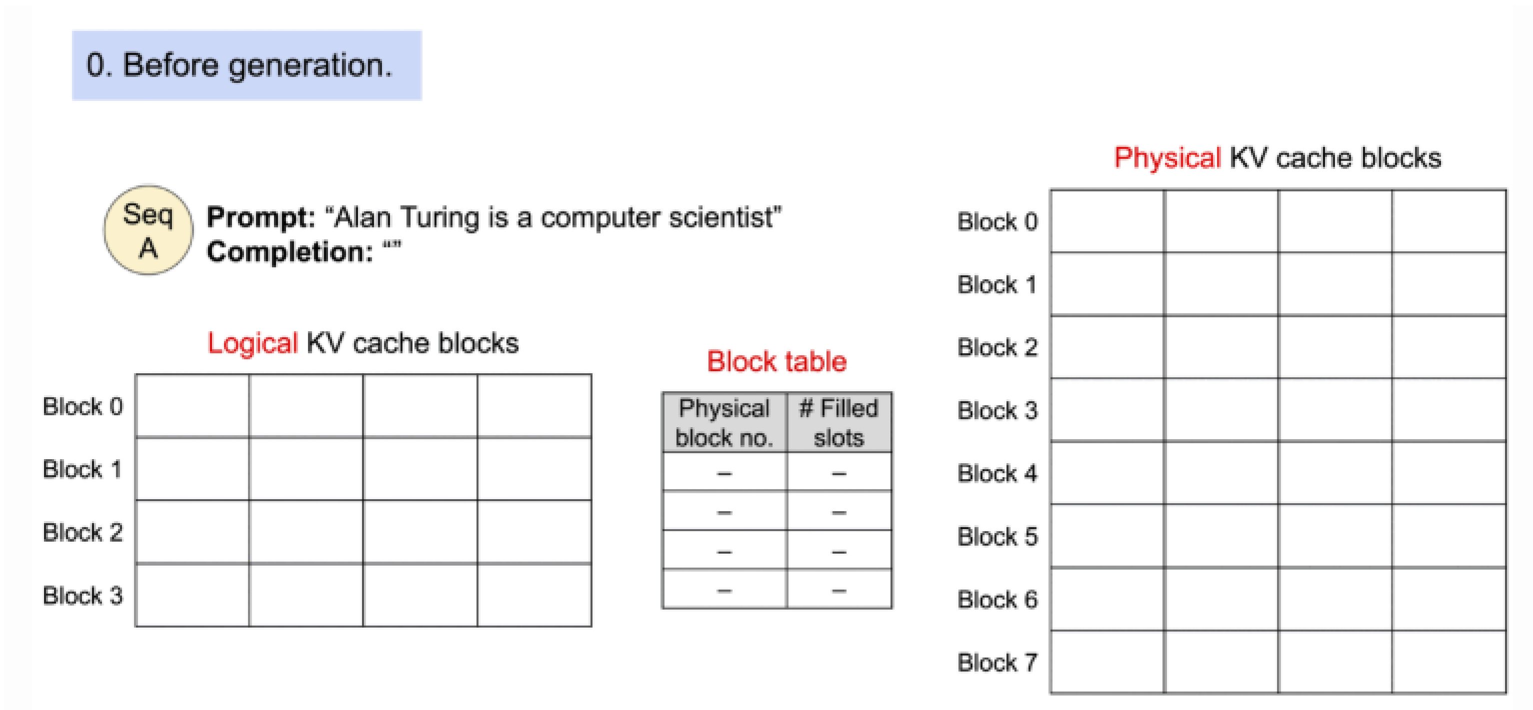
Aij is the row vector of attention score in jth KV block.

KV Cache Manager

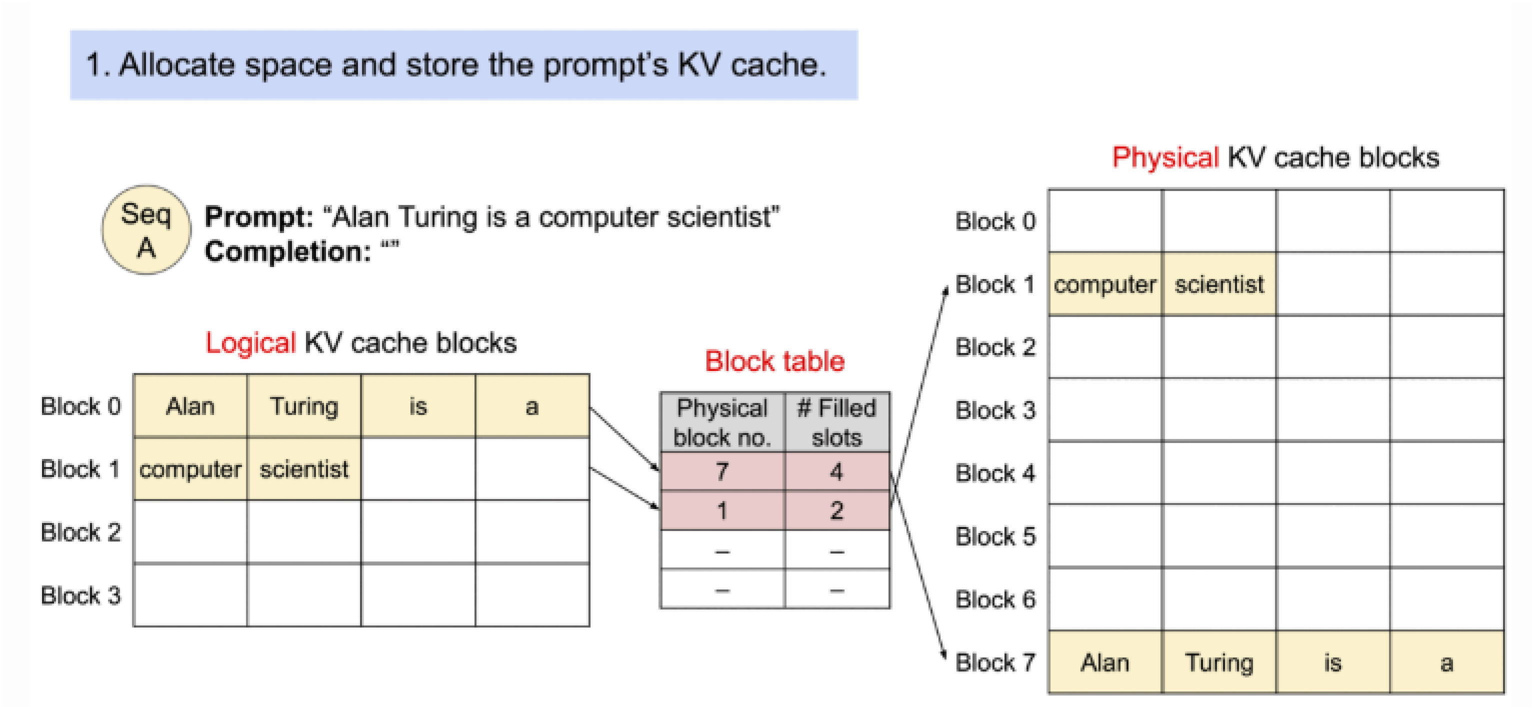
OS partitions memory into fixed-sized pages and maps user programs' logical pages to physical pages. Contiguous logical pages can correspond to non-contiguous physical memory pages, allowing user programs to access memory as though it were contiguous.

