Compress, Gather, and Recompute: REFORMing Long-Context Processing in Transformers

Woomin Song [1 2 †] **Sai Muralidhar Jayanthi** [1] **Srikanth Ronanki** [1] **Kanthashree Mysore Sathyendra** [1]
Jinwoo Shin [2] **Aram Galstyan** [1] **Shubham Katiyar** [1] **Sravan Babu Bodapati** [1]

Abstract

As large language models increasingly gain popularity in real-world applications, processing extremely long contexts, often exceeding the model's pre-trained context limits, has emerged as a critical challenge. While existing approaches to efficient long-context processing show promise, recurrent compression-based methods struggle with information preservation, whereas random access approaches require substantial memory resources. We introduce REFORM, a novel inference framework that efficiently handles long contexts through a two-phase approach. First, it incrementally processes input chunks while maintaining a compressed KV cache, constructs cross-layer context embeddings, and utilizes early exit strategy for improved efficiency. Second, it identifies and gathers essential tokens via similarity matching and selectively recomputes the KV cache. Compared to baselines, REFORM achieves over 52% and 34% performance gains on RULER and BABILong respectively at 1M context length. It also outperforms baselines on ∞-Bench, RepoEval, and MM-NIAH, demonstrating flexibility across diverse tasks and domains. Additionally, REFORM reduces inference time by 30% and peak memory usage by 5%, achieving both efficiency and superior performance.

1. Introduction

The ability to handle extremely long contexts, often exceeding the original model's pre-trained context limits, has emerged as a critical challenge for the advanced usage of large language models (LLMs) in real-world scenarios. This capability is essential for various applications, such as processing life-long user interactions, understanding and debugging repository-level codebases, and handling multi-modal inputs (interleaved sequences of text and visual information can result in extremely long contexts). However, under existing Transformer-based language model architectures (Dubey et al., 2024; Jiang et al., 2023), processing such long sequences often causes significant computational challenges, requiring substantial computation as well as memory resources. These demanding requirements often prove infeasible in practical deployment settings, necessitating new technologies that can handle extremely long sequences with reasonable computational resources.

Current approaches to efficient context window extrapolation broadly fall into two categories: recurrent context processing and random access mechanisms. Recurrent methods (Xiao et al., 2023a; Zhang et al., 2023b; Oren et al., 2024; Kim et al., 2024) divide the input into manageable chunks and iteratively process them while maintaining a summarized representation of prior chunks, typically by compressing or evicting parts of the Key-Value (KV) cache. While these approaches reduce memory and computational costs, they often suffer from 'forgetting' due to the loss of critical information during compression and/or eviction.

In contrast, another line of work aims to enable dynamic random-access to the previous inputs by preserving the full KV cache and retrieving relevant portions when processing new chunks (Xiao et al., 2024a; Liu et al., 2024). These methods provide more flexibility in accessing prior context, as they allow selective re-attention to specific segments of the input. However, maintaining the full KV cache requires substantial memory resources, often leading to significant memory overhead and latency increases, especially in practical deployments where CPU memory offloading is necessary. Furthermore, the increased flexibility does not necessarily lead to high retrieval performance. These limitations highlight the need for a more balanced approach that combines efficiency with precise long-context handling.

To address the above challenges, we propose REFORM (**RE**current chunked **F**orwarding with **O**n-demand cache **R**eco**M**putation), a novel inference framework that combines the efficiency of recurrent approaches with the supe-

[†]Work done during an internship at Amazon. [1]Amazon AGI [2]KAIST. Correspondence to: Sravan Babu Bodapati <sravanb@amazon.com>.

3rd Workshop on Efficient Systems for Foundation Models (ES-FoMo III) at ICML, Vancouver, Canada. 2025.

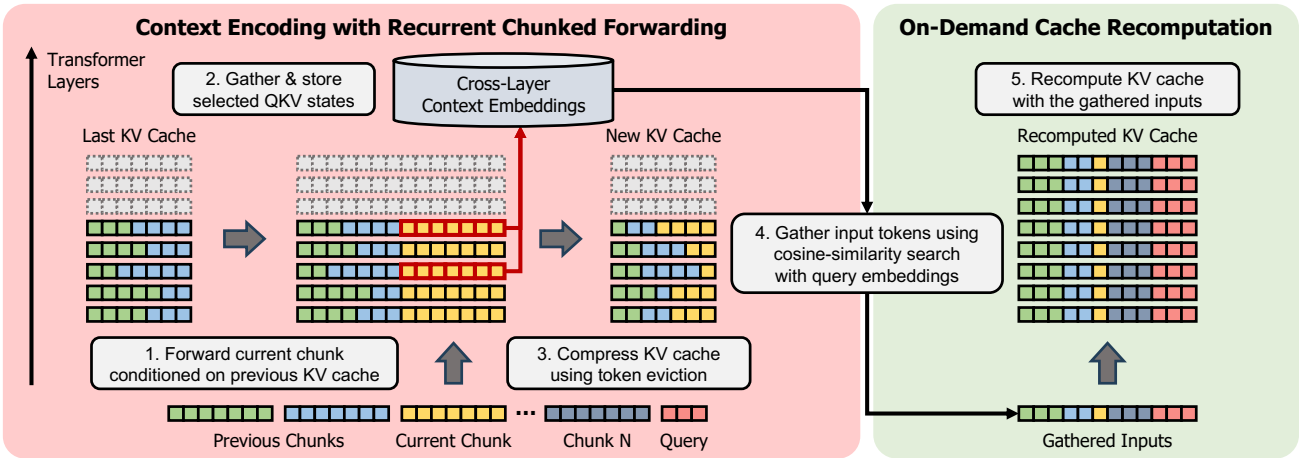

Figure 1. **An overview of the proposed framework.** REFORM efficiently processes long inputs through two phases. In the recurrent chunked forwarding phase, it segments inputs into chunks and processes them iteratively. In each iteration, REFORM (1) forwards each chunk conditioned on the previous KV cache, (2) extracts key QKV states from selected layers and heads for constructing cross-layer context embeddings, and (3) compresses the cache via token eviction (Zhang et al., 2023b). An early exit strategy skips upper layers beyond those used for embedding collection, further improving efficiency. In the on-demand cache recomputation phase, REFORM selects important tokens via similarity search with the query embeddings (last part of the input), gathers them, and recomputes the KV cache for further generation.

rior recall capabilities of random-access methods via a computationally efficient compress-gather-recompute pipeline. In contrast to existing recurrent methods that use compressed KV cache for generation, REFORM uses *compression* to construct and store lightweight token-wise embeddings of the input. Given a query, REFORM then uses these embeddings to *gather* most relevant input tokens via similarity matching, and *recomputes* the full KV cache for those tokens. This process yields high-fidelity yet efficient representations for query-relevant tokens, leading to a superior retrieval ability of the method while still benefiting from reduced memory overhead.

