



CONFERENCE

—NORTH AMERICA 2026—

Prefix Caching for Autoregressive Stages in Multi-Stage Pipelines

 @alex-jw-brooks

Alex Brooks

- Machine Learning Engineer
- vLLM Omni maintainer



 @oglok

Ricardo Noriega

- Software Engineer (currently) working on vLLM Omni
- Blog: <http://oglok.es>

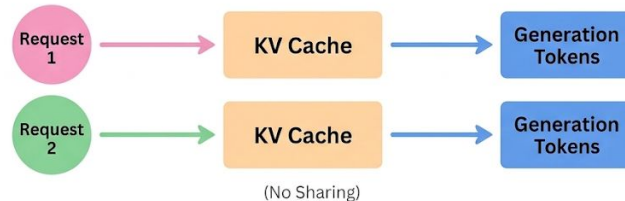


 **PyTorch**
Foundation
AMBASSADOR

What is Prefix Caching?

- KV cache is used to avoid recomputation per generation step within a single request
 - It is discarded once generation completes
- Prefix Caching shares KV tensors across independent requests
 - Matches prefixes using block hashing in the pool
 - It helps reducing Time-To-First-Token (TTFT)

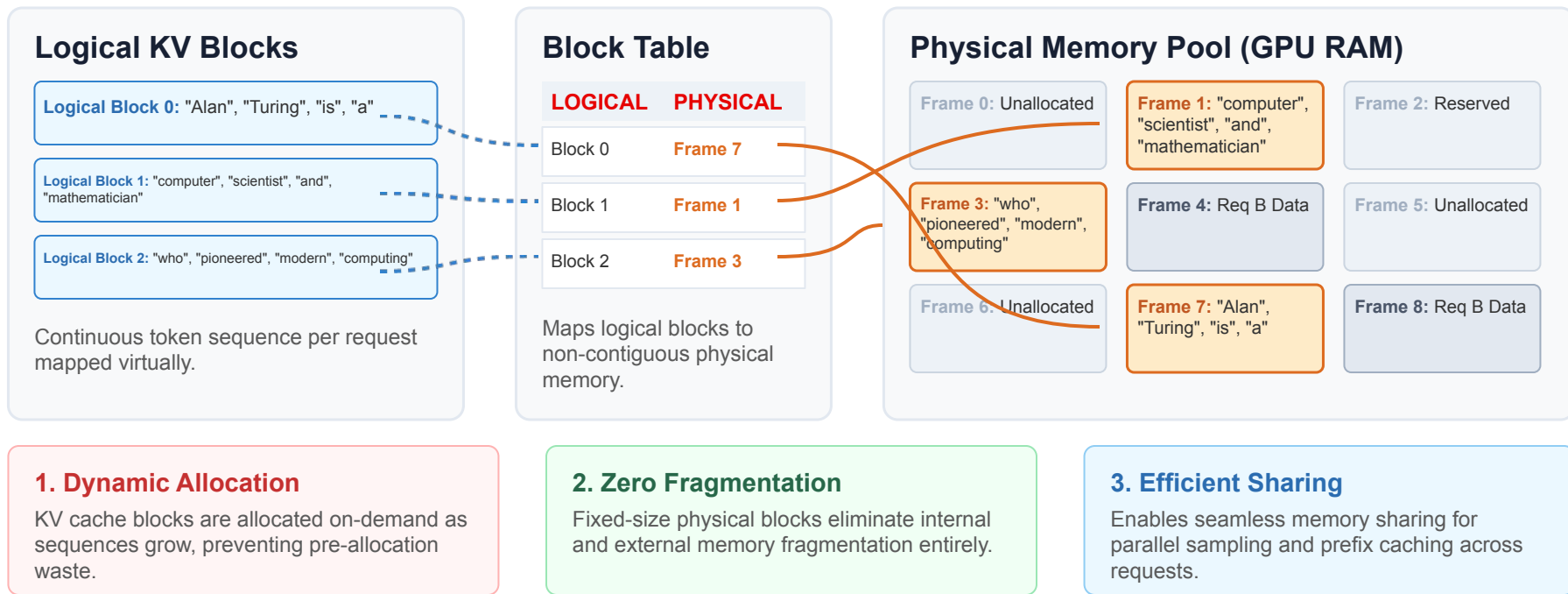
STANDARD KV CACHE



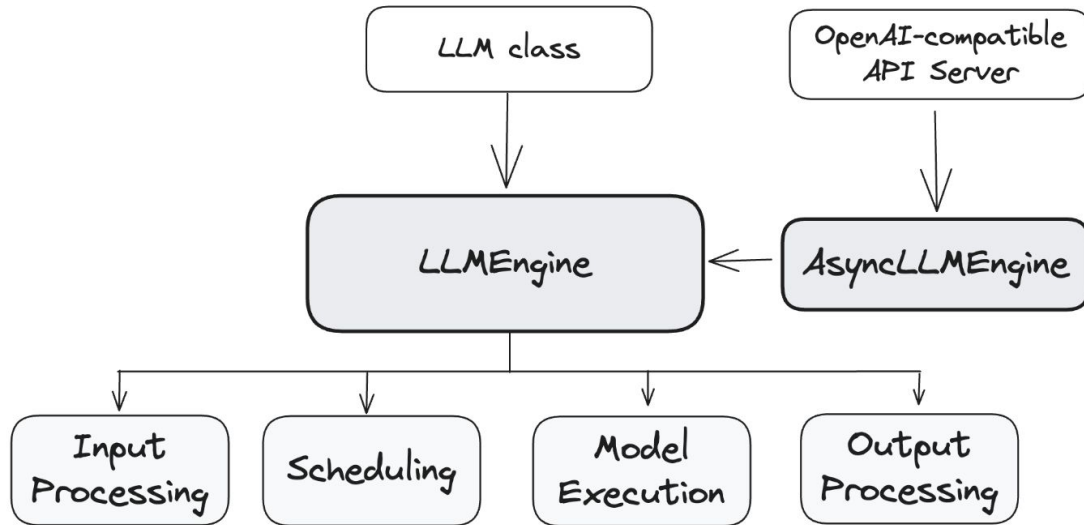
PREFIX CACHING



How PagedAttention Works

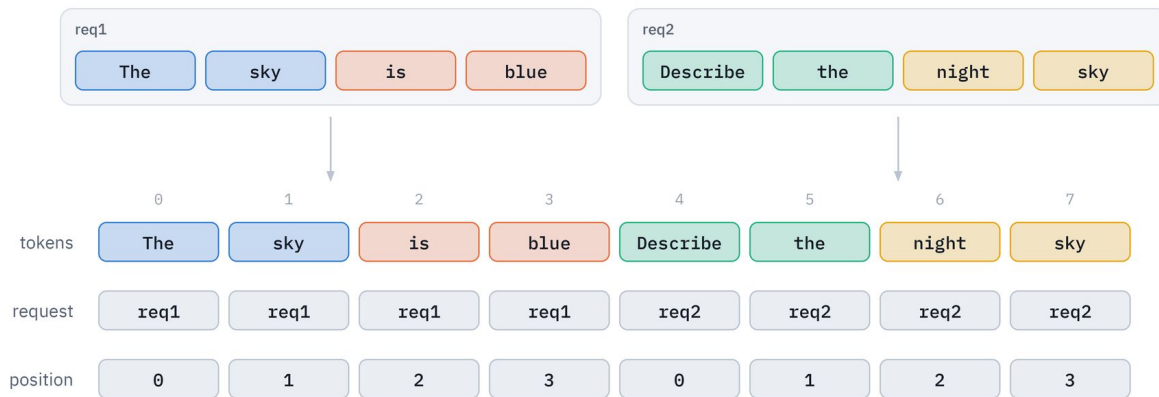


LLM Engine

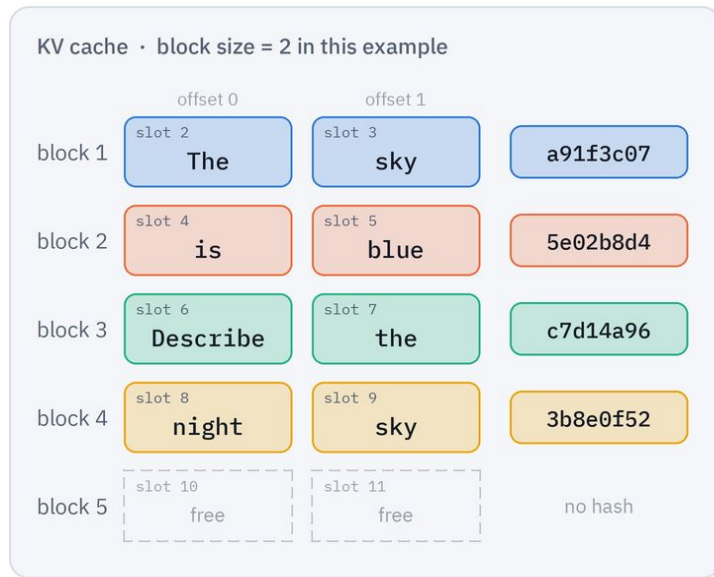
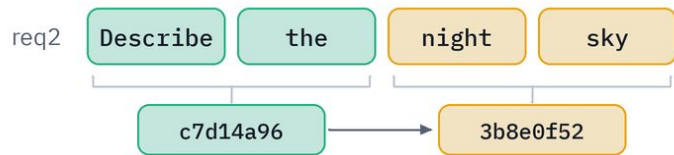
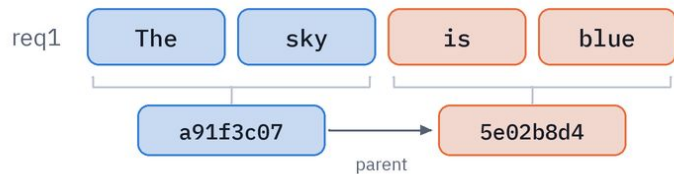


