ArkVale: Efficient Generative LLM Inference with Recallable Key-Value Eviction

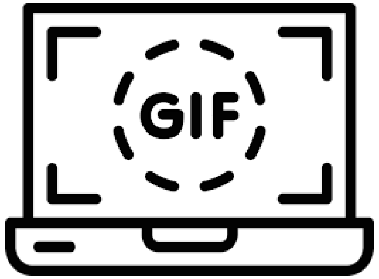
Renze Chen[1], Zhuofeng Wang[1], Beiquan Cao[1], Tong Wu[1], Size Zheng[1],
Xiuhong Li[1], Xuechao Wei[1], Shengen Yan[2], Meng Li[1], Yun Liang[1]

[1]Peking University
[2]Infinigence-AI

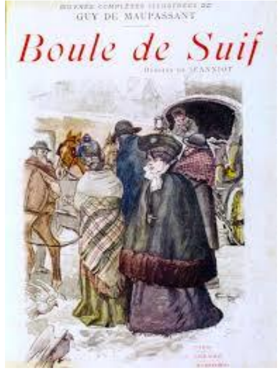
Long-form Contents

There are various long-form contents in our daily lives.

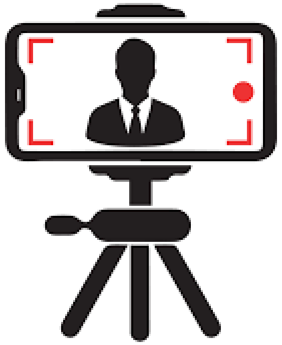
A research paper



