vAttention: Dynamic Memory Management for Serving LLMs without PagedAttention

Alex D’Souza

Paper background

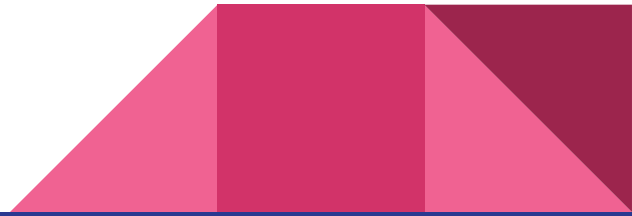
- The paper was published in March 30th 2025 at ACM
- By Microsoft Research in Bengaluru India.

Background

- LLM serving bottlenecks - Modern LLMs are enormous with billions of parameters, serving them efficiently is expensive, each token generation requires computing attention across all prior tokens. For every token, the model needs to store the key-value cache, the memory of everything it has seen before.
- KV Cache - During inference, the transformer’s attention layers store: keys and values for each token and head. This cache is reused for every token generation. If each request gets its own contiguous region of GPU memory, eventually memory becomes fragmented.

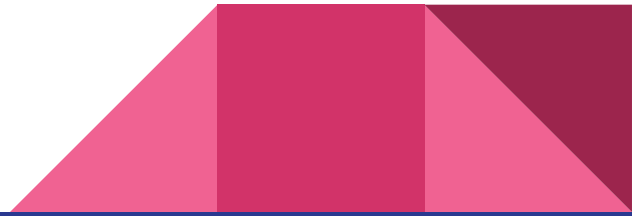
Background cont

- Paged Attention eliminates fragmentation and allows dynamic allocation/deallocation. Enables large batch sizes which gives higher throughput.
- But it **breaks contiguous vm layouts**, kernels like FlashAttention must be rewritten to follow page tables.
- Adds performance overhead due to non coalesced memory access.
- Complex to maintain and debug



Enter vAttention

- vAttention mitigates fragmentation in physical memory while retaining vm contiguity of the KV cache.
- Achieved by decoupling the allocation of virtual and physical memory using CUDA virtual memory management APIs (VMM).
- vAttention is simpler, portable, and performant alternative to PagedAttention: it supports various attention kernels out of the box and improves LLM serving throughput by up to **1.23x** in comparison.

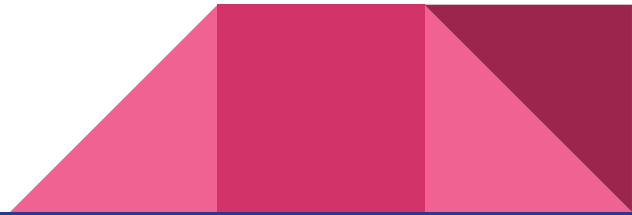


Introduction

- Efficiently allocating GPU memory for KV cache is challenging. First, the per-request KV cache grows slowly (one token per iteration), and second, a request's decode length (or its total KV cache size) is not known ahead of time.
- Some previous systems like Orca and FasterTransformer allocate memory for each request based on maximum context length supported by the model, but average number of decode tokens were far less in practice.

Differences between vAttention and PagedAttention

- Paged attention relies on reservation based memory allocation, where physical memory is allocated even if the corresponding virtual memory is not accessed.
- The paper argues that separating the allocation of virtual memory and physical memory allows for more effective KV cache memory management.
- vAttention decouples the allocation of virtual and physical memory using CUDA virtual memory management (VMM).