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

Decoding with PagedAttention



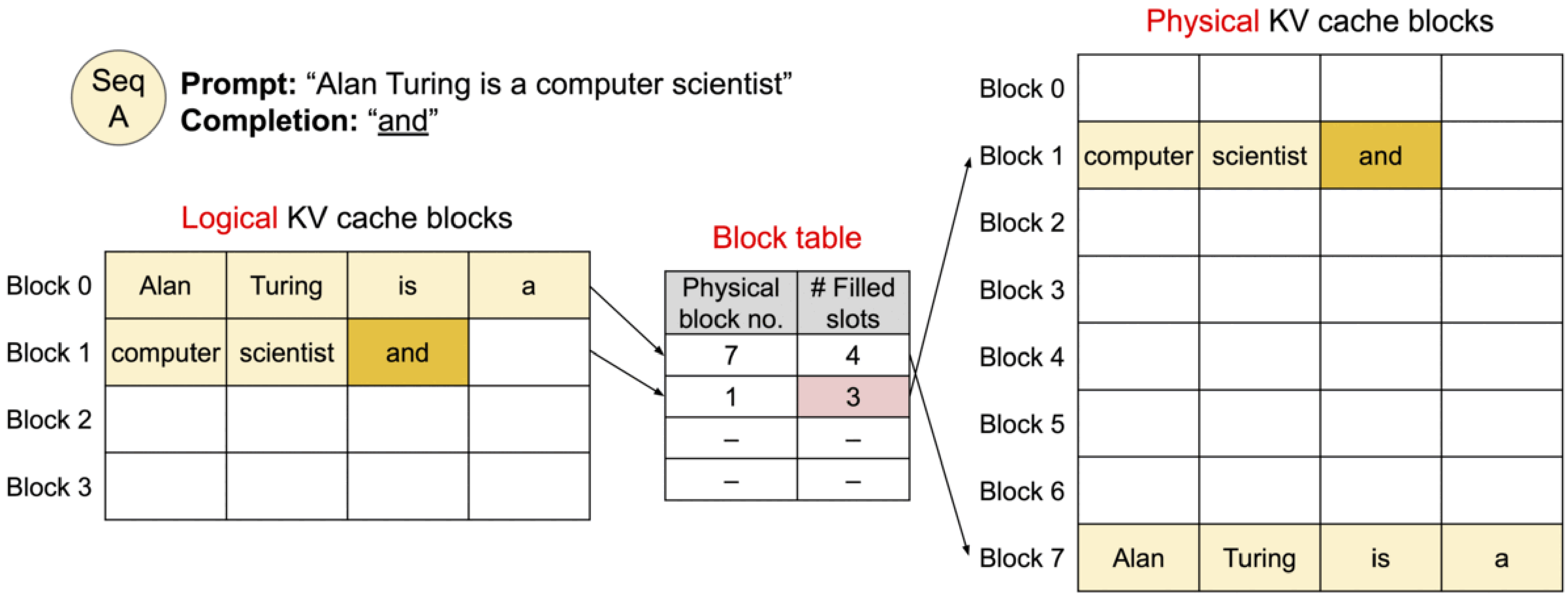
Decoding with PagedAttention



In the prefill step, vLLM generates the KV cache of the prompts and the first output token with a conventional self-attention algorithm