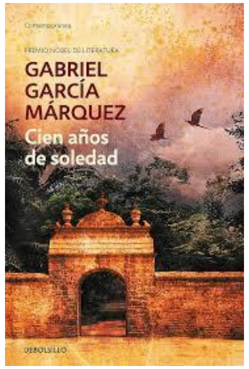
A gif meme



A novella



A short video



A novel

An EP of an anime

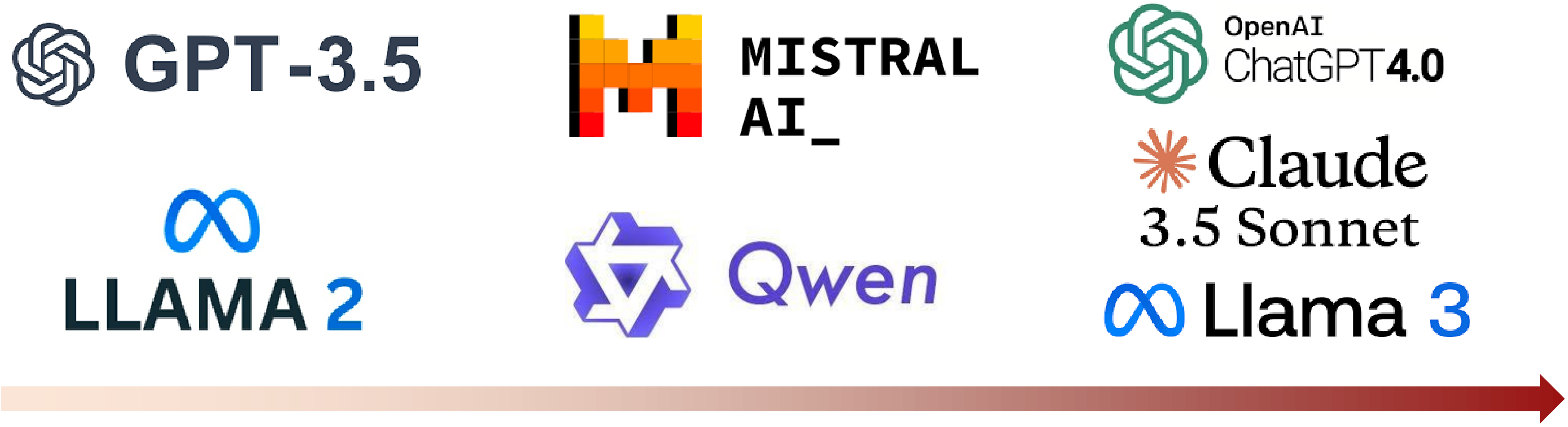
~8k tokens

~32k tokens

~128k tokens

Long Context LLMs

Context-length supported by LLMs also grows rapidly



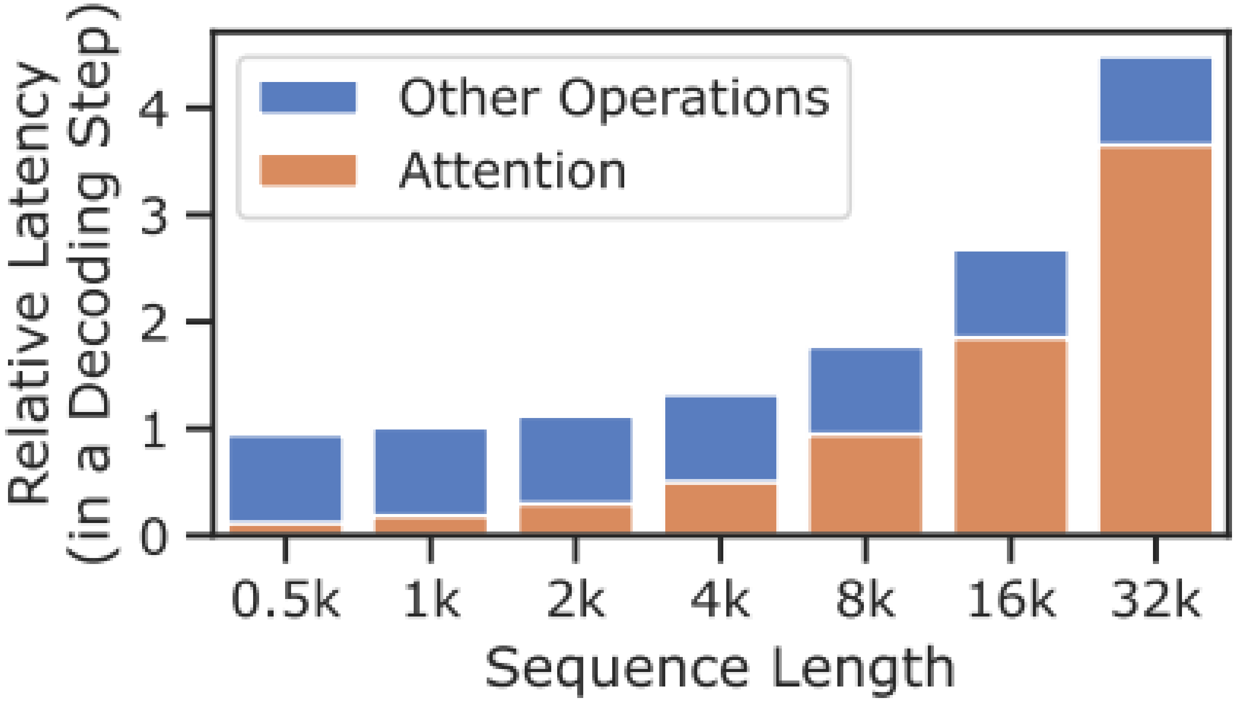
4k tokens

32k tokens

128k tokens

Impact of Long Context

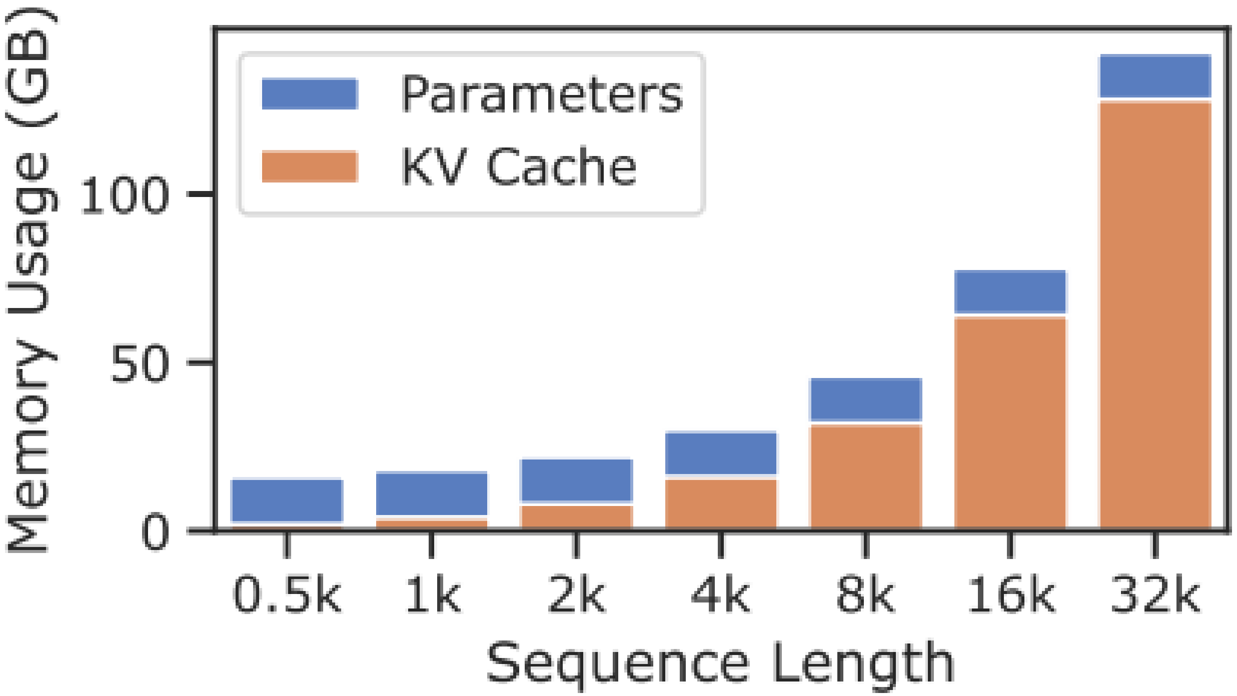
Long context attention can be the latency bottleneck of LLM decoding



LongChat-7b-v1.5-32k (batch-size=8)

