Efficient Sparse Decoding for Test-Time Scaling with KV Cache Disaggregation and Asynchronism

Shuqing Luo*
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC, 27599
luoshuqing5@gmail.com

Yilin Guan*
Department of Computer Science
Johns Hopkins University
Baltimore, MD, 21218
yguan29@jh.edu

Hanrui Wang
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA, 90095
wang@cs.ucla.edu

Tianlong Chen
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC, 27599
tianlong@cs.unc.edu

Abstract

While block-level sparse attention has demonstrated superiority in efficiency on large language model test-time scaling tasks through simple yet effective designs, high concurrency and prolonged decoding can severely degrade its efficiency. In this paper, we introduce a paradigm to optimize conventional sparse decoding on test-time scaling tasks. We propose a disaggregated inference architecture that parallelizes inference computation with cache management and selection to reduce decoding latency and increase model serving throughput. Based on this decoupled design, we introduce an asynchronous KV cache filtering algorithm that delivers token-level and fine-grained contextual sparsity without sacrificing model quality under the same sparsity budgets. We evaluate our method on the classical test-time reasoning benchmark against the strong baselines of block-wise sparse decoding, where our method demonstrates comparable or superior performance compared to the strong baseline of block-level sparsity.

1 Introduction

Test-time scaling (TTS) has recently emerged as one of the most powerful paradigms for enhancing LLM performance, as exemplified by the remarkable successes of GPT-o1 [15] and DeepSeek-R1 [12]. By allocating more computation during inference, TTS enables models to produce more accurate and reliable solutions, either by extending the chain-of-thought (CoT) [34] process or by generating multiple reasoning paths in parallel [40, 33]. These methods have been proven especially effective in challenging domains that demand systematic reasoning, including mathematical problem solving [24, 31, 14], program synthesis [3, 14, 12] and scientific discovery [13, 36].

However, such improvements are achieved at the cost of sharply escalating inference overhead, as extended decoding and multi-sample generations amplify the linear growth of KV-cache memory footprint. In practical LLM serving systems, it is usually the attention core [10] rather than parameter computation that emerges as the dominant bottleneck in test-time scaling tasks [26].

A common line to mitigate these issues is dynamic sparse attention, which employs dynamic rules to select and retain the most critical tokens. Compared to previous methods built on static cache

*Equal contribution. Correspondence to: luoshuqing5@gmail.com.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

filtering rules [5, 8, 38], dynamic sparsity can achieve better performance through context-aware token selection. Query-aware sparsity further sharpens this relevance by exploiting the fact that the criticality of a token in the full KV cache requires the query embedding at the current decoding step [37, 29], but this reliance introduces a sequential dependency and incurs significant latency under long CoT[34] and high concurrency.

In this work, we propose an innovative design that decouples the KV cache filtering operation from the model serving pipeline through asynchronism, first overlapping the overhead of cache filtering with inference computation to deliver minimized decoding latency and maximized serving throughput under the same sparsity budget. We develop a lightweight algorithm to regress the query of the current decoding token with a sliding query window, conducted on the cache ranks to empower token-level KV cache sparsity, which demonstrates superior performance on TTS tasks.

In summary, our contributions are:

- We design a disaggregated inference architecture that parallelizes KV selection and inference computation to reduce latency and boost throughput on TTS tasks.
- We propose a novel asynchronous KV cache selection framework with token-level sparsity to deliver high performance on TTS tasks with long CoT.
- We evaluate the performance of multiple dynamic KV cache sparsity settings on a popular TTS benchmark, where our method achieves the best performance under the same cache budget with fully overlapped cache filtering operations.

2 Related Work

2.1 LLM Reasoning & Test-Time Scaling

Test-time scaling (TTS) is capable of enhancing model reasoning performance by allocating additional inference-time computation [42], typically through sequential scaling and parallel scaling strategies.

Sequential scaling extends test-time computation through Long-CoT [34], where models such as GPT-o1 [15], DeepSeek-R1 [12], QwQ [30], Qwen3 [39], GPT-OSS [1], and LIMO [41] generate substantially longer reasoning trajectories before producing a final answer.

