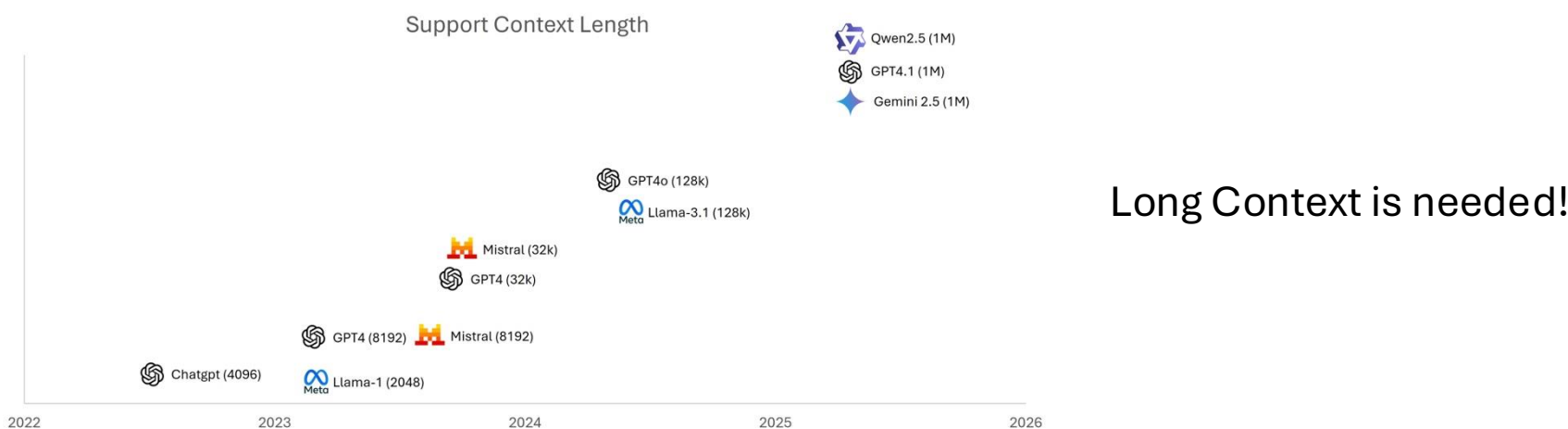


PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling

Authors: Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, Wen Xiao

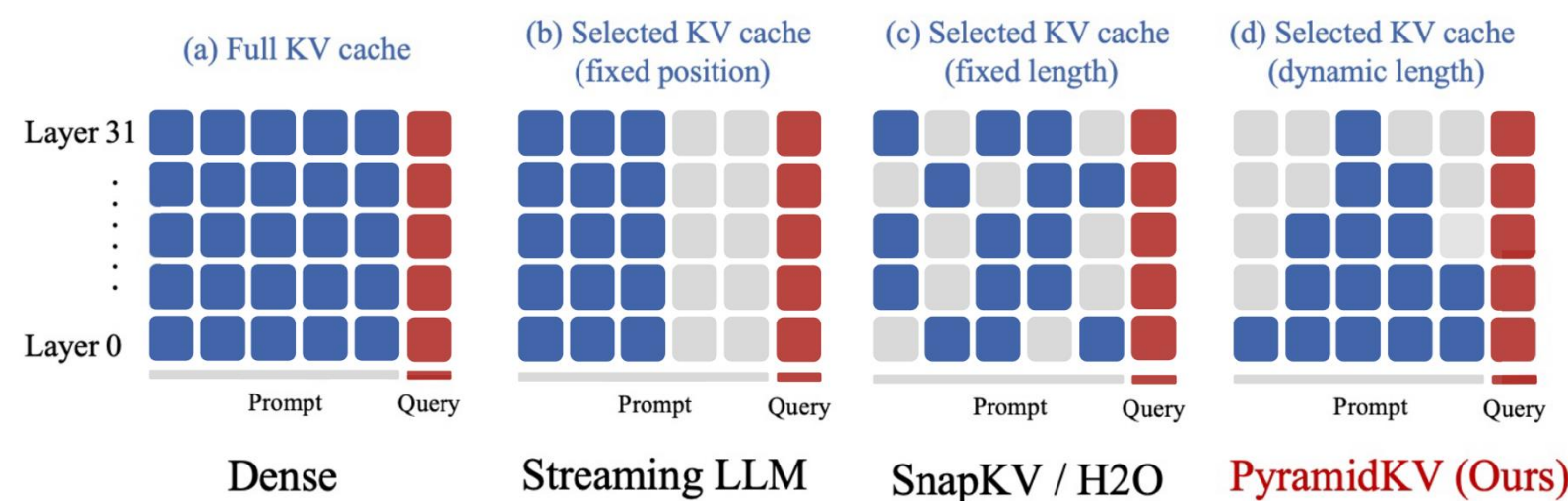


Motivation



Long Context is needed!