Impact of Long Context

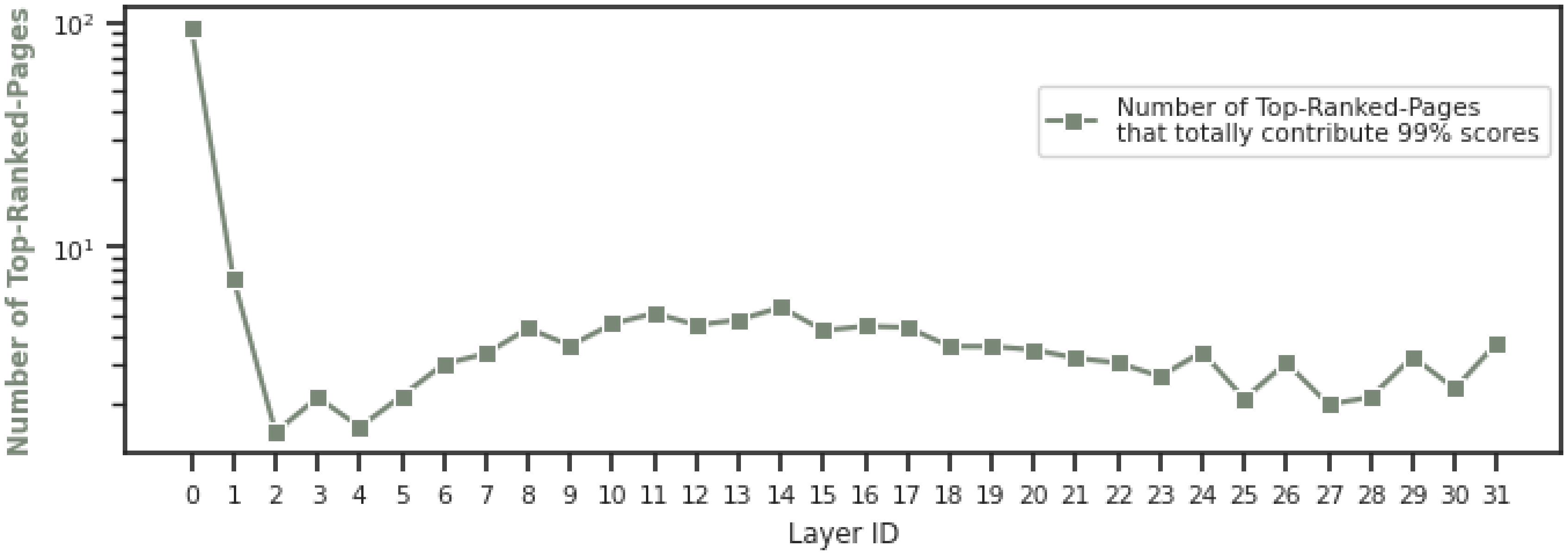
Long context can be the memory bottleneck of LLM decoding, which hampers the use of larger batch-size for serving.



LongChat-7b-v1.5-32k (batch-size=8)

Observation: Token-level Sparsity of KV Cache

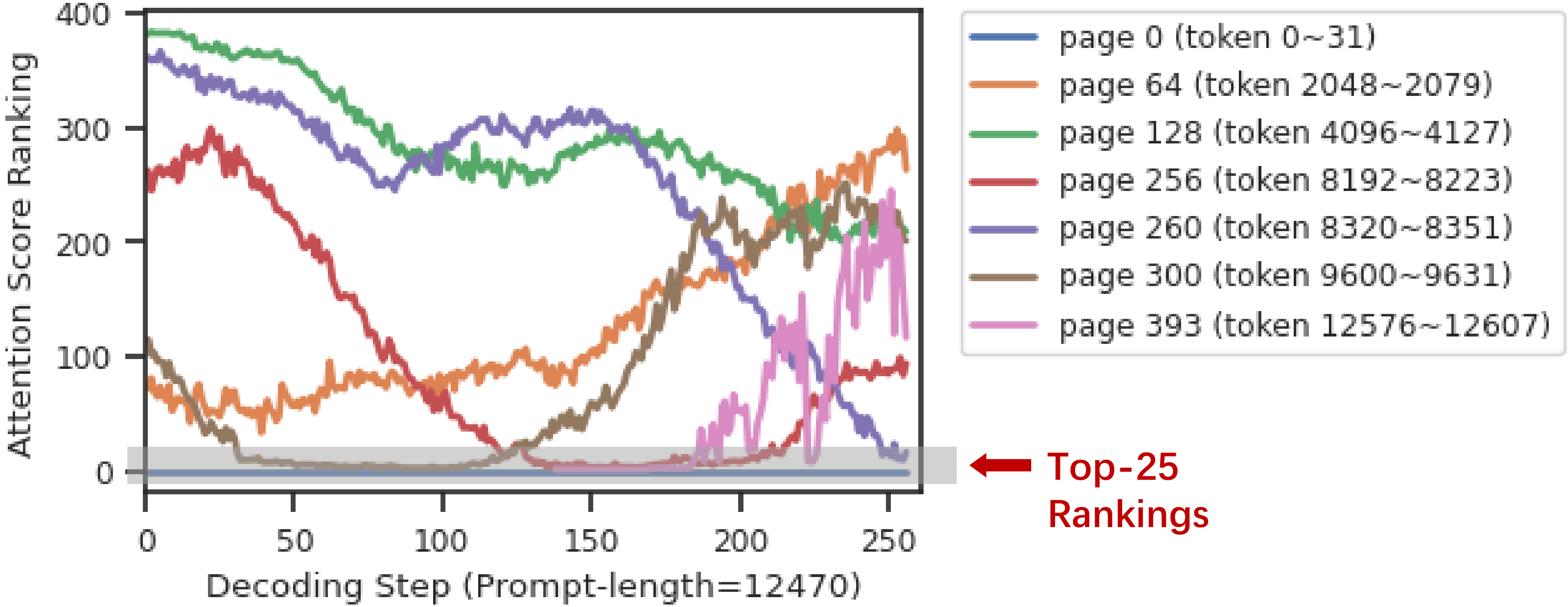
- In most LLM layers, less than 10 KV-cache pages (page-size=32) contributing over 99% of attention scores.



We can only keep those important tokens to save memory and make attention computation more efficient.