vLLM [Engine Step]

- Say we have two scheduled requests, and set `block_size=2`.
 - Request 1: "The sky is blue"
 - Request : "Describe the night sky"



vLLM [Block Pool]



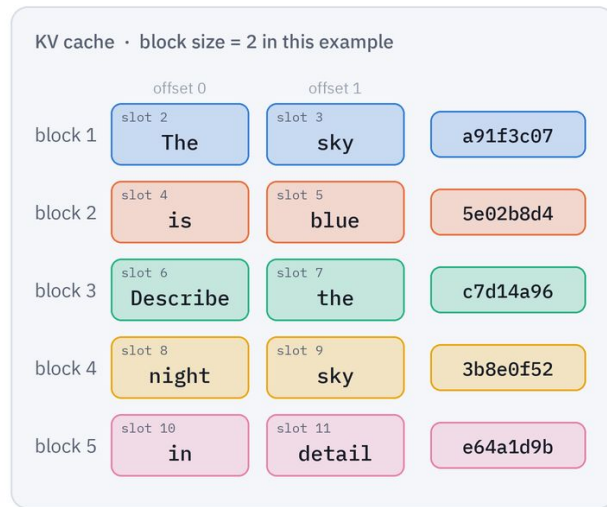
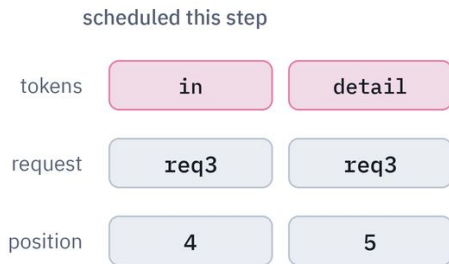
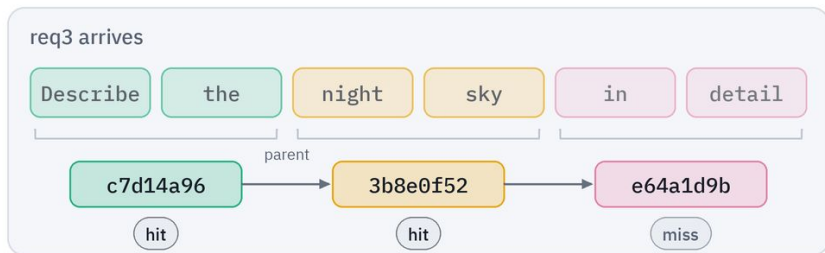
hash → block

$$\text{slot} = \text{block} \times \text{block size} + \text{offset}$$

block size 2 here: “The” = $1 \times 2 + 0 = \text{slot } 2$ “Describe” = $3 \times 2 + 0 = \text{slot } 6$