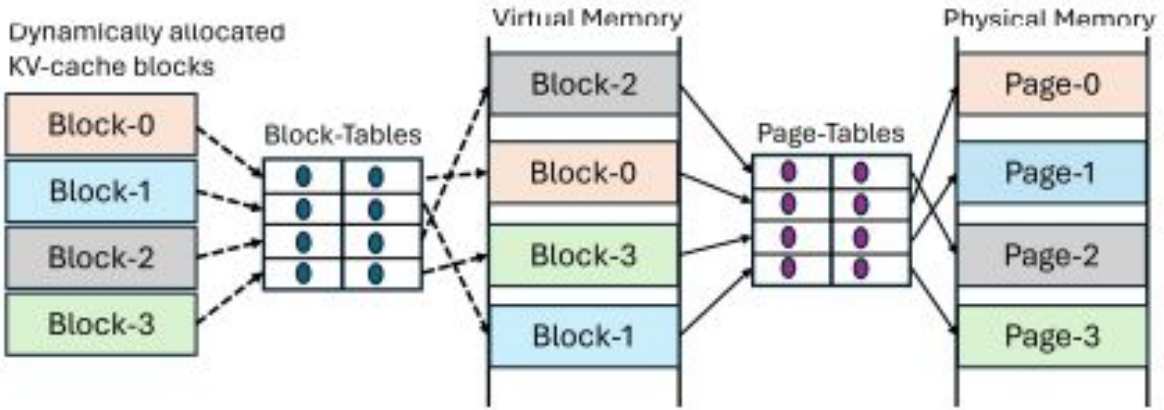


CUDA VMM Challenges

- CUDA VMM support for KV cache management poses two key efficiency challenges.
- Memory allocation using CUDA VMM API incurs high latency because each allocation involves a round trip to the OS kernel. They overcome this with several LLM specific optimizations such as overlapping memory allocation with compute, opportunistically allocating pages ahead of time, and deferring memory reclamation.
- Second, CUDA supports memory allocation only at granularity of large pages in multiples of 2MB. They fixed this by adding support for smaller

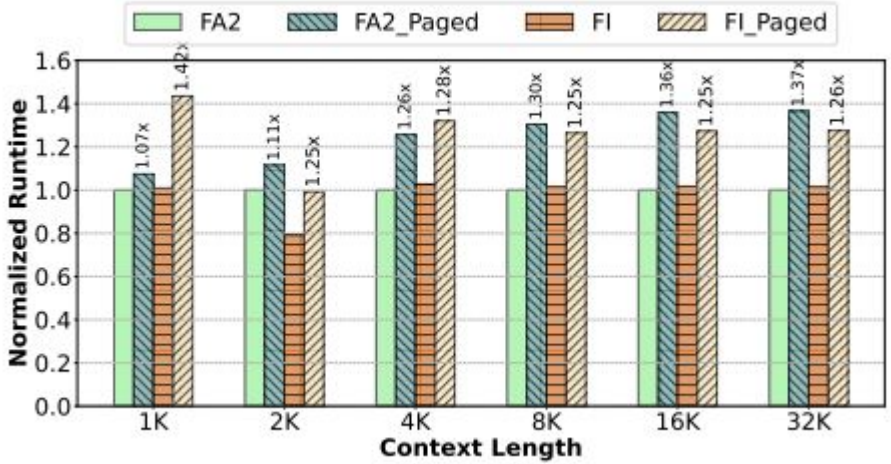
  64KB pages.

Redundancy in PagedAttention



This is what a mapping looks like for PagedAttention, to compute attention that would require these blocks, the approach effectively duplicates what the OS already does for virtual to physical address translation.

