RetroInfer: A Vector-Storage Approach for Scalable Long-Context LLM Inference

Yaoqi Chen, Jinkai Zhang, Baotong Lu , Qianxi Zhang, Chengruidong Zhang, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Jing Liu, Bailu Ding, Xiao Yan, Jiawei Jiang, Chen Chen, Mingxing Zhang, Yuqing Yang, Fan Yang, Mao Yang

Presented by Xiaoqi Li

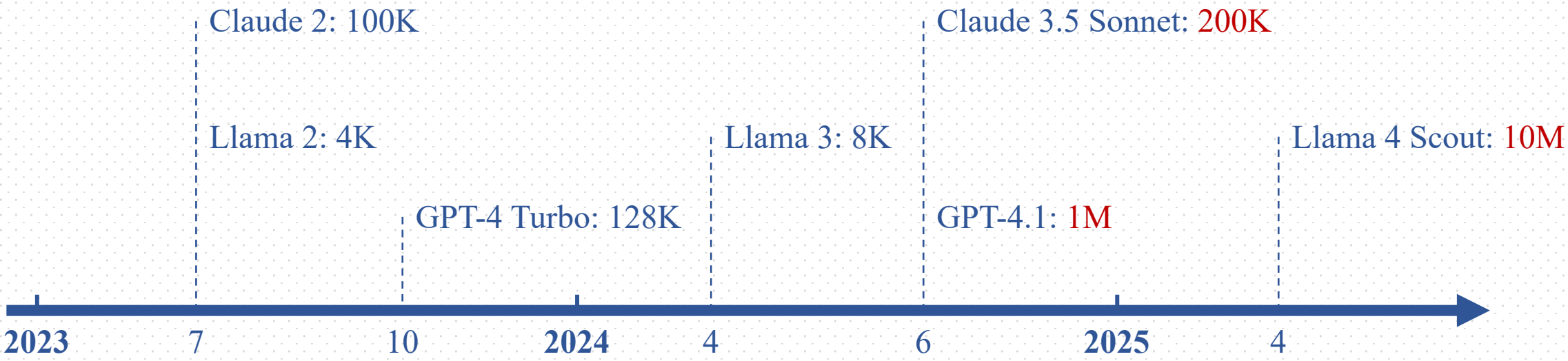
2025-06-03

Contexts

- ❑ **Background**
- ❑ Motivation
- ❑ RetroInfer
- ❑ Evaluations

Longer and Longer Context Window in LLMs

- **Longer context window**
  - **Improved understanding and factual accuracy**
  - **Enhanced summarization and information retrieval**
  - **Advanced in-context learning**



Expensive Long Context Serving

- $O(n^2)$ Inference latency to the context size n
- Most of the time is spent on attention operations

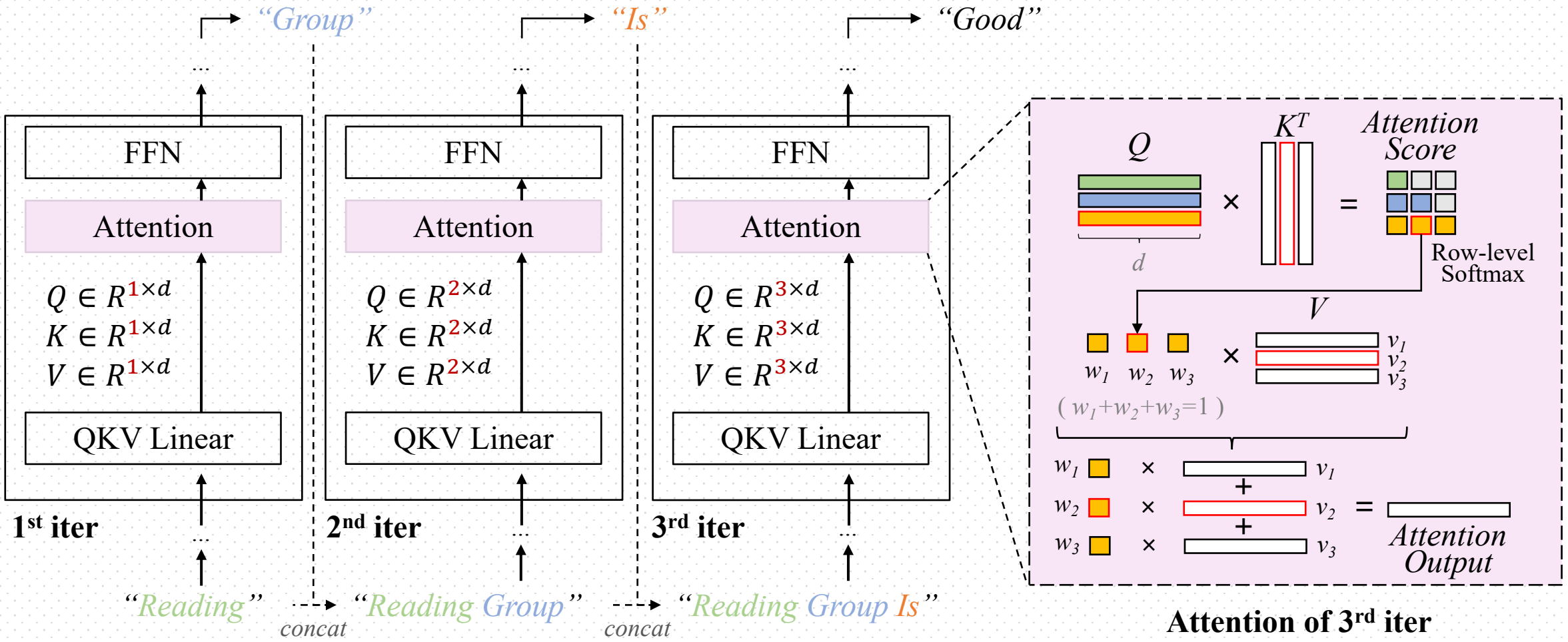
Prompt Length	128K	256K	512K	1M
Total Latency (s)	32.8	111	465	1,765
FFN (s)	7.6	15	31	70
Attention (s)	25.2	96	434	1,695

1695 / 1765 ≈ 96%

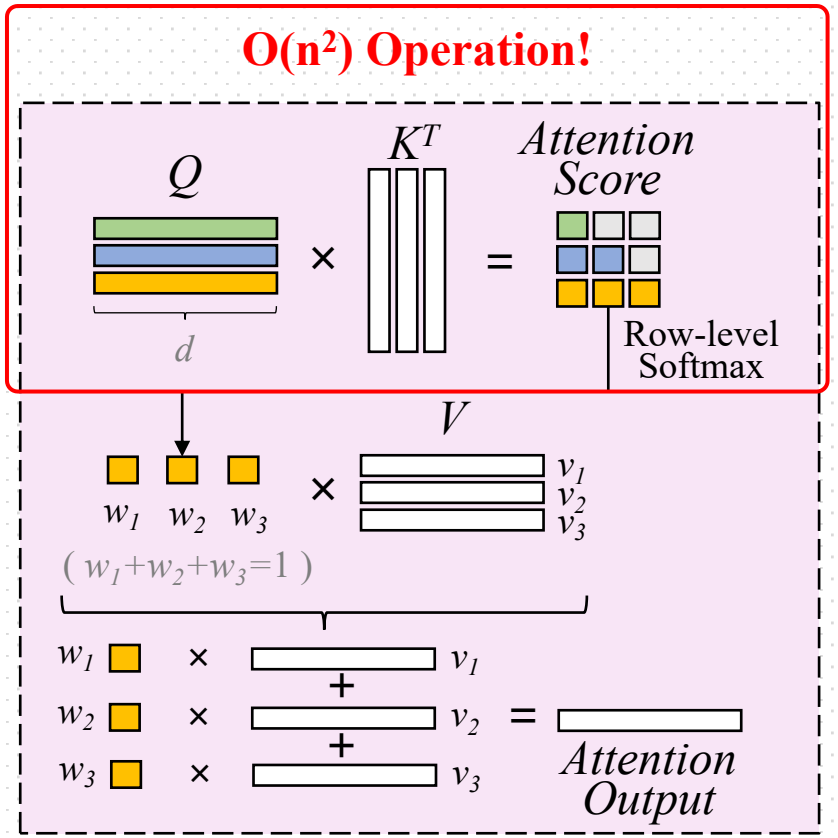
Decoding latency of Llama-3-8B across different context lengths on one A100 GPU[1]

[1] RetrievalAttention: ACCELERATING LONG-CONTEXT LLM INFERENCE VIA VECTOR RETRIEVAL