Figure 1 illustrates the overall approach. In the encoding phase, we process input tokens in chunks through an adaptive caching mechanism called recurrent chunked forwarding: as each chunk is processed, tokens are added to the KV cache and compressed by retaining only the heavy hitters (most influential tokens). Using this progressively sparsified KV cache, we compute representations up to an intermediate transformer layer L, collecting QKV states from multiple layers and heads to generate and store lightweight cross-layer context embeddings for all tokens. This multifaceted efficiency strategy—combining chunked processing, sparse KV cache updates, and early exit—significantly reduces both computation time and memory overhead, as we maintain only small representations for retrieval while dynamically managing KV cache sparsity.

In the recomputation phase, the query tokens (corresponding to the recent context) identify relevant historical tokens through similarity matching with the stored retrieval embeddings, and only these selected tokens undergo full KV cache recomputation across all layers. While this phase requires full computation for selected tokens, this recomputation is crucial: it restores high-fidelity representations for contextually important tokens, ensuring accurate processing of long-range dependencies. By selectively recomputing only the most relevant tokens, we achieve a better balance between computational efficiency and model performance, allowing detailed historical context access while avoiding the costs of maintaining full representations for all tokens.

Our extensive evaluations demonstrate REFORM's effectiveness across various long-context understanding tasks. On complex synthetic benchmarks, REFORM significantly outperforms existing methods, achieving over 52% performance gain on RULER and 34% on BABILong at 1M context lengths with the Mistral-NeMo model, compared to the best-performing baselines. On ∞-Bench evaluations, REFORM achieves 31.2% average accuracy on four real-world tasks with the same model, substantially exceeding the baseline performance of 24.8%.

Operating at the transformer architecture level, REFORM is modality-agnostic and applicable to any domain/modality the base model supports. For example, REFORM significantly outperforms the baselines in RepoEval, a repository-level code completion task, using Qwen2.5-Coder-1.5B, and shows superior performance in multimodal needle-in-a-haystack datasets using Pixtral model.

Finally, REFORM delivers substantial efficiency improve-

ments over recent state-of-the-art long-context processing methods. Compared to InfLLM (Xiao et al., 2024a) and InfiniPot (Kim et al., 2024), REFORM reduces inference time by 80% and 33% and peak memory usage by 32% and 5% respectively, in evaluations with 256k token inputs. These results demonstrate that REFORM effectively combines the benefits of both recurrent compression and random access approaches while mitigating their respective limitations.

2. Method

In this section, we present the details of our proposed method. In Section 2.1, we first describe REFORM's recurrent chunk forwarding phase in detail. This phase efficiently constructs token-level, cross-layer context embeddings by segmenting the long input into multiple chunks and repeatedly processing them while conditioning on a compressed previous KV cache. We further elaborate on how we construct the cross-layer context embeddings in Appendix C.1. Then, in Section 2.2, we describe how we use the context embeddings to identify the relevant input segments and highlight our on-demand cache recomputation framework that enables random access to previous contexts while maintaining the integrity of the KV cache. We outline the full procedure in Figure 1 and Algorithm 1.

2.1. Embedding Extraction with Recurrent Chunked Forwarding and Early-Exit

Encoding long contexts with pre-trained Transformers is often infeasible due to the quadratic computational cost and the model's limited context window. To overcome this problem, we focus on recurrent KV cache compression approaches that allow the processing of infinite context under limited resources and context windows. Here, we describe the encoding process in detail, discuss key efficiency benefits, and present our early exit strategy that provides further efficiency gains when using recurrent chunked forwarding to create context embeddings.

Embedding extraction with recurrent chunked forwarding. To process extremely long inputs under limited computational budget and context windows, we adopt an iterative KV cache compression approach (Xiao et al., 2023a; Zhang et al., 2023b; Oren et al., 2024). Specifically, we segment the long input into larger chunks (32k tokens for our experiments) and apply KV cache compression after forwarding each chunk to better utilize parallel computation. For KV cache compression, we employ attention-based token eviction following H2O (Zhang et al., 2023b). After compression, we reassign the position IDs so that the tokens in the compressed cache have consecutive position IDs. This position reassignment allows the model to handle longer sequences beyond its pre-trained context limit.

Unlike existing approaches that directly use these compressed KV cache for generation, we use it only to construct context embeddings that will later be used to identify which part of the input is required. We outline more details for embedding construction in Appendix C.1.

Early exit. Utilizing a compressive approach for creating embeddings introduces an additional benefit: efficiency can be further improved by employing an early exit strategy. As observed in Appendix C.1, high-performing embeddings are often available in the lower Transformer layers. Therefore, forwarding the inputs through the remaining layers after the topmost layer used for embedding extraction is unnecessary. The proposed early exit strategy reduces both computation and memory requirements because we do not need to keep the KV cache for the upper layers.

2.2. On-Demand Cache Recomputation

To enable random access to the previous inputs, we utilize the cross-layer context embeddings to identify the input segments that are relevant to the last part of the input. Then, we gather the corresponding input embeddings and forward them through the model again, re-constructing the KV cache with the most relevant inputs. We conduct the detailed process as follows.

Identification of significant inputs. After constructing the cross-layer context embeddings corresponding to the input, we perform a token-level similarity search between the query (the last part of the input) and the remaining inputs using the context embeddings. Then, we max-pool the similarity scores over the query to tokens to ensure that each token is assigned a single score. To preserve the continuity of the identified inputs, we further max-pool each token's score with the 128 adjacent tokens. After processing the significance scores, we identify the tokens with the highest scores. We always keep the initial and final 256 tokens to maintain coherence.

On-Demand Cache Recomputation. Once we identify the relevant segments, we gather the corresponding input embeddings and forward them through the model again, recomputing the KV cache. The new KV cache is then used for the further decoding process. By introducing on-demand cache recomputation, we avoid the need of storing the full KV cache while enabling random access to previous inputs, significantly reducing the memory requirements.

3. Experiments

Experimental setup and baselines. We compare REFORM against training-free context extrapolation methods, including StreamingLLM (Xiao et al., 2023a), TOVA (Oren et al., 2024), H2O (Zhang et al., 2023b), InfiniPot (Kim et al., 2024), and InfLLM (Xiao et al., 2024a). We also include

Table 1. **Evaluation on RULER and BABILong.** We measure the performance on an extended version of the RULER (Hsieh et al., 2024) and BABILong (Kuratov et al., 2024) benchmark. We report the averaged performance of all tasks at different context lengths. The best values are highlighted in **bold**.