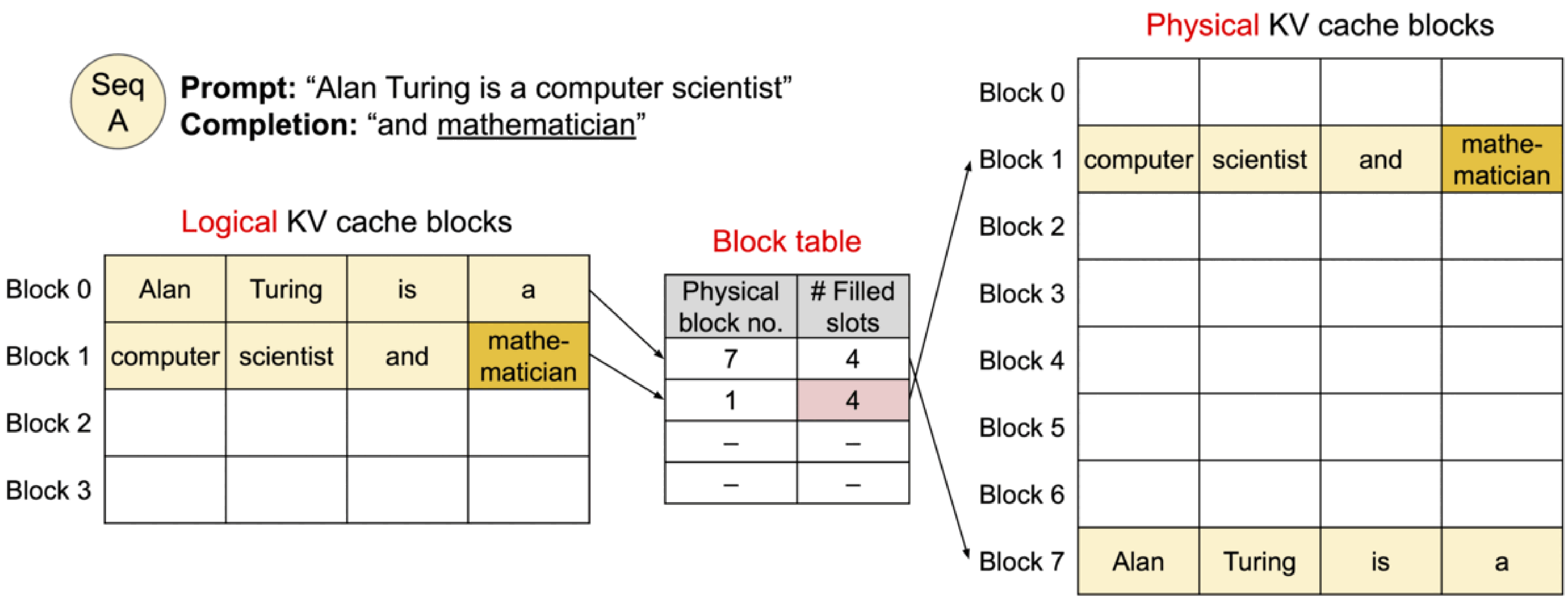
Decoding with PagedAttention

2. Generated 1st token.



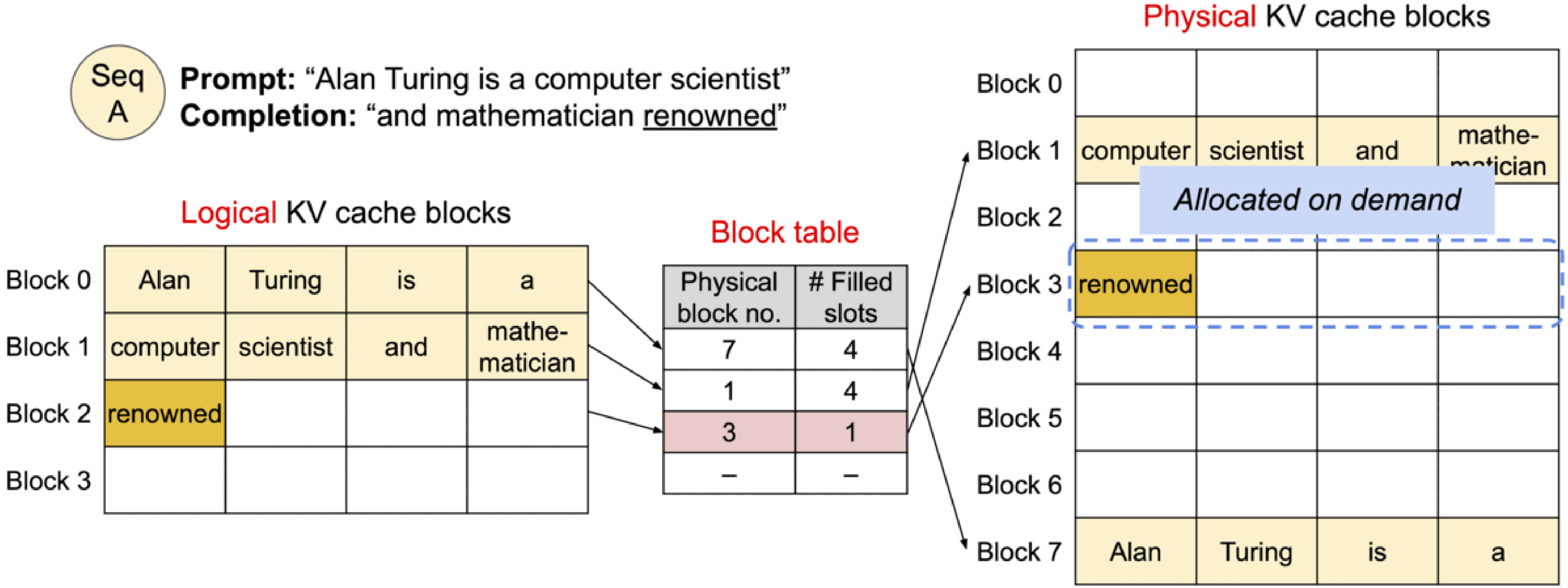
Decoding with PagedAttention

3. Generated 2nd token.



Decoding with PagedAttention

4. Generated 3rd token. Allocate new block.



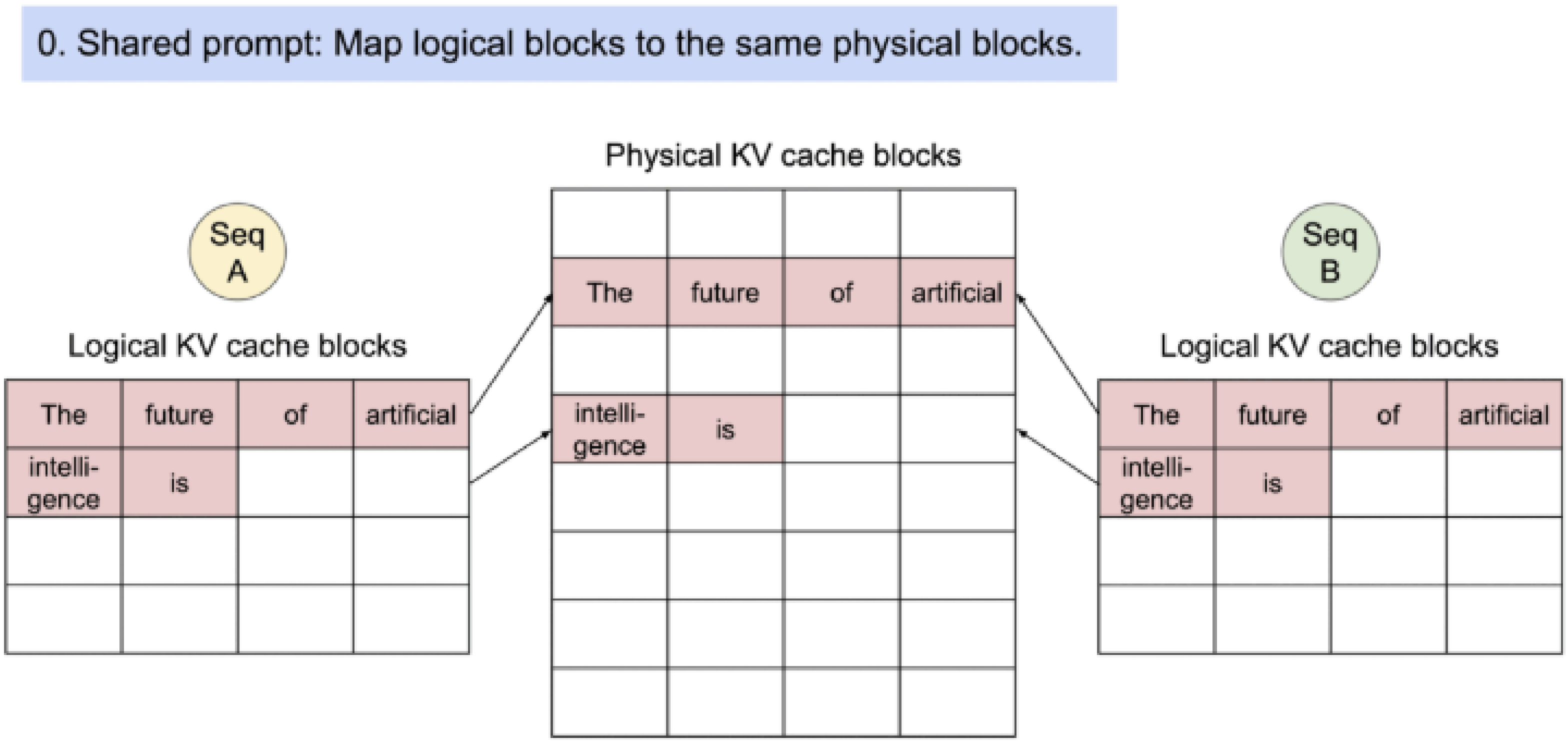
Decoding with PagedAttention

Second, the existing systems cannot exploit