Vanilla Attention: $O(n^2)$ Complexity



Vanilla Attention: $O(n^2)$ Complexity

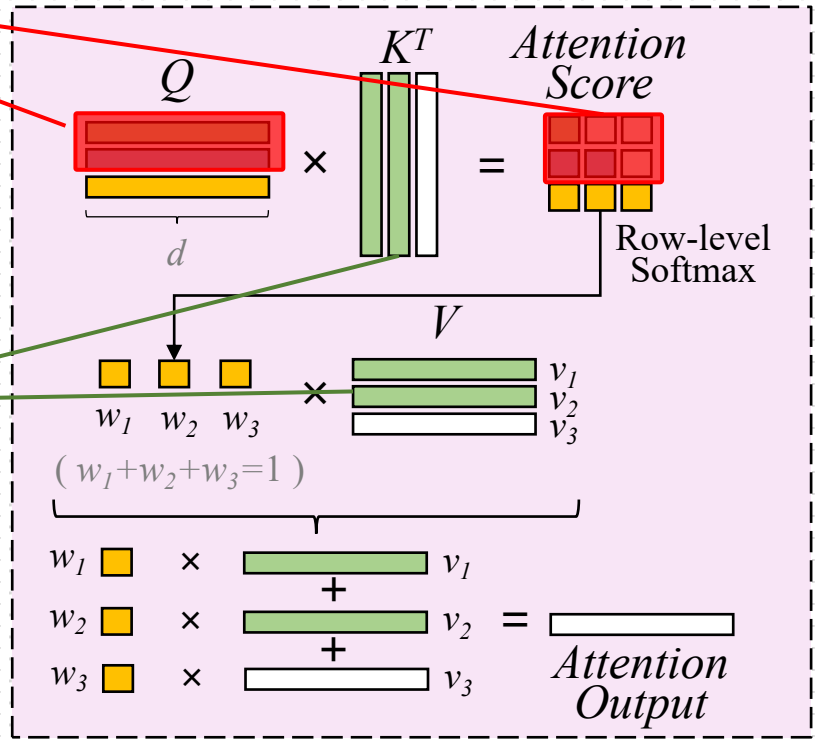


Attention Computation

Optimize Vanilla Attention

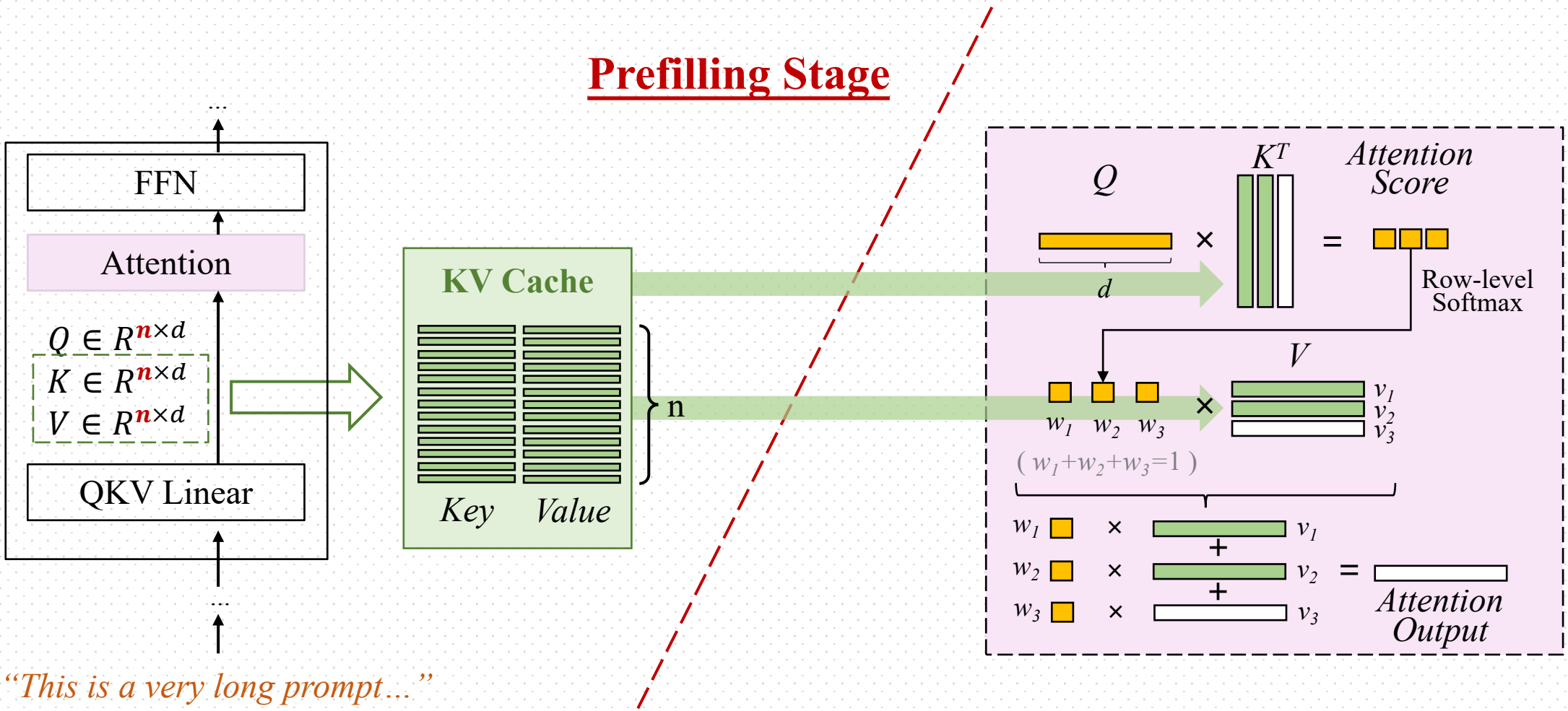
1. Remove Unneeded Operation

2. Cache KV



Attention Computation

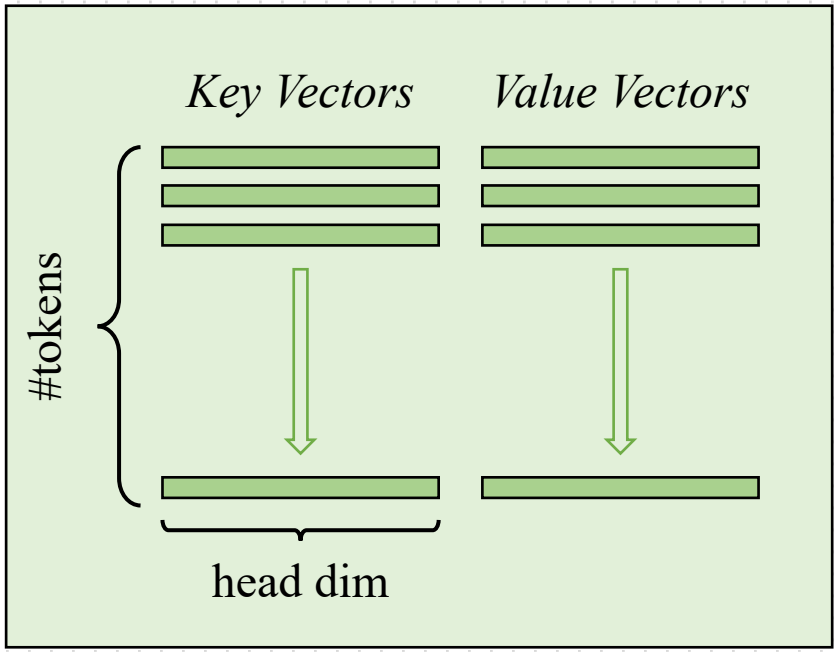
Using KV Cache to Accelerate Decoding



But KV Cache Is Not Enough!

❑KV cache grows linearly with context window length!

LLM Model: Llama3-8B			
#Layers	Hidden Dim	GQA Compress Ratio	Data Type
32	4096	4	BF16



KV Cache

KV Cache size:

- A single token: $2 * 32 * 4096/4 * 2B = 128KB$
- 1M tokens: $128KB * 1M = 128GB$

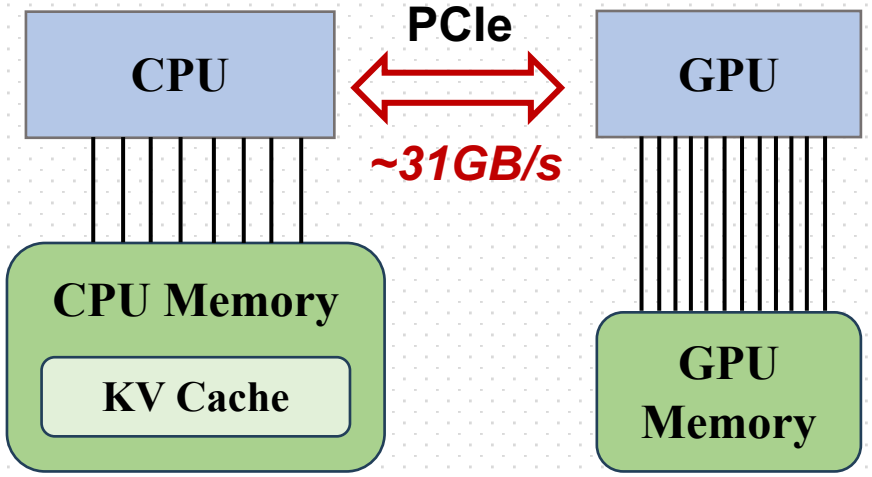
GPU memory capacity:

- RTX5090: 32GB
- A100: 40/80 GB

KV Cache Optimizations

❑Opt. 1: Offload

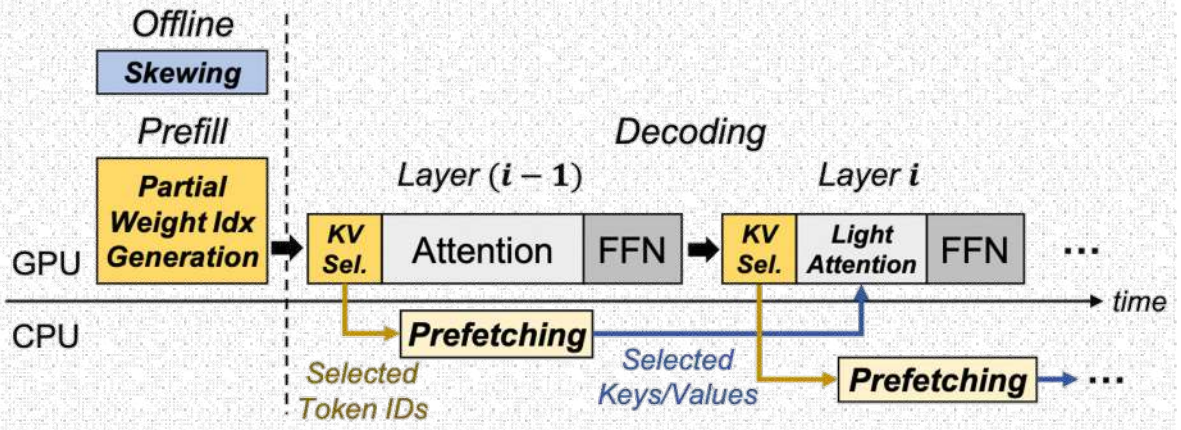
- ❖Offload KV cache to CPU DRAM
- ❖PCIe bandwidth is limited



Cheap & Large Expensive & Small

❑InfiniGen (OSDI 24)

- ❖Prefetching next layer's KV vectors

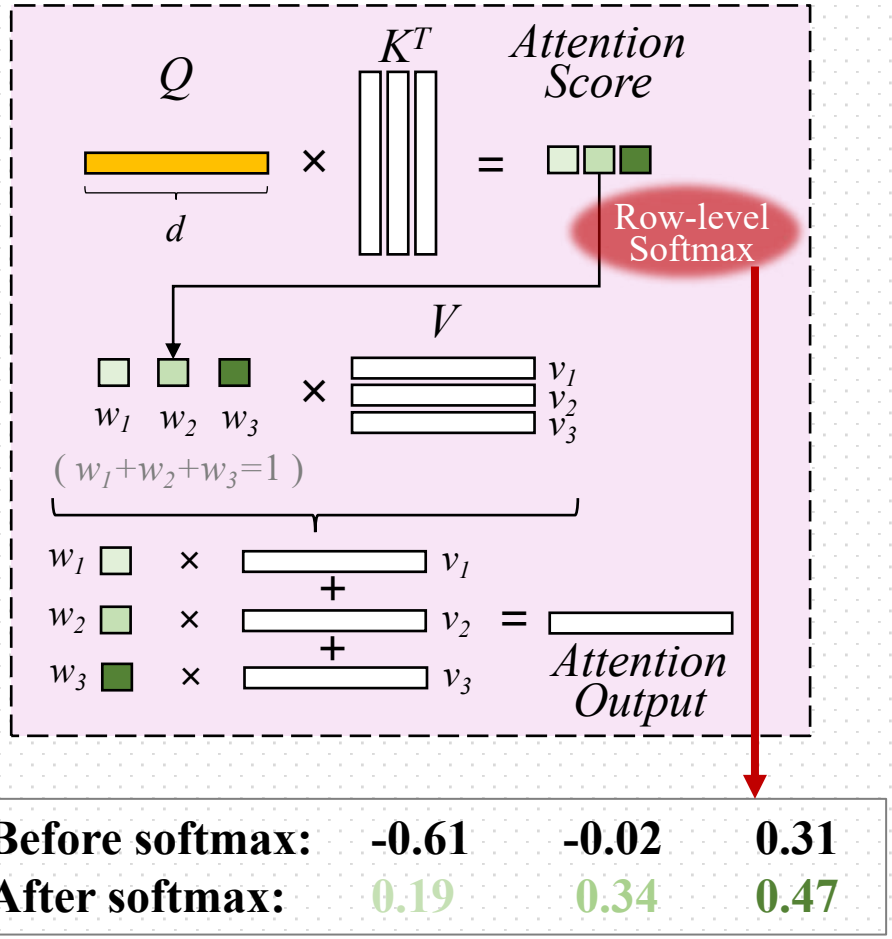


InfiniGen leverages prefetching to overlap data transfer

KV Cache Optimizations

❑Opt. 2: TopK Attention

- ❖Attention is sparse[1].
- ❖In most transformer layers, n% top KV can carry enough information to maintain model accuracy.
- ❖Only use top n% token with largest attention weights