Observation: Dynamism of Token Importance

Importance of KV-cache token/page can dynamically change overtime

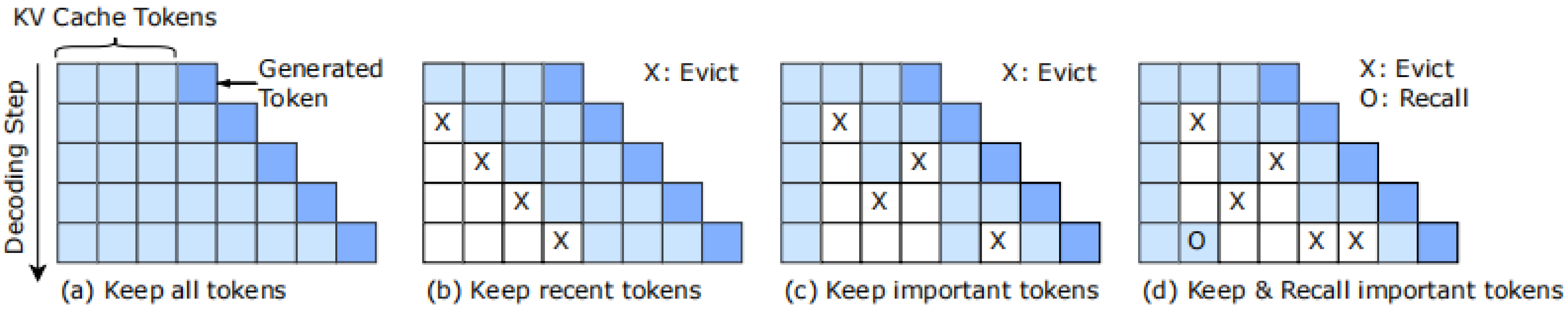


Top-25 Rankings

The attention score ranking of different KV-cache pages over decoding steps.

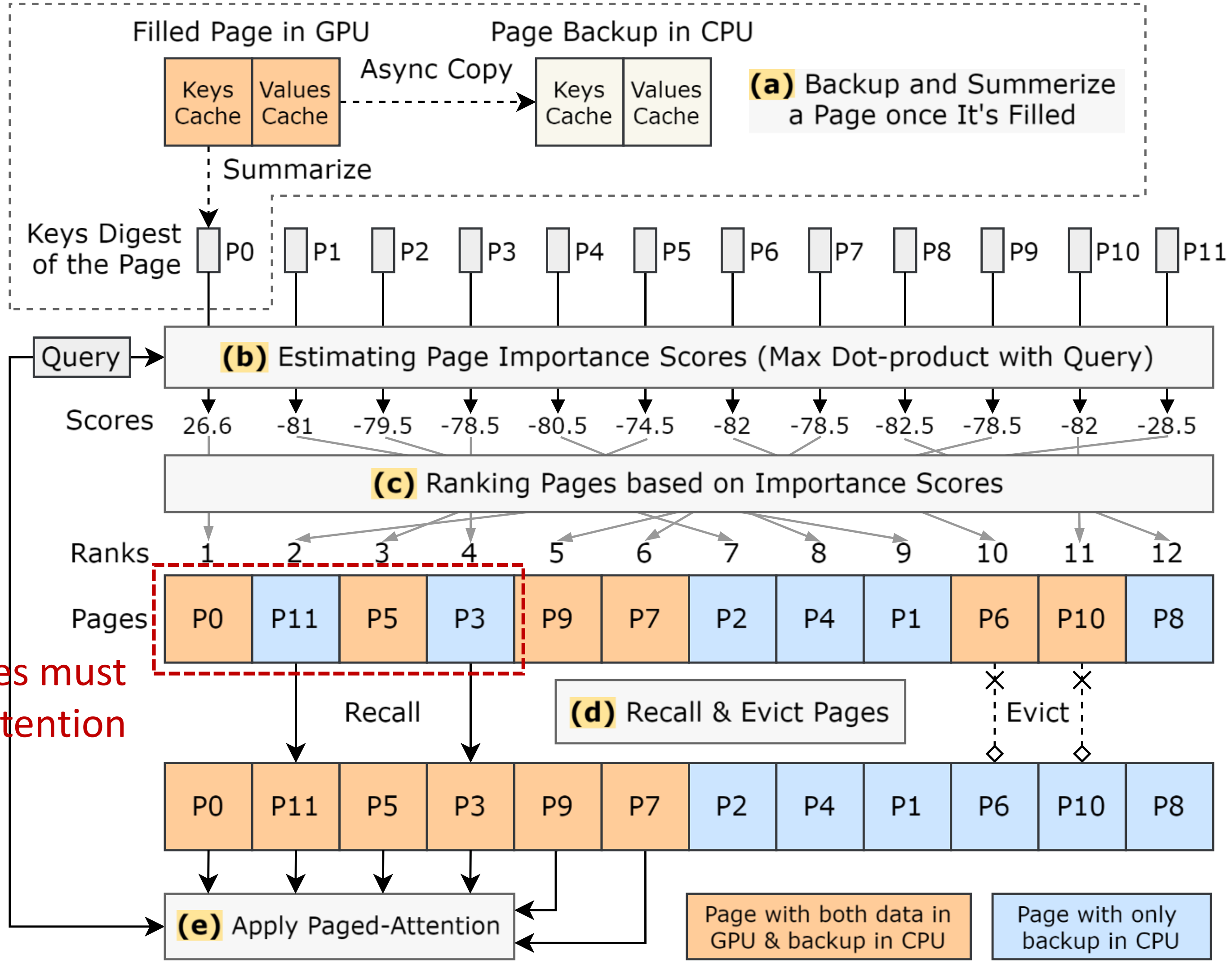
Observation: Dynamism of Token Importance

- Previous works permanently evict unimportant tokens based on history attention scores, but the evicted tokens may be important in the future.
- We propose a method named ArkVale to properly recall important tokens as well as evict unimportant ones during LLM decoding.