the opportunities for memory sharing.

In parallel sampling and beam search scenarios,
the request consists of multiple sequences
that can partially share their KV cache.

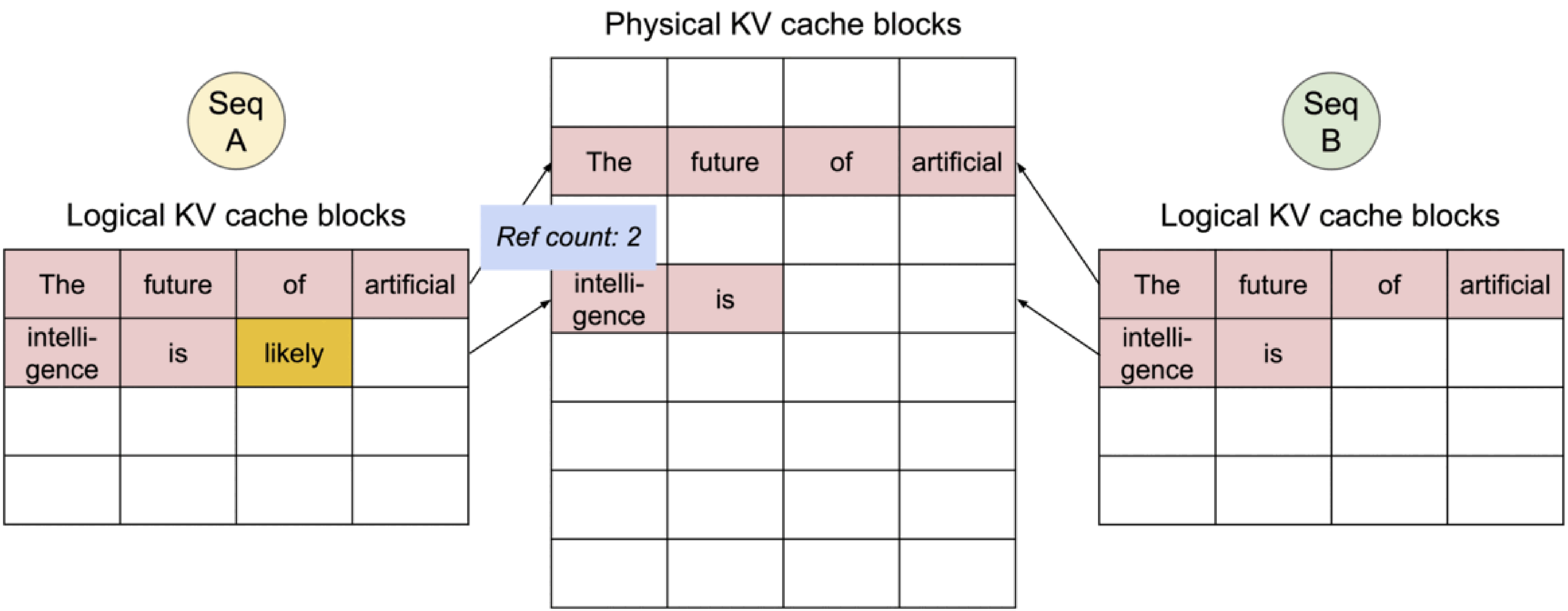
However, memory sharing is not possible in the
existing systems because the KV cache of
the sequences is stored in separate contiguous spaces.

