Can LLM attention enable efficient KV caching with frugal memory without hurting long-context reasoning?

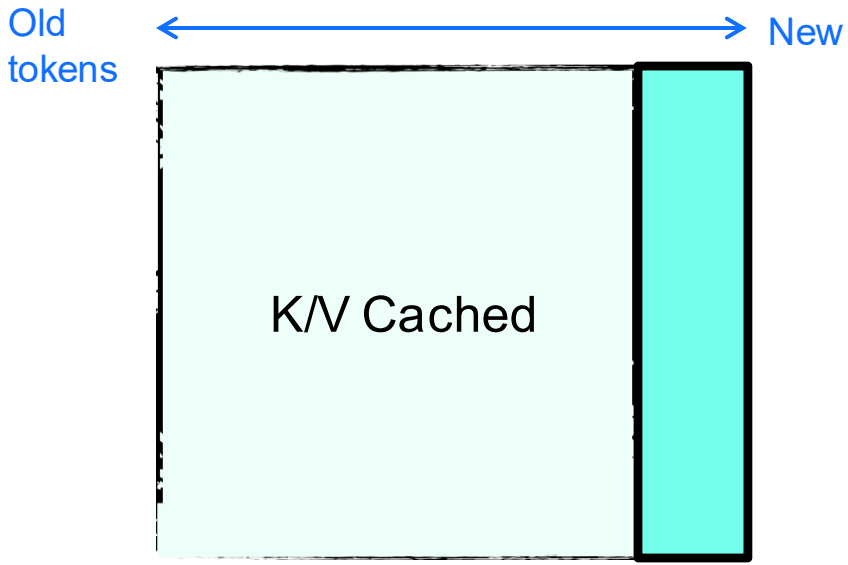
CSCI 601-771 (NLP: Self-Supervised Models)

Ernie Chu, Jonathan Lee

https://self-supervised.cs.jhu.edu/fa2025/

KV-Cache Eviction Dimension

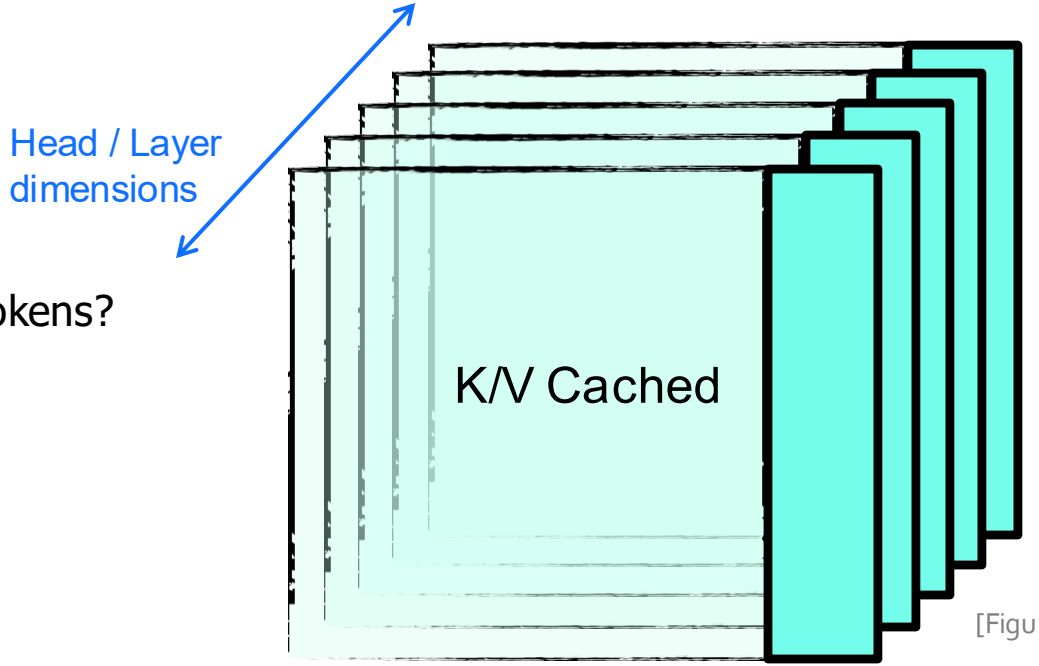
- Dropping irrelevant tokens. But how to determine the relevancy?



[Figure credit: Arman Cohan]

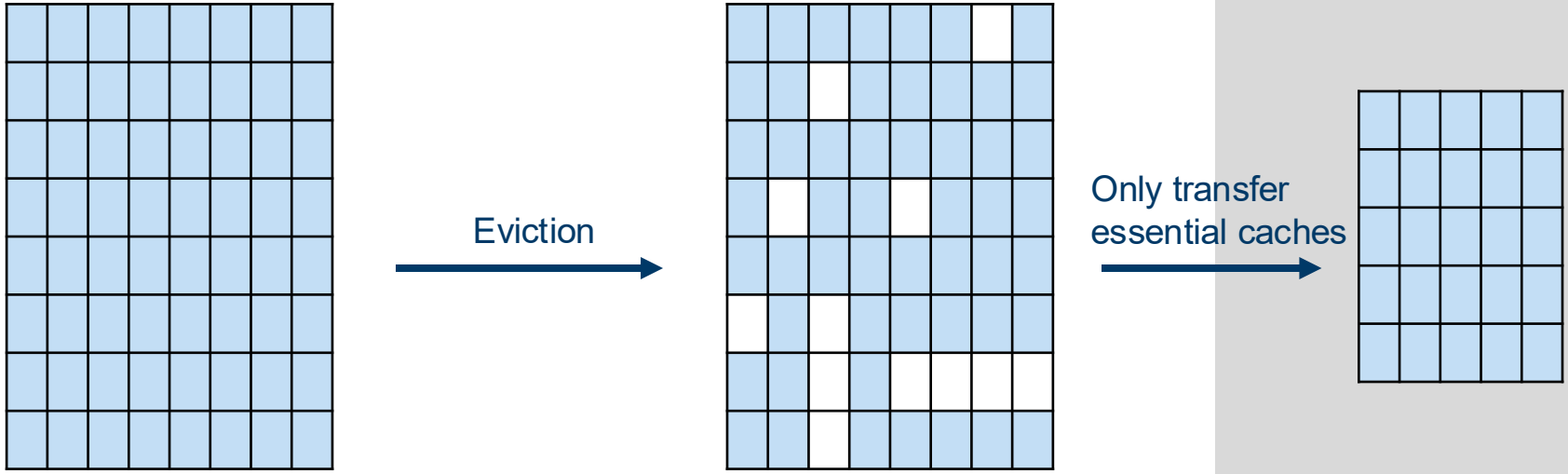
KV-Cache Eviction Dimension

- Eviction in other dimensions?
- Evict non-adjacent tokens?



[Figure credit: Arman Cohan]

KV-Cache Eviction



Cheap CPU memory / disk

Costly GPU memory

Analogy to Software Engineering

- Static
  - Code analysis
- Dynamic
  - Runtime state analysis
- Redesigned
  - Rewriting the code

KV-Cache Eviction Types

- Static - prefill stage
  - Evict KV cache based on the initial prompts (contexts)
- Dynamic - decode stage
  - Evict KV cache based on runtime tokens
- Redesigned - train stage
  - Restricting attending position by finetuning (e.g., use local attn in training)

KV-Cache Eviction Types

- Static - prefill stage
  - Evict KV cache based on the initial prompts (contexts)
  - Pros: minimal latency in runtime
  - Cons: cannot handle future tokens (generation, follow-up questions)
- Dynamic - decode stage
  - Evict KV cache based on runtime tokens
- Redesigned - train stage
  - Restricting attending position by finetuning (e.g., use local attn in training)

KV-Cache Eviction Types

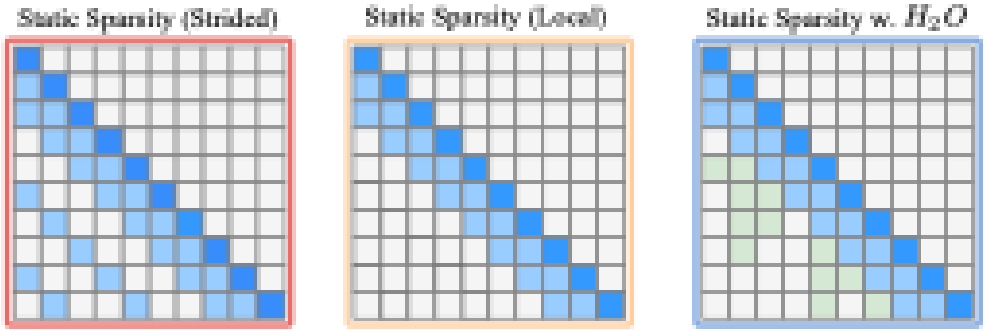
- Static - prefill stage
  - Evict KV cache based on the initial prompts (contexts)
  - Pros: minimal latency in runtime
  - Cons: cannot handle future tokens (generation, follow-up questions)
- Dynamic - decode stage
  - Evict KV cache based on runtime tokens
  - Pros: preserve all info
  - Cons: runtime analysis overhead
- Redesigned - train stage
  - Restricting attending position by finetuning (e.g., use local attn in training)