- Scaling LLMs to extremely long contexts naturally leads to a **significant delay** as attention needs quadratic computation
- Solution: **caching the key and value states (KV)** of previous tokens
- KV cache **grows linearly with context length × #layers**
- Becomes dominant memory cost in long-context inference
- Example:
 - LLaMA-2-7B @ 2K tokens → <1 GB KV
 - Same model @ 100K tokens → 50+ GB KV



- **StreamingLLM (SLM)**: keeps only the first few and most recent tokens — inspired by the attention sink phenomenon.
- **SnapKV/ H2O (Heavy Hitter Oracle)**: selects a fixed number of KV entries per layer based on attention scores within an instruction window.

All previous work keeps the same KV size per layer

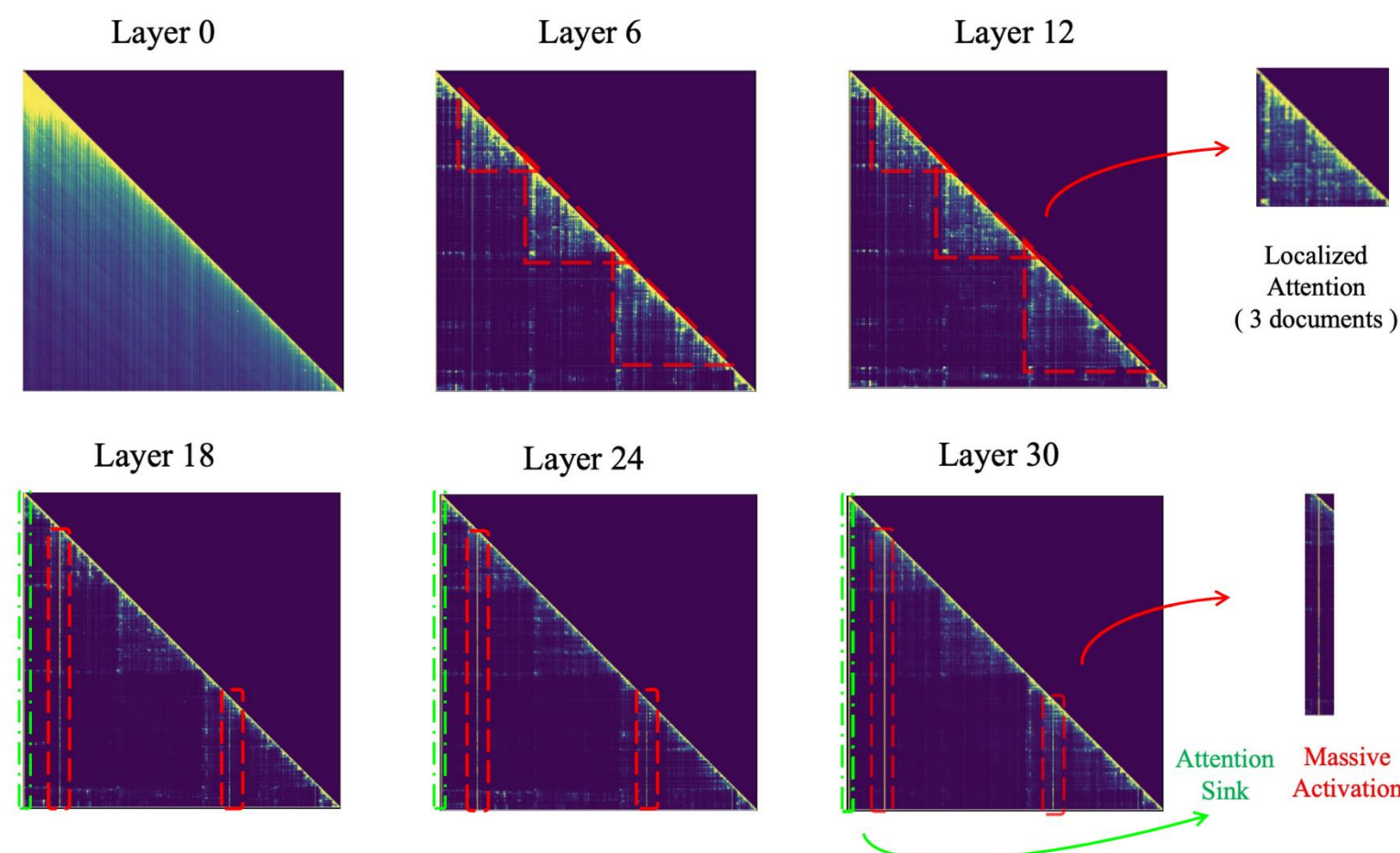
Does each layer show the same behavior?

