HEADINFER : Memory-Efficient LLM Inference by Head-wise Offloading

Cheng Luo [1] **Zefan Cai** [2] **Hanshi Sun** [3] **Jinqi Xiao** [4] **Bo Yuan** [4] **Wen Xiao** [5] **Junjie Hu** [2] **Jiawei Zhao** [1] **Beidi Chen** [3] **Anima Anandkumar** [1]

Abstract

Transformer-based large language models (LLMs) demonstrate impressive performance in long context generation. Extending the context length has disproportionately shifted the memory footprint of LLMs during inference to the key-value cache (KV cache). In this paper, we propose HEADINFER , which offloads the KV cache to CPU RAM while avoiding the need to fully store the KV cache for any transformer layer on the GPU. HEADINFER employs a fine-grained, head-wise offloading strategy, maintaining only selective attention heads' KV cache on the GPU while computing attention output dynamically. Through roofline analysis, we demonstrate that HEADINFER maintains computational efficiency while significantly reducing memory footprint. We evaluate HEADINFER on the Llama-3-8B model with a 1-million-token sequence, reducing the GPU memory footprint of the KV cache from 128 GB to 1 GB and the total GPU memory usage from 207 GB to 17 GB, achieving a 92% reduction compared to BF16 baseline inference. Notably, HEADINFER enables 4-million-token inference with a Llama3-8B model, and 2.7 million-token inference with a MAGI-4.5B model on a single consumer GPU with 24GB memory (e.g., NVIDIA RTX 4090) without approximation.

1. Introduction

Modern Large Language Models (LLMs) increasingly support extremely long inputs: Llama-3 (Dubey et al., 2024) handles up to 128K tokens, Claude (Anthropic, 2024) supports up to 1 million tokens, while Gradient AI (Pekelis et al., 2024) extends Llama-3 to 4 million tokens. Video Generation Model, like MAGI-1 (Sand-AI, 2025), can also support up to 4 million tokens. These extended context lengths improve performance on tasks such as book summarization (Pal et al., 2023) and video generation (Liu et al., 2024a), both of which require millions of tokens.

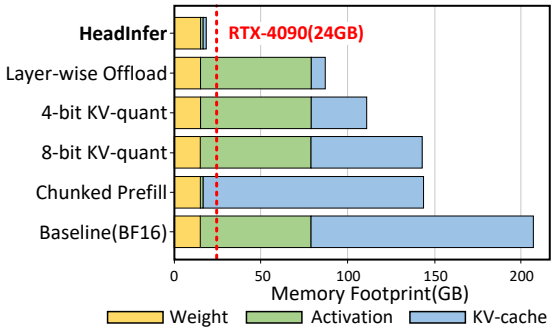

Figure 1. Estimated memory consumption of inference a Llama-3-8B model with 1 million token.

As context lengths increase, memory usage and latency grow significantly due to self-attention in transformers. To improve inference efficiency (Zhou et al., 2024), most LLM inference consists of two phases: prefill for input processing and decoding for token generation, with key and value states from attention cached for reuse (KV cache). However, as the context length increases, the KV cache memory grows rapidly, posing significant challenges for storage and efficiency. For example, generating 1 million tokens using a Llama-3-8B model requires 207 GB of memory (15 GB for pre-trained parameters, 64 GB for activation, and 128 GB for KV cache [1]) as shown in Figure 1. Similarly, generating a one-minute video using MAGI-1 4.5B (Sand-AI, 2025) requires 170 GB (17 GB for pre-trained parameters, 5 GB for activation, and 158 GB for KV cache). This makes long-context inference unfeasible on consumer-level GPUs such as the NVIDIA RTX-4090 with 24 GB of memory.

To address this memory challenge, modern LLM serving systems have introduced offloading strategies that transfer data to CPU memory, enabling efficient LLM inference within hardware constraints (Sheng et al., 2023; Lee et al., 2024; Aminabadi et al., 2022). By moving portions of the

[*]Equal contribution [1]California Institute of Technology [2]University of Wisconsin-Madison [3]Carnegie Mellon University [4]Rutgers University [5]Microsoft. Correspondence to: Cheng Luo <chengluo@caltech.edu>, Anima Anandkumar <anima@caltech.edu>.

Proceedings of the 2nd Workshop on Long-Context Foundation Models, Vancouver, Canada. 2025. Copyright 2025 by the author(s).

[1]The calculation is based on standard inference generation using Hugging Face with BF16 precision and a batch size of 1. Details of how it is calculated and more model analysis can be referred to in Appendix A.4.

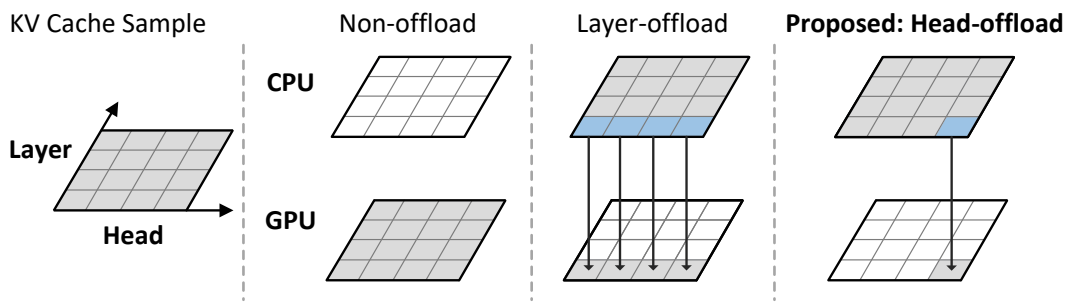

Figure 2. Demonstrations of KV cache policies in inference. Full KV cache contains two main dimensions: layer and head. Layer-wise offloads KV cache in the layer's dimension, with a cache budget of all heads per layer. HEADINFER further reduces GPU memory by adaptively reallocating cache budgets in the head's dimension, with a cache budget of one head.