KV-Cache Eviction Types

- Static - prefill stage
  - Evict KV cache based on the initial prompts (contexts)
  - Pros: minimal latency in runtime
  - Cons: cannot handle future tokens (generation, follow-up questions)
- Dynamic - decode stage
  - Evict KV cache based on runtime tokens
  - Pros: preserve all info
  - Cons: runtime analysis overhead
- Redesigned - train stage
  - Restricting attending position by finetuning (e.g., use local attn in training)
  - Pros: no discrepancy between training/inference
  - Cons: cannot generalize to different tasks and models (need FT)

Static Token Eviction

- Determine evicted tokens in prefill stage
  - Local (with sink)
  - Strided (new toks have higher resolutions)
  - Base on attention score -> H_2O, PyramidKV, KVzip



[Figure credit: Zhang et al. H2O]

In this seminar

- Static - prefill stage
  - Previous works
- Dynamic - decode stage
  - The first paper: **OmniKV**
- Redesigned - train stage
  - The second paper: **DuoAttention**

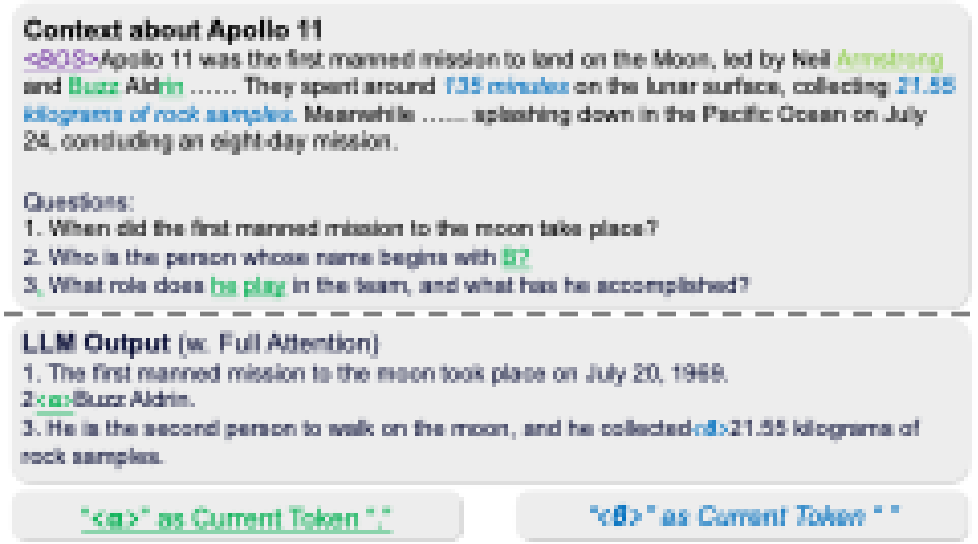
OmniKV: Dynamic Context Selection for Efficient Long-Context LLMs

(Jitai Hao, Yuke Zhu, Tian Wang, Jun Yu, Xin Xin, Bo Zheng, Zhaochun Ren, Sheng Guo. ICLR 2025)

Limitations of Static Eviction

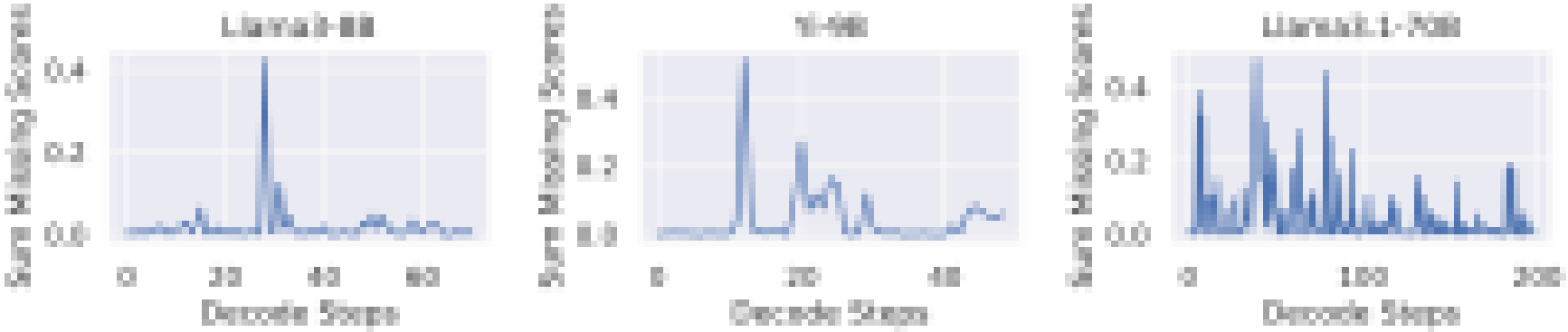
- The set of important tokens varies depending on the reasoning step

Greens: <α> attends to
Blues: <β> attends to



Limitations of Static Eviction

- Given an important token subset from static eviction like H_2O
- For each generated token, check if the top tokens it attends to are missing



- Different tokens attend to different positions

Limitations of Static Eviction

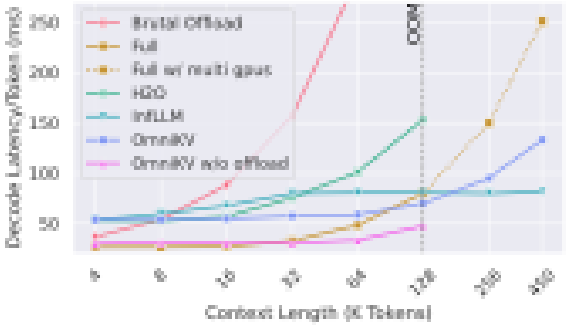
- Static attention scores cannot indicate the future importance of tokens in subsequent generation iterations

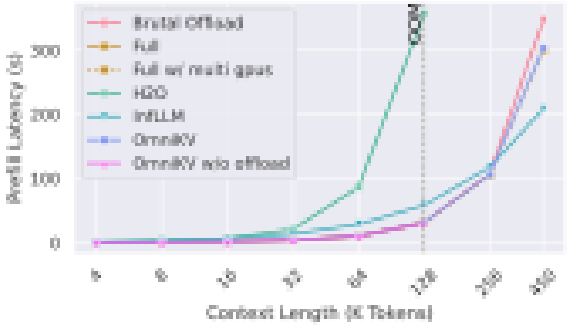
- Tokens that have low attention scores in the prefill stage may be recalled as important tokens in subsequent reasoning steps.

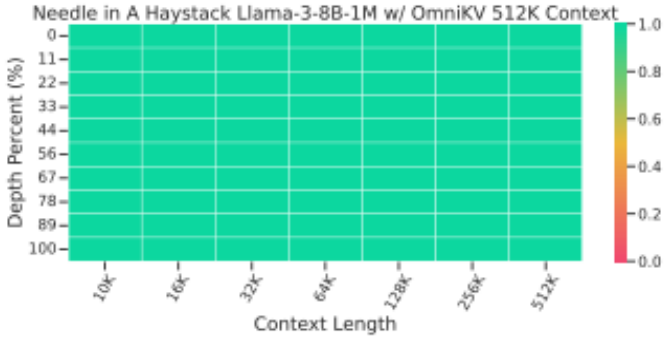
OmniKV: Retaining all KV Cache

- OmniKV retains all KV cache to ensure that performance remains unaffected, while dynamically selecting a sparse subset of tokens for attention computation in the decode stage