Is it optimal to keep the same number of KV Cache in each layer?

PyramidKV: higher layers with smaller KV Cache Budget and lower layers with more KV Cache Budget

An interesting finding:

Pyramidal Information Funneling



- Figure**: Attention Heatmap of LLAMA-3 to do QA task based on multi documents
- Six Figure**: Each figure represents attention visualization for one layer. (layer=0, 6, 12, 18, 24, 30)
- Uniform Distribution in Bottom Layer**: Attention distribution in layer 0 has not a focus
- Localized Aggregation**: Each **red triangle** represents one document, Attention is aggregated to the first tokens inside the document
- Global Aggregation**: In top layers, Attention is aggregated to Attention Sink and some massive activations

PyramidKV

Information is more distributed in lower levels, and more concentrated in higher levels.

To maximize the efficiency, we keep **more kv cache in the lower levels**, while **less kv cache in the higher levels**

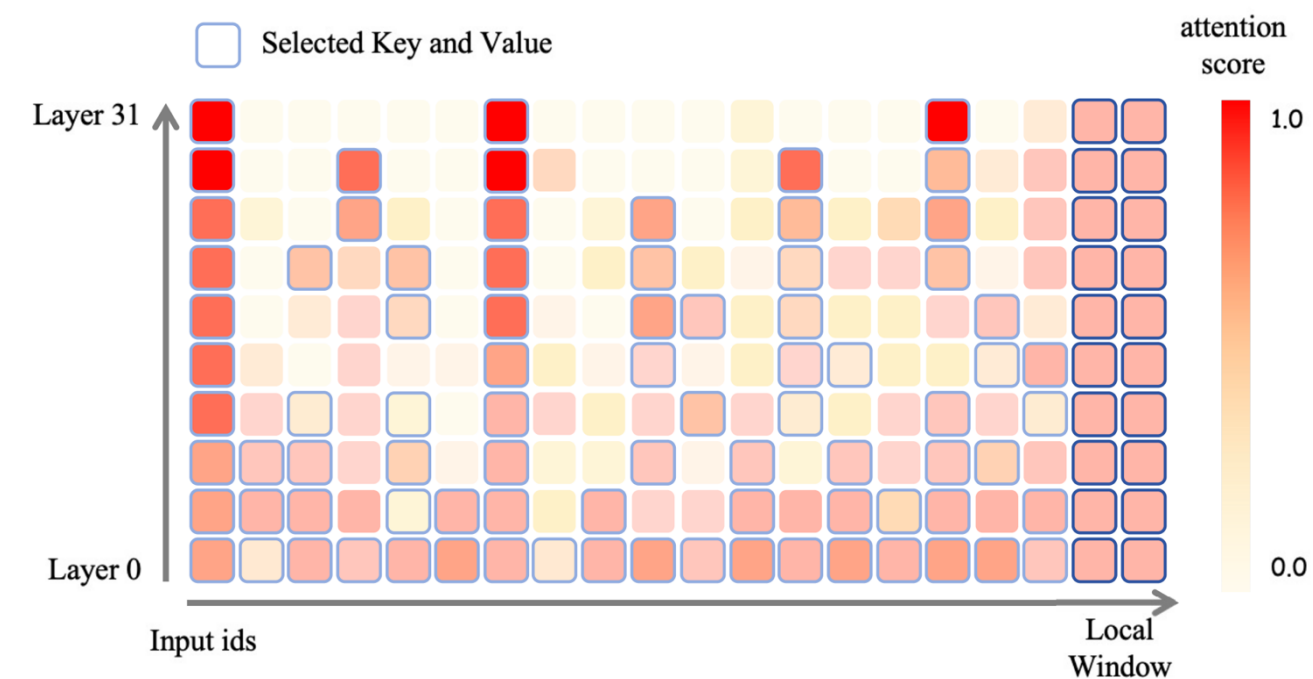
Step 1: Budget Allocation

- **Instruction tokens** - retain the KV cache for the **last α tokens** of the input across all layers
 - They are the last tokens, which are more important based on the Next Token Prediction task
 - They usually contains more task-related information, would be used to select tokens
- Budget Computation:
 - k^{total} - total KV budget (has subtracted the instruction tokens already, avg k- α per layer)
 - m – number of layers
 - β – determine the shape of the pyramid.

