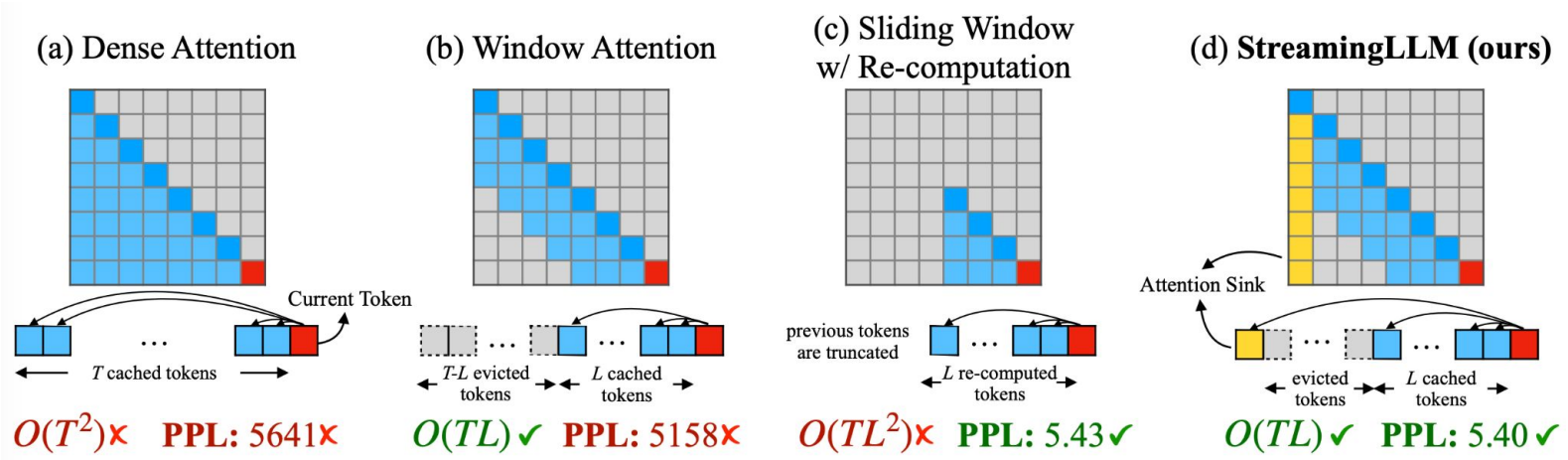
Training Free Exponential Context Extension via Cascading KV Cache

Jeffrey Willette[1], Heejun Lee[1], Youngwan Lee[12], Myeongjae Jeon[3], Sung Ju Hwang[14]
[1]KAIST [2]ETRI [3]POSTECH [4]DeepAuto.ai

Sliding Window Attention (Streaming LLM)

Recent works have tackled the problem of stabilizing sliding window attention during inference.