- Spoiler

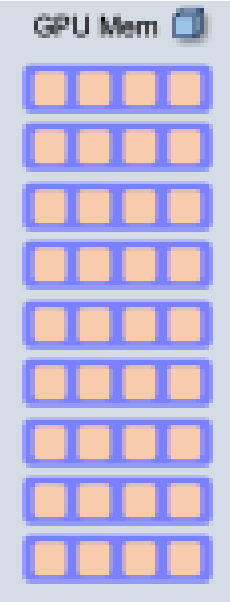






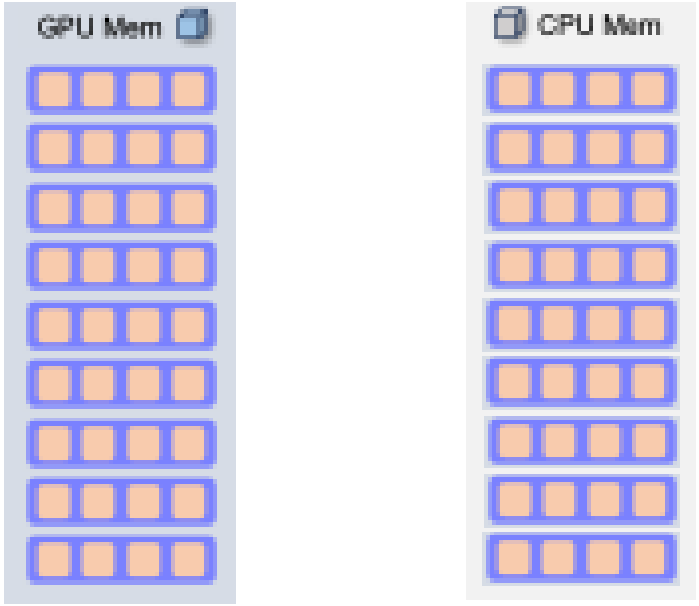
Determine Eviction per Layer

- Given a full attention transformer



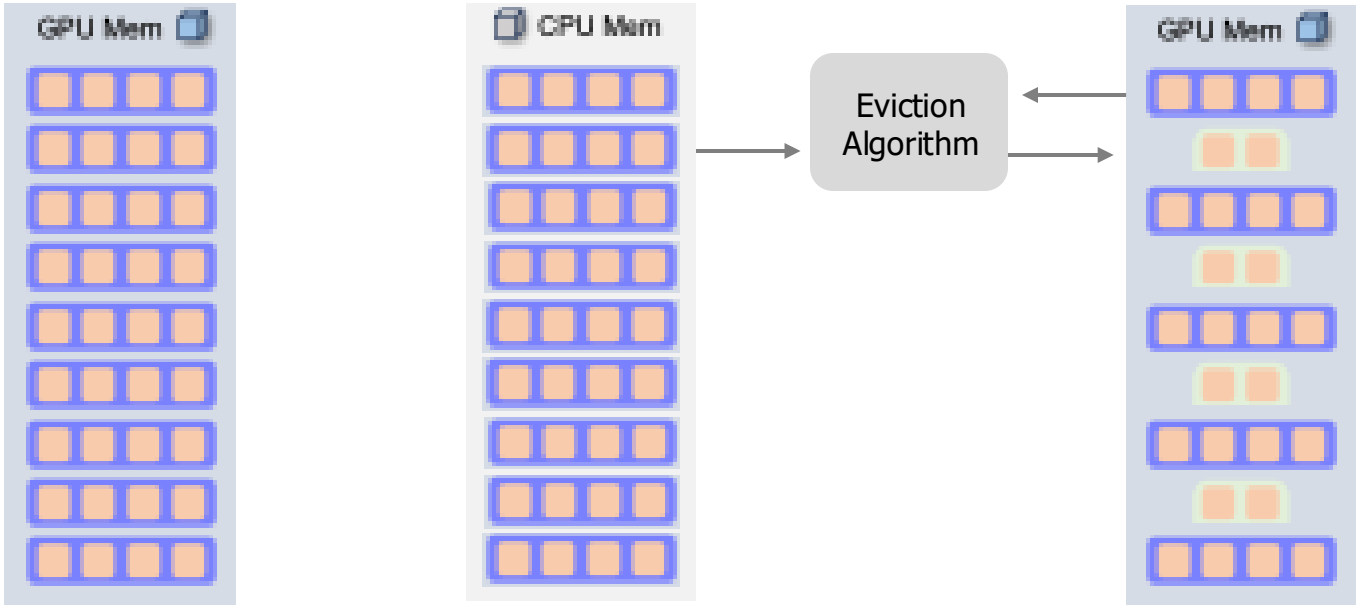
Determine Eviction per Layer

- Given a full attention transformer



Determine Eviction per Layer

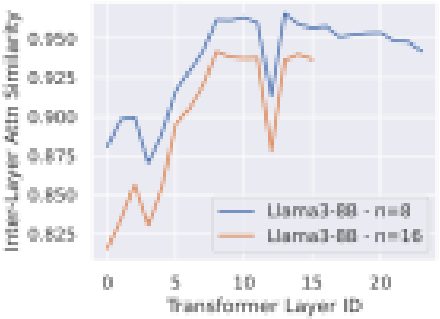
- Given a full attention transformer, assume adjacent layers have similar attn-score

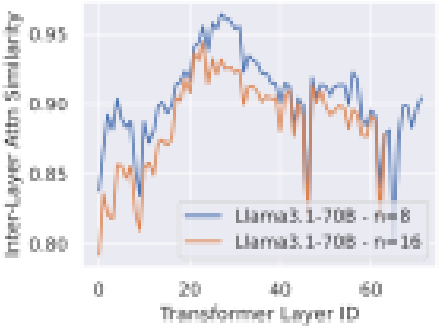


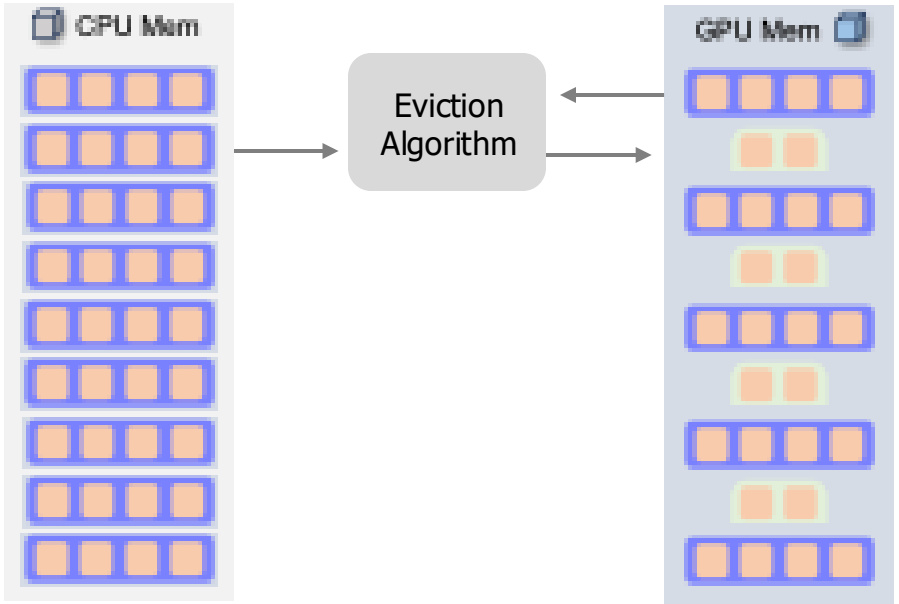
Determine Eviction per Layer

- Given a full attention transformer, assume ~~adjacent~~ nearby layers have similar attn-score

Figure 1





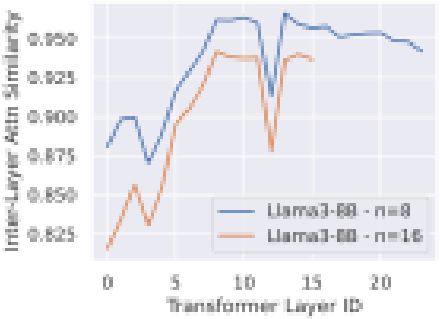


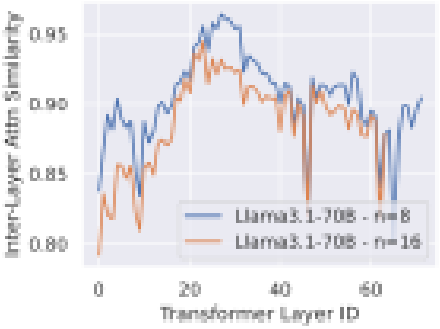
Determine Eviction per Layer

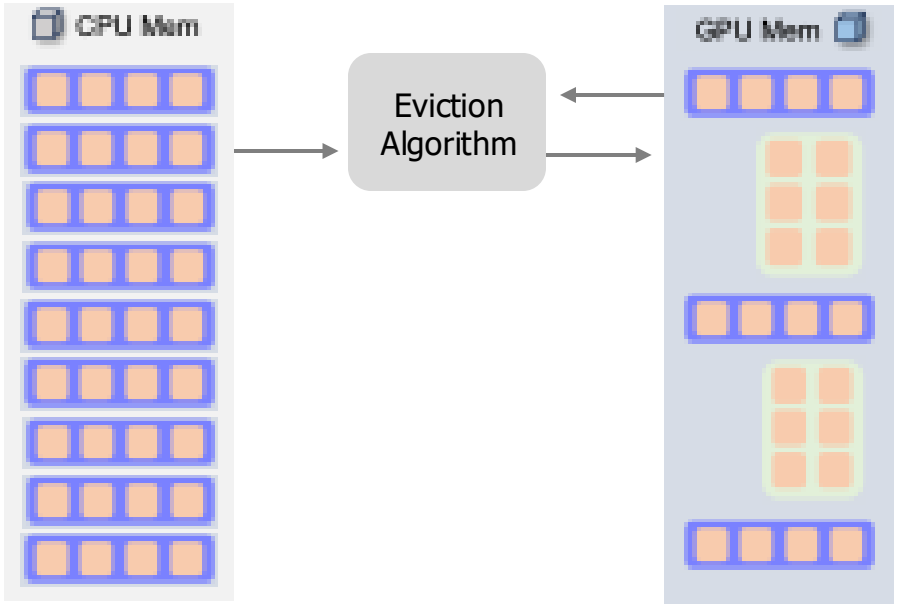
nearby

- Given a full attention transformer, assume ~~adjacent~~ layers have similar attn-score