	RULER							BABILong				
	64k	128k	200k	300k	400k	500k	1M	64k	128k	256k	512k	1M
Mistral-Nemo-Instruct-2407												
Truncation	32.6	20.4	17.8	15.2	12.3	12.5	10.8	32.2	26.2	17.0	13.6	14.0
StreamingLLM	27.6	13.8	11.6	9.3	7.2	7.1	4.7	38.8	23.4	15.4	11.0	6.2
TOVA	21.6	15.3	14.0	11.8	7.9	8.7	4.6	37.8	23.6	14.4	9.6	3.4
H2O	15.1	7.4	7.8	5.6	4.2	5.7	3.6	38.0	25.2	16.2	7.2	3.6
InfiniPot	26.9	19.4	15.6	14.5	12.7	13.4	12.0	39.6	26.8	18.6	11.2	8.8
InfLLM	52.7	39.7	28.5	24.9	20.9	22.0	23.3	40.6	34.0	23.6	13.0	9.6
REFORM (Ours)	**79.9**	**81.1**	**83.0**	**84.6**	**84.1**	**83.5**	**75.5**	**57.4**	**51.4**	**50.6**	**47.6**	**48.8**
Qwen2.5-7B-Instruct												
Truncation	46.3	25.1	21.8	17.4	14.9	15.2	11.3	48.4	33.4	27.4	20.0	15.6
StreamingLLM	43.5	25.3	18.7	17.3	11.8	11.8	9.1	53.4	40.6	33.2	23.8	19.6
TOVA	66.2	27.7	25.7	25.8	21.9	20.4	17.0	56.0	46.6	40.6	29.4	21.8
H2O	51.8	20.9	18.5	17.1	11.6	12.1	8.7	57.0	41.6	36.4	24.6	18.8
InfiniPot	65.7	51.7	39.2	33.9	27.8	26.7	23.7	59.6	51.0	53.4	48.2	40.2
InfLLM	47.1	34.2	29.2	24.0	22.0	23.2	23.8	43.0	29.2	20.4	15.4	11.4
REFORM (Ours)	**78.2**	**75.8**	**74.7**	**74.9**	**74.9**	**73.0**	**75.1**	**61.6**	**60.4**	**59.8**	**58.8**	**58.8**

a truncation baseline, which simply drops the middle part of the input. All recurrent baselines operate with a KV cache budget and chunk size of 32k tokens, and InfLLM also uses 32k active KV cache budget. We always keep the initial and recent 256 tokens in cache for all baselines and REFORM to maintain the coherency of the text. For REFORM, we use a recomputation budget of 8k tokens for Mistral-Nemo-Instruct-2407 and 16k tokens for all other models. We provide more details in Appendix C.2.

Performance on RULER and BABILong. Here, we demonstrate the performance of our approach in challenging synthetic benchmarks. Specifically, we evaluate different methods on an extended version of the RULER (Hsieh et al., 2024) and BABILong (Kuratov et al., 2024) benchmarks. RULER is a synthetic long-context benchmark consisting of diverse and challenging needle-in-a-haystack tasks, as well as some aggregation and question-answering tasks. BABILong further challenges the model by introducing more difficult tasks, such as multi-hop reasoning. Although the original version of RULER only supports up to 128k tokens, we further extend the dataset to 1M using the same recipe to evaluate on longer inputs.

We highlight the evaluation results on RULER and BABILong in Table 1. In both benchmarks, REFORM outperforms the baselines by a large margin, indicating its superiority in tasks that require precise recall of essential parts of the context, benefiting both from the ability to locate essential contexts from long inputs and the removal of distribution shifts in the KV cache that commonly come with recurrence-based or random-access approaches.

Further evaluations. While we focus on the RULER and BABILong evaluations in the main text, we conducted extensive evaluation across more diverse tasks and models. We provide the evaluation results in Appendix D. Specifically, in Appendix D.1, we evaluate our approach using needle-in-a-haystack benchmark and showcase that REFORM can retrieve relevant contexts from any position for long inputs up to 1 million tokens. In Appendix D.2, we evaluate REFORM on more diverse realistic benchmarks with a wide range of models, including ∞-Bench (Zhang et al., 2024) for text models, RepoEval (Zhang et al., 2023a) for code models, and MM-NIAH (Wang et al., 2024a) for multi-modal models. In Appendix D.3, we compare REFORM with retrieval-augmented generation. In Appendix D.4, we ablate on each component of REFORM. Finally in Appendix D.5, we highlight that REFORM achieves superior performance while also being the most efficient in terms of memory requirements and inference latency.

4. Conclusion

We introduce REFORM, a novel inference framework for efficient long-context processing. REFORM incrementally processes input chunks while maintaining a compressed KV cache, extracting key QKV states to construct cross-layer context embeddings. An early-exit strategy enhances efficiency, and a similarity-based selection mechanism identifies and gathers essential tokens for KV cache recomputation. REFORM outperforms existing methods across long-context benchmarks while reducing inference time and memory usage. Furthermore, its modality-agnostic design

makes it applicable to a wide range of use cases including multi-modal applications.

References

Agrawal, P., Antoniak, S., Hanna, E. B., Bout, B., Chaplot, D., Chudnovsky, J., Costa, D., De Monicault, B., Garg, S., Gervet, T., et al. Pixtral 12b. *arXiv preprint arXiv:2410.07073*, 2024.

bloc97. Ntk-aware scaled rope allows llama models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation. https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_%20scaled_rope_allows_llama_models_to_have/, 2023.

Bulatov, A., Kuratov, Y., and Burtsev, M. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

Chen, S., Wong, S., Chen, L., and Tian, Y. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023a.

Chen, Y., Qian, S., Tang, H., Lai, X., Liu, Z., Han, S., and Jia, J. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023b.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Dao, T. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

Dong, H., Yang, X., Zhang, Z., Wang, Z., Chi, Y., and Chen, B. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. *arXiv preprint arXiv:2402.09398*, 2024.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

gkamradt. Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023.

Han, C., Wang, Q., Xiong, W., Chen, Y., Ji, H., and Wang, S. Lm-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*, 2023.

Hooper, C., Kim, S., Mohammadzadeh, H., Mahoney, M. W., Shao, Y. S., Keutzer, K., and Gholami, A. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 2024.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Dang, K., et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Jin, H., Han, X., Yang, J., Jiang, Z., Liu, Z., Chang, C.-Y., Chen, H., and Hu, X. Llm maybe longlm: Self-extend llm context window without tuning, 2024.

kaiokendev. Things i'm learning while training superhot. https://kaiokendev.github.io/til#extending-context-to-8k./, 2023.

Kang, H., Zhang, Q., Kundu, S., Jeong, G., Liu, Z., Krishna, T., and Zhao, T. Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

Kim, M., Shim, K., Choi, J., and Chang, S. Infinipot: Infinite context processing on memory-constrained llms. *arXiv preprint arXiv:2410.01518*, 2024.