(a) Keep all tokens (b) Keep recent tokens (c) Keep important tokens (d) Keep & Recall important tokens

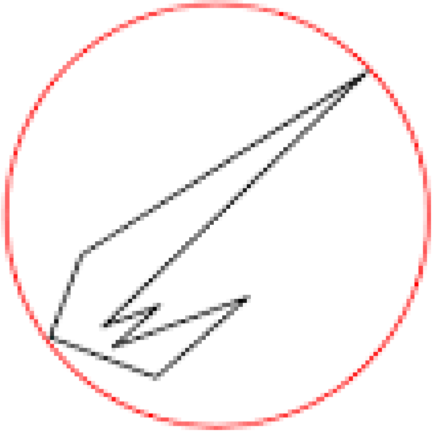
ArkVale: Workflow Overview


Filled Page in GPU
Page Backup in CPU
Async Copy
Keys Cache
Values Cache
Keys Cache
Values Cache
(a) Backup and Summerize a Page once It's Filled
Summarize
Keys Digest of the Page
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11
Query
(b) Estimating Page Importance Scores (Max Dot-product with Query)
Scores
26.6 -81 -79.5 -78.5 -80.5 -74.5 -82 -78.5 -82.5 -78.5 -82 -28.5
(c) Ranking Pages based on Importance Scores
Ranks
1 2 3 4 5 6 7 8 9 10 11 12
Pages
P0 P11 P5 P3 P9 P7 P2 P4 P1 P6 P10 P8
Recall
(d) Recall & Evict Pages
Evict
P0 P11 P5 P3 P9 P7 P2 P4 P1 P6 P10 P8
(e) Apply Paged-Attention
Page with both data in GPU & backup in CPU
Page with only backup in CPU

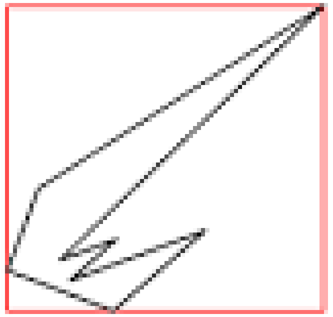
top-k (k=4) pages must participate attention

Page Summarization & Importance Estimation

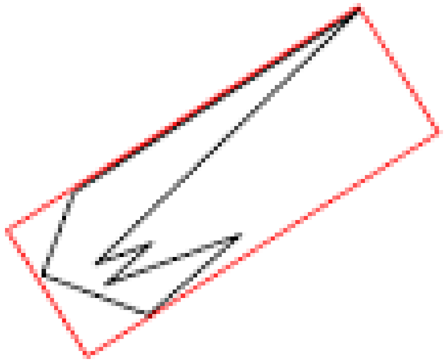
- **Definition:** For a page with keys $K = \{\mathbf{k}^{(i)}\}_{i=1}^{n}$ $(\mathbf{k}^{(i)} \in \mathbb{R}^d)$, and a query $\mathbf{q} \in \mathbb{R}^d$, the *page importance* of K in terms of $\mathbf{q}$ is defined as $\max_{\mathbf{k} \in K} \mathbf{q} \cdot \mathbf{k}$
- **Observation:** $\mathbf{k}' = \operatorname{argmax}_{\mathbf{k} \in K} \mathbf{q} \cdot \mathbf{k}$ must be one of the “outmost” points of K
- **Solution:** We can use the concept of *bounding-volume* (from computer graphics area) for page summarization and importance estimation.



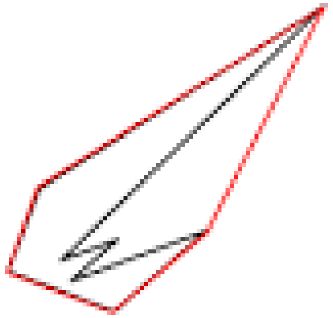
Bounding-Sphere



Bounding-Cuboid
(Axis-Aligned Bounding Box)