Parallel scaling generates multiple candidates to expand the solution space. Self-repetition [33] and multi-sample decoding [27] instantiate this idea by generating multiple trajectories from the same model. Mixture-of-models strategies coordinate generations across multiple LLMs, further amplifying diversity and coverage [7, 32]. Beyond sampling-based methods, search-based approaches (e.g., tree search, Monte Carlo Tree Search, or graph-based exploration) explicitly structure generation into expanded trajectories that probe a combinatorial space of reasoning paths [6, 40].

2.2 Dynamic Sparsity of KV Cache

A natural approach to mitigating LLM inference cost is reducing the size of the KV cache through dynamic sparsity, *i.e.*, selectively retaining only the most relevant tokens during decoding. SparQ [25] approximates scores from top query components and fetches only the top-k key–value pairs with mean-based compensation. InfiniGen [18] mitigates offloading bottlenecks by speculatively prefetching critical KV pairs of the next attention layer to the GPU while retaining the full cache pool on the CPU. FastGen [11] adaptively compresses the KV cache by profiling attention heads and discarding tokens according to their structural patterns. ShadowKV [28] retains compressed pre-RoPE key caches on GPU and offloads values, while using chunk-level approximations and accurate sparse KV selection to minimize decoding latency. Quest [29] leverages query-aware sparsity by tracking key ranges within cache pages to load KV cache pages selectively. MoBA [19] partitions the context into blocks and employs a gating mechanism to dynamically route queries.

3 Preliminaries

Sparse decoding can boost model serving throughput via reduced inference FLOPs from streamlined KV cache [2] while matching the accuracy of full attention under constrained sparsity budgets.

There have been some recent works on training-free dynamic KV cache sparsity, such as Quest [29], InfLLM [37], and Kinetics [26], along with some papers on training-dependent dynamic sparsity, such as MoBA [19]. These works typically select the cache pages [17] or blocks [10] based on the dynamically changed query embedding during inference. The criticality of a page or block is estimated by summarizing the key cache into a vector, usually through average pooling [26, 19]. While this dynamism can well preserve the performance of full attention, the overhead of the criticality estimation operation [29] has long been neglected. This operation typically suffers from three aspects:

- **High Computational Complexity**: Criticality estimation requires traversing the full context of the key cache for each attention head, leading to linear computational complexity.
- **Hardware Inefficiency**: GeMV-based criticality estimation delivers low arithmetic intensity [35], which can only be scheduled to execute on CUDA cores, leaving the GPU's higher-throughput tensor cores [20, 9] idle and severely under-utilizing the hardware.
- **Non-Parallelizable Execution**: Criticality estimation depends on the query embeddings of the current decoding token and cannot be parallelized with the inference pipeline.

4 Method

4.1 Disaggregated Cache Architecture

Our sparse inference engine is built upon a novel disaggregated cache architecture, inspired by the classic chatbot serving framework Mooncake [23]. We split the GPUs for model inference into 2 categories as `inference rank` and `cache rank`. We describe their functionalities as follows:

1. `Inference Rank`: We load the model parameters and generation requests on the `Inference Rank` for both prefilling and decoding stages. The `Inference Rank` only keeps the filtered KV cache for attention core computation, while the newly generated key & value embeddings are transmitted asynchronously to the cache rank.
2. `Cache Rank`: We preserve the full KV cache on the `Cache Rank`, which selects the critical tokens at each decoding step after enabling sparse decoding, reorganizes them into a continuous tensor, and transmits the filtered cache asynchronously to the inference rank.

4.2 Asynchronism of Cache Selection

When decoding for token t in sequential order, we do not filter the full KV cache based on the current query embedding, adhering to previous work Quest [29] or Kinetics [26]. Instead, we preserve a sliding window with a fixed size for each layer and each attention head containing $W + 1$ tokens, *i.e.*, the query embedding for token $\{t - W - 1, ..., t - 1\}$. In this way, the KV cache context can be selected asynchronously on the `Cache Rank`, overlapped with the inference computation on the `Inference Rank`, including parameter computation and attention core [10].

4.3 Token Criticality Approximation

For the sliding query window with size $W + 1$ on the `Cache Rank`, we regress the Q_{t-1} instantaneously based on $\{Q_{t-W-1}, ..., Q_{t-2}\}$ with a *Softmax*ed vector v_{t-1}, and directly apply it on $\{Q_{t-W}, ..., Q_{t-1}\}$ to approximate Q_t before decoding for token t. The closed-formed solution of v_{t-1} can be obtained explicitly by minimizing the mean square error along the head dimension.