Figure 1





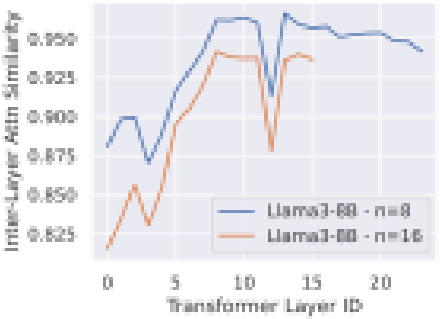


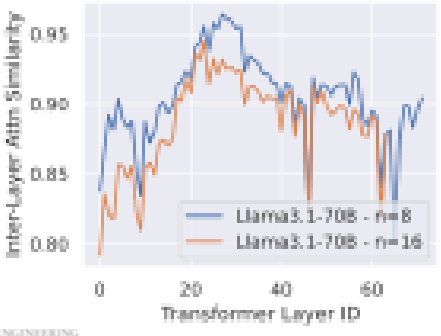
Determine Eviction per Layer

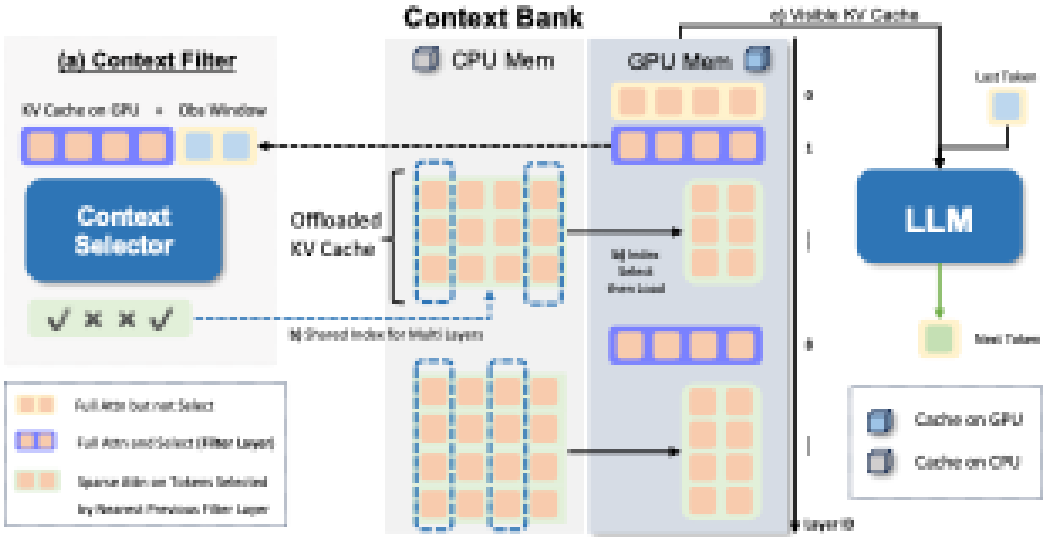
nearby

- Given a full attention transformer, assume ~~adjacent~~ layers have similar attn-score

Figure 1

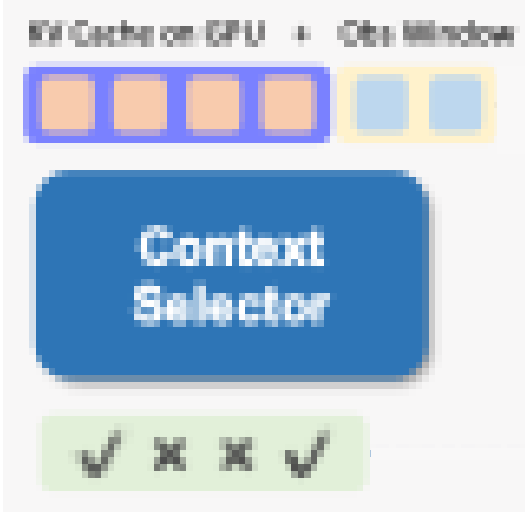






Eviction Algorithm Details

- Obs Window: the most recent tokens (~16 toks)
- Aggregation methods:
  - Uniform (avg)
  - Exponential (emphasize on new toks)
  - Last Token (window_size = 1)



Eviction Algorithm Details

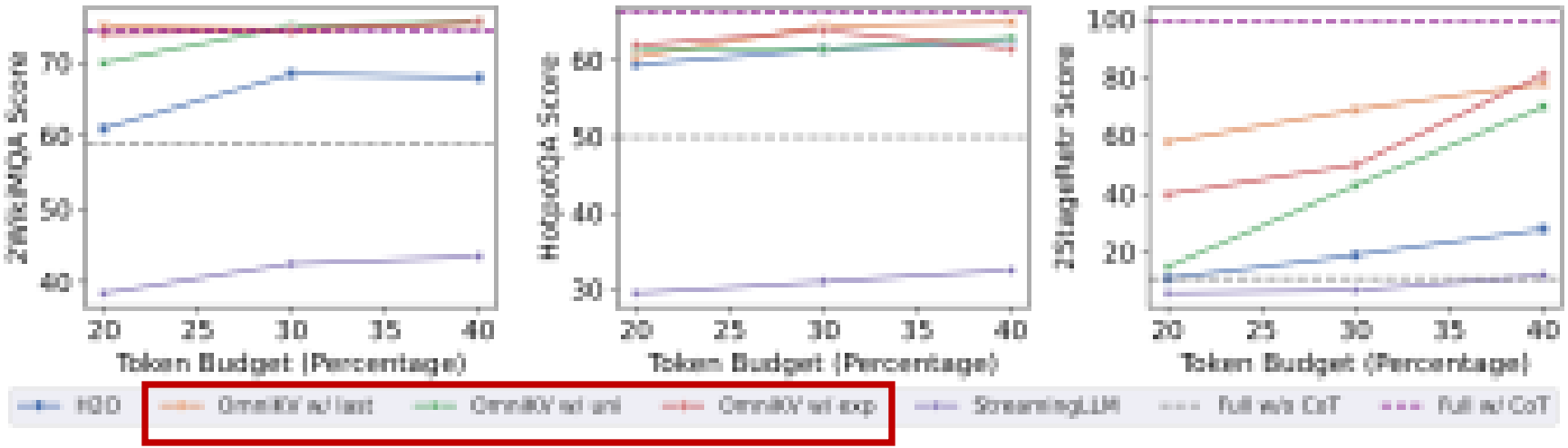


Figure 3

Eviction Algorithm Details

- **Last Token** is the most robust across various benchmarks
  - Fast, easy to implement

Methods	%Mem	Single-Doc QA	Multi-Doc QA	Summarize	Few-Shot	Synthetic	Code	Avg.
Llama-3-8B-262K	100%	29.2	22.9	24.9	65.9	43.5	48.9	39.2
H2O	30%	27.7	20.2	23.8	62.9	**42.7**	43.3	36.8
InfLLM	~30%	28.1	15.3	19.1	62.7	36.5	48.5	35.0
StreamingLLM	30%	19.3	17.3	18.6	49.7	11.7	**52.3**	28.1
OmniKV w/ uni	30%	**29.6**	**23.3**	23.7	64.0	41.5	48.4	38.4
OmniKV w/ exp	30%	29.5	22.9	23.9	**65.1**	41.5	35.1	38.3
OmniKV w/ last	30%	29.5	22.9	**24.4**	64.9	41.2	48.4	**38.5**
Yi-9B-200K	100%	28.6	33.3	20.2	71.2	31.0	67.3	41.9
H2O	30%	26.0	33.5	17.5	68.7	**31.1**	**66.6**	40.6
InfLLM	~30%	25.9	33.6	20.0	70.8	26.3	65.9	38.8
StreamingLLM	30%	12.4	11.3	13.5	56.3	2.9	60.7	26.2
OmniKV w/ uni	30%	**28.2**	**34.3**	**20.0**	71.0	29.1	65.0	41.3
OmniKV w/ exp	30%	28.1	34.0	20.0	**71.2**	30.6	65.5	**41.6**
OmniKV w/ last	30%	28.0	33.6	19.7	70.8	30.9	65.3	41.4
Llama-3.1-70B	100%	42.2	44.8	25.5	68.9	58.0	55.7	49.2
H2O+	20%	36.6	38.9	24.1	61.2	24.5	54.0	39.9
InfLLM	~20%	39.3	36.1	18.4	62.5	41.1	39.8	39.5
StreamingLLM	20%	16.7	12.3	18.7	45.6	7.5	61.1	27.0
OmniKV w/ uni	20%	40.8	43.8	23.6	67.7	**57.7**	53.8	47.9
OmniKV w/ exp	20%	42.0	44.3	24.6	**68.4**	57.6	**55.2**	**48.7**
OmniKV w/ last	20%	**42.0**	**44.3**	**24.7**	68.2	57.6	54.8	48.6