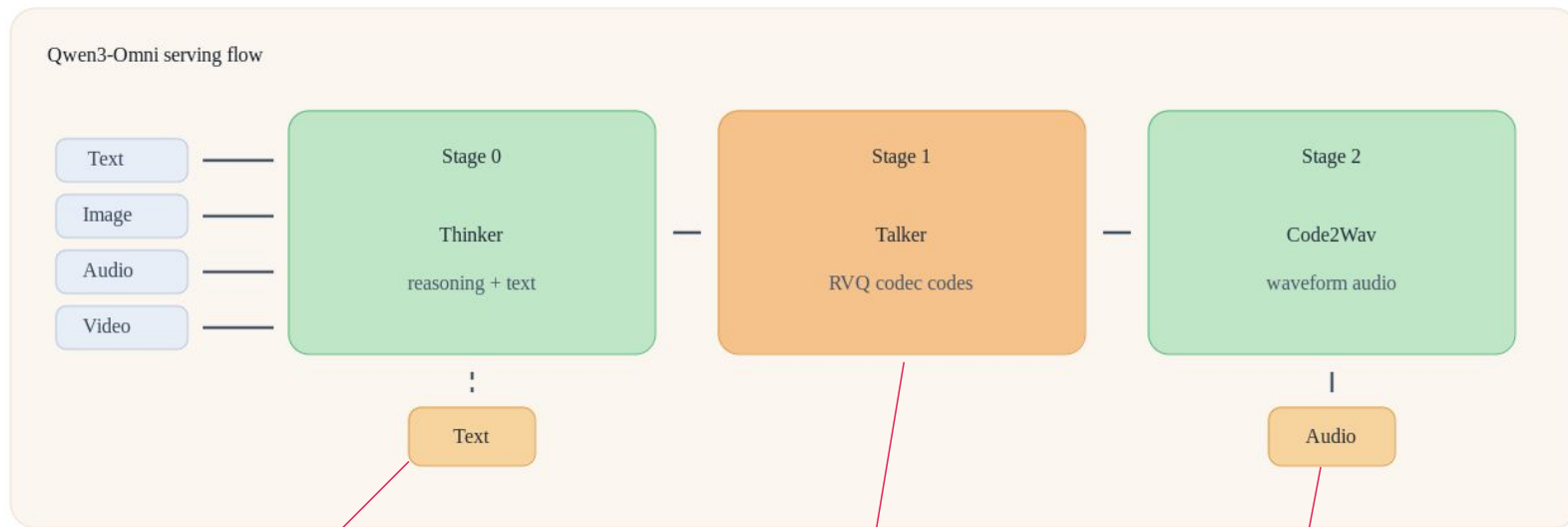
- Now, say we get a new request, “Describe the night sky in detail”; what happens?

vLLM [Prefix Cache Hit for KV Cache Reuse]



- 1 hash each full block (with its parent)
- 2 c7d14a96 → block 3, 3b8e0f52 → block 4: hits
- 3 e64a1d9b: new, gets block 5
- 4 only in detail runs, at positions 4 and 5

Qwen3-Omni



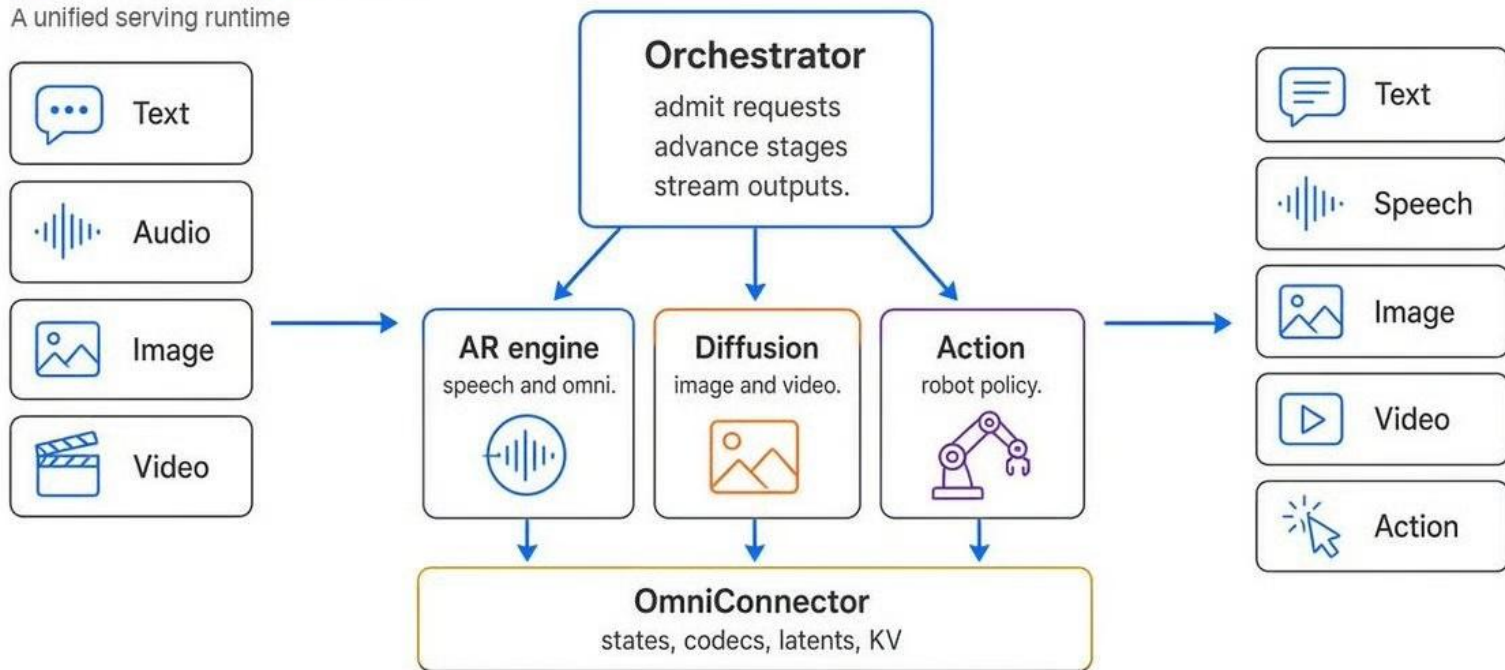
This can run in vLLM (AR decoder)