[1] Memory-efficient Transformers via Top-k Attention

KV Cache Optimizations

❑Opt. 2: TopK Attention

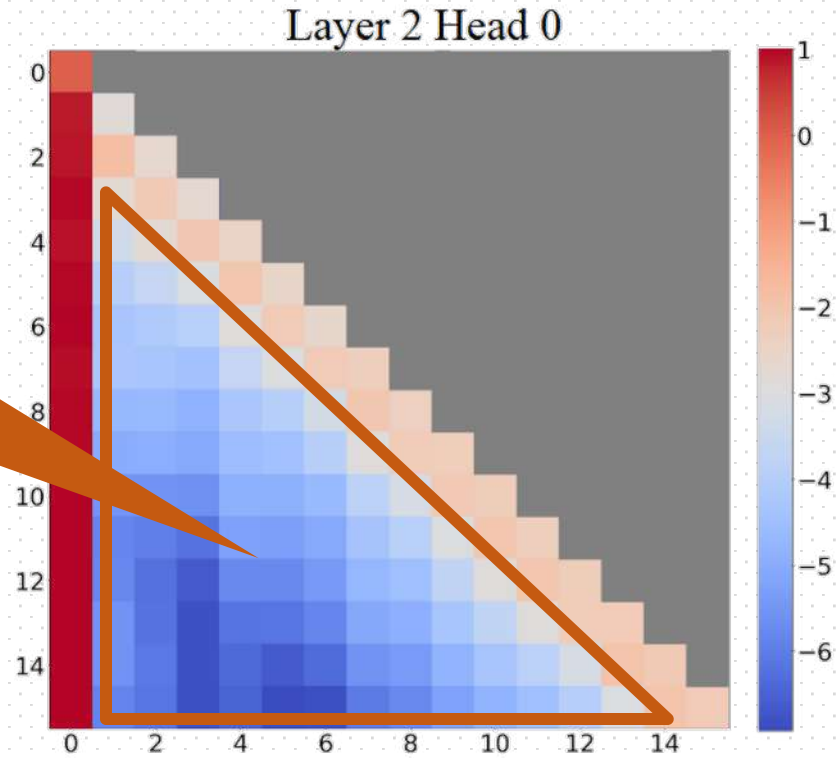
❖Attenti

❖In mos
top KV
informa
accurac

❖Only use top n% token with largest attention weights

Drop unimportant keys (StreamingLLM, H2O, …)
- Easy to implement
- Memory saving

Sounds good, but…



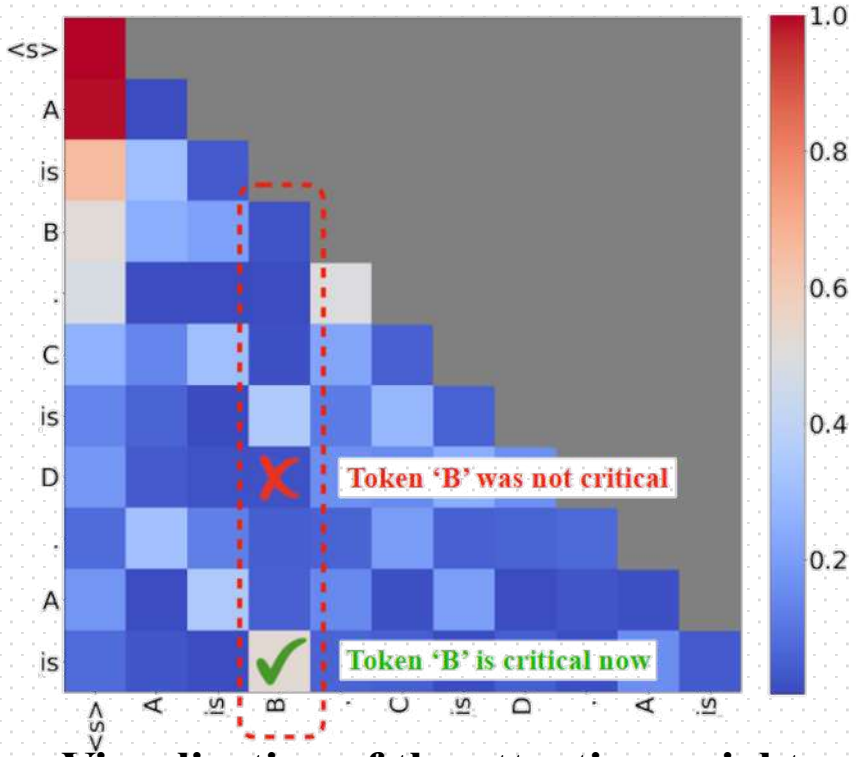
Visualization of the attention weights in Llama-2-7B over sentence with a length of 16.[2]

[1] Memory-efficient Transformers via Top-k Attention
[2] StreamingLLM: Efficient Streaming Language Models with Attention Sinks

KV Cache Optimizations

❑Opt. 2: TopK Attention

- ❖Attention is sparse[1].
- ❖In most transformer layers, n% top KV can carry enough information to maintain model accuracy.
- ❖Only use top n% token with largest attention weights



Visualization of the attention weights of prompt "A is B. C is D. A is".[2]

Critical tokens depend on the query

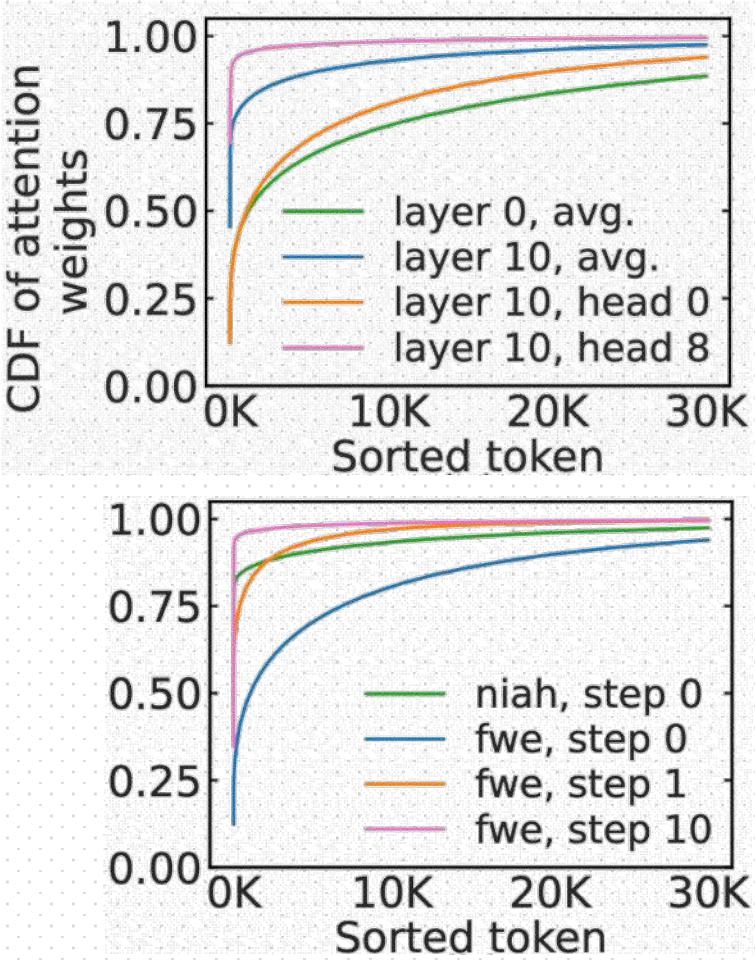
[1] Memory-efficient Transformers via Top-k Attention
[2] QUEST: query-aware sparsity for efficient long-context LLM inference

KV Cache Optimizations

❑Opt. 2: TopK Attention

- ❖Attention is sparse[1].
- ❖In most transformer layers, n% top KV can carry enough information to maintain model accuracy.
- ❖Only use top n% token with largest attention weights

The sparsity exhibits high variability



Variety of Attention Sparsity

[1] Memory-efficient Transformers via Top-k Attention

KV Cache Optimizations

❑Opt. 2: TopK Attention

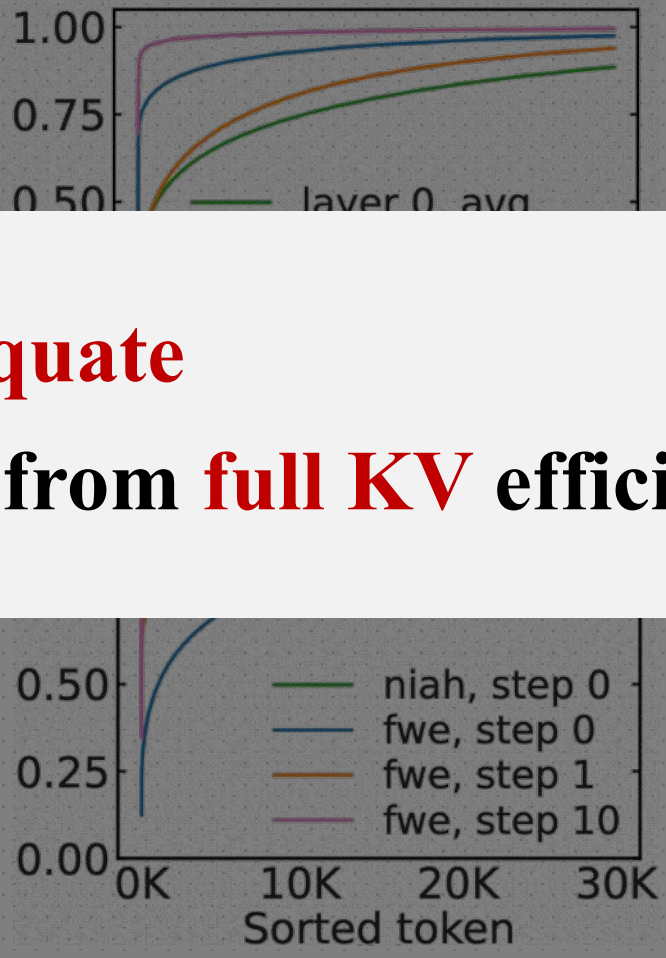
❖Attention is sparse[1].

Static methods is inadequate

We need some strategy to find topk token from full KV efficiently

largest attention weights

The sparsity exhibits high variability



Variety of Attention Sparsity

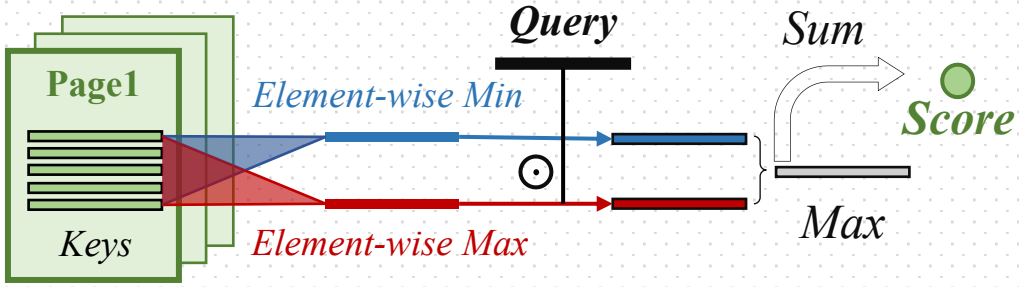
[1] Memory-efficient Transformers via Top-k Attention

KV Cache Optimizations

❑Opt. 2: TopK Attention

❑Quest (ICLR 24)

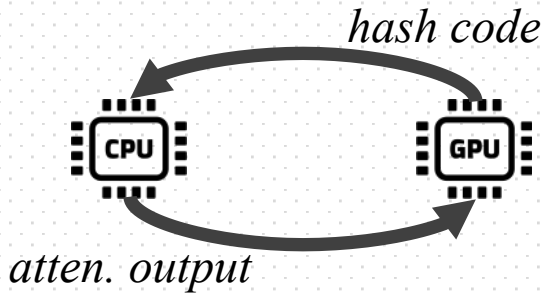
- ❖KV vectors stored in pages, each with a pair of min/max key vector