Kuratov, Y., Bulatov, A., Anokhin, P., Rodkin, I., Sorokin, D., Sorokin, A., and Burtsev, M. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *arXiv preprint arXiv:2406.10149*, 2024.

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024a.

Li, Z., Li, C., Zhang, M., Mei, Q., and Bendersky, M. Retrieval augmented generation or long-context llms? a comprehensive study and hybrid approach. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 881–893, 2024b.

Liu, X., Li, R., Guo, Q., Liu, Z., Song, Y., Lv, K., Yan, H., Li, L., Liu, Q., and Qiu, X. Reattention: Training-free infinite context with finite attention scope. *arXiv preprint arXiv:2407.15176*, 2024.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Munkhdalai, T., Faruqui, M., and Gopal, S. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.

Oren, M., Hassid, M., Yarden, N., Adi, Y., and Schwartz, R. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., Gatford, M., et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

Singhania, P., Singh, S., He, S., Feizi, S., and Bhatele, A. Loki: Low-rank keys for efficient sparse attention. *arXiv preprint arXiv:2406.02542*, 2024.

Song, W., Oh, S., Mo, S., Kim, J., Yun, S., Ha, J.-W., and Shin, J. Hierarchical context merging: Better long context understanding for pre-trained llms. *arXiv preprint arXiv:2404.10308*, 2024.

Su, J. Rectified rotary position embeddings. https://github.com/bojone/rerope, 2023.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Tang, H., Lin, Y., Lin, J., Han, Q., Hong, S., Yao, Y., and Wang, G. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024.

Wang, W., Dong, L., Cheng, H., Liu, X., Yan, X., Gao, J., and Wei, F. Augmenting language models with long-term memory. *arXiv preprint arXiv:2306.07174*, 2023.

Wang, W., Zhang, S., Ren, Y., Duan, Y., Li, T., Liu, S., Hu, M., Chen, Z., Zhang, K., Lu, L., et al. Needle in a multimodal haystack. *arXiv preprint arXiv:2406.07230*, 2024a.

Wang, Z., Jin, B., Yu, Z., and Zhang, M. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*, 2024b.

Wu, W., Wang, Y., Xiao, G., Peng, H., and Fu, Y. Retrieval head mechanistically explains long-context factuality, 2024. URL https://arxiv.org/abs/2404.15574.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022.

Xiao, C., Zhang, P., Han, X., Xiao, G., Lin, Y., Zhang, Z., Liu, Z., Han, S., and Sun, M. Infllm: Unveiling the intrinsic capacity of llms for understanding extremely long sequences with training-free memory. *arXiv preprint arXiv:2402.04617*, 2024a.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023a.

Xiao, G., Tang, J., Zuo, J., Guo, J., Yang, S., Tang, H., Fu, Y., and Han, S. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024b.

Xiao, S., Liu, Z., Zhang, P., and Muennighoff, N. C-pack: Packaged resources to advance general chinese embedding, 2023b.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

Yu, T., Xu, A., and Akkiraju, R. In defense of rag in the era of long-context language models. *arXiv preprint arXiv:2409.01666*, 2024.

Zhang, F., Chen, B., Zhang, Y., Keung, J., Liu, J., Zan, D., Mao, Y., Lou, J.-G., and Chen, W. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*, 2023a.

Zhang, X., Chen, Y., Hu, S., Xu, Z., Chen, J., Hao, M., Han, X., Thai, Z., Wang, S., Liu, Z., et al. ∞ bench: Extending long context evaluation beyond 100k tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, 2024.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023b.

Zhu, D., Yang, N., Wang, L., Song, Y., Wu, W., Wei, F., and Li, S. Pose: Efficient context window extension of llms via positional skip-wise training. *arXiv preprint arXiv:2309.10400*, 2023.

A. REFORM algorithm

We illustrate the overall process of REFORM through a pseudocode in Algorithm 1.

Algorithm 1 Overview of REFORM

```
procedure FORWARDCHUNK(chunk, cache, emb)
    /* Initialize hidden states */
    hs ← input
    /* Forward with early exit */
    for layer in model_layers[:early_exit_layer] do
        hs, cache, qkv ← layer.Forward(hs, cache)
        /* Save selected embeddings */
        emb.SaveSelected(qkv)
    end for
    /* Evict less important tokens */
    cache ← Compress(cache)
    return cache, emb
end procedure
procedure REFORM(input)
    /* Initialize */
    cache, emb ← EmptyInit()
    /* Prepare input chunks */
    context, query ← SplitQuery(input)
    chunks ← ChunkInputs(context) + [query]
    /* Recurrent chunked forwarding */
    for c_i in chunks do
        cache, emb ← ForwardChunk(c_i, cache, emb)
    end for
    /* Gather relevant inputs */
    relevant_inputs ← GatherRelevant(input, emb)
    /* On-demand recomputation */
    cache ← model.Forward(relevant_inputs)
    return cache
end procedure
```

B. Related Works

In this section, we discuss the existing approaches for extending LLM's native context window to efficiently handle extremely long inputs. We categorize these approaches into two groups: methods that use recurrent context processing and methods that leverage random access.

Recurrent context processing. To address the computational challenges of long-context processing, several studies explore the use of recurrence for greater efficiency. A line of works (Dai et al., 2019; Bulatov et al., 2022; Munkhdalai et al., 2024) introduce architectural changes to Transformers, enabling chunk-level recurrence operations to process long contexts in smaller, manageable units. However, these approaches typically necessitate extensive training of the model, and therefore is not directly applicable to existing pre-trained large language models. More recent efforts leverage KV cache eviction to iteratively encode input chunks and compress the KV cache, avoiding architectural modifications or additional training. For instance, StreamingLLM (Xiao et al., 2023a) maintains fluent generation by preserving initial and most recent tokens while compressing intermediate ones. Later approaches (Zhang et al., 2023b; Oren et al., 2024; Kim et al., 2024) identify important tokens from the prior context, enabling more informative cache compression. Despite their efficiency, the process of compressing prior inputs often results in the loss of critical information, leading to 'forgetting' issues. Consequently, these methods may struggle with tasks requiring precise retrieval of earlier inputs. REFORM addresses this issue through its gather and recompute phases, which yields high-fidelity representation of all query-relevant input tokens.

Random access approaches. An alternative direction is to enable random access to prior context, akin to full attention, but in a more computationally efficient manner. These methods typically store the full KV cache in memory and dynamically retrieve relevant tokens as needed. Some approaches train the model (Wu et al., 2022) or an auxiliary side-network (Wang et al., 2023) to utilize the retrieved tokens effectively. More recently, training-free strategies have emerged, which store the full KV cache in memory and retrieve it dynamically (Xiao et al., 2024a; Liu et al., 2024). While these methods allow random access to any part of the input sequence, they introduce significant memory overhead due to the need to maintain large caches. In practice, this often necessitates CPU offloading, which can further increase latency. Furthermore, the flexibility to access previous context may not necessarily lead to high retrieval performance. In contrast, REFORM uses KV cache compression and constructs compact token-level embeddings using only the high-performing heads to reduce memory overhead while still maintaining high retrieval performance.