Decoding with PagedAttention



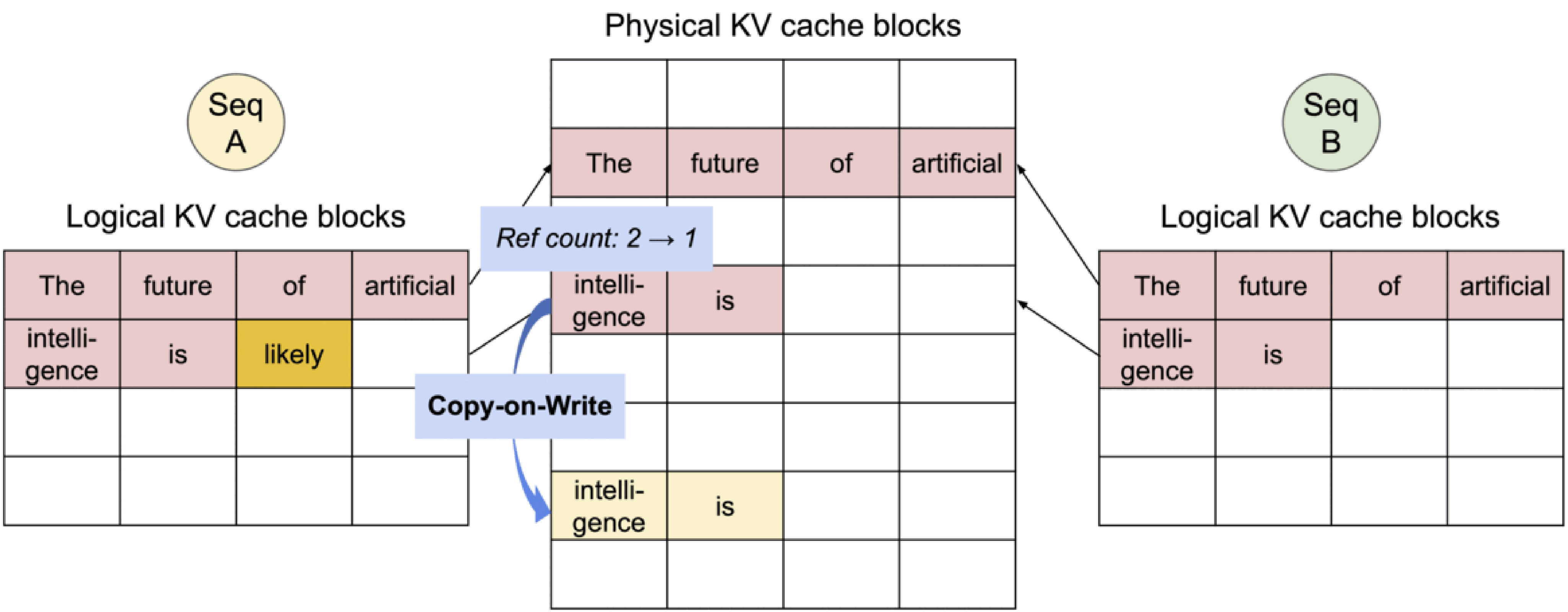
Decoding with PagedAttention

1. Seq A generated 1st token.



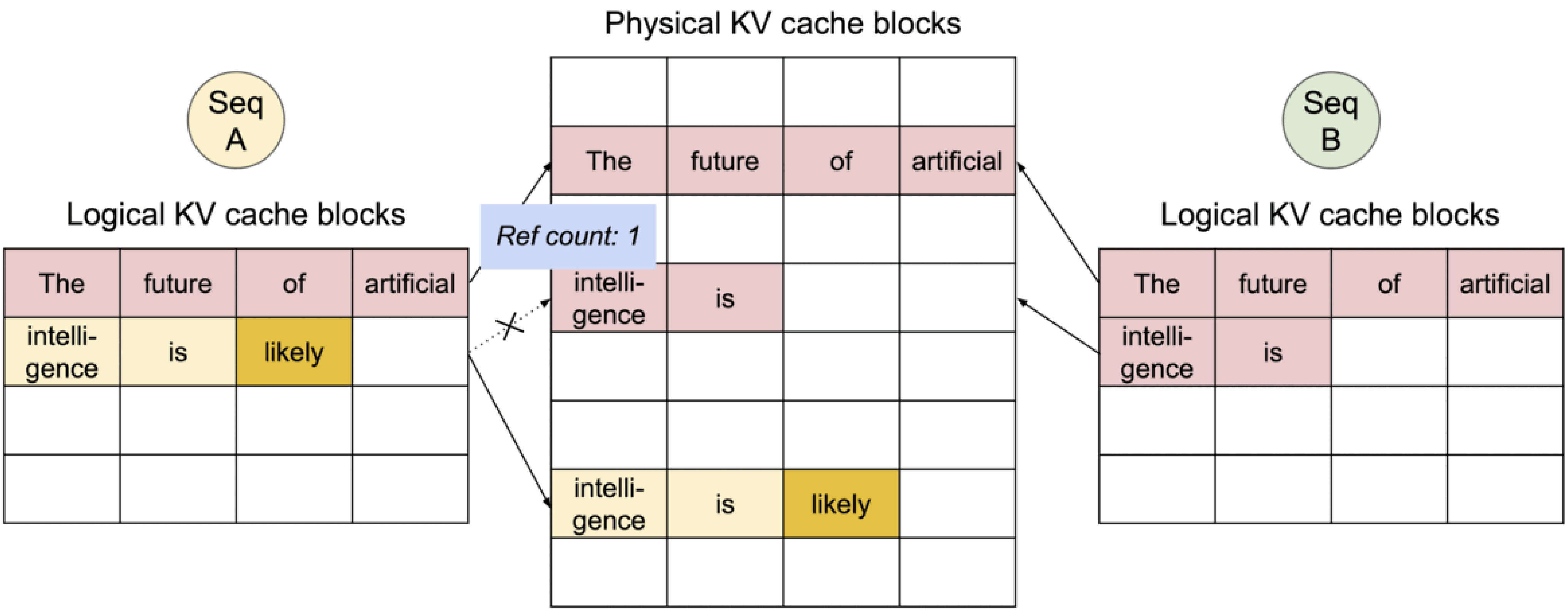
Decoding with PagedAttention

2. Copy-on-Write: Copy to a new block.



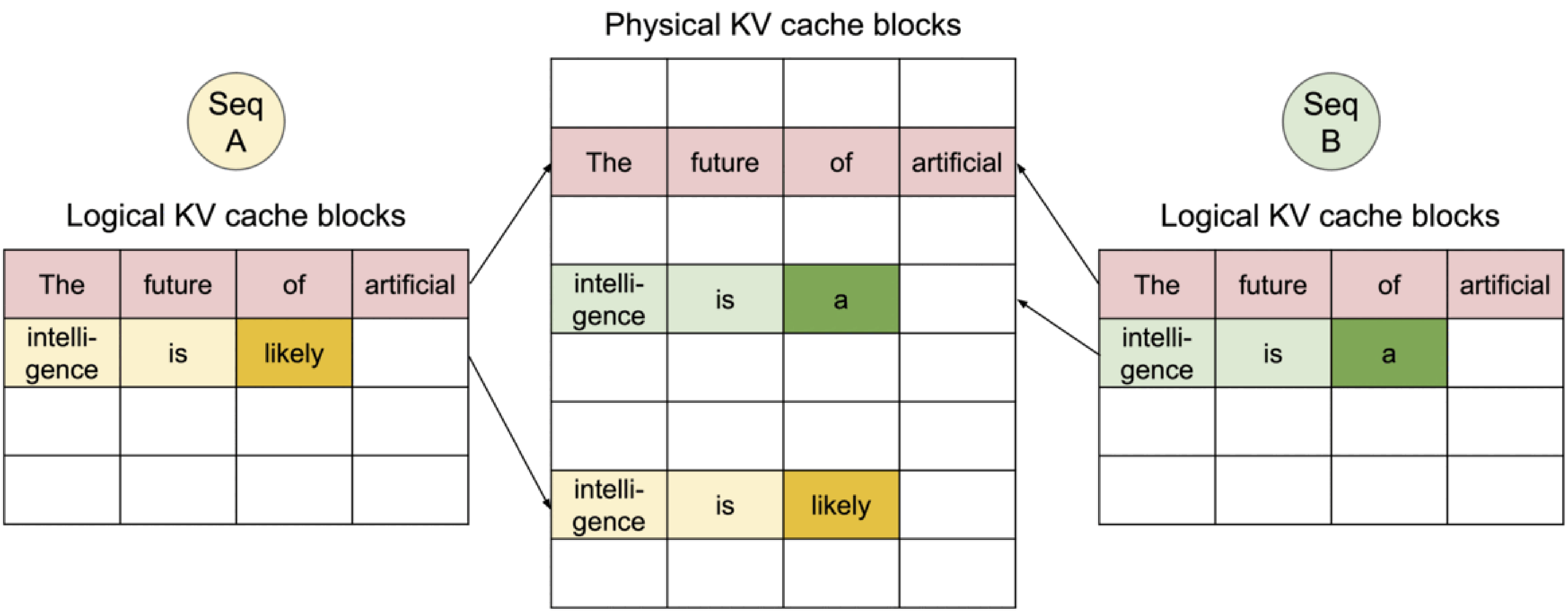
Decoding with PagedAttention

2. Copy-on-Write: Copy to a new block.

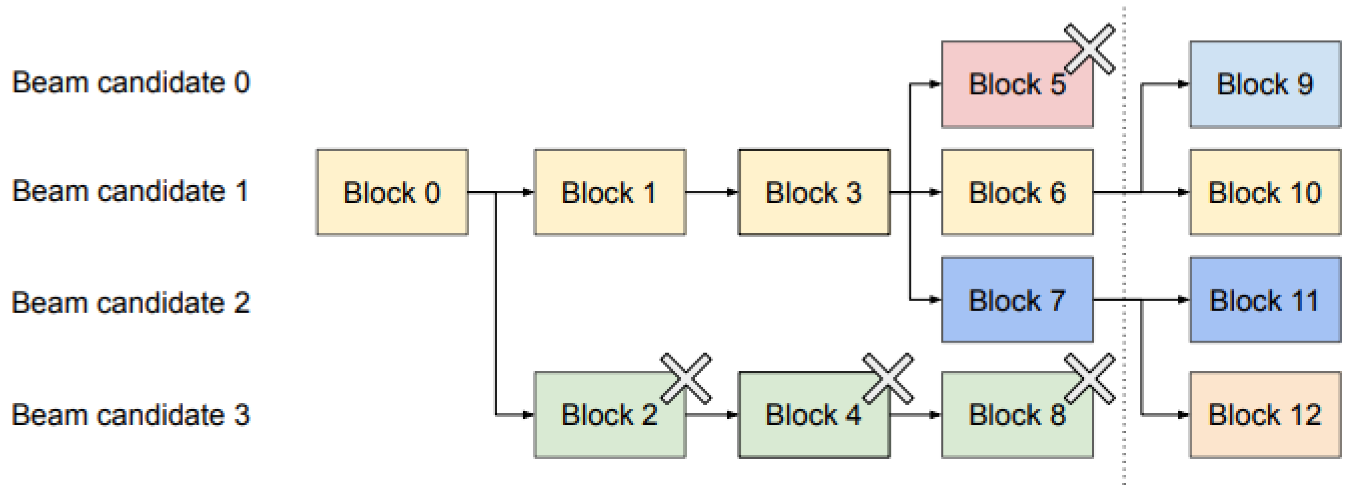


Decoding with PagedAttention

3. Seq B generated 1st token. No copy needed.



Decoding with PagedAttention

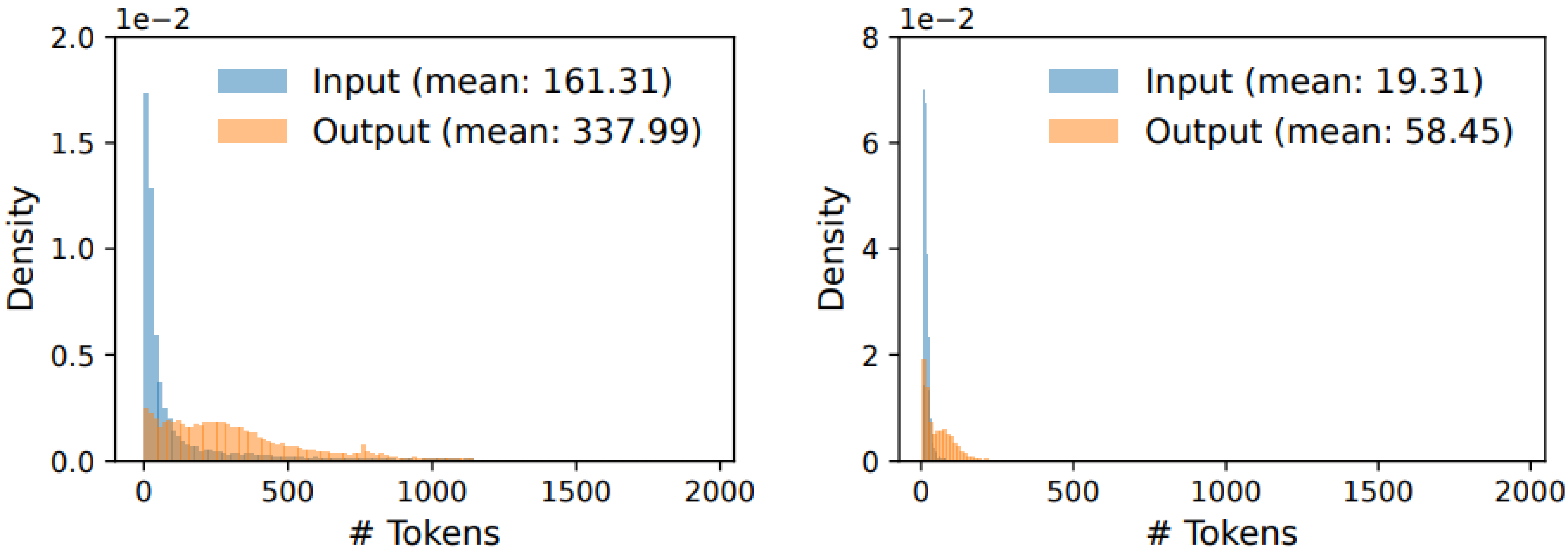


Experiment

Model = OPT-13B, OPT-66B, OPT-175B, LLaMA-13B