Quest use min/max key to find topk pages

❑MagicPIG (ICLR 25)

- ❖Use Locality Sensitive Hashing (LSH) to sample important Key/Value



MagicPIG leverages LSH to do sample

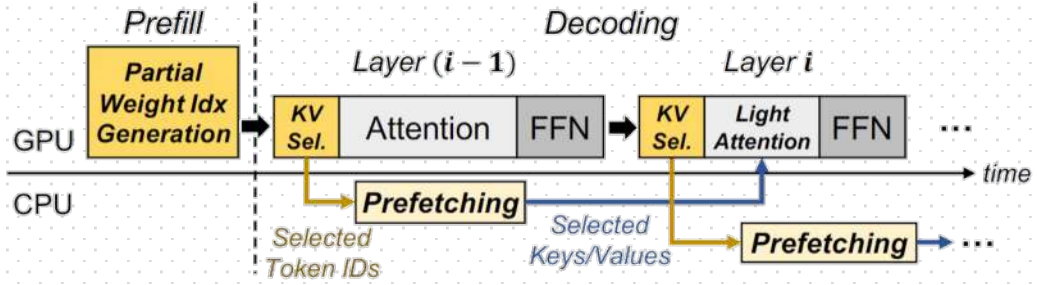
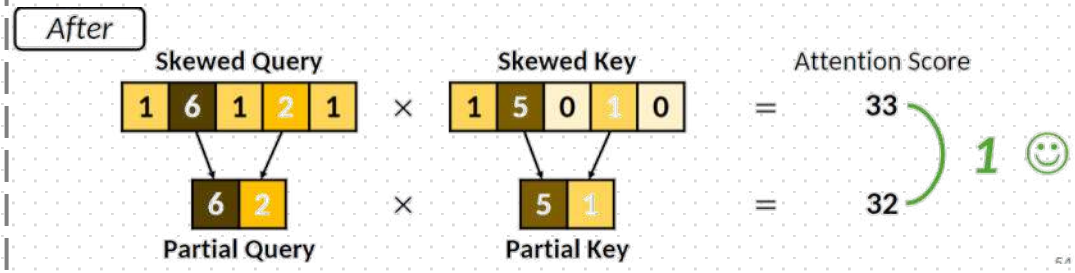
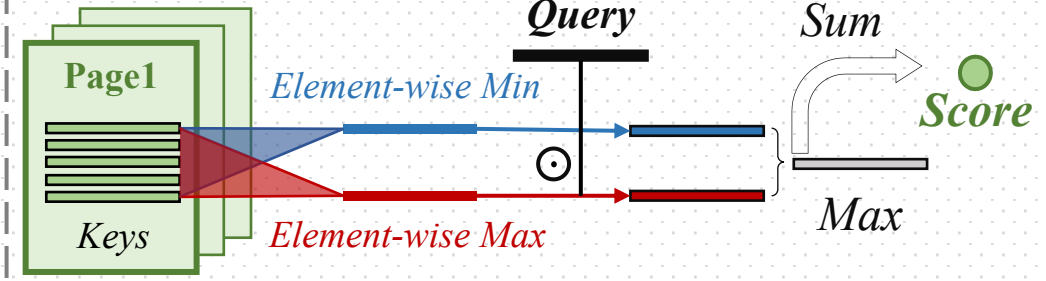
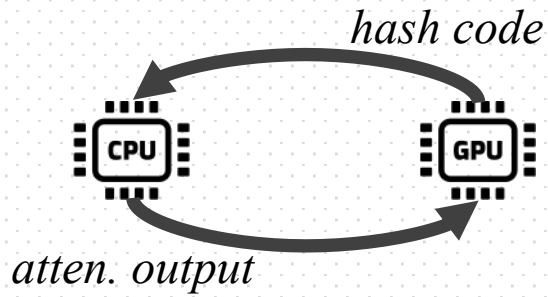
Existing Long-Context Inference

- Existing approaches face challenges in accuracy and efficiency

- Drop-out based *(e.g. StreamingLLM)*
  - Assuming static token importance ➔ Accuracy loss

- Other Leading Methods
  - See next page…

Existing Long-Context Inference: SOTAs

	Data transfer	*TopK Selection*
InfiniGen OSDI 24	 Prefetching, but frequent data transfer	 Use reduced dimension to find topk, still costly
Quest ICLR 24	✗	 Use page level reduced keys, accuracy loss
MagicPIG ICLR 25	Attention is done on CPU 	Use Locality Sensitive Hashing (LSH) to sample important Key/Value, accuracy loss

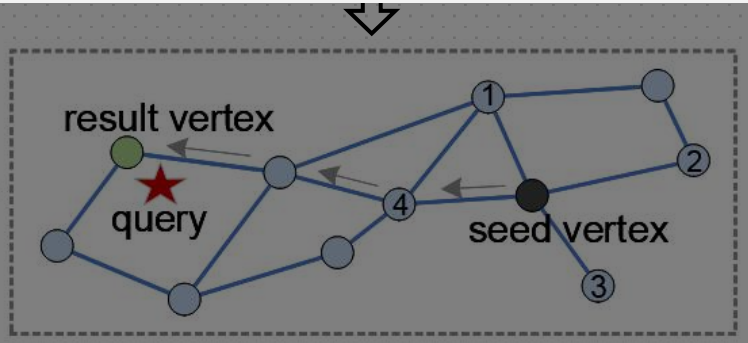
Contexts

- Background
- **Motivation**
- RetroInfer
- Evaluations

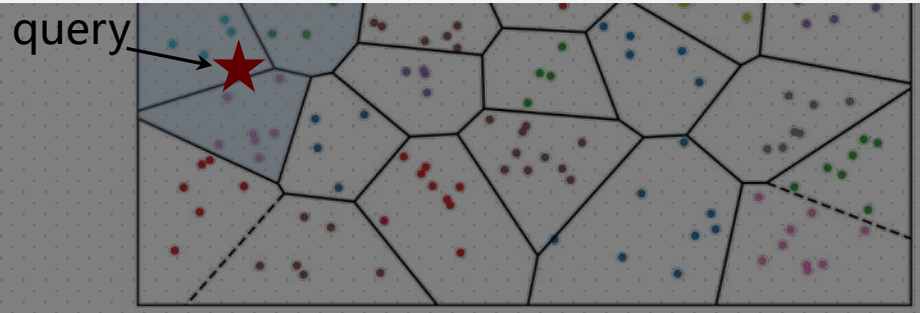
Top-K Selection Is Natively Supported in ANNS

- ANNS (Approximate Nearest Neighbor Search)
  - Say, <5% search yields >90% recall

Can We Leverage ANNS?



Graph-based



Cluster-based

Challenges: When ANNS Meets KV Cache

❑Index traversal and construction costs

❖Quest latency [1] : 3.61 s (prefilling) and 14.86 ms (decoding) on Llama-2-7B using A100 GPU, 32k context

Algo.	ANNS Test Results		KV Cache Estimations	
	Index construction	Avg. latency	Index construction (in prefilling stage)	Latency per token (in decoding stage)
IVF FLAT	562 ms	2.58 us	575.5 s	2.64 ms
HNSW	300 ms	1.02 us	307.2 s	1.04 ms

ANNS test achieving 0.9 recall
on SIFT subset (32k samples, 128-dim, float16)
CPU: Intel(R) Xeon(R) 6982P-C

[1] LServe: Efficient Long-sequence LLM Serving with Unified Sparse Attention

Challenges: When ANNS Meets KV Cache

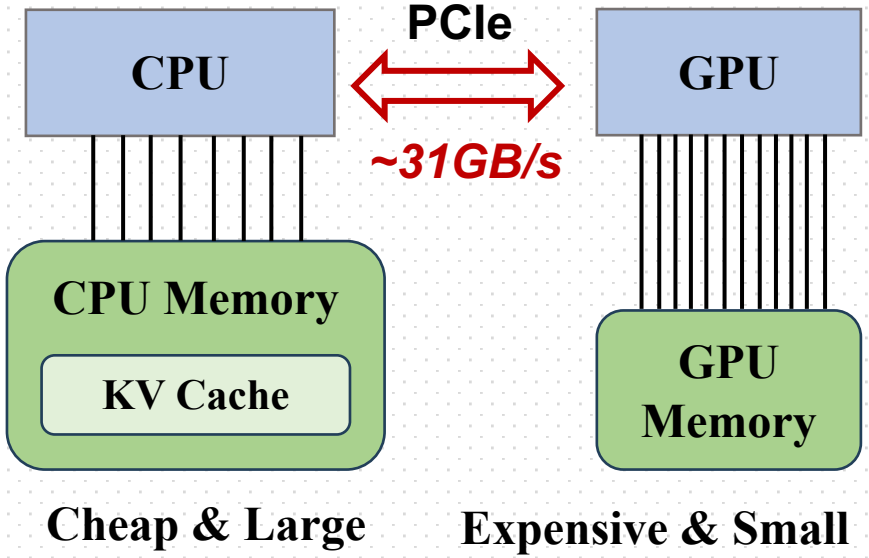
❑Collaboration between GPU and CPU

❖GPU

- ➢Highspeed computation, fast memory access
- ➢**Limited memory capacity**

❖CPU

- ➢Abundant memory
- ➢**Lower computational throughput**
- ➢The control logic for managing index structures is often better suited for CPU execution



❑Questions

❖Where to place data and computation?

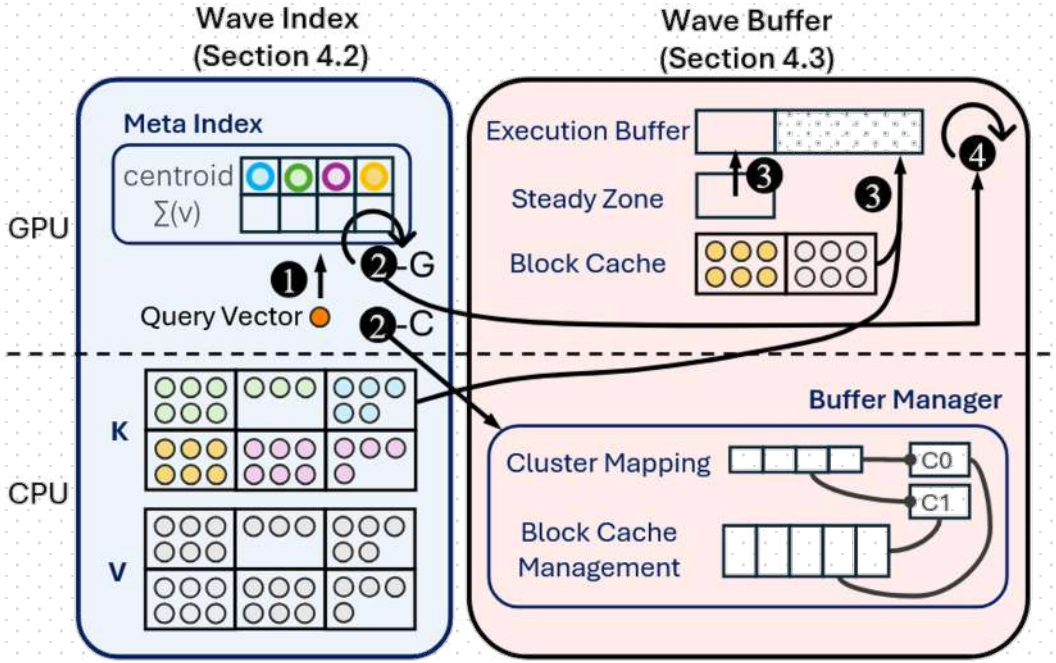
❖How to leverage parallelism and overlap to fully utilize both processors?