Performance Overhead in PagedAttention

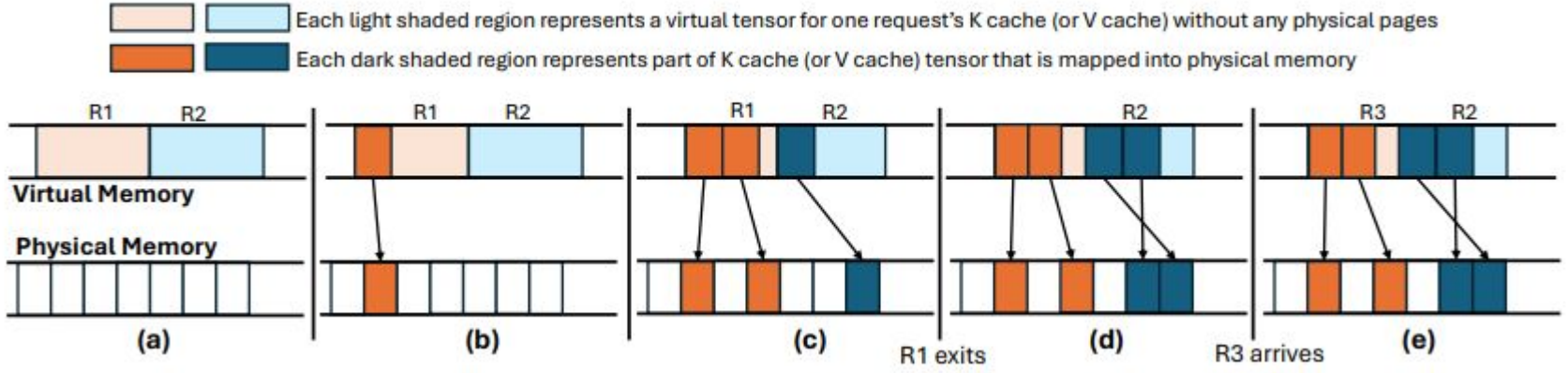


The figure shows how incorporating PagedAttention has significant Overhead in comparison to non paged. FA = FlashAttention-2 FI = FlashInfer both are gpu kernels

vAttention VM allocation

- Since virtual memory is abundant, they pre-allocate it in sizes that are large enough to hold the KV cache of the maximum batch size (configurable) that needs to be supported.
- A serving framework maintains separate K and V tensors for each layer of the model. Therefore, they reserve $2\times N$ buffers on a worker where N is the number of layers managed by that worker
- the maximum size of a buffer is BS = B × S where
- B is maximum batch size
- S = L × H × D × P
- L = Maximum context length supported by the model
- H = Number of KV heads on a particular worker
- D = Dimension of each attention head
- P = Number of bytes needed to store one element

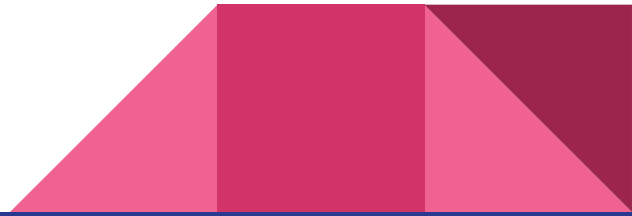
vAttention dynamic memory management deferring



a. Shows a virtual tensor for a batch of two requests and no physical memory allocation yet.
b. R1 is allocated one physical page
c. R1 is allocated two pages and R2 is allocated one page
d. R1 has completed by vAttention does not reclaim its memory(deferred reclamation)
e. R3 arrives and vAttention assigns R1's tensor to it which is already backed by physical memory

Request Termination

- A request terminates when it reaches user specified or maximum context length or when the model produces a special end of sequence token
- vAttention may unmap the physical pages of a completed request or defer them to be freed later.
- They trigger memory reclamation only when the number of page-groups cached in vAttention falls below a certain threshold (e.g., less than 10% of GPU memory). They delegate both deferred reclamation and eager allocation to the background thread that the step API spawns.



Challenges Faced - Continuous Batching

- Problem - In LLM serving, multiple requests are processed together in a batch for GPU efficiency. Requests finish at different times → holes appear in the KV cache, wasting memory.
- Solution - Introduces a mapping table (cache_batch_idx),
- Maps logical Q batch indices → actual KV cache slots.
- Supports arbitrary reordering of requests.
- Enables dynamic reuse of holes left by finished requests.

Challenges Faced - VMM API latency

- Problem - Using CUDA VMM to decouple virtual memory (VM) and physical memory (PM) introduces high latency due to OS/kernel interactions when mapping/unmapping pages.
- Overlapping Memory Allocation with Compute - Memory demand per decode iteration is predictable (1 token per request) → allocate pages while computing the current token.
- Deferred reclamation - Reuse memory from completed requests by remapping their pages to new requests.
- Eager allocation - Proactively allocate a small number of

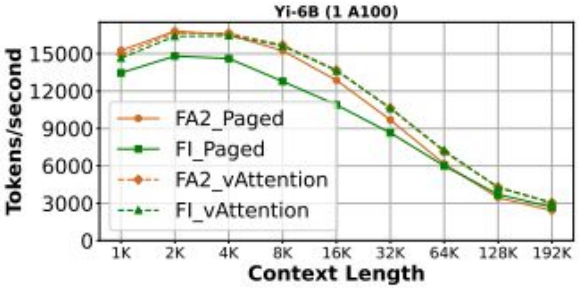
  page-groups ahead of time to reduce allocation stalls.