$$\text{bottom } k^0 = (2 \cdot \bar{k}^{total}) / m \quad \text{Two times of the avg \# tokens}$$

$$\text{top } k^{m-1} = k^{total} / (\beta \cdot m) \quad \text{Varies depending on beta, determining the shape of the pyramid}$$

$$\text{Intermediate } k^l = k^{m-1} - \frac{k^{m-1} - k^0}{m} \times l.$$



Step 2: Token Selection

- For each head, given the attention matrix A^h
- Score of each token:
 - The total attention each token receives from the instruction tokens

$$s_i^h = \sum_{j \in [n-\alpha, n]} A_{ij}^h$$

- We then select top k tokens to retain in each layer (as computed in the previous step)

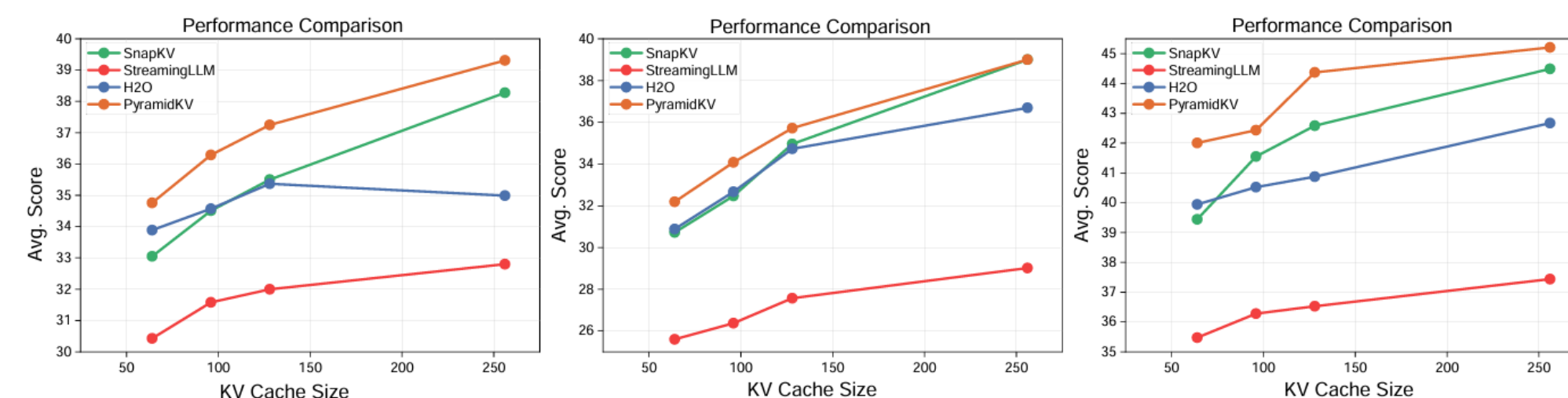
TakeAway

- Attention in lower layers is more evenly distributed, while on higher layers is more concentrated on certain tokens.
- To maximize the efficiency, we propose PyramidKV, allocating more budget on lower layers and less budget on higher layers
- PyramidKV shows a superior performance especially in extreme memory-saving scenarios

Experiment

- LongBench : 17 datasets across 6 tasks, avg length 1235 - 18409
- Model: LLaMa-3-8B-Instruct, Mistral-7B-Instruct , LLaMa-3-70B-Instruct
- Baselines: SnapKV , H2O, StreamLLM
- Setting: The same avg kv cache / layer for all method

In low-budget scenarios, our PyramidKV dominates baseline methods across budgets from 64 to 256 (avg kv cache per layer)