Extending LLMs to handle extremely long inputs. To extend the context windows of Large Language Modles (LLMs) efficiently, various approaches have been proposed. A significant body of work focuses on modifying positional embeddings. These include scaling Rotary Positional Embeddings (RoPE) (Su et al., 2024) beyond the model's context limit (Chen et al., 2023a; kaiokendev, 2023; bloc97, 2023; Peng et al., 2023), applying attention masks (Han et al., 2023), or adjusting the relative distances between tokens to fall within a predefined range (Su, 2023; Jin et al., 2024). Another line of research explores fine-tuning techniques to adapt models for longer contexts (Zhu et al., 2023; Chen et al., 2023b). While these methods enable models to handle extended inputs, they do not address the significant computational and memory costs introduced by the self-attention mechanism, limiting their practical utility for extremely long contexts. Hence, we did not include them as baselines in our experiments.

Other approaches for efficient long context processing. Together with the recurrent KV cache compression approaches, a large volume of recent works focus on reducing the size of the KV cache to enable more efficient inference at long contexts. For example, SnapKV (Li et al., 2024a) proposes to forward the full input through the model, and then compress the cache by evicting tokens based on attention scores. While efficient at decoding-time, it requires the model to first process the full input, and therefore is not applicable to extremely long inputs that exceed the model's pre-trained context window. Alternatively, HOMER (Song et al., 2024) proposes to use a hierarchical divide-and-conquer approach to combine the encoding and eviction process. Some works propose to further enhance KV cache compression by merging tokens instead of evicting them (Wang et al., 2024b; Dong et al., 2024), but their experiments also only consider inputs within the model's context limit, and their extrapolation capabilities remain unknown. Some recent works propose another direction to keep the full cache only for some selected attention heads known as 'retrieval heads' (Tang et al., 2024; Xiao et al., 2024b), reducing the memory burden of preserving the full KV cache. Other works investigate quantization (Hooper et al., 2024; Kang et al., 2024) and low-rank cache compression (Singhania et al., 2024) to further reduce the memory requirements of the KV cache. However, these methods also cannot extrapolate to longer sequences beyond the model's pre-trained context limit.

C. Experimental Details

C.1. Constructing Cross-Layer Context Embeddings

Prior research has revealed the existence of specialized Transformer heads distributed across different layers that can accurately retrieve relevant information from long context input (Wu et al., 2024). To construct informative embeddings, we thus analyze the retrieval performance of various heads and embeddings in Transformers to determine the most suitable ones for our method. Specifically, we compare the token-level retrieval performance of attention scores (without positional encoding, to make it applicable to extremely long inputs), and cosine similarity between hidden states, or the attention QKV states (i.e. embeddings resulting from QKV projection in attention layers) across Transformer layers.

Table 2. **Comparing similarity search methods.** Best-3 MNR scores (lower is better) corresponding to different similarity search methods including attention, and cosine similarity search using hidden states (HS) or attention QKV states. Scores are measured with Mistral-Nemo-Instruct-2407, and averaged over 500 Multi-hop QA examples.

Type	Dim.	Top-1	Top-2	Top-3	Avg.
Attention	160	6.91	7.70	7.81	7.47
Cosine-HS	5120	9.40	9.63	9.80	9.61
Cosine-Q	160	6.48	6.74	6.93	6.72
Cosine-K	160	6.77	7.31	7.41	7.16
Cosine-V	160	5.77	6.57	6.57	6.30

Embedding head identification. We conducted a set of experiments on a multi-hop question-answering dataset to identify the best embeddings to use for the similarity search; see Appendix C.2 for details. We report the MNR scores for the top-performing layers and heads in Table 2. Somewhat surprisingly, we observe that cosine similarity search with attention

QKV states often outperform the widely-used attention scores. It also outperformed using cosine similarity between the hidden states despite having a much smaller size. This finding suggests that, with careful selection of the appropriate heads, directly using the QKV states from the attention layer and using them for cosine similarity search can achieve a very high performance, while requiring minimal memory.

Figure 2 illustrates the distribution of the top-performing value heads. Notably, the best-performing heads are not necessarily from the final Transformer layers. This observation implies that forwarding inputs through the upper layers may not be required for constructing effective retrieval embeddings, serving as a key motivation for our early exit strategy described in the previous section.

To create a universal embedding that is useful for a wide range of tasks, we do additional experiments to identify heads that can effectively represent complex input patterns. Specifically, we create a synthetic key-value retrieval task, which involves embedding multiple sentences of the format *"The value corresponding to the id {key} is {value}."* within the WikiText (Merity et al., 2016) corpus, where keys and values are random 10-character ASCII strings.

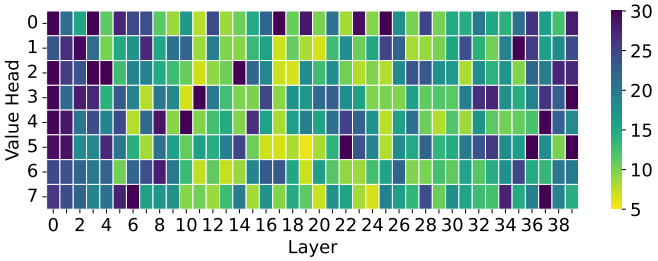

Figure 2. **MNR Scores for Value Heads.** The distribution of the MNR scores (lower is better) across value states of different attention heads, measured by Mistral-Nemo-Instruct-2407 model for 500 synthetic multi-hop QA examples. 256-token heavy hitter budget was used for computation.

We selected the top-performing embeddings for each synthetic dataset after evaluating 500 samples each. We highlight that although head selection is based on relatively short synthetic data (8k tokens), the benefits extrapolate to longer contexts involving millions of tokens.

Combining multiple heads. After identifying the top-performing heads, we combine their embeddings to create a single, token-level embedding. In our preliminary experiments, we observed that using an average of retrieval scores obtained from different heads often improves final retrieval performance. Accordingly, we concatenate the gathered embeddings after normalizing them:

$$e_{\texttt{comb}} = \texttt{concat}\left(\left\{\frac{e_{\texttt{i}}}{||e_{\texttt{i}}||}, \texttt{i} \in \texttt{selected\_heads}\right\}\right)$$

This approach ensures that performing a cosine similarity search using the resulting embedding is mathematically equivalent to independently computing cosine similarity scores for each head and then averaging them.