Table 1

Latency

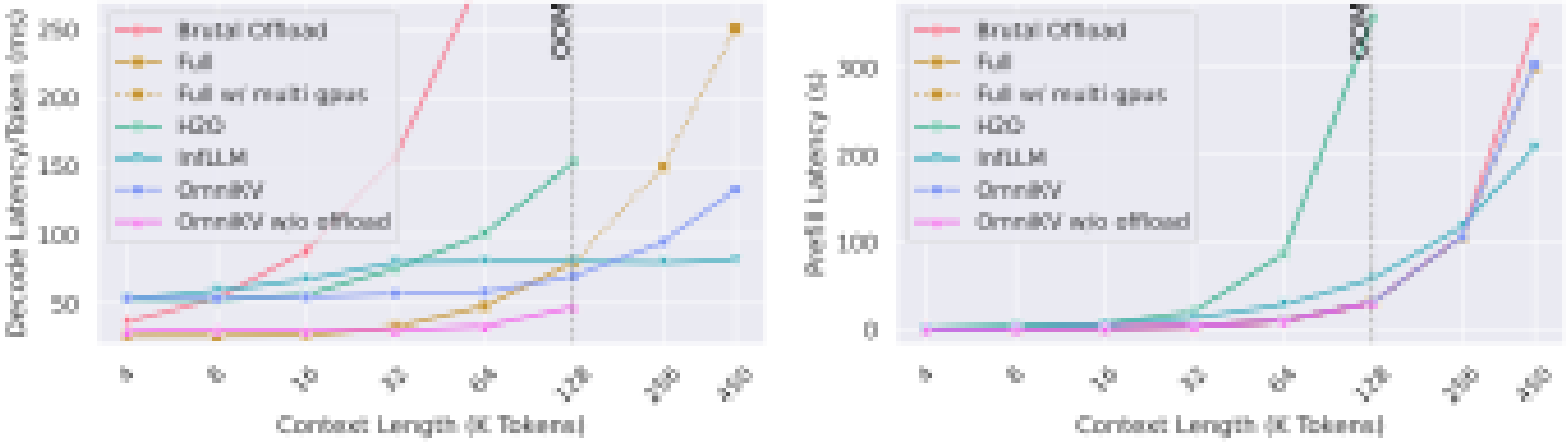


Figure 4

Latency & Performance Tradeoff

Figure 5

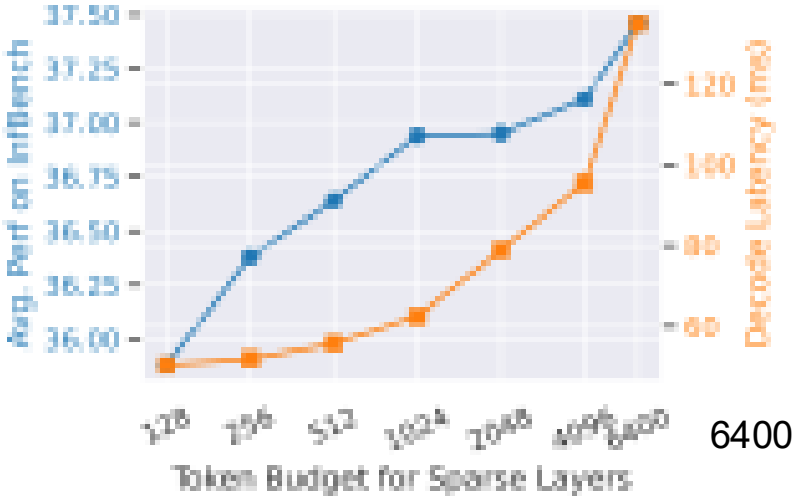


Table 2

Methods	%Mem	Avg.
LLaMA-3-8B-262K	100%	38.1
H2O+	30%	35.2
InfLLM	~30%	34.6
OmniKV w/ last	30%	**37.4**

6400 toks = 30% of full attn

Specialized Dataset for Runtime Deps

- Existing datasets may not require runtime dependency
  - Models can "guess" the important tokens using prior knowledge/common sense
- 2StageRetr(ieve)
  - Similar to NIAH, but the key is computed in the runtime

2StageRetr Example

Lets play a game. You have a dict and a mathematical addition equation. The keys of a dictionary can be any number. You need to find the corresponding key value in the dictionary after performing the addition and output the value corresponding to that key. The Dict is {0: lime, 1: yellow, 2: red, 3: black, 4: brown, ... 17: brown, 18: maroon, 19: teal, 20: red, ... 28: brown, 29: violet}
The equation is 8 + 10 = ? Answer the corresponding color based on the addition result.

Example output: Since 8 + 10 = 18, the corresponding color is "maroon".

2StageRetr Results

Figure 3

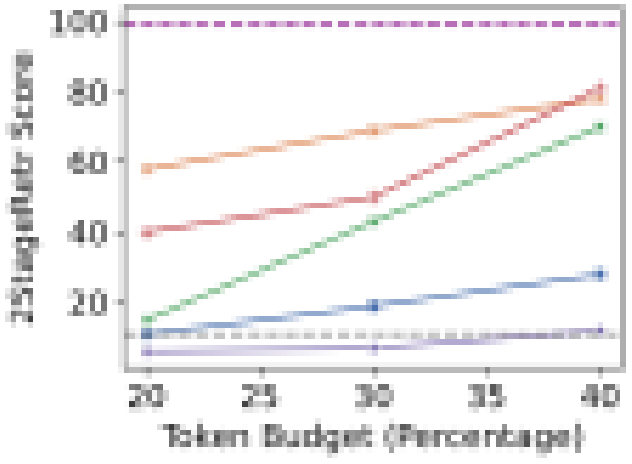
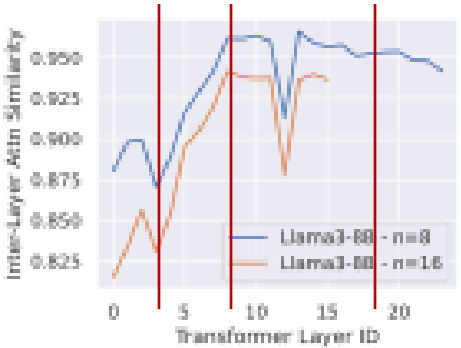


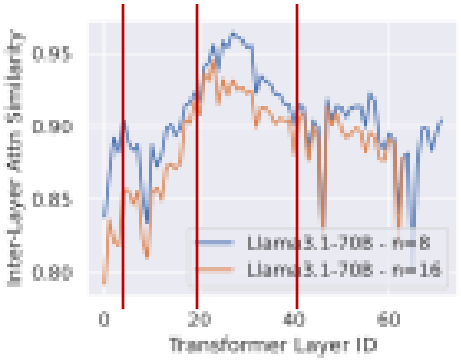
Table 6

Model	CoT	%Mem	2StageRetr
Yi-9B-200K	✗	100.0%	-
Yi-9B-200K	✓	100.0%	36.0
H2O	✓	30.0%	13.0
H2O	✓	40.0%	19.0
H2O	✓	50.0%	26.0
OmniKV	✓	30.0%	14.0
OmniKV	✓	40.0%	32.0
OmniKV	✓	50.0%	31.0

Choices of Full-Attn Layers (Filter Layers)



Llama3-8B
{2, 8, 18}



Llama3.1-70B
{4, 19, 41}

Table 8: Performance on LongBench of different filter layers settings.