AR scheduling, takes hidden states from thinker

Not AR, converts audio codec tokens to waveform

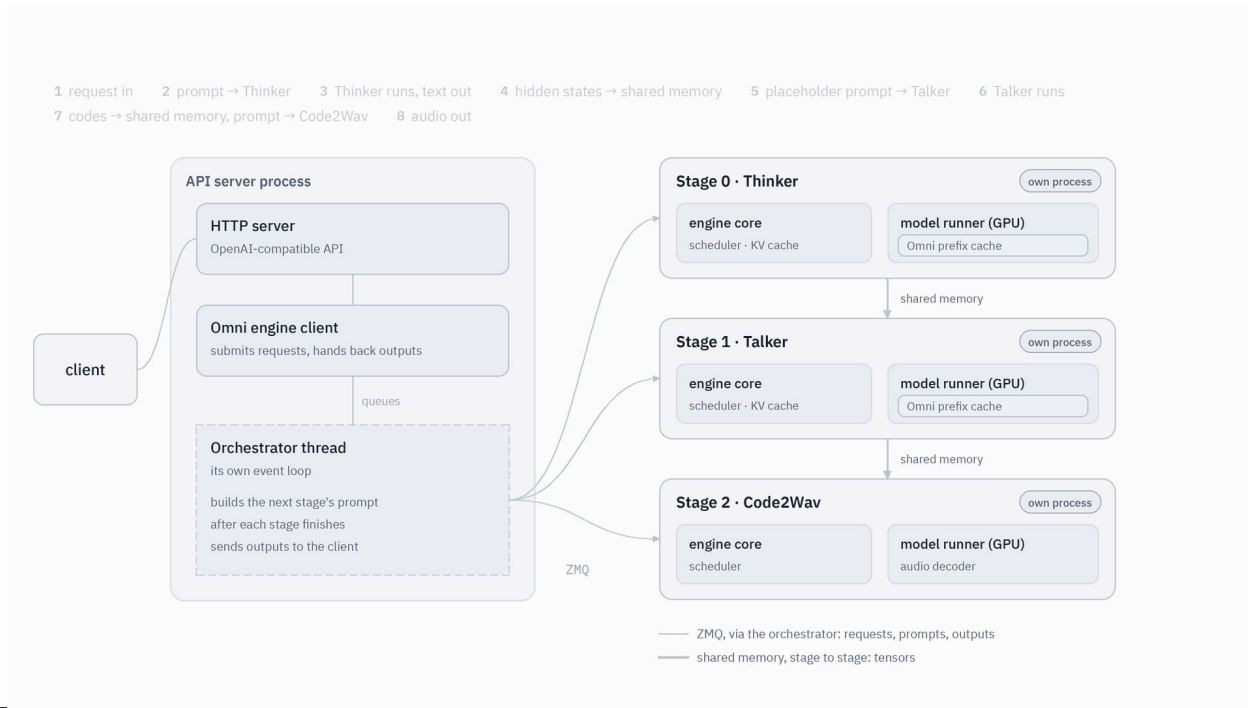
LLM-Omni

A unified serving runtime



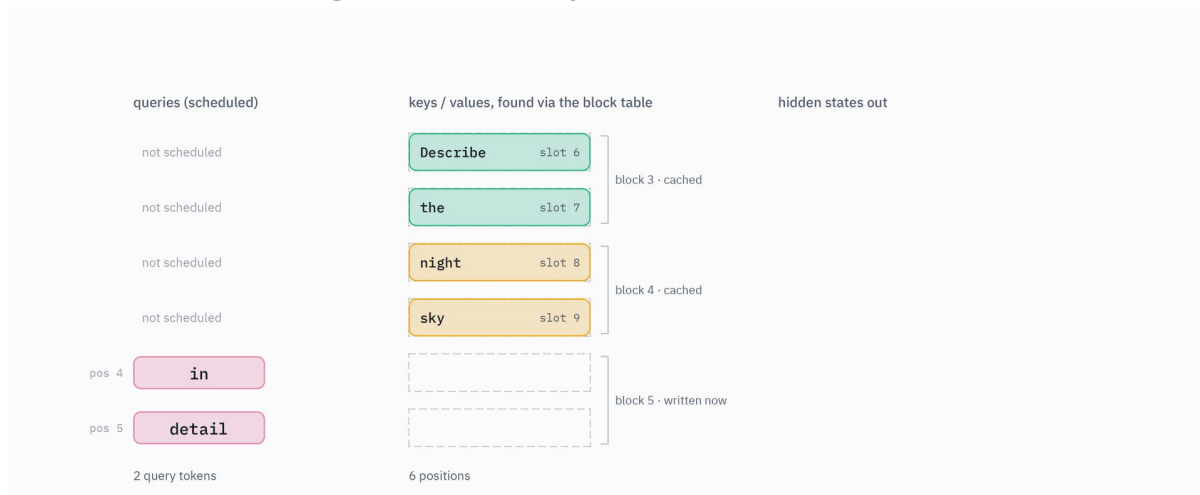
arXiv:2610.09307

vLLM Omni + Qwen3-Omni



[vLLM Omni] What's the Problem?

For autoregressive decoding, we mostly need the last row for sampling.



But if we need hidden states, not having the tokens scheduled means we don't have an easy way to get the hidden states corresponding to the cache hits.

[vLLM Omni] Core Idea: Reuse vLLM's Block Pool

If tensors are 1:1 with input tokens and need to be reused, we can hold our own cache, and use the indices from the block table