Contexts

- ❑Background
- ❑Motivation
- ❑RetroInfer
- ❑Evaluations

RetroInfer

- **RetroInfer**
  - **A Vector-Storage System for KV Cache**
- **Key designs**
  - **Wave index**
    - **Wave: Attention-aWare VEctor index**
  - **Wave buffer**



RetroInfer Overview

Wave Index: 3 Zones

❑Distinguish 3 zones within the token sequence

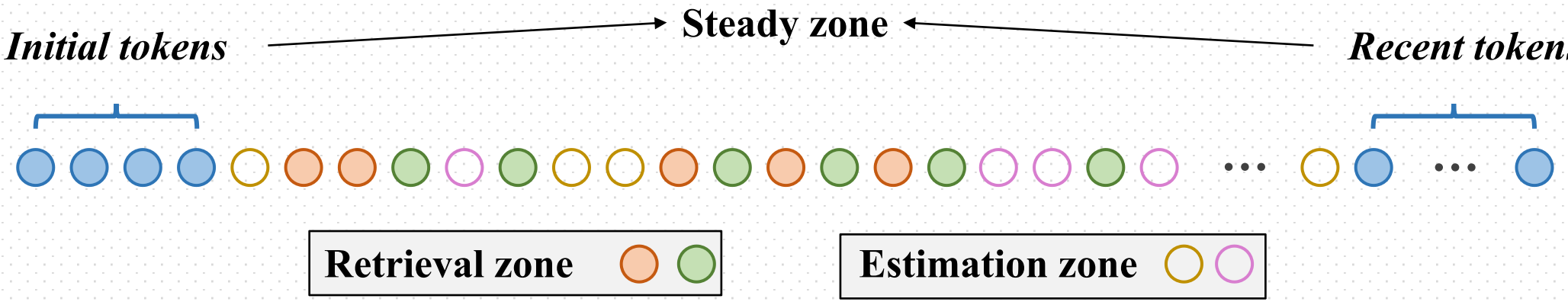
❖Steady zone: initial + recent tokens[1]

❖Retrieval zone: retrieve using cluster-based index

❖Estimation zone: unretrieved clusters

Physical

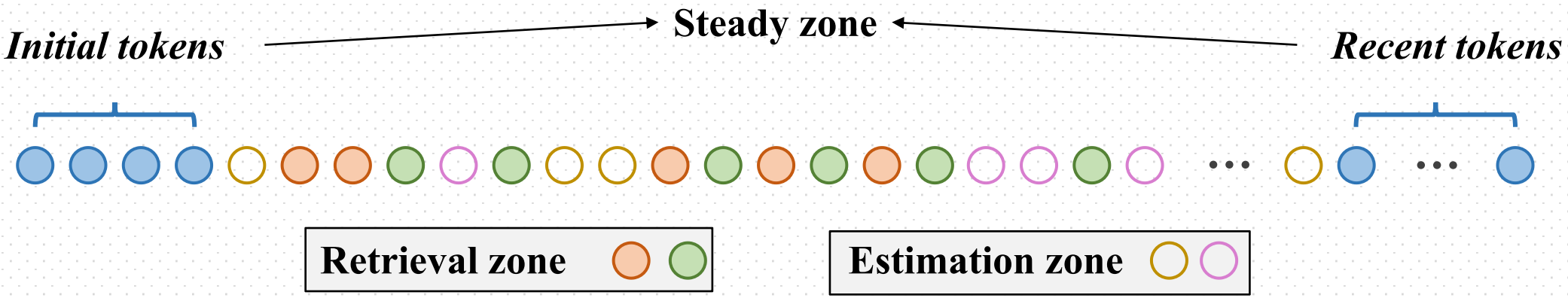
Logical



[1] StreamingLLM: Efficient Streaming Language Models with Attention Sinks

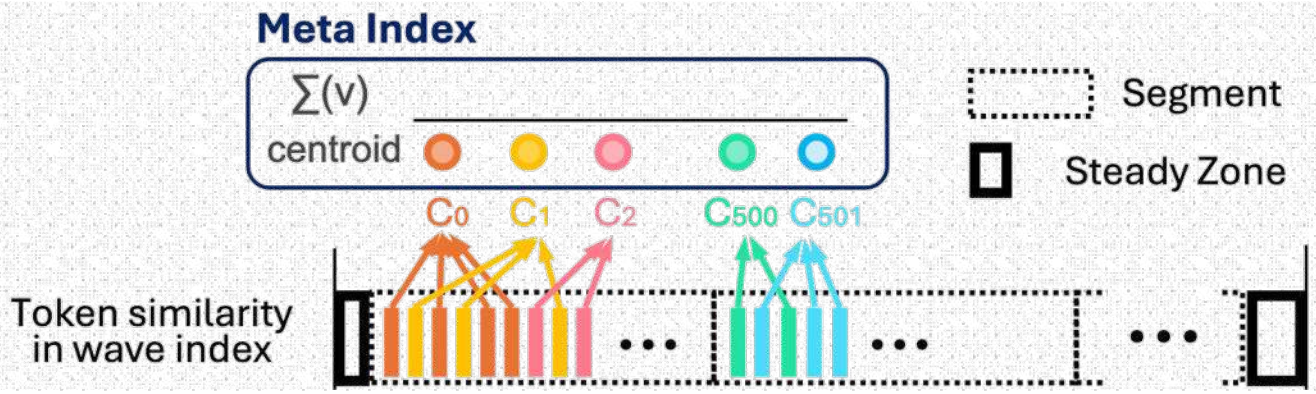
Wave Index: 3 Zones

- **Tripartite Attention Approximation**
  - **Steady zone o^0: precise attention**
  - **Retrieval zone o^1: precise attention**
  - **Estimation zone o^2: estimated attention**



Wave Index: Meta Index

- **Meta index**
  - **Spherical k-means clustering**
  - **Segment-level clustering**
    - **To save index construction overhead**
    - **Based on the coarse-grained spatial locality ?**



Meta index design

Wave Index: Estimation Zone

❑Accuracy-Bounded Attention Estimation

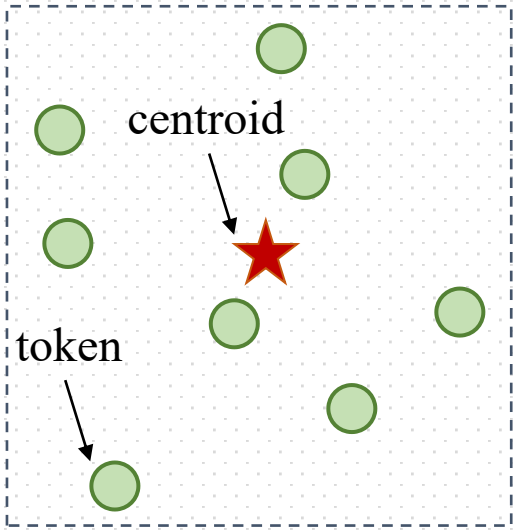
❖For un-retrieved clusters, use centroids to generate estimate attention

$$token\ atten. = softmax\left(\frac{qK^T}{\sqrt{d}}\right)_{token} v_{token}$$

use centroid's weight

$$\tilde{a}_i = \frac{e^{\frac{q \cdot C_i^T}{\sqrt{d}}}}{\underbrace{\sum_{j=1..p} e^{\frac{q \cdot K_j^T}{\sqrt{d}}}}_{p \text{ tokens in steady zone}} + \underbrace{\sum_{j=1..m} \left(s_j \cdot e^{\frac{q \cdot C_j^T}{\sqrt{d}}}\right)}_{\text{all } m \text{ clusters with } s_j \text{ tokens each}}}, i = 1..m$$

$\tilde{a}_i$: i-th centroid weight



i-th cluster and centroid

Wave Index: Estimation Zone

❑Accuracy-Bounded Attention Estimation

❖For un-retrieved clusters, use centroids to generate estimate attention

$$token\ atten. = softmax\left(\frac{qK^T}{\sqrt{d}}\right)_{token} v_{token}$$

use centroid's weight

$$\tilde{a}_i = \frac{e^{\frac{q \cdot C_i^T}{\sqrt{d}}}}{\sum_{j=1..p} e^{\frac{q \cdot K_j^T}{\sqrt{d}}} + \sum_{j=1..m} \left(s_j \cdot e^{\frac{q \cdot C_j^T}{\sqrt{d}}}\right)}, i = 1..m$$

i-th centroid weight

p tokens in steady zone

all m clusters with s_j tokens each

$$ith\ cluster\ atten.$$
$$= \sum_{token} softmax\left(\frac{qK^T}{\sqrt{d}}\right)_{token} v_{token}$$
$$\approx \sum_{token} \tilde{a}_i\, v_{token} = \tilde{a}_i \sum_{token} v_{token}$$
$$= \tilde{a}_i \sum v$$

Wave Index: Estimation Zone

❑Accuracy-Bounded Attention Estimation

❖For un-retrieved clusters, use centroids to generate estimate attention

$$\tilde{a}_i = \frac{e^{\frac{q \cdot C_i^T}{\sqrt{d}}}}{\sum_{j=1..p} e^{\frac{q \cdot K_j^T}{\sqrt{d}}} + \sum_{j=1..m} \left(s_j \cdot e^{\frac{q \cdot C_j^T}{\sqrt{d}}}\right)}, i = 1..m$$

i-th centroid weight

p tokens in steady zone

all m clusters with s_j tokens each

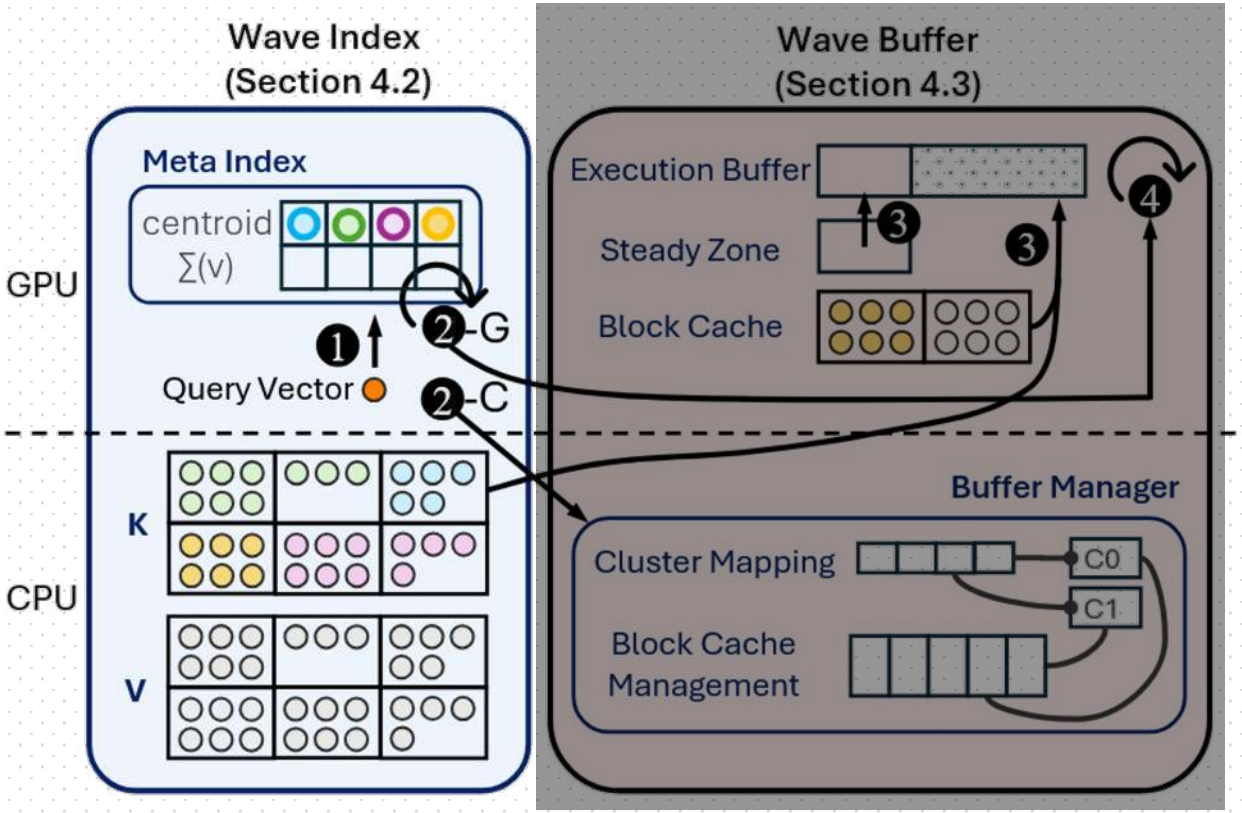
$$\begin{cases} C_i = \dfrac{\sum_{j=1..s_i} K_j}{s_i} \\ e^{\frac{q \cdot C_i^T}{\sqrt{d}}} \leq \dfrac{\sum_{j=1..s_i} e^{\frac{q \cdot K_j^T}{\sqrt{d}}}}{s_i} \end{cases}$$