Filter Layers	Single-Doc QA	Multi-Doc QA	Summarization	Few-Shot	Synthetic	Code	Avg.
2,4,18	27.0	19.9	21.4	58.1	42.8	46.0	35.8
2,5,18	29.3	22.8	23.4	63.4	42.5	47.1	38.1
2,7,18	29.3	22.1	23.6	65.2	41.8	47.2	38.2
2,8,18	29.6	22.9	**24.4**	65.0	41.3	**48.4**	38.6
2,9,18	28.6	22.5	24.1	65.1	41.8	47.4	38.2
2,10,18	30.2	**23.0**	24.0	65.1	42.8	47.5	38.8
2,11,18	**31.0**	22.7	23.9	**65.4**	**43.0**	48.2	**39.0**
2,12,18	29.8	21.2	23.0	63.2	42.1	47.8	37.8
2,13,18	30.1	22.5	23.7	65.4	42.5	48.2	38.7

Takeaways

- Leverage redundancy in per-layer attention scores to identify the most informative subset of the KV cache.
  - by exploiting Inter-Layer Attention Similarity, allowing selected "filter" layers to identify a shared, sparse index of crucial tokens that is propagated to non-filter layers, enabling efficient KV cache retrieval and acceleration during decoding.

Takeaways

- Leverage redundancy in per-layer attention scores to identify the most informative subset of the KV cache.
  - by exploiting Inter-Layer Attention Similarity, allowing selected "filter" layers to identify a shared, sparse index of crucial tokens that is propagated to non-filter layers, enabling efficient KV cache retrieval and acceleration during decoding.
- My comments:
  - An in-depth analysis on finding the optimal filter layers would be nice
    - Lossy compression theory (rate-distortion)
    - Analogy to video compression (I-frame = filter layers, what about P, B?)

Takeaways

- Leverage redundancy in per-layer attention scores to identify the most informative subset of the KV cache.
  - by exploiting Inter-Layer Attention Similarity, allowing selected "filter" layers to identify a shared, sparse index of crucial tokens that is propagated to non-filter layers, enabling efficient KV cache retrieval and acceleration during decoding.
- My comments:
  - An in-depth analysis on finding the optimal filter layers would be nice
    - Lossy compression theory (rate-distortion)
    - Analogy to video compression (I-frame = filter layers, what about P, B?)
  - A potential new direction
    - Given an LLM, find k filter layers that give the lowest distortion in attn-score

Takeaways

- Leverage redundancy in per-layer attention scores to identify the most informative subset of the KV cache.
  - by exploiting Inter-Layer Attention Similarity, allowing selected "filter" layers to identify a shared, sparse index of crucial tokens that is propagated to non-filter layers, enabling efficient KV cache retrieval and acceleration during decoding.
- My comments:
  - An in-depth analysis on finding the optimal filter layers would be nice
    - Lossy compression theory (rate-distortion)
    - Analogy to video compression (I-frame = filter layers, what about P, B?)
  - A potential new direction
    - Given an LLM, find k filter layers that give the lowest distortion in attn-score
  - More Information Retrieval (IR) methods other than full attention in filter layers

In this seminar

- Static - prefill stage
  - Previous works
- Dynamic - decode stage
  - The first paper: **OmniKV**
- Redesigned - train stage
  - The second paper: **DuoAttention**

DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads

(Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, Song Han. ICLR 2025)

Motivation – Long Context LLMs

- Long-context inference in LLM is challenging because the KV cache of all previous tokens grows linearly with context length and the attention mechanism's cost grows quadratically
  - For the Llama-3-8B model, storing FP16 KV-cache for ~1 million tokens would require ~137 GB of memory – beyond a single 80GB GPU.
- Key insight: **not all attention heads** are equally important for long-context dependency; some heads (**retrieval**) attend globally and need full KV cache, others (**streaming**) mostly focus on recent tokens or sinks and can use a light cache.

Key Observation

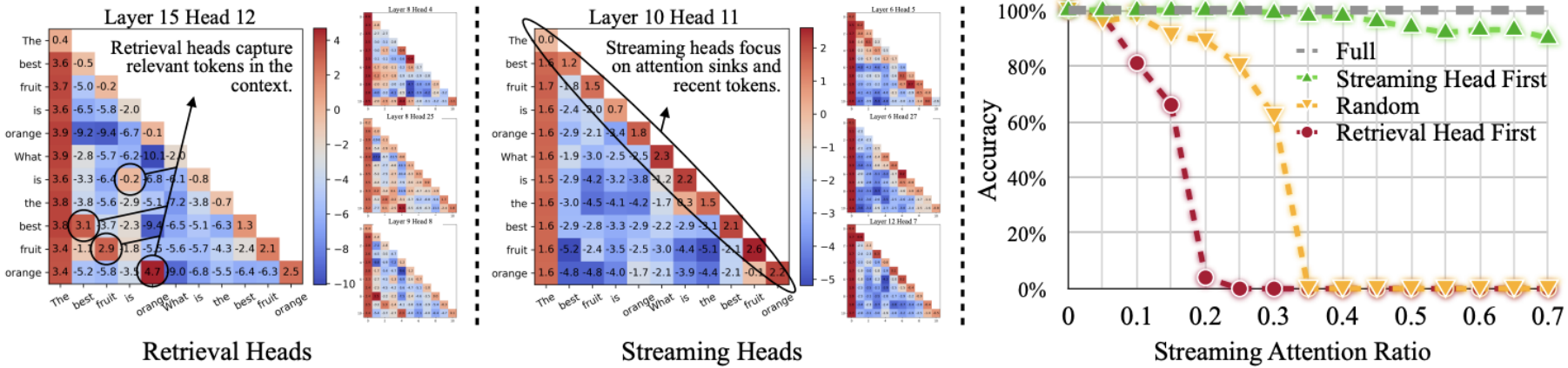
Retrieval Heads

- Small fraction of heads
- Require full attention across all tokens
- Crucial for long context retrieval
- Significantly alter model outputs when restricted to recent tokens and attention sinks.

Stream Heads

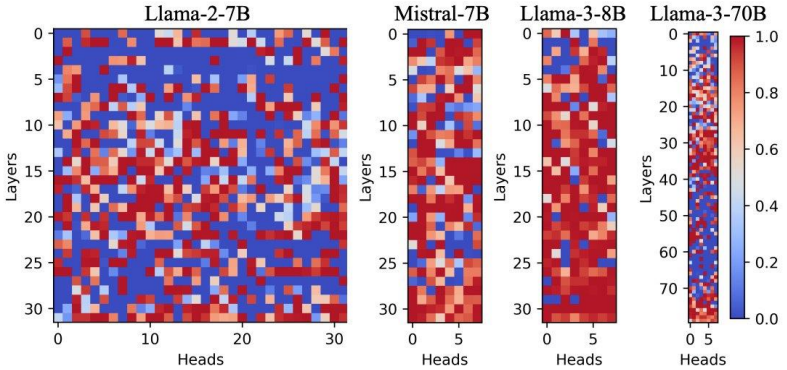
- Majority of heads
- Recent tokens and sinks

Sentence: "The best fruit is orange. What is the best fruit? Orange."

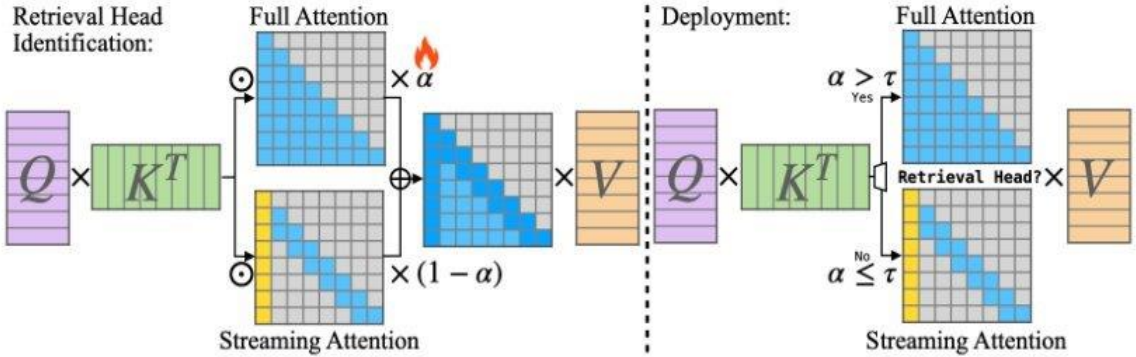


Motivation to Big Picture

- Training phase:
  - Freeze the model and **learn one scalar gate α per attention head**
  - These gates represent how much each head depends on long-range context.
- Deployment/ Inference:
  - Heads with **α > τ** → **Retrieval heads** (keep full KV cache).
  - Heads with **α ≤ τ** → **Streaming heads** (keep only recent + sink tokens).



Red = Retrieval-heavy heads, Blue = Local heads.