5 Experiments

5.1 Experimental Setups

We conducted all the experiments on a single node with 8 NVIDIA H200 GPUs. We customized the HuggingFace inference code to integrate multiple sparsity strategies and evaluate the test-time reasoning performance of Qwen3-14B model [39] on the AIME24 [21] and AIME25 [22] benchmarks.

5.2 Profiling Results

We profile the equivalent FLOPs [26] at the decoding stage across multiple model inference configurations, since it can better approximate the runtime latency compared to FLOPs. The definition of this

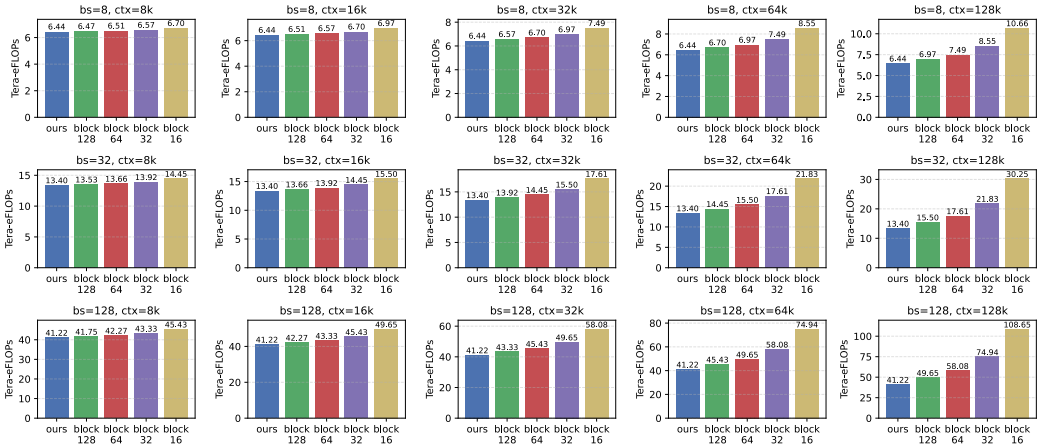

Figure 1: **Decoding Tera-eFLOPs under Varied Batch Size and Context Length.** Profiling is conducted for one decoding step with Qwen3-14B and $2k$ selected tokens. We only examine our method on the inference rank, since cache filtering operations on the cache rank have been overlapped.

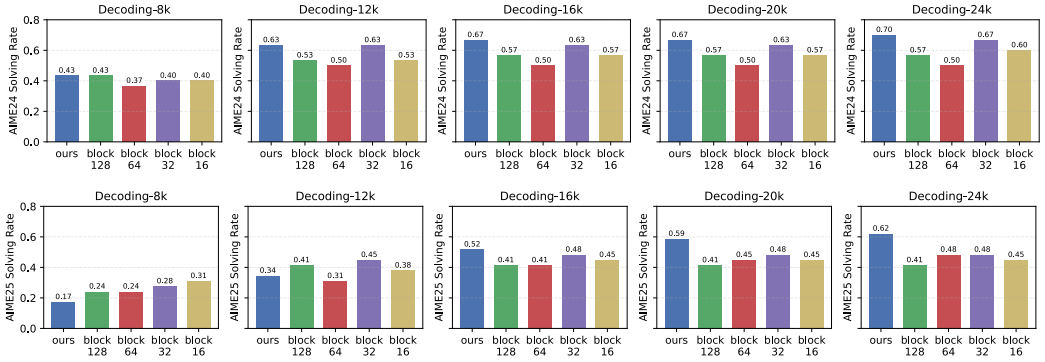

Figure 2: **Performance Comparison on AIME24&25 with Different Decoding Sparsity Strategies on Qwen3-14B**. We report the solving rate with $2k$ selected tokens and varied decoding length.

metric is formulated in Appendix A. The configurations used in our profiling vary in (i) context length to emulate CoT depth, and (ii) batch size to emulate model serving concurrency. The overhead of criticality estimation [29] got amplified under (i) long CoT reasoning, since it is proportional to the context length, and (ii) high concurrency, since it cannot be converted from GeMV to GeMM to utilize GPU tensor cores [20] for acceleration when scaling the batch size beyond 1. For page size 16, long decoding and high concurrency scenarios, the equivalent computation overhead of KV cache filtering can even be heavier than the forward pipeline itself. On the contrary, the disaggregated design of our method enables minimized eFLOPs for model inference, therefore delivering minimized latency and maximized serving throughput on TTS tasks.