Oriented Bounding Box



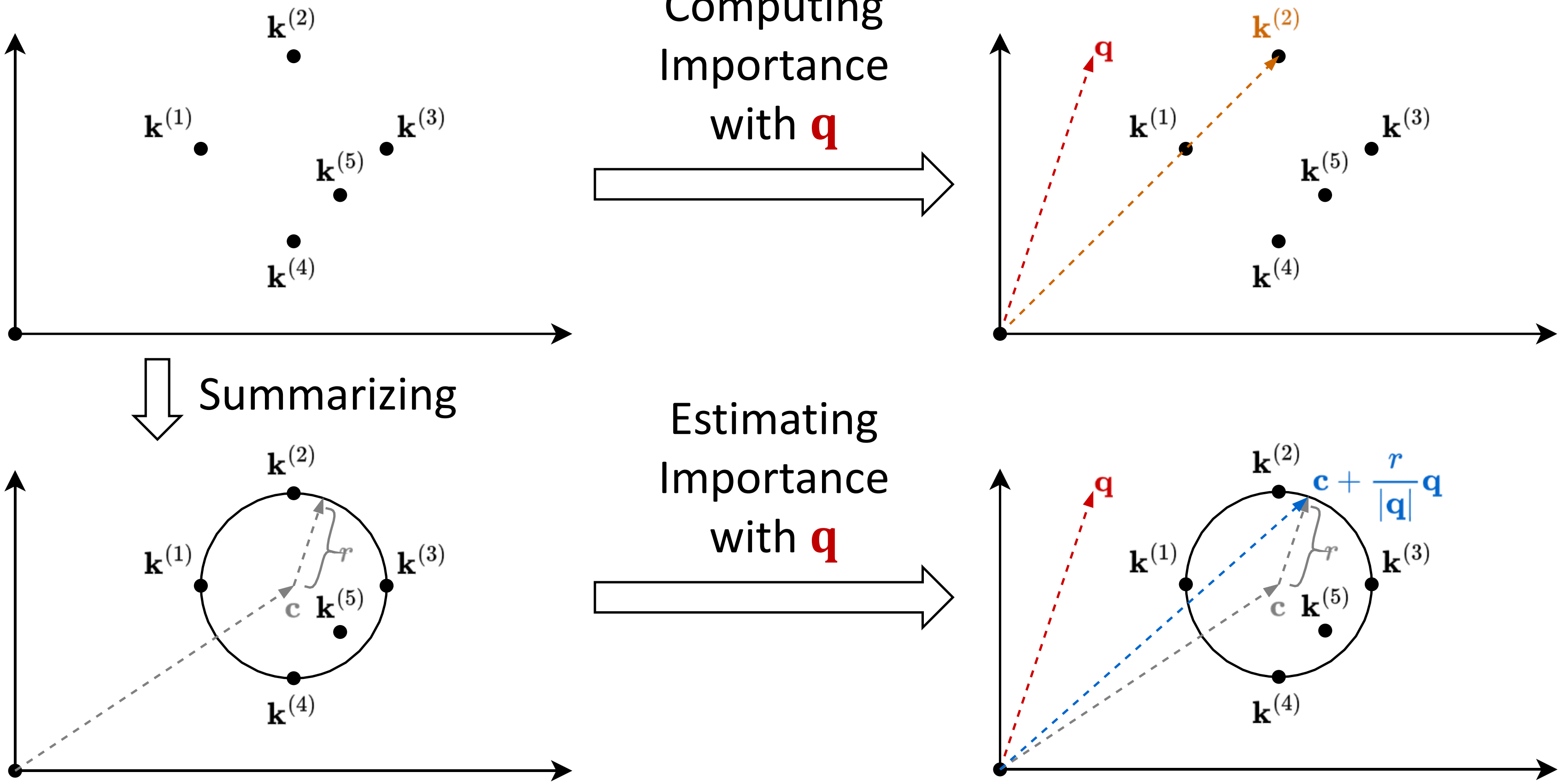
Convex Hull

https://www.ncollide.org/bounding_volumes/

Bounding-Sphere for Summarization & Estimation

Page Keys: $K = \{\mathbf{k}^{(i)}\}_{i=1}^{5}$

Computed Importance: $\mathbf{q} \cdot \mathbf{k}^{(2)}$



Computing Importance with $\mathbf{q}$

Summarizing

Estimating Importance with $\mathbf{q}$

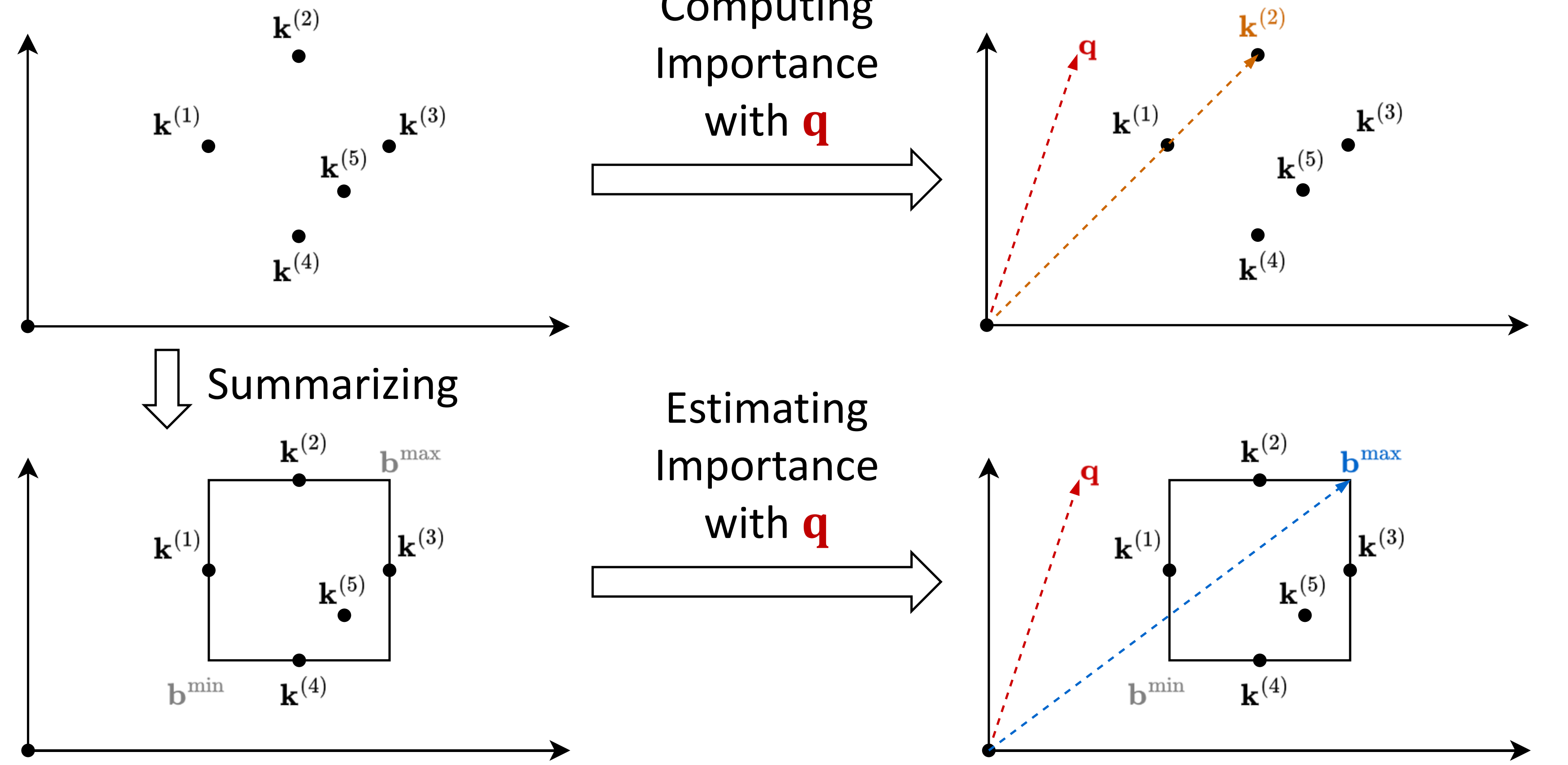
Page Digest: Sphere $(\mathbf{c}, r)$

Estimated Importance: $\mathbf{q} \cdot (\mathbf{c} + \frac{r}{|\mathbf{q}|}\mathbf{q})$

Bounding-Cuboid for Summarization & Estimation

Page Keys: $K = \{\mathbf{k}^{(i)}\}_{i=1}^{5}$

Computed Importance: $\mathbf{q} \cdot \mathbf{k}^{(2)}$



Computing Importance with $\mathbf{q}$