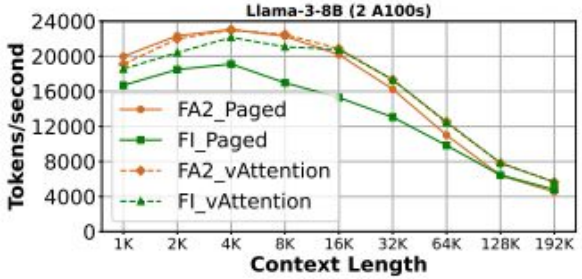
Challenges Faced - Granularity Size

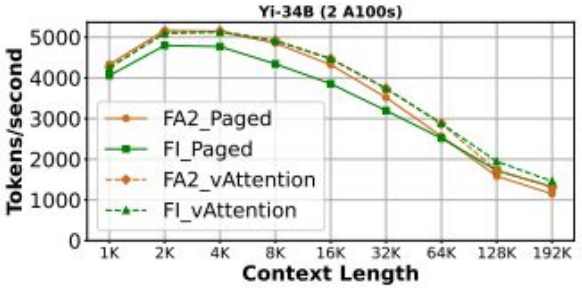
- Problem - CUDA VMM initially only supports large pages (2MB) → causes internal fragmentation for small requests.

- Solution - Implement custom APIs in the open-source NVIDIA drivers to support smaller page-groups (64KB, 128KB, 256KB).

Evaluations - Prefill throughput



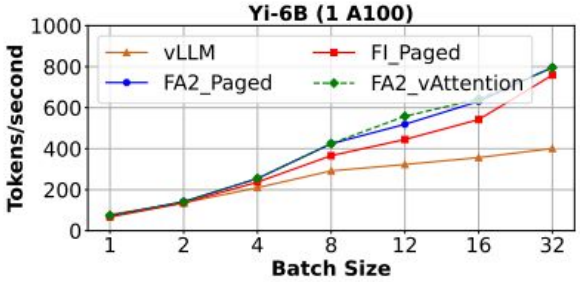


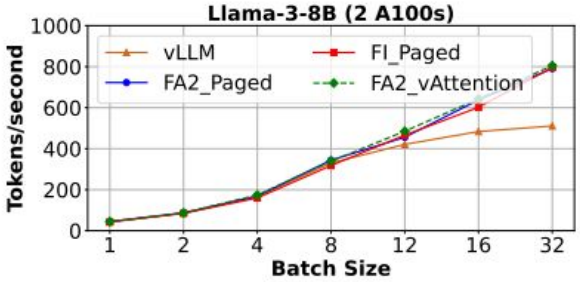


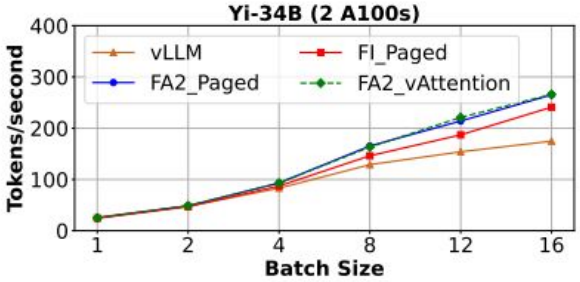
Prefill phase is when the model processes all input tokens to fill the KV cache, or the speed of processing the initial input prompt.

Prefill throughput shows that the vAttention backed systems outperform the paged counter parts of FlashAttention-2 and FlashInfer.