Jensen's inequality

The estimation is a lower bound

Wave Index: Data Placement and Workflow

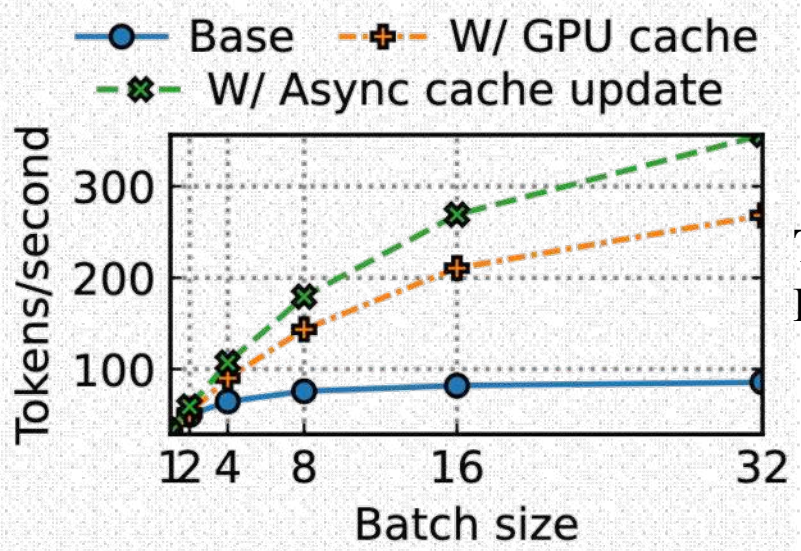
- **Physical placement**
  - CPU: KV vectors (stored in blocks)
  - GPU: centroids + ∑v
- **After prefilling**
  - GPU performs segmented clustering
  - KV vectors offload asynchronously
  - CPU construct related data structure asynchronously
- **Decoding**
  - GPU search centroids
  - CPU send retrieved KV blocks



RetroInfer Overview

Wave Buffer: Why and How Is it Possible

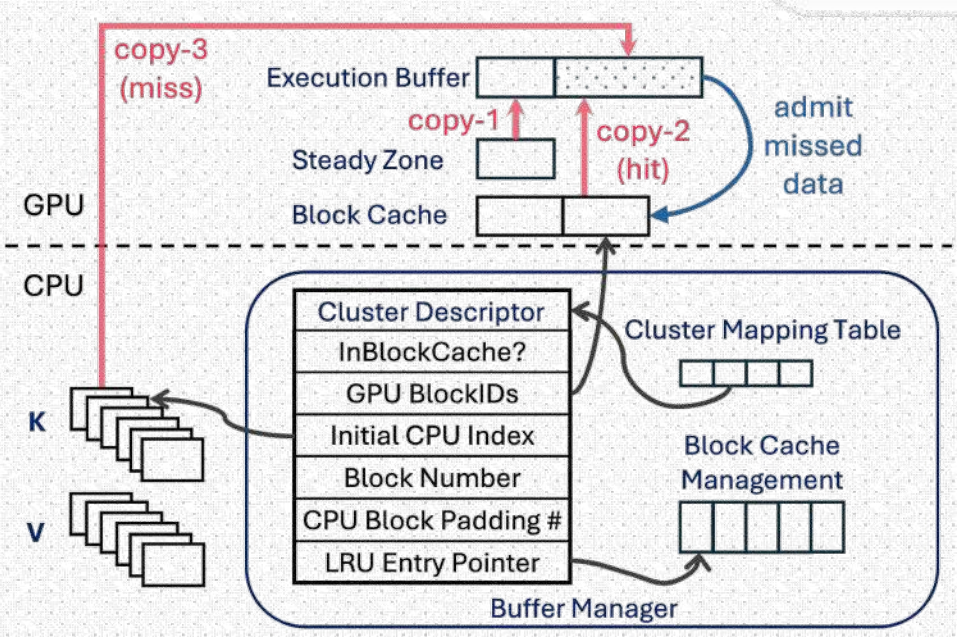
- **Data traffic is still high**
  - **Though wave index reduce it to <2% of full attention**
- **A GPU buffer is designed**
  - Based on a phenomenon *"adjacent decoding steps for neighboring query vectors show strong temporal locality"*



Throughput with batch sizes
Base is constrained by PCIe bandwidth

Wave Buffer: Overview

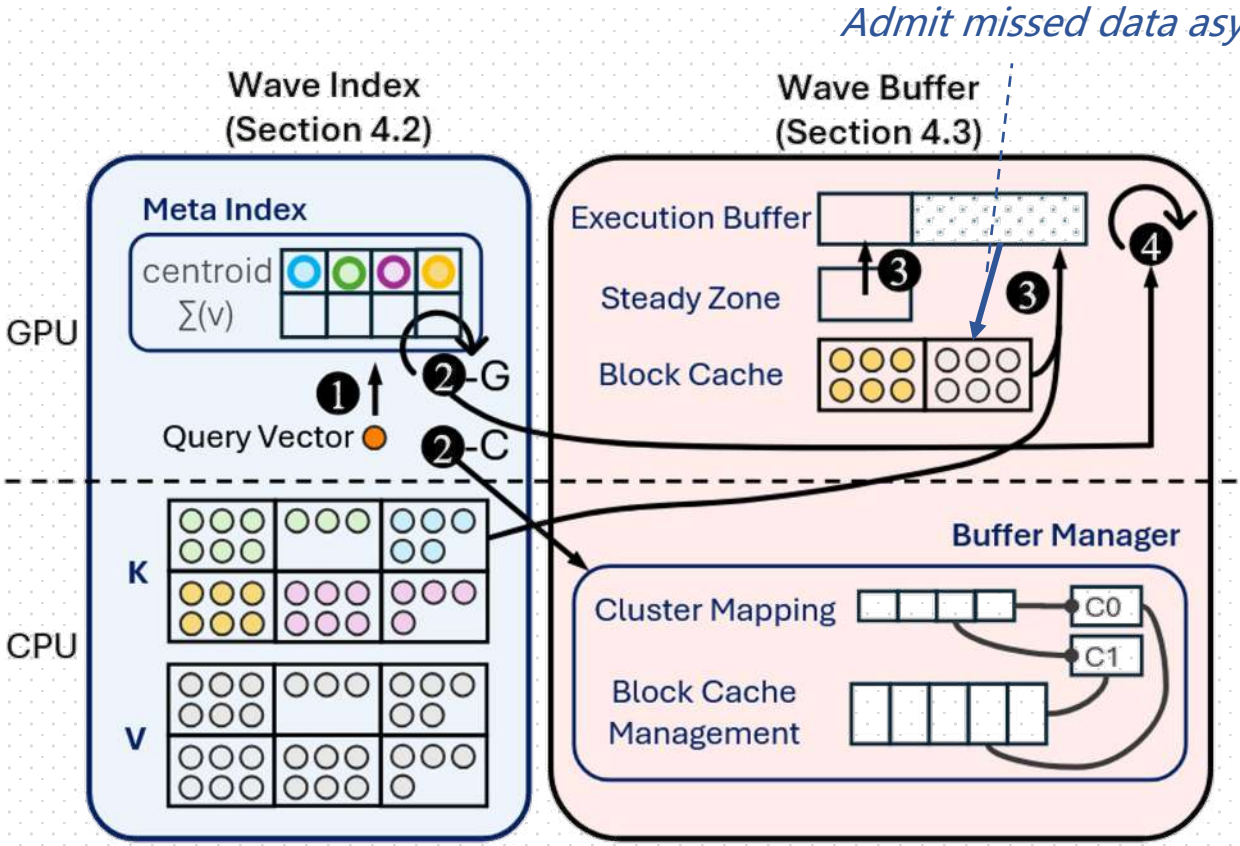
- Buffer contents on GPU
  - Execution buffer
  - Steady zone
  - Block cache
- During decoding: 3 Data copy
- Control plane on CPU
  - A table containing cluster-id to infos
  - Cache policy: just LRU



Wave buffer design

Coordination between CPU/GPU

- **Synchronous cache access**
  - The KV vectors are ready as soon as possible
- **Asynchronous cache update**
  - Cache update
  - Admit missed data
  - Removed from critical path

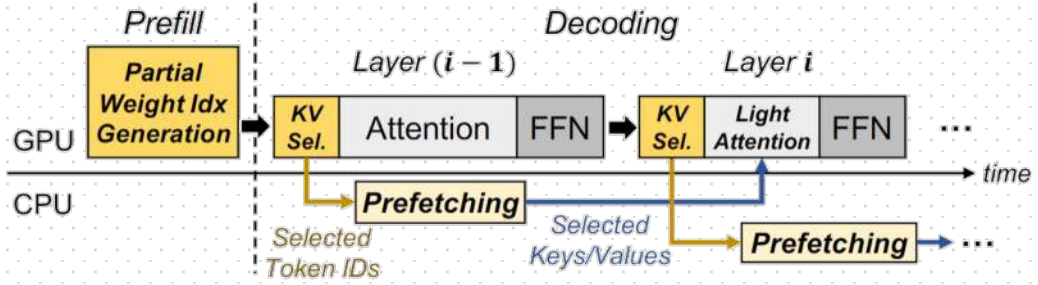
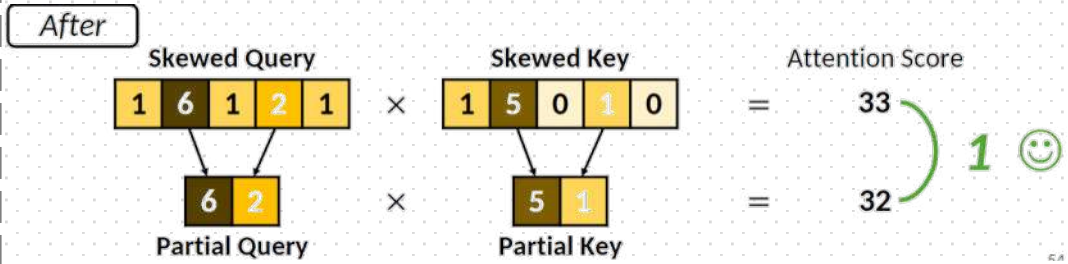
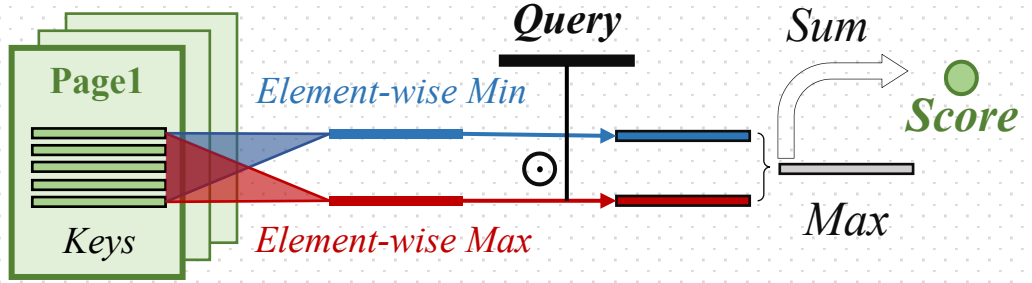
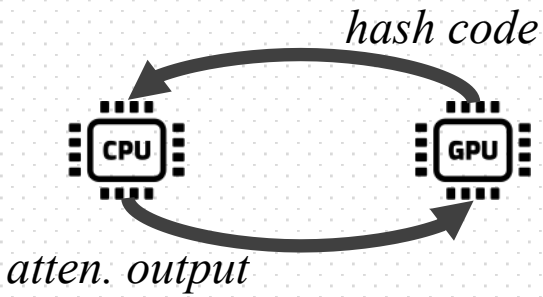