Summarizing

$$\mathbf{b}^{\min} = \min_{\mathbf{k} \in K} \mathbf{k}$$

$$\mathbf{b}^{\max} = \max_{\mathbf{k} \in K} \mathbf{k}$$

Estimating Importance with $\mathbf{q}$

Page Digest: Cuboid $(\mathbf{b}^{\min}, \mathbf{b}^{\max})$

Estimated Importance: $\mathbf{q} \cdot \mathbf{b}^{\max} =$ $\mathrm{sum}(\max(\mathbf{q} \odot \mathbf{b}^{\min}, \mathbf{q} \odot \mathbf{b}^{\max}))$

Evaluation Setup

Platform
Intel(R) Xeon(R) Gold 6348 CPUs
NVIDIA A100 80GB PCIe GPU

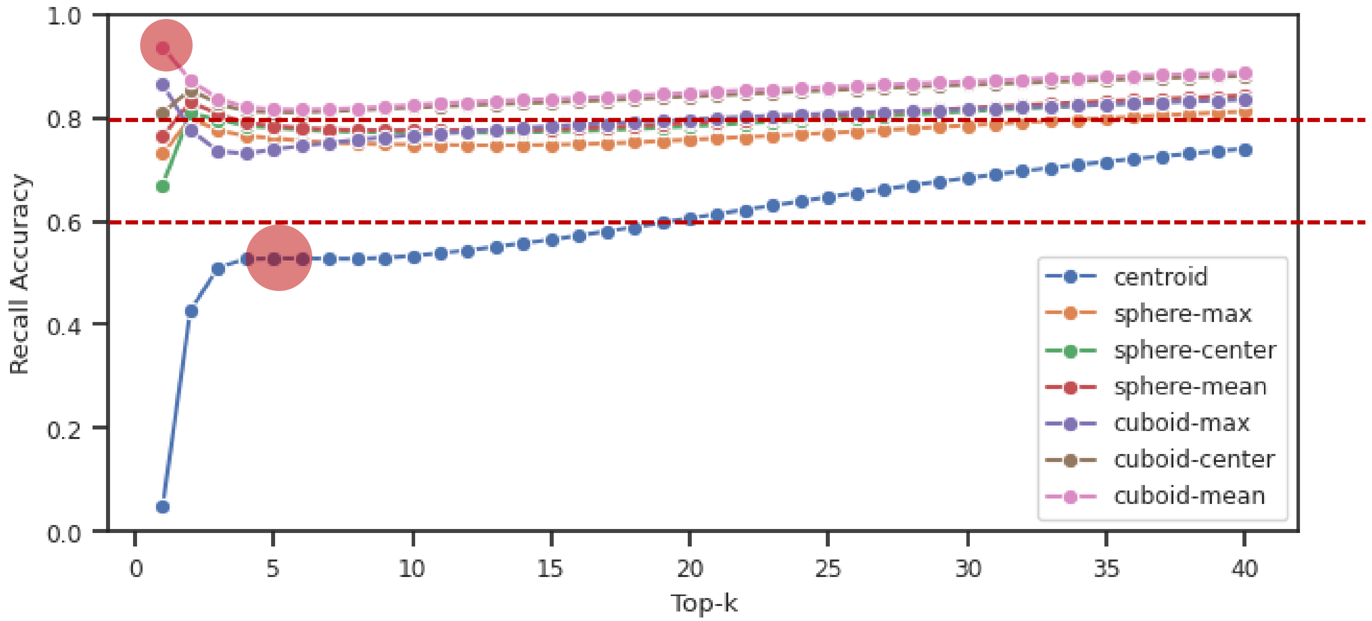
Baseline	Method
Origin/Full	Without KV-cache eviction
StreamingLLM (ICLR’23)	Retain initial tokens + recent tokens
H2O (NIPS’23)	Evict tokens based on history scores
TOVA	Evict tokens based on history scores