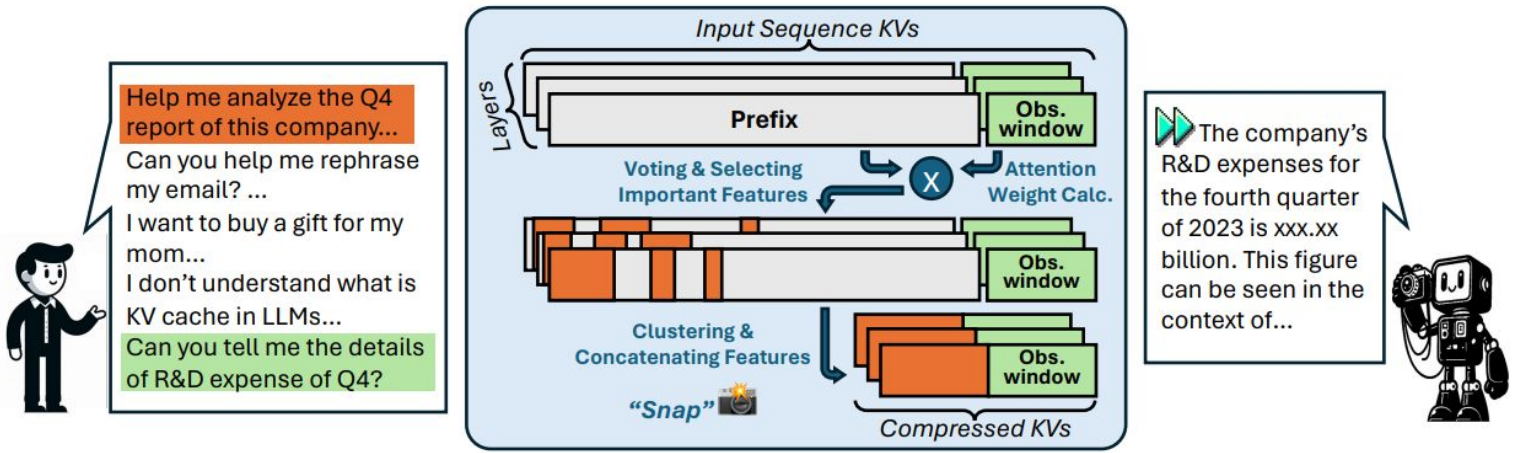


However, the token eviction strategy is naive, which limits the effectiveness of the model.

[Image] Xiao, G., Tian, Y., Chen, B., Han, S., & Lewis, M. (2023). Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

KV Caching (Recent Works)

Recent works (such as H20 [1] and SnapKV [2]) have proposed KV cache compression.

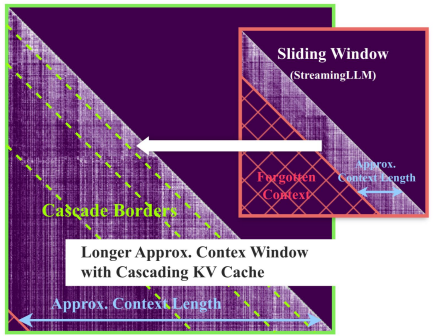


However, these methods still require quadratic complexity during the prefill stage and only prune tokens after prefill and before decoding.

[1] Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., ... & Chen, B. (2023). H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, *36*, 34661-34710.
[2] Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., ... & Chen, D. (2024). Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, *37*, 22947-22970.

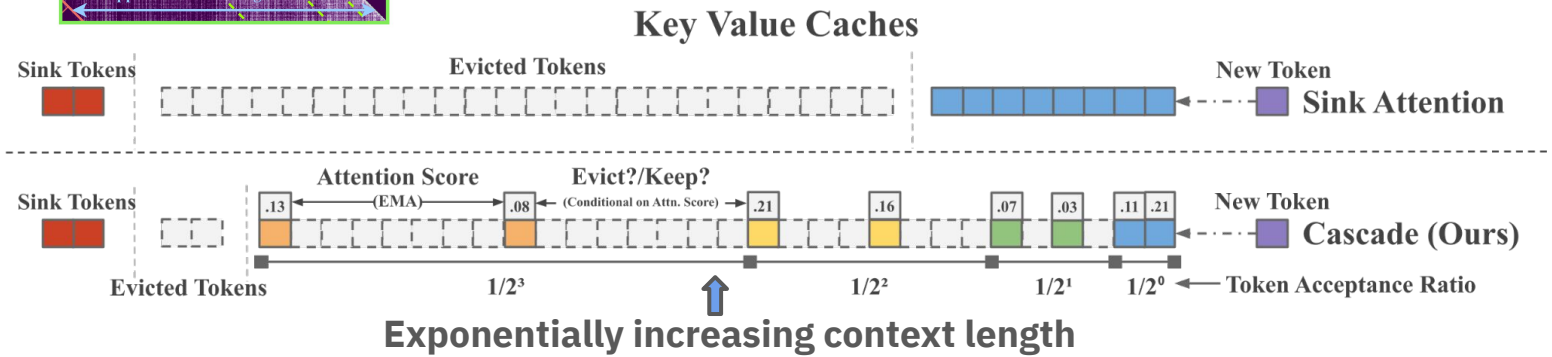
Cascading KV Cache: Exponential Window Extension

We proposed a method called Cascading KV caching to further extend the context window size of any pretrained transformer by retaining more important tokens while evicting less important ones.