RetroInfer Overview

Contexts

- ❑Background
- ❑Motivation
- ❑RetroInfer
- ❑Evaluations

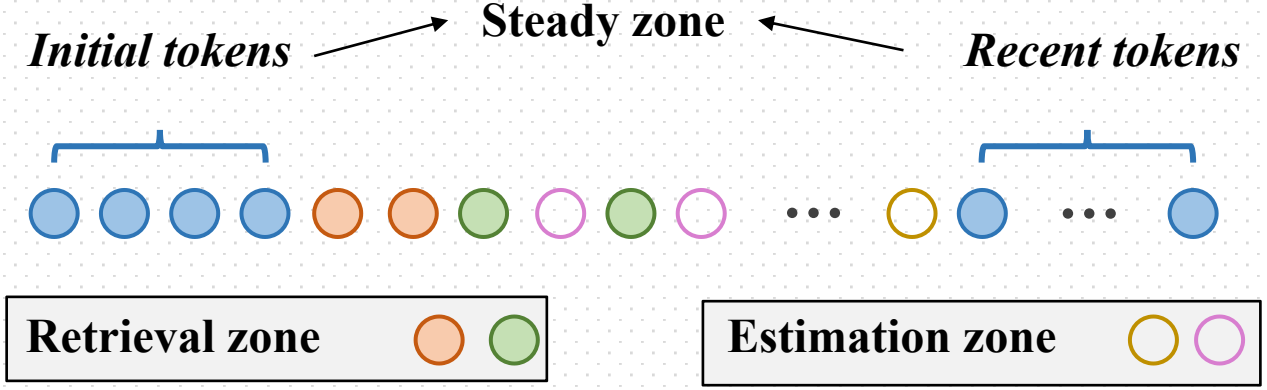
Baselines

	Data transfer	*TopK Selection*
InfiniGen OSDI 24	 **Prefetching, but frequent data transfer**	 **Use reduced dimension to find topk, still costly**
Quest ICLR 24	**Quest-Offload is implemented in this paper**	 **Use page level reduced keys, accuracy loss**
MagicPIG ICLR 25	**Attention is done on CPU** 	**Use Locality Sensitive Hashing (LSH) to sample important Key/Value, accuracy loss**

Parameters

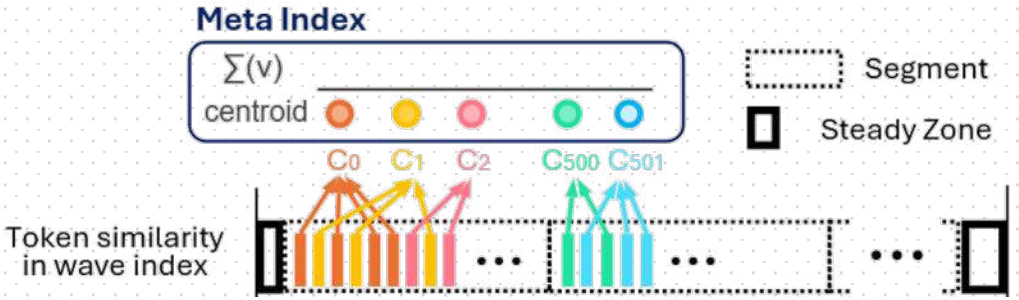
❑Zones setup

- ❖Steady zone: 4 + 64
- ❖Retrieval zone: 1.8%
- ❖Estimation zone: 23%



❑Segment clustering

- ❖8K tokens in a segment
- ❖#clusters = 1/16 #tokens



Meta index design

Parameters

- **Cache Setup**
  - **5% of all KV vectors**
  - **Page size: 2KB**
- **Baseline setup**
  - **Follow the paper statements**

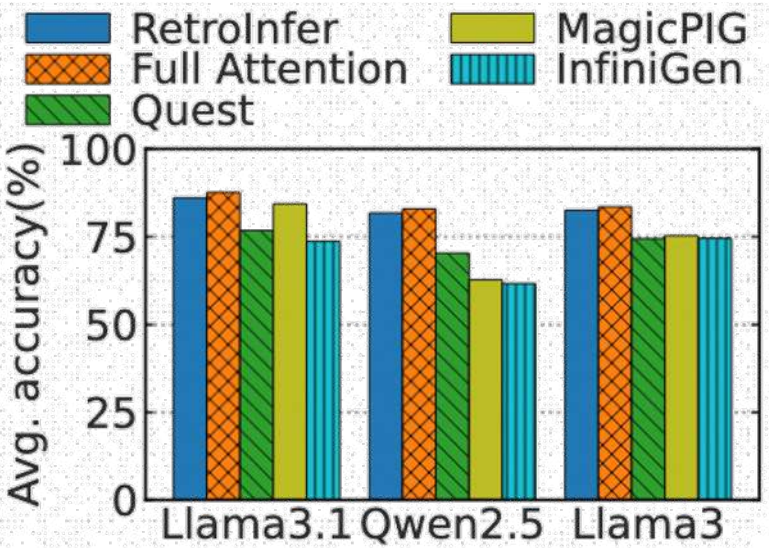
Other Configuration

- **Hardware**
  - 1 × NVIDIA A100 GPU (80GB memory)
  - AMD EPYC 7V12 64-core CPU (1900GB memory)
  - PCIe 4.0 ×16 (~31GB/s)
- **Models**
  - Llama3.1-8B
  - Qwen2.5-7B

  } 128K context
  - Llama38B-1048K

Accuracy

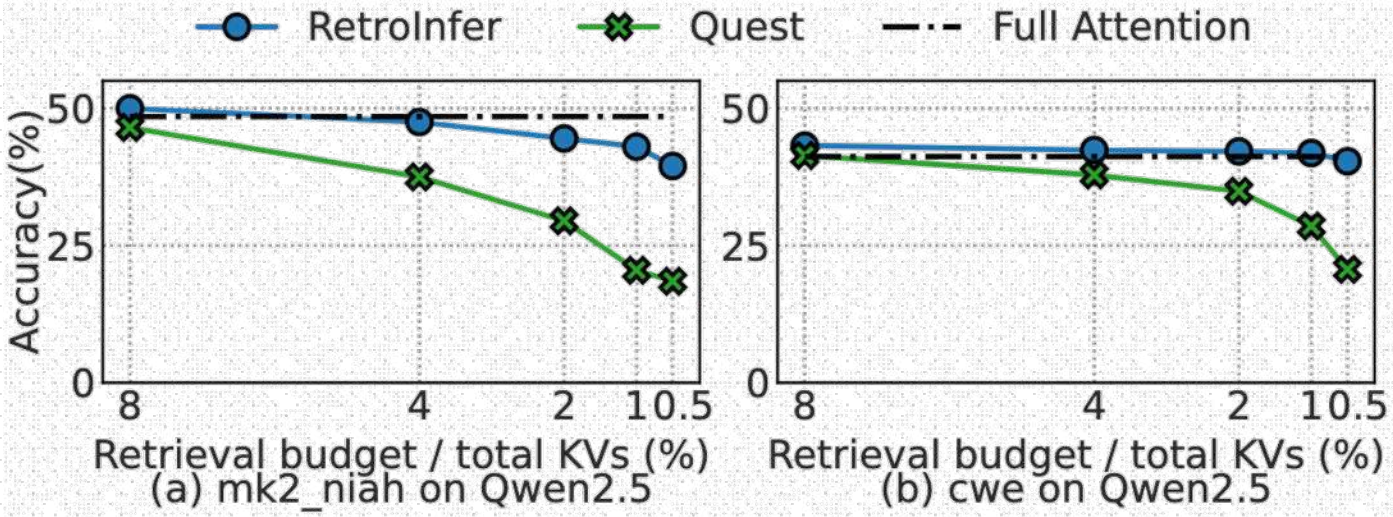
- RetroInfer always approaches full attention accuracy
  - Drop 0.73%/0.78%/1.46% compared to full attention
  - Achieves 5.48%/15.51%/3.33% accuracy improvements compared to the best-performing baselines on each model



Average Model Accuracy
(RULER, 8K to 128K Context)

Accuracy

- RetroInfer maintains accuracy close to full attention, even with a low retrieval budget
- RetroInfer sometimes outperforms full attention



Accuracy vs. Retrieval Budgets

Efficiency

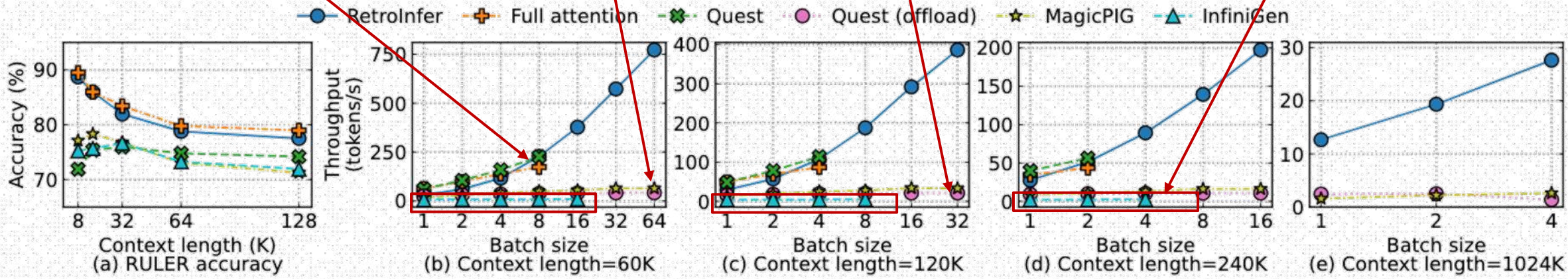
❑**RetroInfer scales well and achieves the best throughput**

MagicPIG is constrained by CPU compute capacity

InfiniGen exhibits worst because of its expensive speculative operations and frequent data transfer over PCIe