How the Model Learns Head Importance (α-Gates)

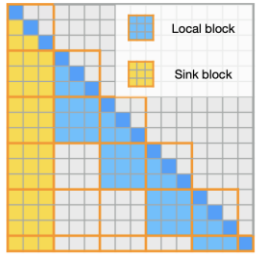
- Optimization-based head gating using **passkey retrieval dataset.**
- Step 1. Start with all α = 1 (every head treated equally). $\alpha_{i,j} \in [0,1]$
- Step 2. During training, blend full and truncated attention for each head:

$$\texttt{attn}_{i,j} = \alpha_{i,j} \cdot \texttt{full\_attn} + (1 - \alpha_{i,j}) \cdot \texttt{streaming\_attn}$$

$$\texttt{full\_attn} = \text{softmax}(\boldsymbol{QK}^T \odot \boldsymbol{M}_{\text{causal}})\boldsymbol{V}, \quad \texttt{streaming\_attn} = \text{softmax}(\boldsymbol{QK}^T \odot \boldsymbol{M}_{\text{streaming}})\boldsymbol{V}$$

  α ↑ ⇒ relies on full attention; α ↓ ⇒ prefers local attention.

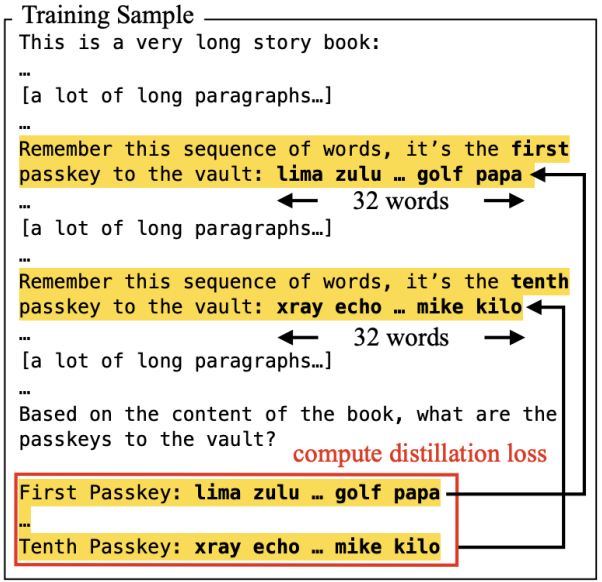
- Step 3. Adjust α so that the model with truncated cache can imitate the full-cache output.



Streaming Mask
(Block Granularity)

Synthetic Passkey Dataset

- Generate **long sequences (hundreds K – millions of tokens)**.
- Insert hidden "passkey" strings at random deep positions.
- Ask the model to output the passkeys at the end.
- **Training signal:** heads that recover the key get higher α → retrieval heads.



The distillation loss compares teacher (full cache) and student (truncated cache + α) predictions on this task.

Loss Function

- Compare hidden states of the **full model** vs the **α-gated model** on passkey data.
- Distillation loss:

$$\mathcal{L}_{\text{distill}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=T-l+1}^{T} (\boldsymbol{H}_{\text{full}}^{(i)}[j] - \boldsymbol{H}_{\text{mixed}}^{(i)}[j])^2$$

- Regularization:

$$\mathcal{L}_{\text{reg}} = \sum_{i=1}^{L} \sum_{j=1}^{H} |\alpha_{i,j}| \,.$$

- Final loss:

$$\mathcal{L} = \mathcal{L}_{\text{distill}} + \lambda \mathcal{L}_{\text{reg}}.$$

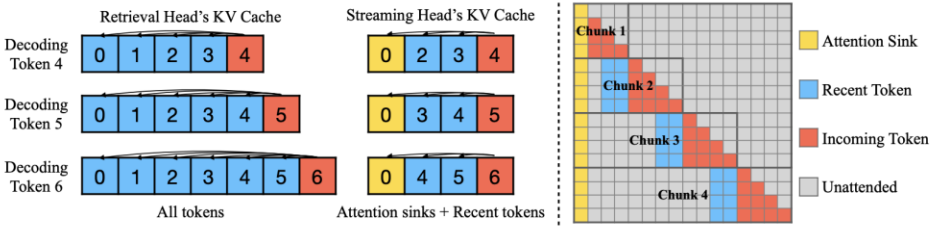
L_2 term makes α reproduce teacher behavior, L_1 term keeps only the necessary heads active.

Deployment

- **Head Assignment:**

$$\texttt{attn}_{i,j} = \begin{cases} \texttt{full\_attn} & \text{if } \alpha_{i,j} > \tau \\ \texttt{streaming\_attn} & \text{otherwise} \end{cases}$$

- **Head Reordering:**
  - Group retrieval and streaming heads contiguously in Q, K, V.
  - Enables faster slicing and decoding.
- **Dual KV-Caches:**
  - Maintain two caches per layer:
    - Retrieval → store all tokens.
    - Streaming → store attention sinks + recent tokens.
  - Combine outputs during decoding.
- **Chunked Pre-filling:**
  - Use FlashAttention-2 to fill caches chunk-by-chunk.
  - Prune old streaming KV after each chunk → lower memory (O(K)).

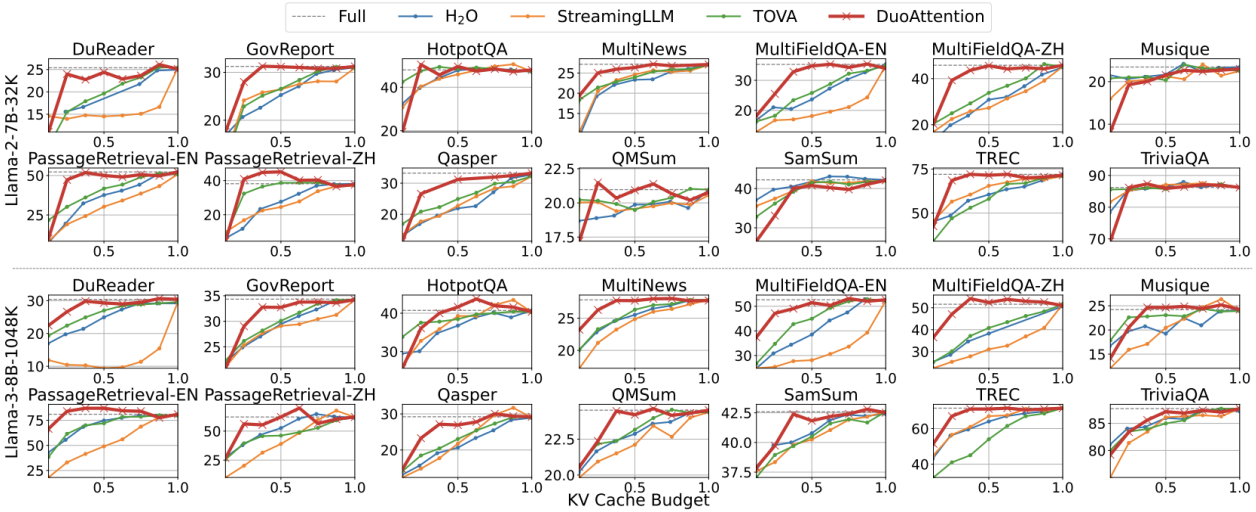


Dataset, Model, Baselines

- Long-context Benchmark: **LongBench**, **Needle-in-a-Haystack (NIAH)**
- Short-context Benchmark: **MMLU**, **MBPP**, **MT-Bench**
- Models:
  - Short-context:
    - **LLaMA-2-7B-32K-Instruct** (MHA), **LLaMA-3-[8B, 70B]-Instruct** (GQA)
  - Long-Context:
    - **LLaMA-2-7B-32K-Instruct** *(MHA)*
    - **LLaMA-3-8B-Instruct-Gradient-1048k** *(GQA)*
    - **Mistral-7B-v0.2-Instruct (**GQA)
- Baselines: **H_2O**, **TOVA**, **StreamingLLM**

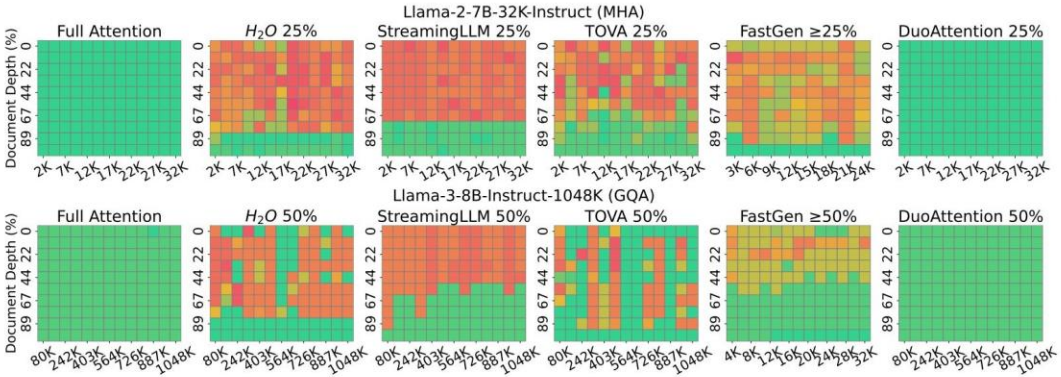
Result – Long Context (LongBench)