The dynamic cascading KV cache:

- Evicts less relevant tokens
- Keeps important tokens for as long as possible
- Maintains important patterns in the attention matrix
- Maintains linear inference complexity
- Uses the same memory as a fixed sliding window

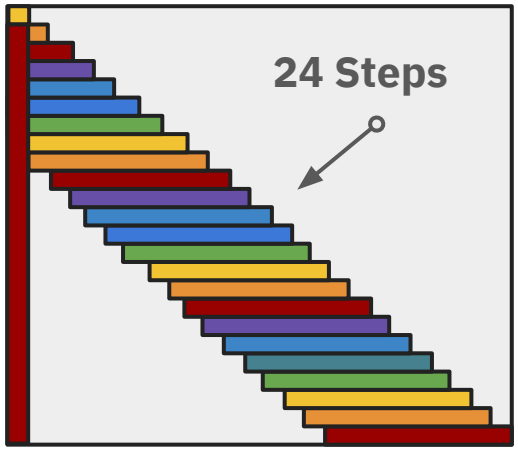


[1] Willette, J., et. al. (2024). Training-Free Exponential Extension of Sliding Window Context with Cascading KV Cache. arXiv preprint

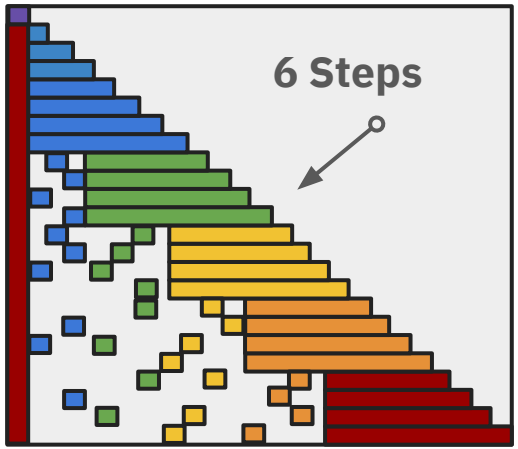
Cascading KV Cache: Batch Processing Prefill Tokens

The key to realizing actual latency improvements over other linear sliding window methods lies in the way our Cascading KV Cache processes batches of tokens at a time instead of single tokens.

Streaming LLM
(single token per step)



Cascading Cache
(N tokens per step)



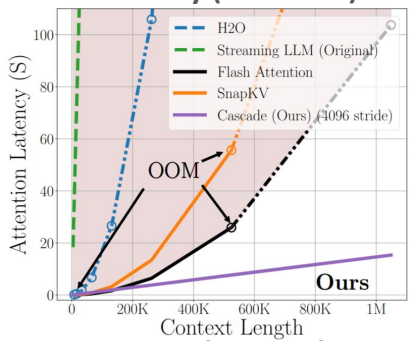
Using this method, Cascading KV Cache delivers a more than 2 order of magnitude improvement in latency over Streaming LLM.

[1] Willette, J., et. al. (2024). Training-Free Exponential Extension of Sliding Window Context with Cascading KV Cache. arXiv preprint

Cascading KV Cache: Latency and Long Context Tasks

Given the same cache size, our method shows stronger performance on long context tasks, while **scaling latency linearly, and keeping memory size constant**.

Latency (Llama 3.1)