Omni cache: block hash $\rightarrow$ hidden states

[vLLM Omni] Core Idea: Reuse vLLM's Block Pool

To implement it, we can use the indexing into the block table to reindex into our own 3D cache!

the KV cache: blocks $\times$ 2 slots



Advantages:

- 3D cache can stay on CPU
- Block reuse is handled automatically; the indices will just clobber the data in our tensor cache

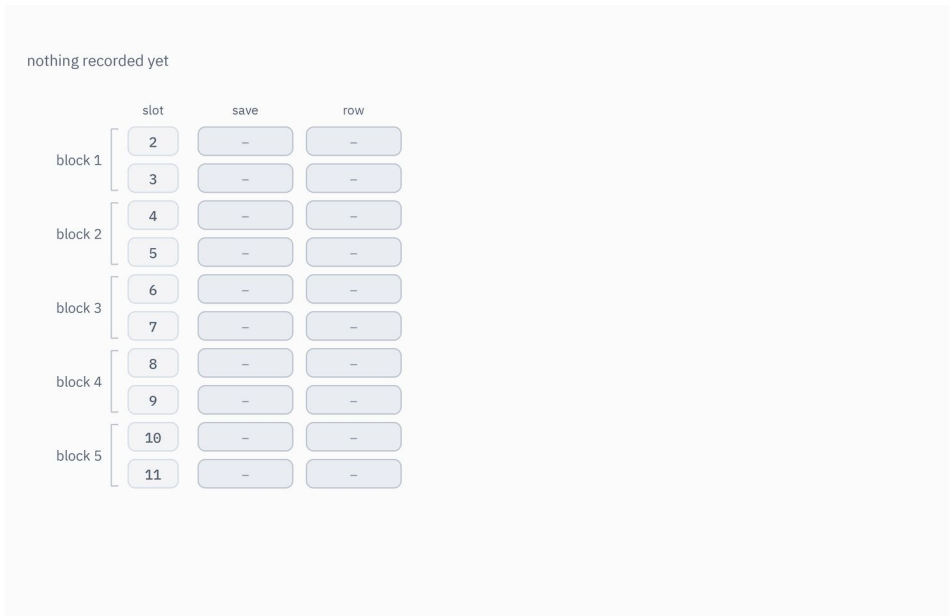
Disadvantages:

- Lots of copying to the separate tensor cache

[vLLM Omni] Minimizing Copies

Instead of copying to a separate 3D tensor, we can instead create a new 'save' on each engine step, which holds the tensors unique to that engine step.