Evaluations - Decode throughput



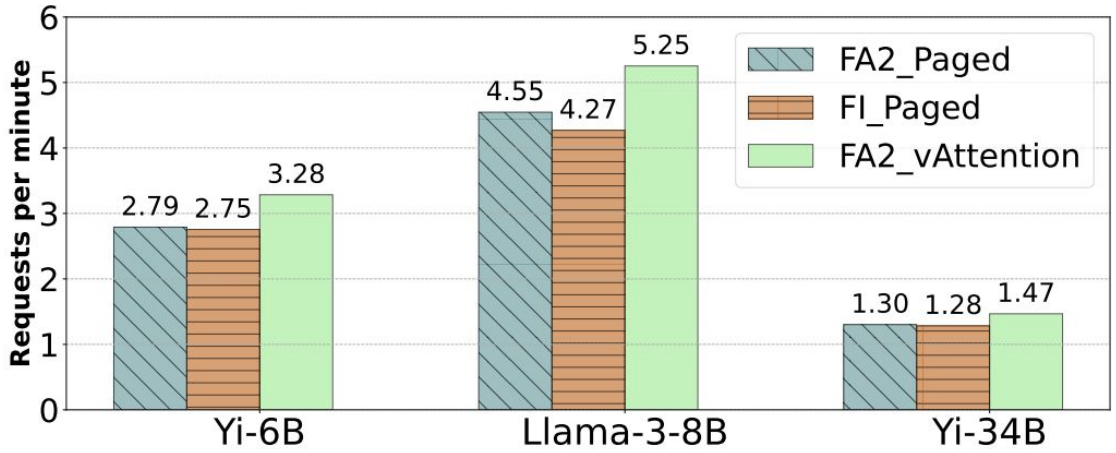




Decode is the phase where the model generates the rest of the output one by one.

FA2_vAttention is on par with FA2_paged which is the best among all PagedAttention based alternatives while out performing FI_Paged and vLLM.

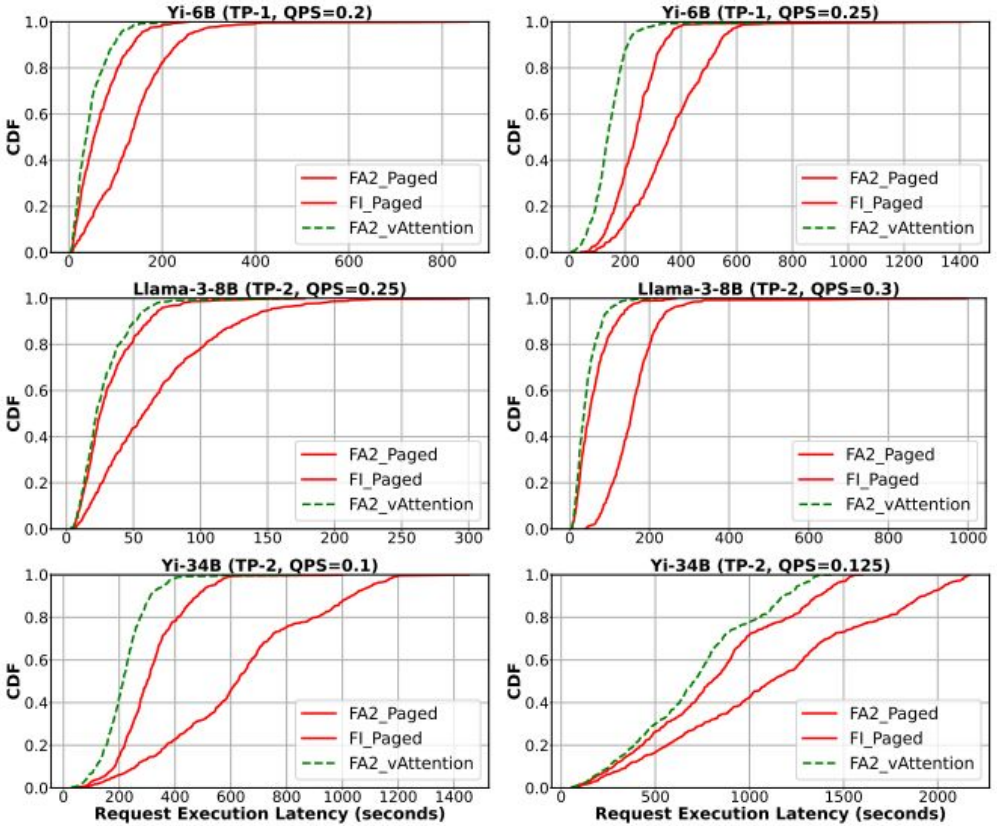
Evaluations - Offline Inference Throughput