5.3 Performance Comparison

We report the Pass@1 accuracy of different sparsity strategies with the same selected KV cache budget on AIME24&25 benchmarks. Benefiting from the fine-grained token-level sparsity and appropriate approximation of the current query embedding, our method can achieve comparable and better performance than the strong baseline of block-level query-aware sparsity, with fully overlapped KV cache filtering overhead to deliver optimal model serving performance.

6 Conclusions

In this paper, we introduce a novel approach for training-free fine-grained KV cache sparsity on test-time scaling tasks, tackling the long-standing efficiency tradeoff in conventional block-level query-aware sparse decoding methods. We propose a novel disaggregated inference architecture, decoupling the full KV cache from the dynamically selected KV cache on different GPU ranks

during model inference, along with a simple asynchronous cache filtering algorithm executed on the cache rank instantaneously to emulate the next query embedding during sparse decoding. Profiling experiments and evaluations on popular test-time reasoning benchmarks consistently demonstrate the superiority of our method in both model serving efficiency and model reasoning capabilities.

References

[1] Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, 2024.

[3] Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding. *arXiv preprint arXiv:2504.01943*, 2025.

[4] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, 2023.

[5] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[6] Antoine Chaffin, Vincent Claveau, and Ewa Kijak. Ppl-mcts: Constrained textual generation through discriminator-guided mcts decoding. In *NAACL 2022-Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1–15, 2022.

[7] Souradip Chakraborty, Sujay Bhatt, Udari Madhushani Sehwag, Soumya Suvra Ghosal, Jiahao Qiu, Mengdi Wang, Dinesh Manocha, Furong Huang, Alec Koppel, and Sumitra Ganesh. Collab: Controlled decoding using mixture of agents for LLM alignment. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=7ohlQUbTpp`.

[8] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[9] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.

[10] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[11] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=uNrFpDPMyo`.

[12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[13] Xiaoke Huang, Juncheng Wu, Hui Liu, Xianfeng Tang, and Yuyin Zhou. m1: Unleash the potential of test-time scaling for medical reasoning with large language models. *CoRR*, 2025.

[14] Yichen Huang and Lin F Yang. Gemini 2.5 pro capable of winning gold at imo 2025. *arXiv preprint arXiv:2507.15855*, 2025.

[15] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *CoRR*, 2024.

[16] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

[17] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with paged attention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[18] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 155–172, 2024.

[19] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

[20] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 522–531. IEEE, 2018.

[21] Mathematical Association of America. American invitational mathematics examination 2024, 2024. URL `https://www.maa.org/sites/default/files/pdf/AMC/aime/2024/AIME-2024-Questions.pdf`. Accessed: 2025-09-06.

[22] Mathematical Association of America. American invitational mathematics examination 2025, 2025. URL `https://www.maa.org/sites/default/files/pdf/AMC/aime/2025/AIME-2025-Questions.pdf`. Accessed: 2025-09-06.

[23] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a {KVCache-centric} architecture for serving {LLM} chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 155–170, 2025.

[24] ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

[25] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient LLM inference. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024. URL `https://openreview.net/forum?id=Ue8EHzaFI4`.

[26] Ranajoy Sadhukhan, Zhuoming Chen, Haizhong Zheng, Yang Zhou, Emma Strubell, and Beidi Chen. Kinetics: Rethinking test-time scaling laws. In *ICML 2025 Workshop on Long-Context Foundation Models*, 2025.

[27] Hanshi Sun, Momin Haider, Ruiqi Zhang, Huitao Yang, Jiahao Qiu, Ming Yin, Mengdi Wang, Peter Bartlett, and Andrea Zanette. Fast best-of-n decoding via speculative rejection. *Advances in Neural Information Processing Systems*, 37:32630–32652, 2024.

[28] Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=oa7MYAO6h6`.

[29] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: query-aware sparsity for efficient long-context llm inference. In *Proceedings of the 41st International Conference on Machine Learning*, pages 47901–47911, 2024.

[30] Qwen Team. Qwq: Reflect deeply on the boundaries of the unknown. *Hugging Face*, 2024.

[31] Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

[32] Junlin Wang, Jue WANG, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=h0ZfDIrj7T.