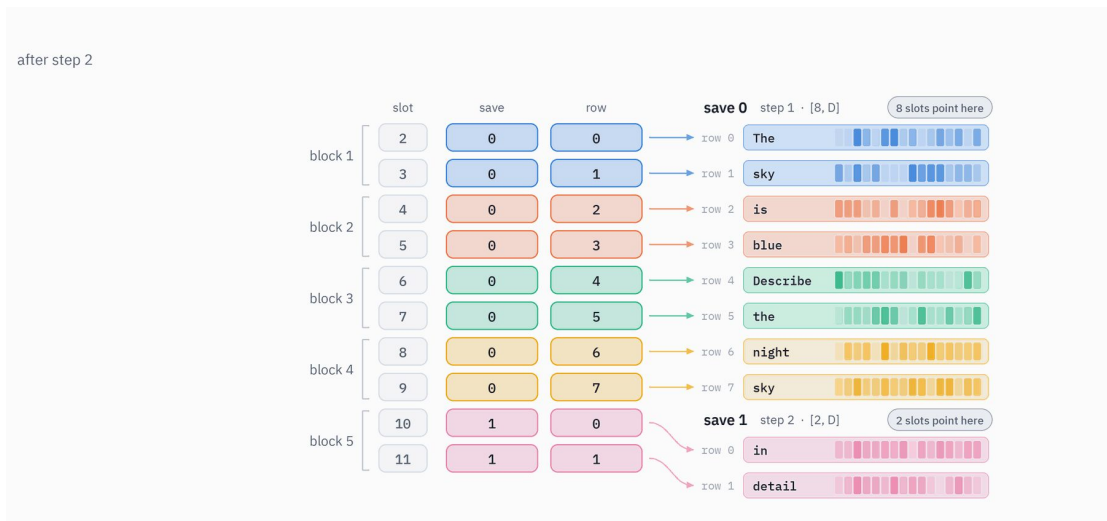
This lets us avoid turn this into a reference counting problem, but we need to handle updates and deletions.



[vLLM Omni] Handling Updates / Deletes

When a block is reused,
we can update the
reference counts in
existing saves.

If a save no longer has any
references, it can be
purged. Otherwise we can
try other approaches, e.g.,
compaction.

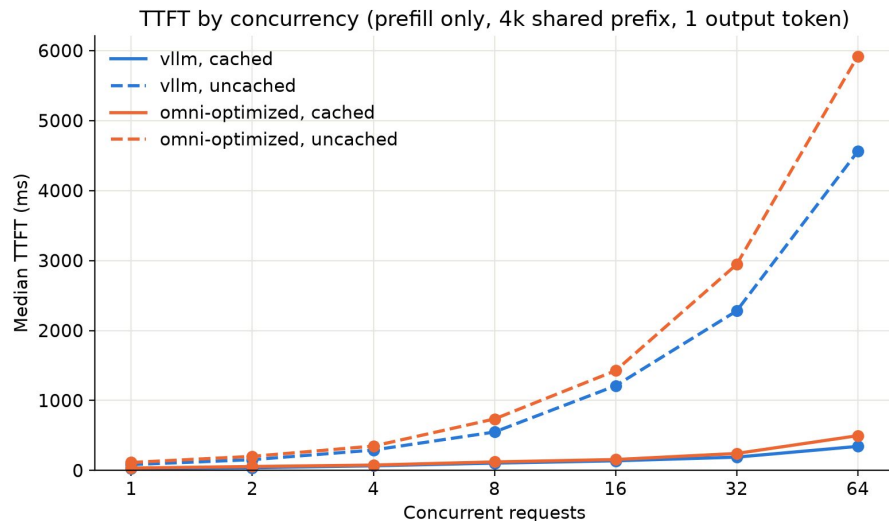


Performance: Prefix Caching in vLLM vs vLLM Omni

We can evaluate the performance of vLLM Omni's implementation against vLLM by running an autoregressive decoder in isolation!