C.2. Evaluating Retrieval Heads and Embeddings

Dataset preparation. To evaluate the embeddings, we constructed a synthetic dataset based on multi-hop question answering. In this setup, we embedded documents from the HotPotQA dataset (Yang et al., 2018) at random positions within a long text corpus derived from the WikiText dataset (Merity et al., 2016). Each question was appended at the end of the context, and token-level labels were created, where tokens from the golden documents were marked as ground truth. All samples were designed to be 8k tokens long, which is within the context window of the Mistral-7B-Instruct-v0.2 model.

Embedding extraction. To simulate long-context scenarios where full attention computation is infeasible due to computational or memory constraints, we employed a recurrent chunk forwarding method based on H2O (Zhang et al., 2023b), elaborated in Section 2.1. For attention, we compute the retrieval scores using the dot product between query states (Q) and the key states (K) without applying positional encoding. For all other embeddings, we compute the significance scores using cosine similarity between question embeddings and context embeddings, followed by max-pooling over question tokens. Additionally, retrieval scores for each context token were smoothed by mean-pooling with 20 neighboring tokens.

Performance measurement. Retrieval performance was quantified using the Mean Normalized Rank (MNR), which is calculated as the average normalized rank of the golden tokens. Lower scores correspond to higher performance, as the golden tokens have a high rank.

$$\texttt{MNR} = \frac{1}{\texttt{len(gold\_doc)}} \sum_{t \in \texttt{gold\_doc}} \frac{\texttt{rank(t)}}{\texttt{num\_tokens}}$$

C.3. Multimodal Evaluations

Baseline details. In the multi-modal experiments, we evaluate the model performance using recurrence-based methods only, as the codebase for InfLLM only supports text-based models. For InfiniPot, the NuC (novelty under compression) score cannot be utilized for cache compression for multi-modal models because the vision tokens do not output a logit. Therefore, we only apply the CaP (catalyst prompt) score for the InfiniPot baseline in multi-modal experiments.

C.4. Comparison against RAG

Experiment setup. We follow the setup in OP-RAG (Yu et al., 2024) for the RAG experiments, segmenting the inputs to 128-token chunks and preserving the order of the chunks instead of rearranging them according to the retrieval scores. We use Mistral-NeMo-Instruct-2407 as the base LLM. We use BM25 (Robertson et al., 1995) as the sparse retriever, and bge-large-en-v1.5 (Xiao et al., 2023b) as the dense retriever. For each sample, 8k tokens are retrieved in total, matching the KV size with our approach to ensure fair comparison.

C.5. Efficiency Measurements

Experiment setup. We measure the average inference time and peak memory usage for generating 10 tokens conditioned on 256k tokens. All measurements are made on a single H100 GPU, and we apply Flash Attention 2 (Dao, 2024) for all measurements. We further elaborate the experiment setup for InfLLM, as the inference speed and memory consumption can largely vary depending on the configuration. We use the default configuration provided in their GitHub repository, while modifying the number of retrieved blocks to keep 32k active tokens in the cache. The maximum number of blocks cached in GPU was set to be the twice as large as the number of retrieved blocks, following the convention in their official configuration file.

C.6. Embedding Construction and Similarity Search for REFORM

Embedding head selection. We construct the context embeddings by combining four QKV embeddings, where two heads are identified using the pattern matching dataset and the other two are identified using the multi-hop QA dataset. To balance between performance and efficiency gains, we select the top-performing heads from layers with depth under 70% for pattern matching heads. See Appendix D.6 for a more detailed discussion.

For Mistral-NeMo-Instruct-2407, the following heads are used: Query head 9 at layer 15, Value head 5 at layer 19, Value head 0 at layer 27, Value head 7 at layer 27.

For Qwen2.5-7B-Instruct, the following heads are used: Value head 3 at layer 7, Key head 0 at layer 14, Value head 3 at layer 14, Value head 0 at layer 19.

For Qwen2.5-Coder-1.5B-Instruct, the following heads are used: Query head 3 at layer 8, Value head 1 at layer 11, Key head 0 at layer 14, Value head 0 at layer 15.

For Qwen2.5-Coder-7B-Instruct, the following heads are used: Value head 2 at layer 13, Key head 0 at layer 14, Value head 3 at layer 14, Query head 4 at layer 14.

For Pixtral-12B-2409, the following heads are used: Value head 3 at layer 10, Value head 5 at layer 19, Value head 0 at layer 27, Value head 7 at layer 27.

Similarity search. REFORM performs a cosine similarity search between each token in the query (the final part of the input) and the remaining tokens. For better precision in identifying the relevant inputs, we remove the special tokens and the generation prefix (e.g. 'the answer is') when computing the similarity scores.

D. Additional Results

This section further demonstrates the performance of our method across diverse tasks. In Appendix D.1, we begin by showcasing the precise retrieval ability of our approach using a needle-in-a-haystack benchmark. In Appendix D.2, we test REFORM's performance on more realistic tasks including ∞-Bench (Zhang et al., 2024) and RepoEval (Zhang et al., 2023a), as well as highlighting the flexibility of our approach by evaluating it on multi-modal benchmarks. In Appendix D.3, we compare our approach to retrieval-augmented generation, an emerging direction for handling long inputs. Then, we ablate on the key components and analyze the efficiency of our approach in Appendix D.4. In Appendix D.5, we highlight the efficiency benefits of REFORM.

Experimental setup and baselines. Throughout the paper, we mainly compare our approach against training-free context extrapolation methods, with both recurrence-based and random-access approaches. Specifically, we compare our method against StreamingLLM (Xiao et al., 2023a), TOVA (Oren et al., 2024), H2O (Zhang et al., 2023b), InfiniPot (Kim et al., 2024), and InfLLM (Xiao et al., 2024a). We also include a truncation baseline, which simply drops the middle part of the input. For H2O, we restrict attention score computations to the last 128 tokens of each chunk for efficient implementation.

For all text-based experiments, we use Mistral-NeMo-Instruct-2407 (Jiang et al., 2023) and Qwen2.5-7B-Instruct (Yang et al., 2024) models. For code completion experiments and multimodal experiments, we use Qwen2.5-Coder-1.5B/7B (Hui et al., 2024) models and Pixtral-12B-2409 (Agrawal et al., 2024), respectively. All recurrent baselines operate with a KV cache budget and chunk size of 32k tokens, and InfLLM also uses 32k active KV cache budget. We always keep the initial and recent 256 tokens in cache for all baselines and REFORM to maintain the coherency of the text. For REFORM, we use a recomputation budget of 8k tokens for Mistral-Nemo-Instruct-2407 and 16k tokens for all other models. We provide more details in Appendix C.2.