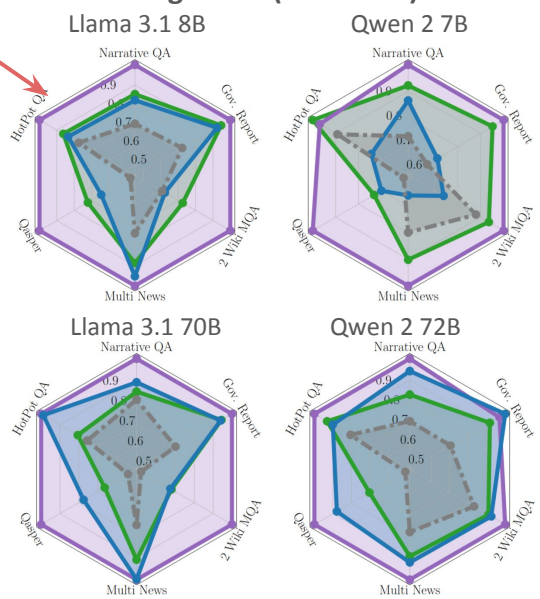


1. Outperforms static window on long context tasks

2. Linearly Scaling Latency

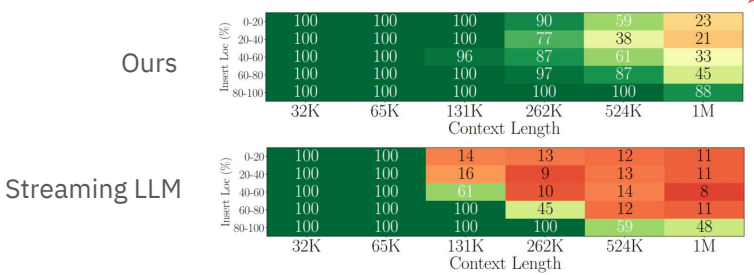
3. Better than random at 16x cache size

LongBench (Llama 3.1)



1M Token Passkey Retrieval (Llama 3.1)

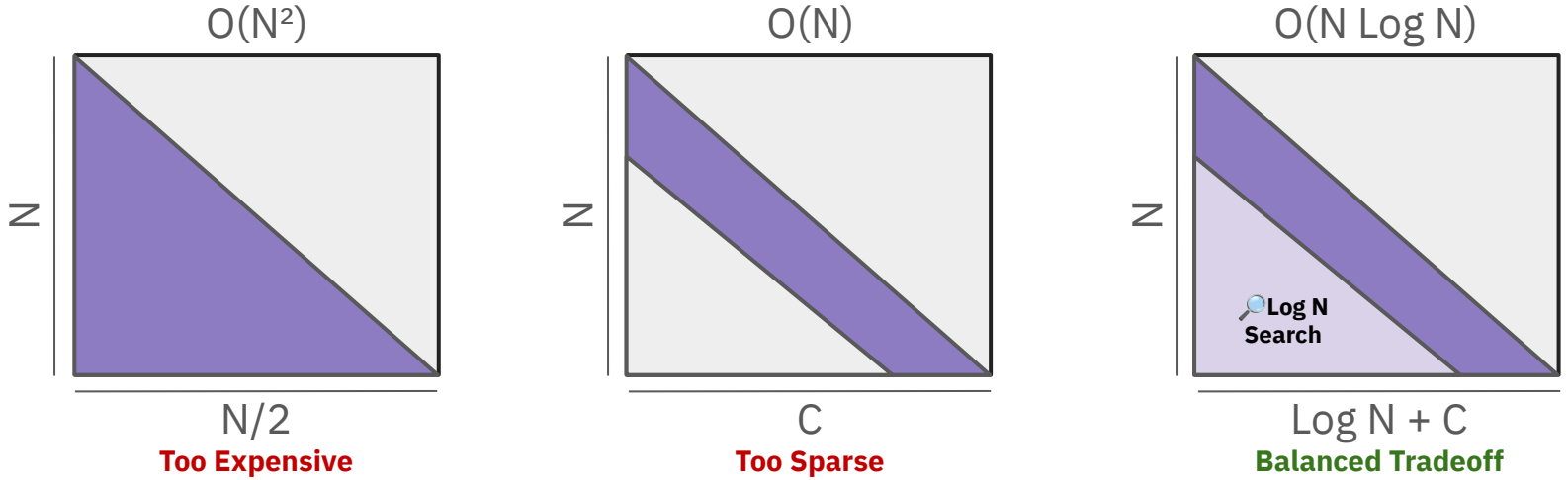
Total Cache Size = 65K, 8 Cascades



[1] Willette, J., Lee, H., Lee, Y., Jeon, M., & Hwang, S. J. (2024). Training-Free Exponential Extension of Sliding Window Context with Cascading KV Cache. arXiv preprint

Future Research: Logarithmic Token Search

Cascading KV cache must eventually evict tokens. However, if all tokens can be stored and relevant tokens can be retrieved in logarithmic time, then an overall O(N log N) transformer would result.



Devising an effective logarithmic token search would allow for practically infinite context lengths while maintaining all tokens in memory so they may retain influence over future predictions.

Thank you! We look forward to seeing you at ICLR 2025!