- e.g., Qwen3-Omni-30B-A3B-Captioner (thinker only Qwen3Omni variant)

This lets us ensure the concurrency scaling is comparable for prefill-heavy workloads, while producing the tensors we'd need to forward to the next stage

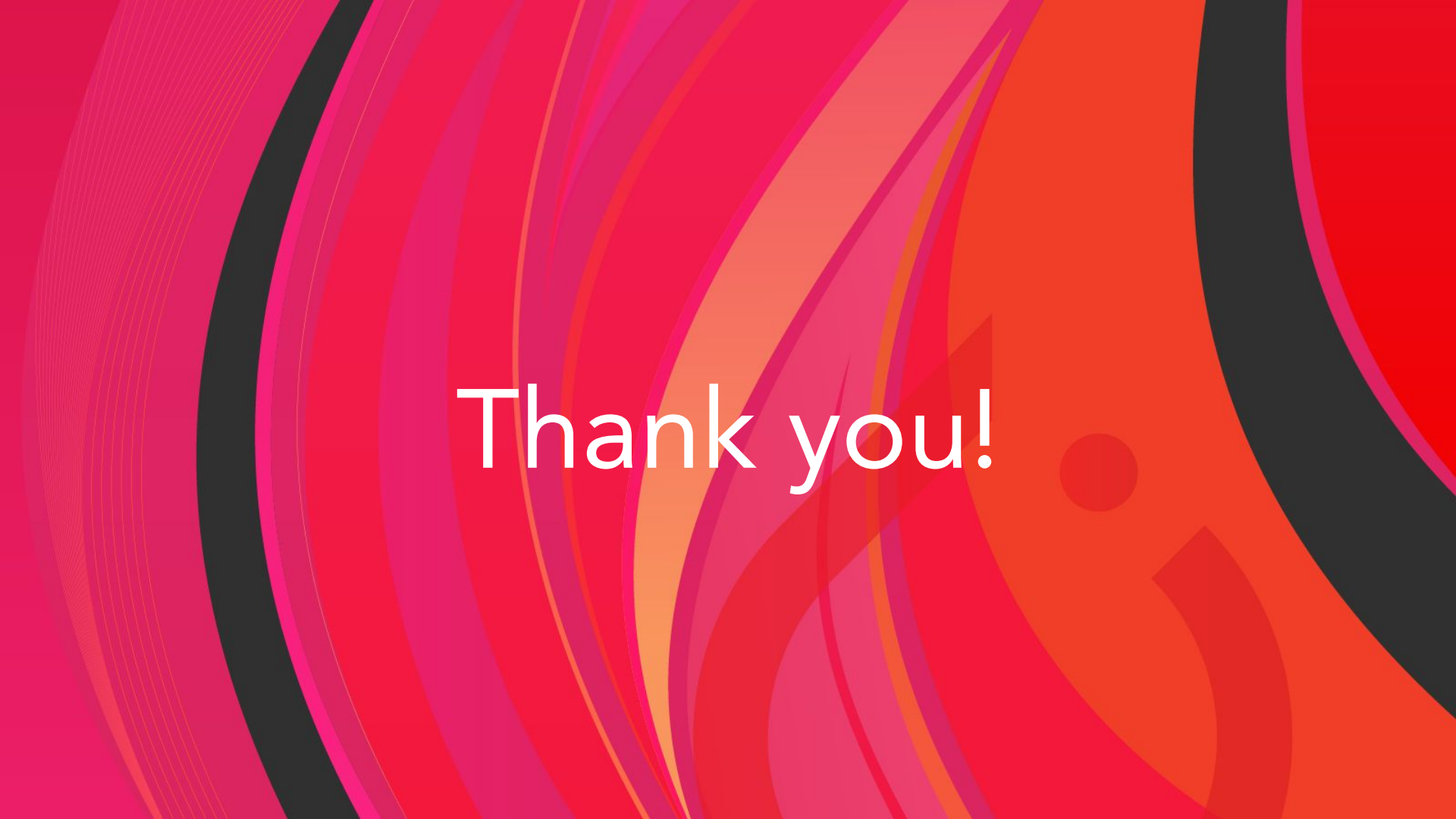


Limitations & Future Work

- Currently only autoregressive stages with a single block table support prefix caching in vLLM Omni
- Work to support prefix caching for non-autoregressive stages and architectures with multiple block tables, e.g., hybrid models, will follow in the future

Refs

- vLLM's [docs](#) on prefix caching
- vLLM Omni's [docs](#) on prefix caching
 - Original implementation of tensor prefix caching in vLLM Omni: [vllm-omni/2164](#)
 - Optimizing with reference counting approach: [vllm-omni/9039](#)



Thank you!