D.1. Needle-In-A-Haystack Evaluation

To evaluate the precise retrieval performance of our approach, we employ the Needle-in-a-Haystack (NIAH) benchmark (gkamradt, 2023). In this task, a specific "needle" sentence (*"The best thing to do in San Francisco is eat a sandwich and sit in Dolores Park on a sunny day"*) is embedded within various depths of irrelevant context consisting of diverse essays by Paul Graham. The model must correctly answer the question: *"What is the best thing to do in San Francisco?"* For evaluation, we consider a response to be correct if it contains all three key phrases: *"eat a sandwich"*, *"sit in Dolores Park"*, and *"a sunny day."*

In Figure 3, we measure the performance of our method at different context lengths and needle depths. Our method demonstrates perfect performance across all setups up to 1M tokens, highlighting our method's robustness in handling extremely long contexts while maintaining precise retrieval performance.

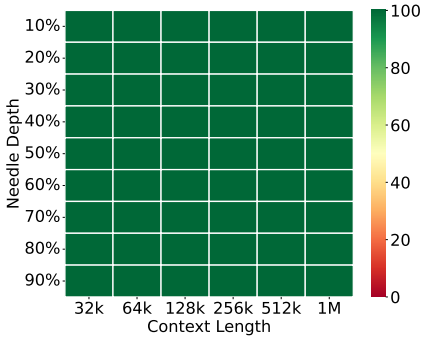

Figure 3. **Needle-In-A-Haystack Evaluation.** We visualize the retrieval accuracy of Qwen2.5-7B-Instruct at different depth and context lengths. Performance is averaged over 20 samples.

D.2. Performance on ∞-bench, RepoEval, and Multi-Modal Evaluations

In this section, we further evaluate the performance on more diverse long context handling tasks. Specifically, we evaluate the performance of REFORM on ∞-bench (Zhang et al., 2024), a more realistic long-context benchmark including tasks derived from long books and dialogues, and RepoEval (Zhang et al., 2023a), a repository-level code completion benchmark, to demonstrate that REFORM is useful in realistic tasks. Furthermore, we highlight the broad applicability of REFORM by demonstrating its performance on a multi-modal benchmark, MM-NIAH (Wang et al., 2024a).

For ∞-bench, we evaluate both Mistral-Nemo-Instruct-2407 and Qwen2.5-7B-Instruct models. For RepoEval, we perform evaluation using code-specific models, namely Qwen2.5-Coder-1.5B/7B-Instruct (Hui et al., 2024). For each sample, we provide the entire repository as the context except for the file that is being completed for the given sample. We report the edit similarity (ES) score as the evaluation metric. Finally for multi-modal evaluations, we use Pixtral-12B-2409 (Agrawal et al., 2024).

Table 3. **Evaluation on ∞-Bench.** We evaluate each method on more realistic datasets from ∞-Bench (Zhang et al., 2024) ranging from long text summarization (En.Sum), question answering from long documents (En.QA, En.MC) and long dialogue understanding (En.Dia). The best values are highlighted in **bold**.

	En.Sum	En.QA	En.MC	En.Dia	Avg.
Mistral-Nemo-Instruct-2407					
Truncation	13.7	16.0	51.1	11.5	23.1
StreamingLLM	12.5	12.6	45.9	6.5	19.3
TOVA	12.3	13.8	47.2	8.0	20.3
H2O	14.2	17.6	49.3	6.0	21.8
InfiniPot	11.9	17.1	52.0	7.0	22.0
InfLLM	16.9	17.4	58.1	7.0	24.8
REFORM (Ours)	**18.2**	**18.0**	**70.3**	**18.5**	**31.2**
Qwen2.5-7B-Instruct					
Truncation	29.0	13.3	43.2	15.0	25.1
StreamingLLM	29.2	8.6	52.4	14.5	26.2
TOVA	29.4	8.6	56.8	15.0	27.4
H2O	**31.0**	11.0	56.3	15.5	28.5
InfiniPot	30.6	11.3	59.0	17.0	29.4
InfLLM	27.6	9.6	38.0	12.0	21.8
REFORM (Ours)	27.8	**16.5**	**61.6**	**21.5**	**31.9**

Table 4. **Evaluation on RepoEval and MM-NIAH.** For RepoEval, we report the edit similarity (ES) score on RepoEval api-level completion task and line-level completion task with 1.5B and 7B models. For MM-NIAH, we report normalized performance across input lengths to ensure equal contribution from each context length range. We do not run multi-modal evaluation for InfLLM, as its implementation only supports text-based models. Best results are in **bold**.

Method	RepoEval				MM-NIAH			
	1.5B API	1.5B Line	7B API	7B Line	Retrieval	Counting	Reasoning	Avg.
Truncate	54.8	63.9	59.2	59.5	72.2	18.7	51.2	47.4
StreamingLLM	55.0	62.7	59.9	58.4	71.9	17.8	49.8	46.5
TOVA	54.7	62.2	59.7	59.8	82.9	18.8	54.1	52.0
H2O	55.1	63.4	61.2	59.6	83.3	18.9	53.5	51.9
InfiniPot	59.4	68.4	66.2	63.8	85.4	18.8	54.7	53.0
InfLLM	61.8	66.8	64.3	66.3	N/A	N/A	N/A	N/A
REFORM (Ours)	**65.3**	**72.4**	**68.7**	**69.4**	**89.2**	**22.0**	**61.3**	**57.5**

As shown in Table 3 and Table 4, REFORM consistently outperforms all baselines in all three benchmarks. This highlights REFORM's superior performance on realistic tasks, and its flexibility to handle diverse inputs, even across modalities.

D.3. Comparison to Retrieval Augmented Generation

We now compare REFORM to Retrieval Augmented Generation (RAG), a popular method for processing long inputs (Li et al., 2024b; Yu et al., 2024). RAG frameworks segment inputs into smaller chunks, which are independently encoded, and use external retrieval models to identify relevant segments. While effective in some scenarios, RAG suffers from key limitations.

First, REFORM avoids the context fragmentation inherent in RAG by conditioning retrieval embeddings on the entire input, ensuring global context continuity and allowing for cohesive processing of long contexts. Second, while RAG frameworks are constrained by the training domain of the retrieval model—requiring domain-specific retraining or advanced adaptations for different domains and modalities—REFORM is inherently flexible and can seamlessly handle diverse domains, including multi-modal applications, without requiring such modifications. Finally, REFORM integrates retrieval functionality directly into the model, eliminating the need for external retrieval models.

In Table 5, we compare the performance of REFORM against RAG approaches using sparse and dense retrievers on the needle-in-a-haystack datasets from RULER at 300k contexts. We provide a more detailed experiment setup in Appendix C.4.

Table 5. **Comparison with RAG.** We compare the performance of RAG methods and REFORM on four groups of needle-in-a-haystack datasets (single, multikey, multivalue, and multiquery) from RULER at 300k contexts, using Mistral-NeMo-Instruct-2407 model.