Benchmark
Long-Bench

Base Model
LongChat-v1.5-32k

Importance Estimation Accuracy

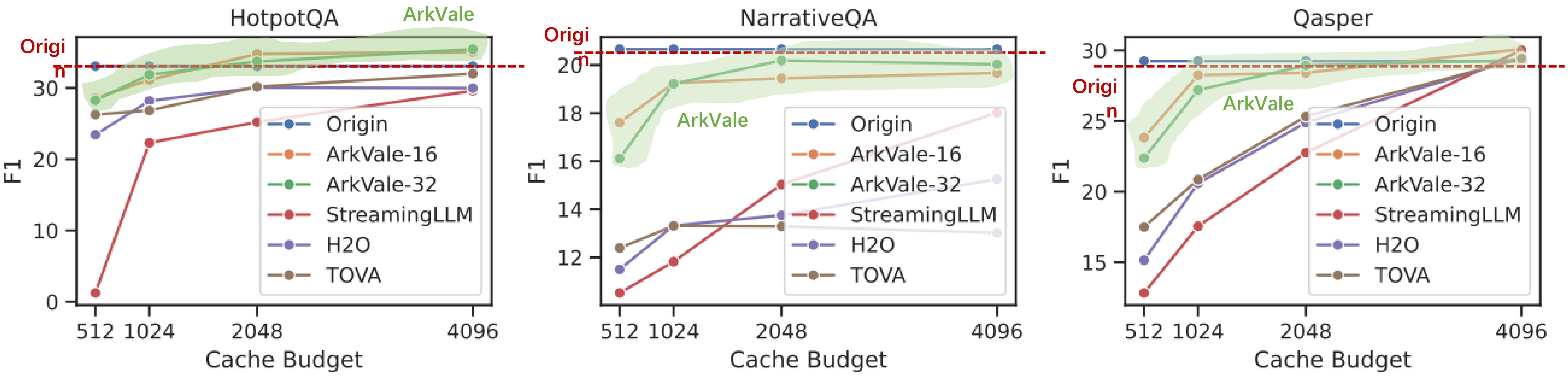
- ◇ Baseline method (centroid) cannot achieve even 60% top-5 recall accuracy.
- ◇ Our methods can achieve 60% top-k recall accuracy for all k.
- ◇ Our cuboid-mean method ensure 95% top-1 recall accuracy, and can achieve 80% top-k recall accuracy for all k.



Top-k recall accuracy of different importance estimation methods

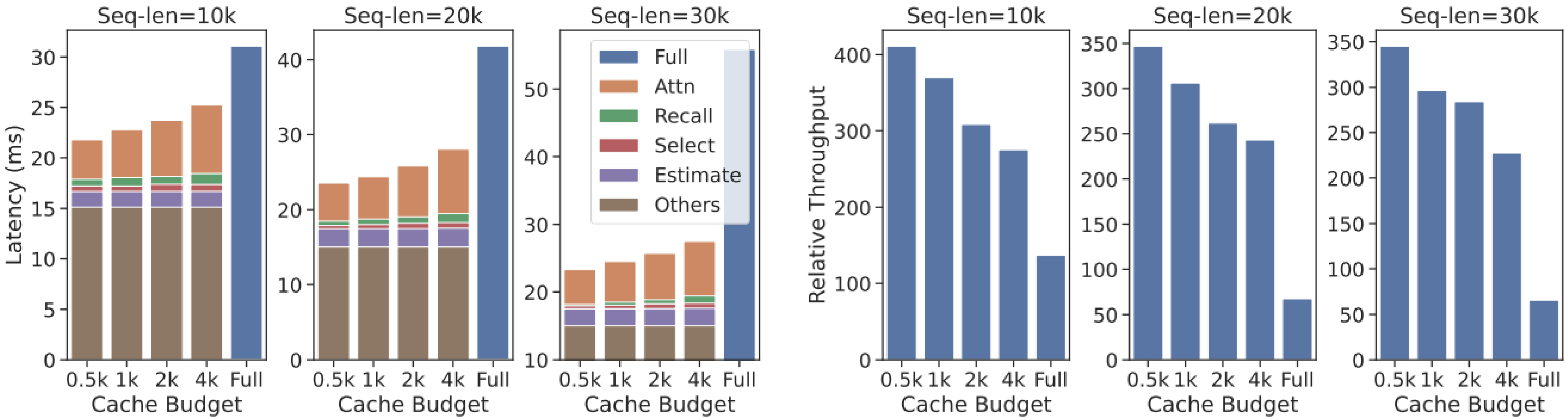
Part of Evaluation Results on Long-Bench

- ◇ArkVale can surpass all baselines with different datasets and cache-budgets.
- ◇ArkVale can approach or even surpass “Origin”.
- ◇ArkVale-16 (page-size=16) usually outperforms ArkVale-32 (page-size=32).



Performance Evaluation

- Allocate 40 GB GPU memory for KV-cache (and page digests) in A100 GPU.
- Compared to baseline, ArkVale can achieve up to 2.2x decoding speedup.
- Compared to baseline, ArkVale can achieve up to 6x decoding throughput.



(a) Latency Breakdown (batch-size=4)

(b) Throughput Comparison

Summary

- ArkVale: Efficient Generative LLM Inference with Recallable Key-Value Eviction
  - Page-based KV-cache Eviction & Recall
  - Page Summarization & Importance Estimation based on Bounding-volume
- ArkVale performs well on various long context tasks with few accuracy loss under a cache budget of 2k~4k and speeds up decoding latency by 2.2× and boosts throughput to 6× in long-context scenarios.

Scan to access our code

Our code is now open-sourced at https://github.com/pku-liang/ArkVale

Thanks for listening! E-mail us to ask follow-up questions: crz@pku.edu.cn