Hardware: NVIDIA A100

Dataset
ShareGPT: Realistic, long and diverse prompts and responses.
(User – Shared Conversation with Chat GPT)
Alpaca: Short, instruction-tuning style tasks
(Instruction Dataset generated by GPT - 3.5)

Experiment



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

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

Experiment

Baseline 1: FasterTransformer.
FasterTransformer is a distributed inference engine highly optimized for latency. Specifically, we set a maximum batch size *B* as large as possible for each experiment, according to the GPU memory capacity.

Baseline 2 : Orca
Orca is a state-of-the-art LLM serving system optimized for throughput.

Orca (Oracle). We assume the system has the knowledge of the lengths of the outputs that will be actually generated for the requests.

Orca (Pow2). We assume the system over-reserves the space for outputs by at most 2×

Orca (Max). We assume the system always reserves the space up to the maximum sequence length of the model, i.e., 2048 tokens.

Experiment

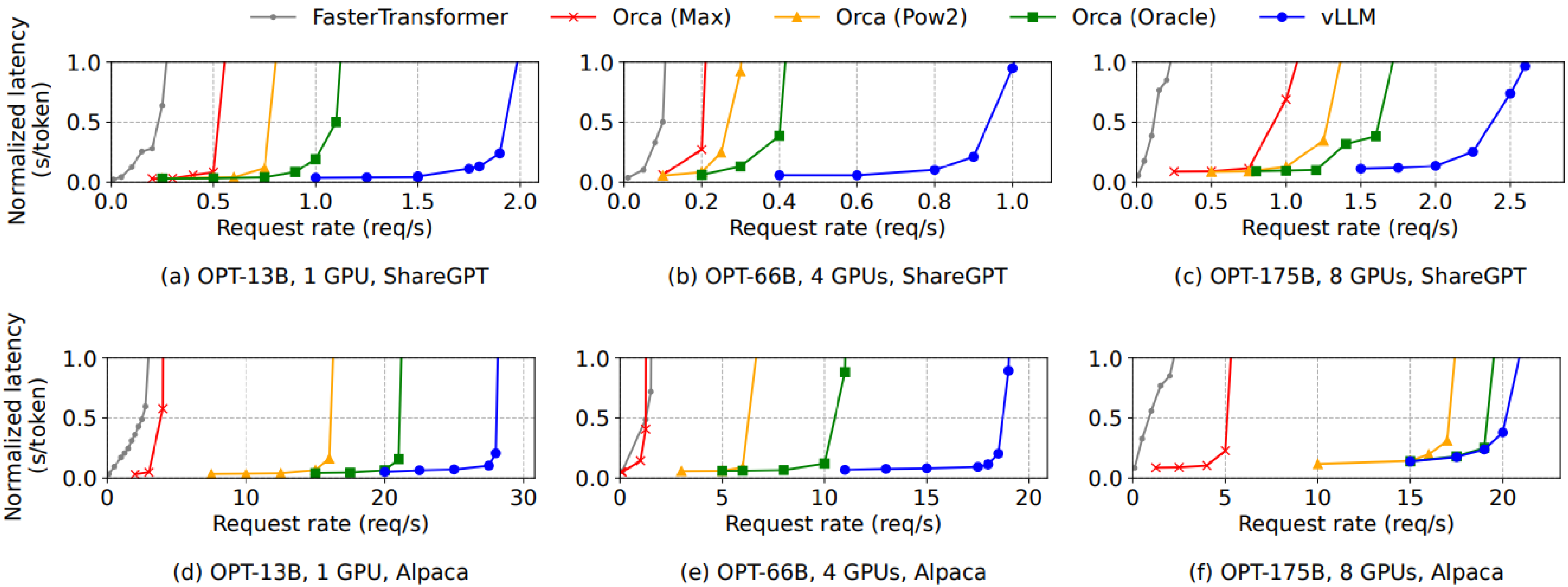


Figure 12. Single sequence generation with OPT models on the ShareGPT and Alpaca dataset