	Single	M.Key	M.Value	M.Query
Sparse RAG	86.7	77.3	88.5	90.0
Dense RAG	87.3	57.3	82.5	78.0
REFORM	**99.3**	93.3	98.5	**100.0**
+ RAG	**99.3**	**94.7**	**99.0**	**100.0**

REFORM consistently outperforms both approaches in all evaluations, demonstrating its robustness and efficiency. Furthermore, we explore a hybrid approach by combining REFORM with a dense retriever, blending REFORM 's token-level significance scores with retrieval scores using a weighted sum (25% for the dense retriever, 75% for REFORM). This approach performs even better, highlighting the complementary strengths of REFORM and RAG.

D.4. Ablation Studies

Table 6. **Ablation study and efficiency analysis.** (a) We report the average performance on RULER 300k and BABILong 512k datasets using Mistral-NeMo-Instruct-2407 model. (b) We compare the inference time and peak memory usage required for generating 10 tokens conditioned on 256k inputs. All measurements are made with the Mistral-NeMo-Instruct-2407 model on a single H100 GPU, and are averaged over 10 samples. The best values are highlighted in **bold**.

(a) Ablation study.

	RULER	BABILong
REFORM (Ours)	**84.6**	**47.6**
w/ StreamingLLM	82.7	44.6
w/ TOVA	81.4	46.8
w/ Random heads	80.3	43.0
w/ Worst heads	44.7	22.8
w/ Kernel size 5	18.4	36.8

(b) Efficiency Analysis.

	Time (s)	Memory (GB)
StreamingLLM	36.58	37.34
H2O	41.33	37.85
TOVA	39.46	37.06
InfiniPot	40.90	37.06
InfLLM	129.14	51.62
REFORM (Ours)	**27.24**	**35.00**

We conduct an ablation study to evaluate the key components contributing to the effectiveness of our approach. Specifically, we analyze the impact of (1) the choice of the recurrent compression method, (2) the selection of attention heads used for retrieval, and (3) size of the maxpool kernel applied during the gather stage.

Choice of recurrent compression method. To demonstrate the generality of our approach, we replace our recurrent compression component with alternative methods, namely StreamingLLM and TOVA. While H2O yields the best results, Table 6a shows that other compression methods achieve comparable performance. This further highlights the flexibility of our framework and its potential for even higher performance with more advanced compression techniques.

Choice of attention heads. To examine the importance of attention head selection for embedding construction, we replace the selected heads with (1) four randomly chosen heads and (2) four worst heads, identified based on poor performance on both synthetic datasets used for head selection. As shown in Table 6a, the heads selected by our mechanism achieve the best performance, demonstrating its effectiveness. Random heads generally show lower but reasonable performance. In contrast, using bad heads results in a substantial performance drop on both benchmarks, underscoring the importance of proper attention head selection to ensure effective embedding construction.

Pooling kernel size. In the on-demand cache recomputation phase, we apply max-pooling over 129-token windows to smooth token-level similarity scores. Reducing the pooling size to 5 tokens significantly degrades performance, highlighting the importance of pooling to maintain contextual information during the recomputation.

D.5. Efficiency Analysis

To highlight the efficiency benefits of our approach, we measure the peak memory usage and inference time required for processing a long input. We outline the results in Table 6b. InfLLM suffers from high inference time due to frequent memory transfer between CPU and GPU and requires large memory to store the cache. Recurrent methods offer faster

inference at lower memory costs, enjoying the benefits of using a fixed-size KV cache. Our approach shows even lower latency and memory requirements compared to the recurrent baselines thanks to the early exit, which saves computation as well as memory by removing the need to keep the KV cache for the upper layers.

D.6. Embedding Head Identification for Pattern Matching Task

Table 7. **Comparing different LLM embeddings.** Best-3 MNR scores (lower is better) corresponding to the hidden states and the attention states, measured by Mistral-Nemo-Instruct-2407. Scores are averaged over 500 synthetic pattern matching examples.

Type	Dim.	Top-1	Top-2	Top-3	Avg.
Hidden States	5120	1.72	1.88	2.10	1.90
Attention	160	1.24	1.36	1.37	1.32
Query	160	1.51	1.56	1.57	1.55
Key	160	1.53	1.65	1.72	1.63
Value	160	0.93	0.95	1.13	1.00

In this section, we present the distribution of MNR scores measured with our pattern matching dataset, similarly to what we presented in Table 2 and Figure 2. The corresponding results for pattern matching dataset is presented in Table 7 and Figure 4. The retrieval performance of QKV heads often outperform that of the hidden states, similarly to the case of multi-hop QA datasets.

Interestingly, the distribution of best-performing heads show a different pattern compared to the milti-hop QA dataset, and the heads at lower layers and middle-to-upper layers show the highest performance. This suggests that different heads show different characteristics depending on the task. It also motivates our approach of using the embeddings identified by the different tasks as it yields more general representations and makes similarity-based retrieval more accurate. It is also important to note that while the upper layer has more good-performing heads, these heads can be also identified in the mid-lower layers (e.g., Layer 16, Head 1). To balance the performance with the efficiency gains provided by early-exit strategy, we select the best-performing pattern-matching heads from layers under 70% of depth. This strategy ensures that we utilize the high-performing heads as well as enjoying the computation savings from early exit.

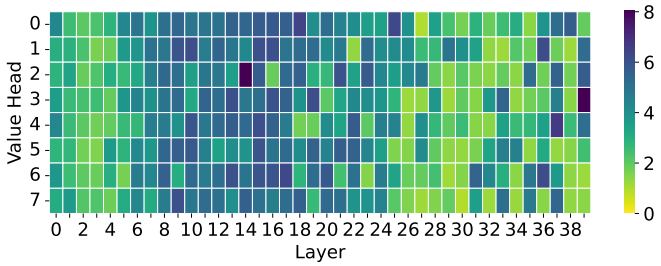

Figure 4. **MNR Scores for Value Heads.** The distribution of the MNR scores (lower is better) across value states of different attention heads, measured by Mistral-Nemo-Instruct-2407 model over 500 synthetic pattern matching examples. Recurrent chunked forwarding with 256-token heavy hitter budget was employed for computing the embeddings.

E. Broader impacts

We believe that the high capability and flexibility will aid everyday use of large foundation models, by extending the model capabilities to efficiently and effectively handle very long contexts. On the other hand, such capabilities of REFORM could potentially enable malicious parties to analyze vast amount of data, enhancing the capabilities of autonomous systems that could be used for manipulation or misinformation.

F. License information for datasets and models

Here, we provide the license for all datasets and models used in our experiments. Apache 2.0 license is applied for Babilong, RULER, Mistral-Nemo-Instruct-2407, Qwen2.5-Coder family, and Pixtral-12B-2409. BSD license is also applied for

some parts of Babilong dataset. MIT license is applied to Needle-in-a-Haystack, InfiniteBench, RepoEval, WikiText, and bge-large-en-v1.5. CC BY-SA 4.0 is applied for HotPotQA.