- **DuoAttention ≈ Full Attention** at 50% KV cache
- **Consistent lead** across retrieval & summarization tasks
- **Stable across models:** Llama-2-7B & Llama-3-8B



Result – Long Context (NIAH)

- **Near-perfect recall** even at 25–50% cache
- **Uniform attention** across document depth
- **Best retrieval consistency** among all methods

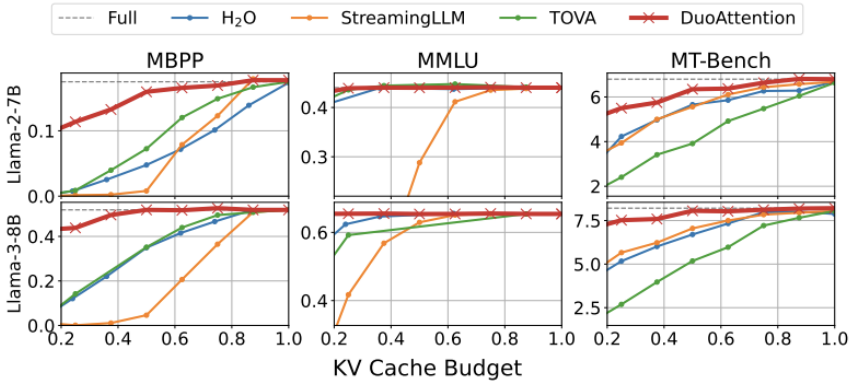


Result – Short Context

- **No drop in short tasks:** MMLU ≈ 79 %, MBPP ≈ 47 %
- **Beats baselines** on all short-context benchmarks
- Generalizes beyond long-context setting

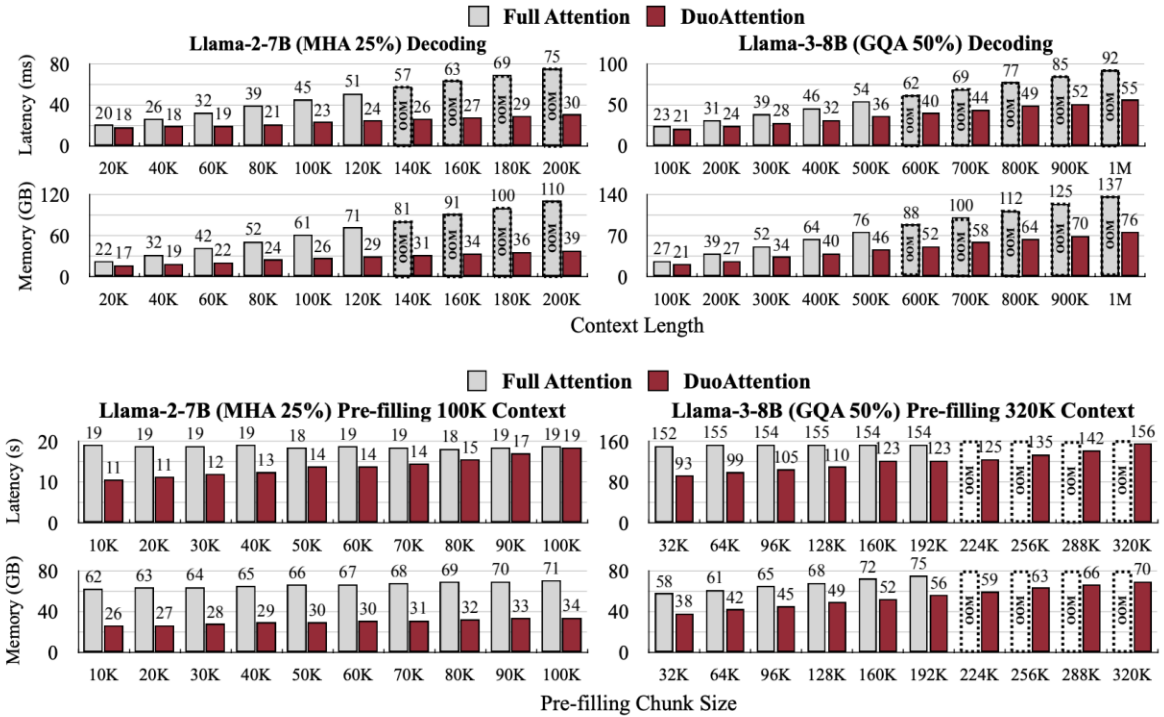
Table 1: Llama-3-70B results on short benchmarks.

	Budget	MMLU	MBPP	MT-B
Full	100%	79.38%	47.85%	8.93
H2O	50%	79.26%	32.12%	7.16
TOVA	50%	79.15%	36.09%	7.96
SLLM	50%	77.46%	5.57%	5.41
DuoAttn	50%	**79.35%**	**47.09%**	**9.14**



Efficiency Study

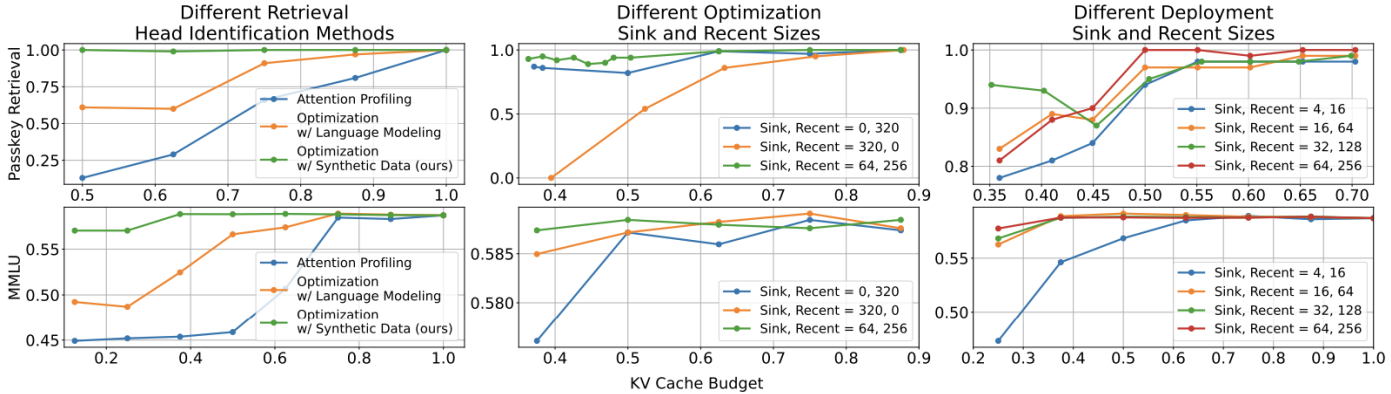
- Decoding speedup:
  - **2.18× (MHA)**, **1.50× (GQA)**.
- Prefill speedup:
  - **1.73× (MHA)**, **1.63× (GQA)**.
- Memory reduction: **2.55 × (MHA)** and **1.67 × (GQA)**.
- **Streaming head complexity:**
  - Runtime: $O(L^2) \rightarrow O(L \cdot K)$;
  - Memory: $O(L) \rightarrow O(K)$



Ablation Study

- Synthetic passkey optimization → **best retrieval & reasoning**.
- Profiling / LM loss methods fail to capture true retrieval heads.

- **Moderate windows (Sink = 32–64, Recent = 128–256)** ≈ full performance.
- Too small windows sharply degrade retrieval accuracy.



- Balanced cache (**Sink = 64, Recent = 256**) gives highest performance.
- All-sink or all-recent setups reduce generalization.

Main Takeaway

My Comments

- DuoAttention reduces KV cache and speeds up inference with minimal accuracy loss (limited to streaming heads).
- Reveals head-level specialization and differing retrieval roles in LLMs.
- A global τ threshold may overlook layer-specific importance patterns.
- Future direction: test on multimodal or dynamic tasks (image / video).

DuoAttention bridges efficiency and interpretability, but scaling to multimodal and adaptive retrieval remains open.

Discussion Questions

1. DuoAttention (Redesigned)
   a) Do specific heads specialize in *particular reasoning or retrieval tasks*??
   b) Do heads generalize *across tasks* or domains
   c) Could head classification (gating) help interpret model behavior or guide pruning?
2. OmniKV (Dynamic)
   a) Given an LLM, find k filter layers that give the lowest distortion in attn-score
   b) More Information Retrieval (IR) methods other than full attention in filter layers
3. In General
   a) Beyond layer- or head-wise eviction, what *other dimensions* can we slice?
   b) More paradigms in addition to Static, Dynamic and Redesigned?