[33] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

[34] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[35] Samuel Webb Williams, Andrew Waterman, and David A Patterson. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Technical report, Technical Report UCB/EECS-2008-134, EECS Department, University of . . . , 2008.

[36] Juncheng Wu, Wenlong Deng, Xingxuan Li, Sheng Liu, Taomian Mi, Yifan Peng, Ziyang Xu, Yi Liu, Hyunjin Cho, Chang-In Choi, et al. Medreason: Eliciting factual medical reasoning steps in llms via knowledge graphs. *arXiv preprint arXiv:2504.00993*, 2025.

[37] Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=bTHFrqhASY.

[38] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

[39] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[40] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[41] Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.

[42] Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, et al. A survey on test-time scaling in large language models: What, how, where, and how well? *arXiv preprint arXiv:2503.24235*, 2025.

A Equivalent FLOPs in Decoding

We report the equivalent FLOPs in the LLM decoding stage, considering both FLOPs and memory access, following Kinetics [26]. To simplify the modeling, we only account for the matrix operations in Qwen3-14B [39] model, including the query, key, value, and output projection in the grouped query attention (GQA) module [4], as well as the gate, up, and down projection in the Feed-Forward Network (FFN) module.

Table 1: **Notations of Model Configurations**.

Symbol	Description	Symbol	Description	Symbol	Description
H	hidden size	i	FFN intermediate size	q	number of attention heads
kv	number of KV heads	h	head dimension	l	number of layers

Table 2: **Notations of LLM Decoding & Hardware Configurations**.

Symbol	Description	Symbol	Description	Symbol	Description
B	batch size	P	page size	C	context length
T	number of selected tokens	I	GPU arithmetic intensity		

Computation Decoding computation is composed of 2 basic aspects, *i.e.*, model parameter computation and KV cache computation (softmax attention computation). For block-level sparse decoding, the cache selection overhead should also be taken into consideration due to the sequential dependence.

$$C_{\text{comp}}^{\text{param}} = 2 \cdot B \cdot l \cdot (2 \cdot H \cdot q \cdot h + 2 \cdot H \cdot kv \cdot h + 3 \cdot H \cdot i) \tag{1}$$

$$C_{\text{comp}}^{\text{cache}} = B \cdot l \cdot (4 \cdot q \cdot h \cdot T) \tag{2}$$

$$C_{\text{comp}}^{\text{filter}} = 2 \cdot B \cdot l \cdot (q \cdot h \cdot (C/P)) \tag{3}$$

Memory Access Memory access overhead also comprises 2 basic parts, *i.e.*, model parameter access and KV cache access. For block-level sparse decoding, the additional cost of cache selection is also recorded. Both model parameters and the KV cache are in BFLOAT16 half-precision format [16]. Since the KV cache of each key & value head is repeatedly loaded from HBM to SRAM for (q/kv) times in the GQA module when launching thread blocks in FlashAttention [10], here we count for the memory access overhead using q rather than kv.

$$C_{\text{mem}}^{\text{param}} = 2 \cdot l \cdot (2 \cdot H \cdot q \cdot h + 2 \cdot H \cdot kv \cdot h + 3 \cdot H \cdot i) \tag{4}$$

$$C_{\text{mem}}^{\text{cache}} = 2 \cdot B \cdot l \cdot (2 \cdot q \cdot h \cdot T) \tag{5}$$

$$C_{\text{mem}}^{\text{filter}} = 2 \cdot B \cdot l \cdot (q \cdot h \cdot (C/P)) \tag{6}$$

eFLOPs for Different Sparsity Strategies Since our method can overlap the cache fetching overhead with the forward inference pipeline, we do not consider the cache filtering overhead. We formulate the equivalent FLOPs for different sparsity strategies as follows:

- For block-level sparsity, eFLOPs = $C_{\text{comp}}^{\text{param}} + C_{\text{mem}}^{\text{cache}} + C_{\text{mem}}^{\text{filter}} + (C_{\text{mem}}^{\text{param}} + C_{\text{mem}}^{\text{cache}} + C_{\text{mem}}^{\text{filter}}) \cdot I$
- For our method, eFLOPs = $C_{\text{comp}}^{\text{param}} + C_{\text{mem}}^{\text{cache}} + (C_{\text{mem}}^{\text{param}} + C_{\text{mem}}^{\text{cache}}) \cdot I$