Experiment

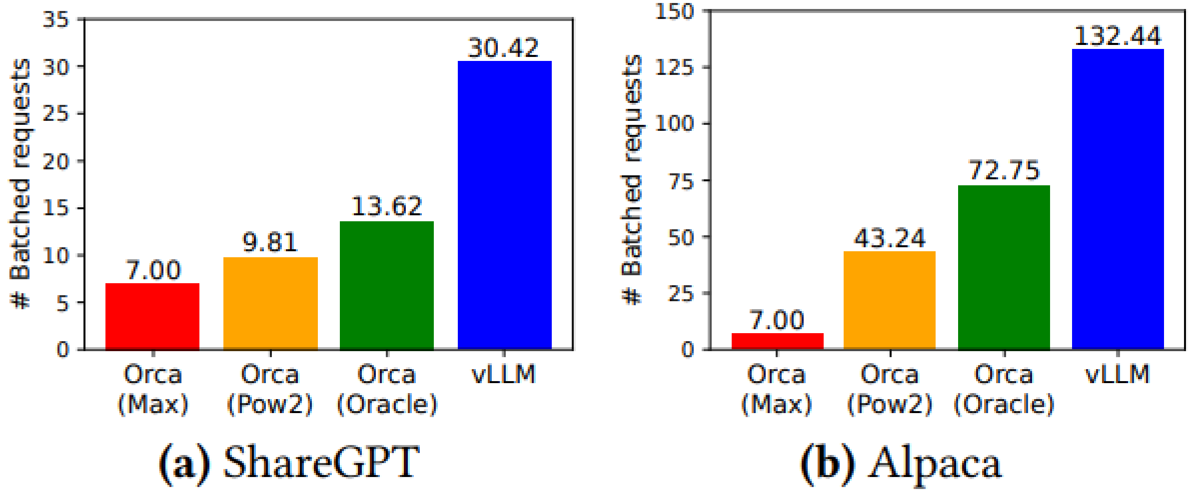


Figure 13. Average number of batched requests when serving OPT-13B for the ShareGPT (2 reqs/s) and Alpaca (30 reqs/s) traces.

Experiment

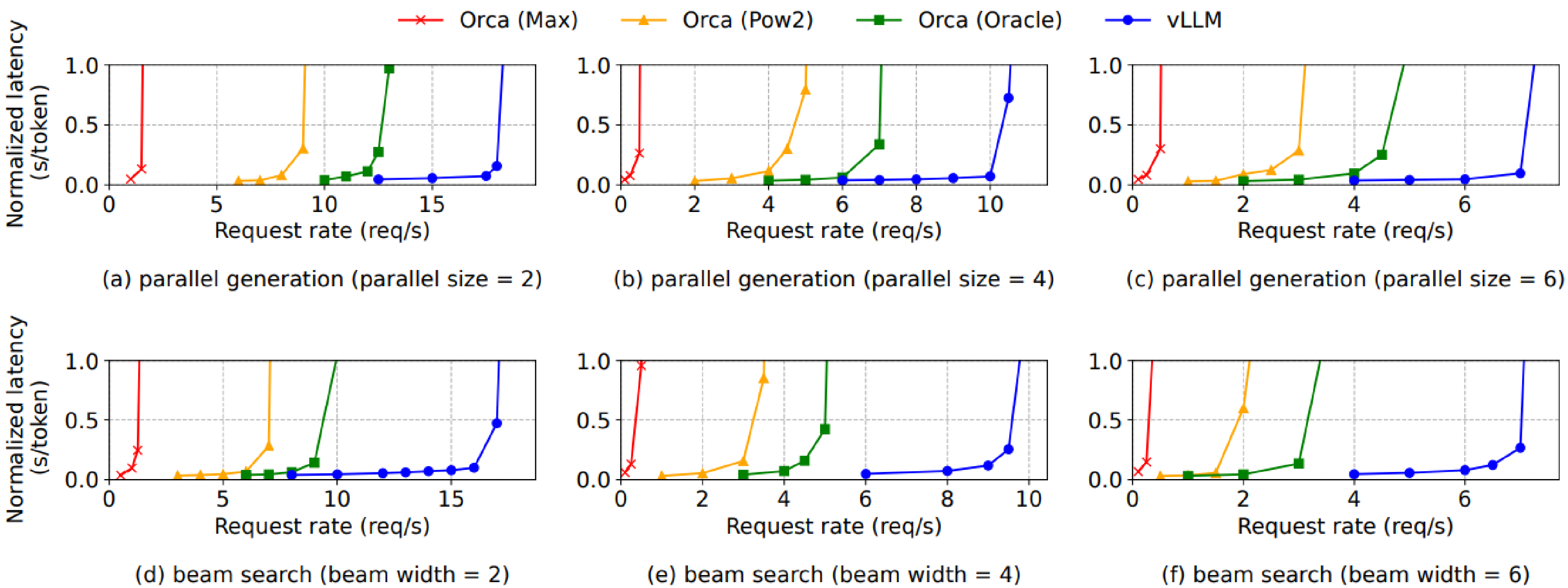
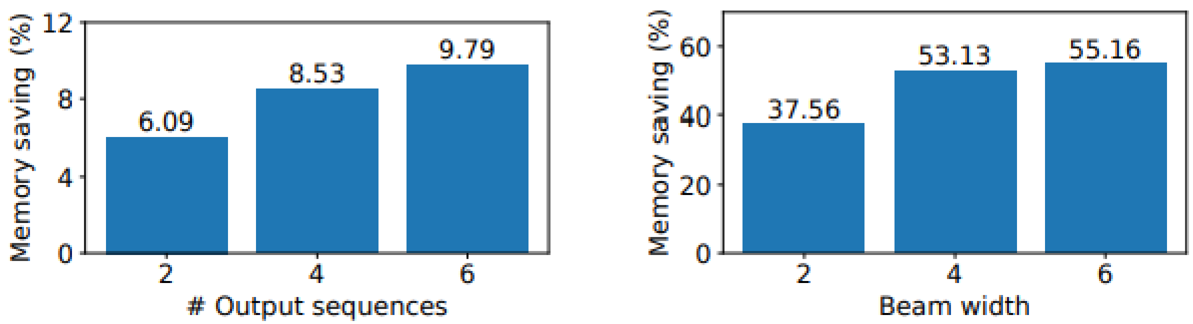


Figure 14. Parallel generation and beam search with OPT-13B on the Alpaca dataset.

Experiment



(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.