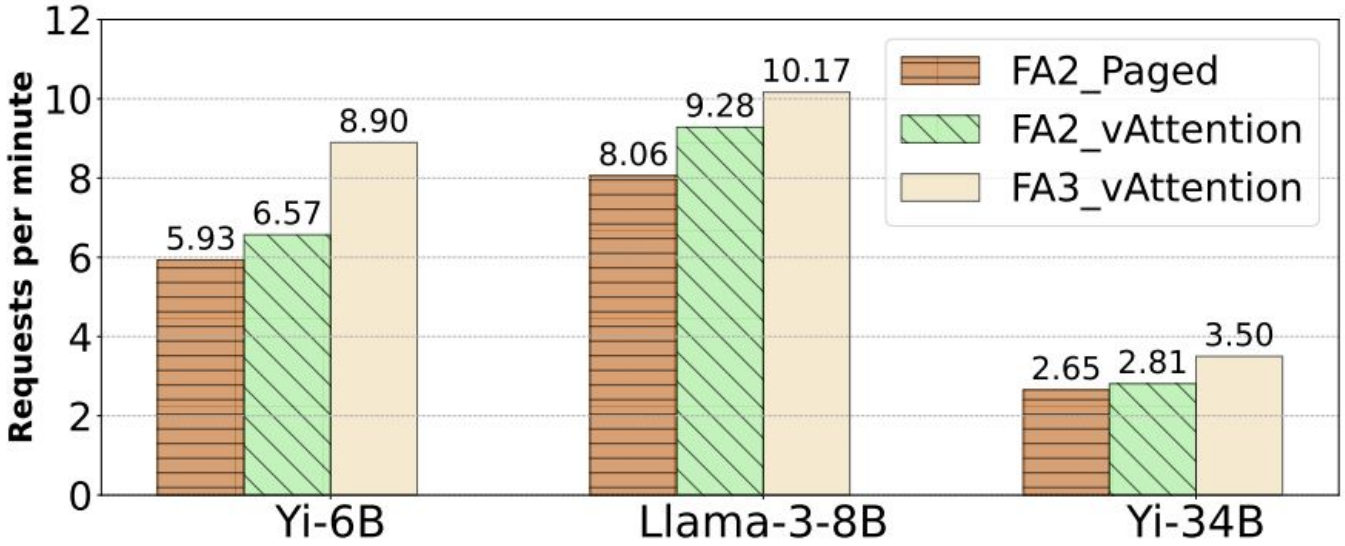
Number of requests completed in comparison, FA2_attention was higher in all three models.

Evaluations - Online inference CDF

- Online inference shows a similar pattern of FA2_vAttention outperforming the others.
- The graphs show in each different hyperparameter variation with tensor parallelism and queries per second FA2_vAttention delivers the lowest and most predictable latency.



Evaluations - vAttention demonstrates portability

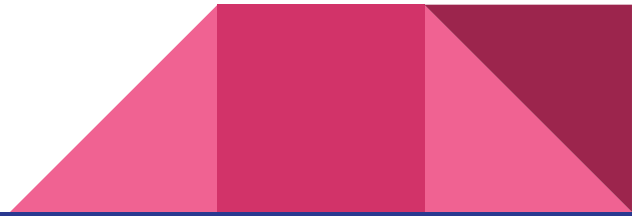


- FA3 is a recently released kernel and optimized for the NVIDIA Hopper architecture and did not support PagedAttention when released.
- vAttention not only enables dynamic memory allocation with FA3, it also requires

  no code changes to deploy FA3. demonstrating portability.

Key Takeaway

- **PagedAttention**: Dynamically allocates smaller VM and PM pages, which allows flexibility but can cause fragmentation and scattered memory access.
- **vAttention**: Allocates larger VM pages while still dynamically allocating PM pages, decoupling VM and PM allocation. This reduces fragmentation and allows coalesced memory access, improving GPU efficiency.



Thank You!