Quest and full attention cannot scale beyond GPU memory

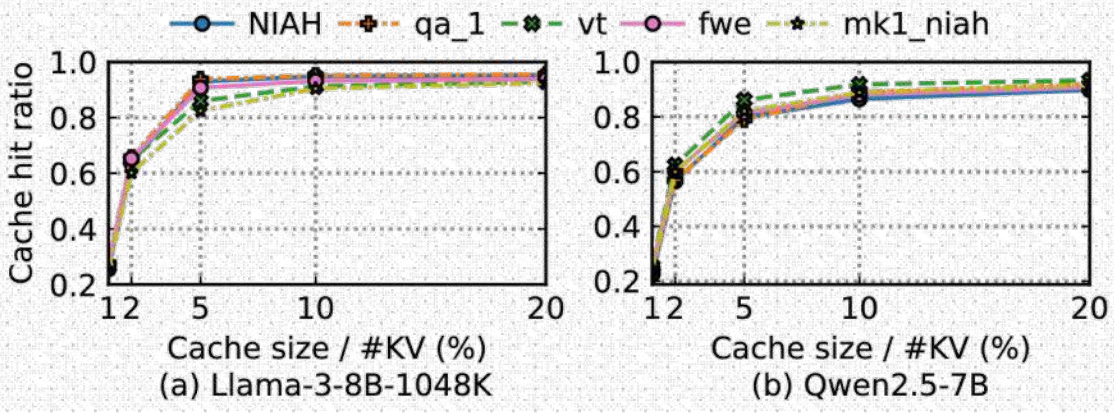
Quest (offload) exhausts the limited PCIe bandwidth



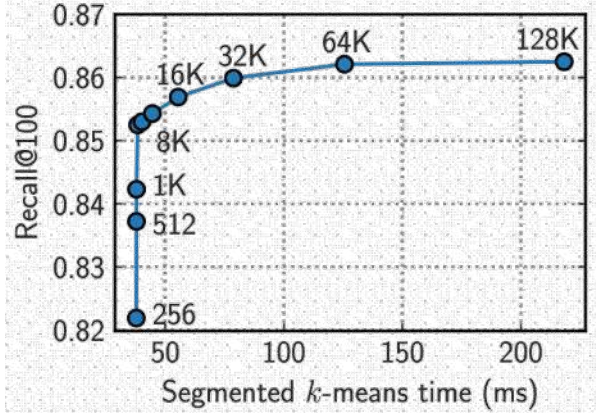
Accuracy and Decoding throughput vs. Context Length and Batch Size
Benchmark: RULER, model: Llama3-8B-1048K

Micro Analysis

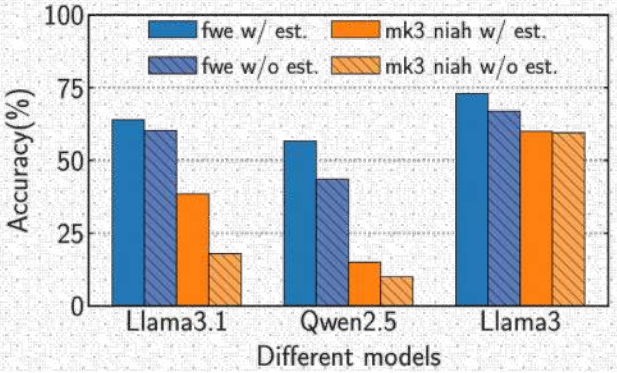
❑Cache ratio: 5% is good



❑Segment size: 8K is good



❑Estimation helps a lot



❑Decoding Latency Breakdown (ms)

Batch size	1	32
Access meta index	0.1447	0.5132
Estimation zone attention	0.1010	0.5398
Access cluster mapping table	0.2405	0.5902
Update cluster mapping table	0.0142	0.5176
Data copy to execution buffer	0.1281	0.9008
Steady and retrieval zone attention	0.0777	0.2950
Admit missed data	0.0572	0.1038

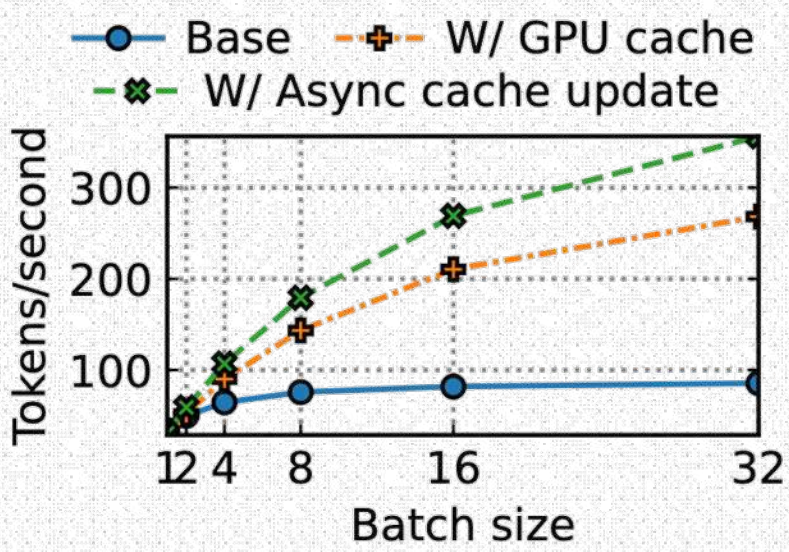
Conclusions

❑Pros

- ❖Innovative technique: estimation via centroids and ∑v
- ❖High completeness of system design
- ❖Excellent experimental results
- ❖Good story: vector storage system

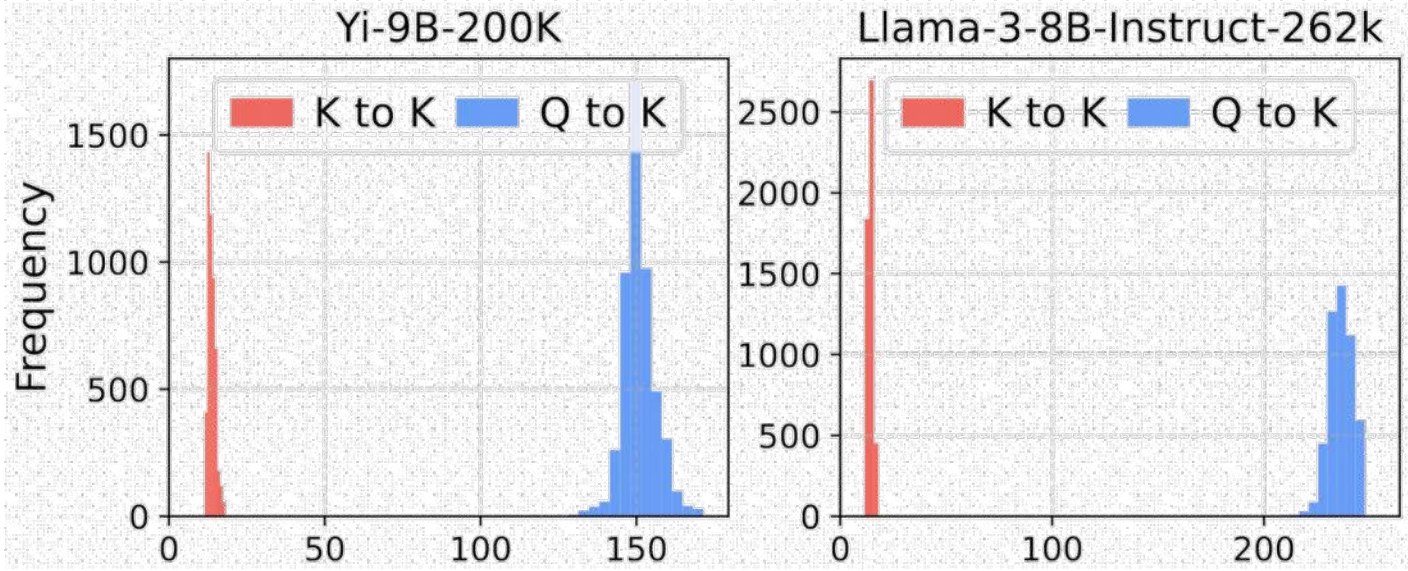
❑Cons

- ❖Buffer contributes a lot…
- ❖Disparity between motivation and solution
- ❖Unclear explanation or lack of deep analysis
  - ➢ Impact of Zone ratio (retrieval / estimation)
  - ➢ ANNS OOD issue (see next page)



OOD Issue: When ANNS Meets KV Cache

- **OOD (out-of-distribution)**
  - **The distribution of queries differs from that of the base data**
- **Search in the KV cache exhibits strong OOD characteristics**

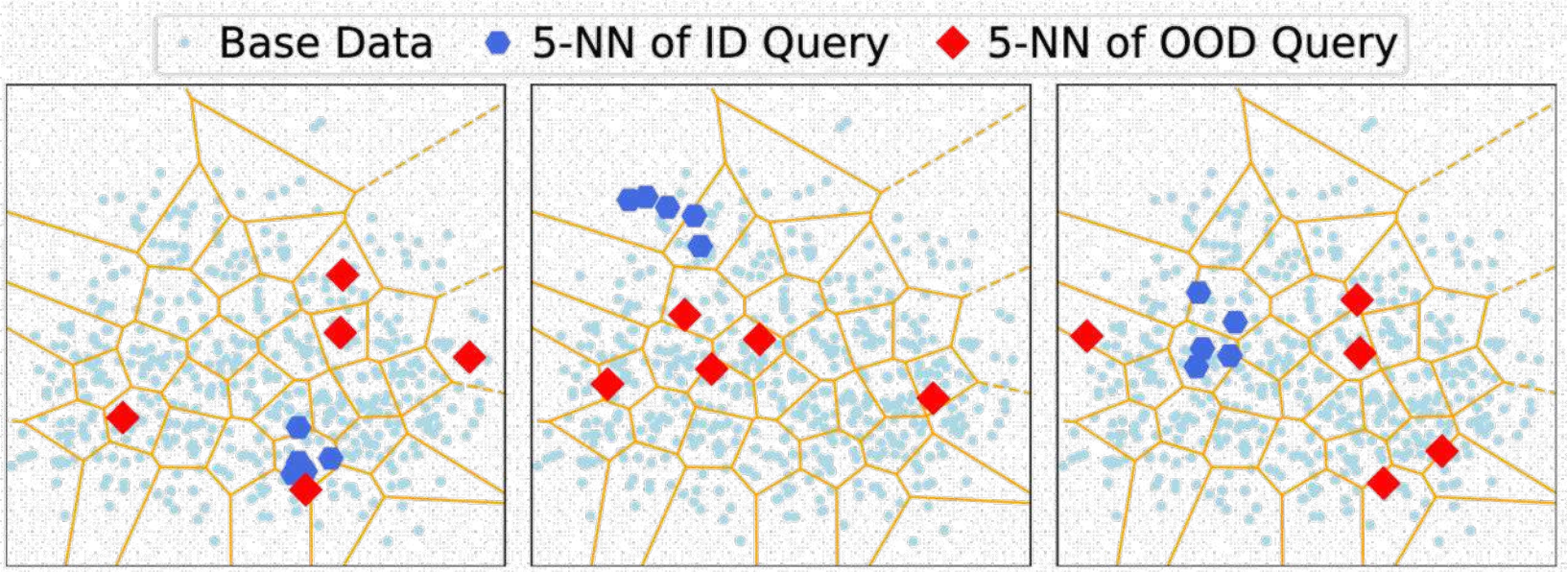


Mahalanobis Distance between 5000 samples of query and key vectors[1]

[1] RetrievalAttention: Accelerating Long-Context LLM Inference via Vector Retrieval

OOD Increases the Searching Difficulty

- ❖Most ANNS indexes are built for in-distribution search
- ❖OOD queries challenge the assumptions of standard ANNS methods.



5 Nearest neighbors' distribution in cluster-based index for ID query and OOD query[1]

[1] RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search

End

- Thank you!
- Q&A