Selected Tasks

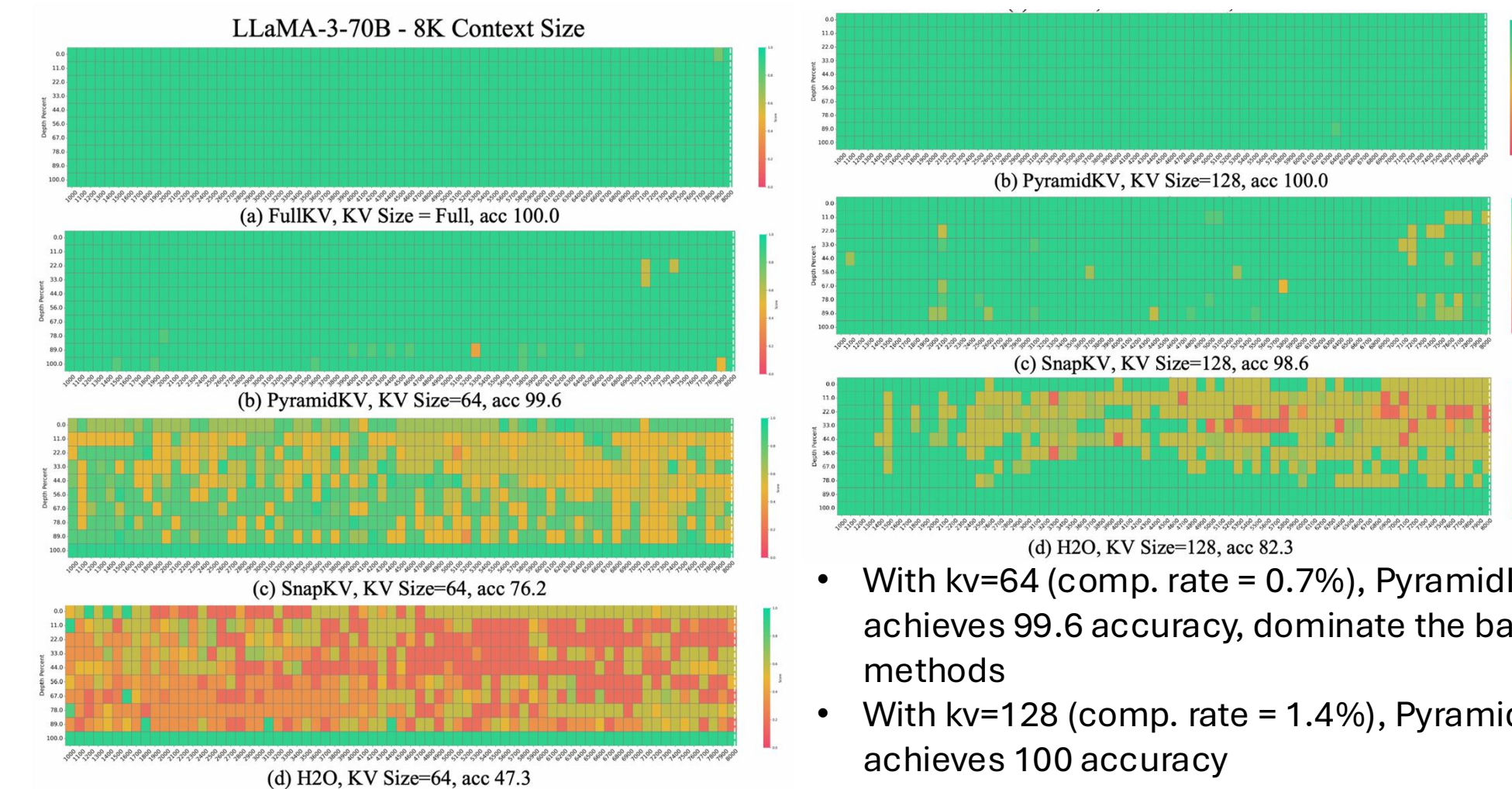
Method	Qasper	MF-en	TREC
SnapKV	19.1	34.6	39.2
StreamingLLM	16.2	27.7	37.7
H2O	20.8	31.0	39.0
PyramidKV	23.7	37.5	58.8

In extreme memory efficient scenario (kv=64), our method has a notably better performance We show the selected tasks in the table above

- Qasper: QA over NLP papers
- MF-en: QA in specific domains, e.g. gov. report, academic papers, legal docs
- TREC: ICL classification task with 50 classes

It shows that our method can effectively aggregate the information, especially in professional field, or ICL scenario, calling for more investigation in these tasks

Needle-in-a-haystack



- With kv=64 (comp. rate = 0.7%), PyramidKV achieves 99.6 accuracy, dominate the baseline methods
- With kv=128 (comp. rate = 1.4%), PyramidKV achieves 100 accuracy

Impact and Follow-up work

Implementation: PyramidKV's codebase received 1.2k stars, a unified framework for diverse KV opimization methods

Product: Novita.ai, Cloudflare and Sea AI Lab deployed PyramidKV for LLM Serving

Research: PyramidKV's paper received 130+ citations

Our Follow-up Works:

[1] Not All Heads Matter: A Head-Level KV Cache Compression Method with Integrated Retrieval and Reasoning (ICLR 2025)

[2] R-KV: Redundancy-aware KV Cache Compression for Reasoning Models (NeurIPS 2025)