KV cache to the CPU, these methods allow for much longer context generation than would otherwise fit in GPU memory. However, these methods can hardly be applied to consumer-level GPUs such as the RTX-4090, where only 8GB of memory remain for the KV cache and activations. This is due to two factors: (1) offloading does not reduce activation memory usage, and (2) because offloading is applied at a coarse granularity, the remaining KV cache on the GPU ends up larger than the rest of the memory usage.

Our approach: We propose head-wise offload (HEADINFER), an inference framework that drastically reduces the GPU memory needed for LLM inference by offloading the KV cache at the level of individual attention heads. The key idea is to leverage attention head independence to decompose the attention computation in a head-wise manner, requiring only one head of the KV cache on the GPU at a time and offloading the rest to the CPU. As shown in Figure 2, HEADINFER frees the GPU system from storing the KV cache of the entire model or layer and instead stores only a single head at any given time. This fine granularity drastically reduces the GPU memory footprint while maintaining exact mathematical equivalence.

Through roofline analysis (Williams et al., 2009) at appendix , we demonstrate that HEADINFER maintains high compute efficiency while significantly reducing GPU memory consumption. We implement HEADINFER on the Hugging Face framework and evaluate it on representative LLMs, including Llama, Qwen, Mistral, and Gemma, with varying model sizes and sequence lengths. More than the 1 million token inferences shown in Figure 1, our results on NVIDIA RTX 4090 demonstrate that HEADINFER extends the Llama-3-8B model's context length from 25K (standard inference) and 45K (standard offload) to 4 million tokens, achieving around 100x improvement in context length extension. HEADINFER can also extend the MAGI-1 4.5B model's context length from 691K (standard offload) to 2.8 million tokens with a 4x improvement in context length extension. The code implementation, extension, and generated videos are available at https://anonymous.4open.science/r/headinfer-213C.

2. HEADINFER : Head-wise Offload

2.1. Background:Offload KV Cache

When the context length S grows on a million scale, or the GPU's on-device High-Bandwidth Memory (HBM, or M_{HBM}) is small on the consumer GPU, it may become insufficient to store the entire key-value cache. In such scenarios, KV cache offloading provides a practical approach to handling memory limitations. Offloading can temporarily move parts of the KV cache to CPU RAM or other external resources (e.g., NVMe storage and CPU disk). However, each offloading strategy introduces potential communication overheads that must be carefully weighed against the gains in usable context length.

For a batch of size B, a transformer with L layers, and the KV cache with the bytes per element $sizeof(datatype)$, the total KV cache size is:

$$Size_{KVcache} = 2 \times B \times L \times S \times D \times sizeof(datatype) \tag{1}$$

If $Size_{KV_{cache}} > M_{HBM}$, the system can offload some portion of the KV cache to external memory to avoid out-of-memory errors. Let $\alpha (0 \leq \alpha \leq 1)$ be the fraction of the KV cache that remains on the GPU, and $1 - \alpha$ be the fraction offloaded to external memory. The memory footprint on the GPU $Size_{on-GPU}$ can be expressed as follows:

$$Size_{on-GPU} = \alpha \times Size_{KVcache} \tag{2}$$

Therefore, we require:

$$Size_{on-GPU} \leq M_{HBM}, \alpha \leq \frac{M_{HBM}}{Size_{KV_{cache}}} \tag{3}$$

Head-wise KV Cache Generation. In transformer architectures, each self-attention layer is split into multiple attention heads H. Each head has its own set of key-value (KV) tensors:

$$K^{(h)} \in \mathbb{R}^{S \times D_h}, V^{(h)} \in \mathbb{R}^{S \times D_h} \tag{4}$$

where D is divided by H so that each head handles a sub-dimension $D_h = D/H$. Therefore, instead of a single

large KV cache, a head-wise approach organizes cache in H separate memory spaces. Formally, at time step t, we have:

$$K_{cache}^{(h)} = [K_1^{(h)}, ..., K_t^{(h)}], V_{cache}^{(h)} = [V_1^{(h)}, ..., V_t^{(h)}] \quad (5)$$

As a result, each head stores its keys and values in a contiguous memory space, enabling selective offloading of certain heads' cache when memory constraints emerge.

During self-attention at the time step t, we calculate the attention output for each head h independently, using:

$$A_t^{(h)} = Softmax(\frac{Q_t^{(h)} K_{cache^{(h)}}^T}{\sqrt{d_k}}) V_{cache^{(h)}} \quad (6)$$

Finally, the outputs ${A_t^{(h)}}_{h=1}^{H}$ are concatenated to form the final output of attention for that layer.

Head-wise Offload. Since the attention computation has a head-wise independence, if we can keep the KV cache of a single head rather than the entire layer, then the memory consumption can be reduced substantially. This leads to our proposed head-wise offload (HEADINFER) strategy.

HEADINFER is designed to minimize the fraction of on-GPU data α (the fraction of the total KV cache stored on GPU). Using Llama-3-8b as an example, we define H_{all} as the total number of attention heads of a given model, which equals the number of heads per layer times the number of layers $H \times L$. H_{on} is the number of heads h retained on the GPU, and H_{off} is the number of heads offloaded to external memory (CPU); obviously, we have $H_{on} + H_{off} = H_{all}$

Define α as the fraction of the KV cache that remains on the GPU. We can store all KV cache on the GPU if $\alpha = 1$ or layer-wise offload if $\alpha = 1/L$. In our head-wise scheme:

$$\alpha = \frac{H_{on}}{H_{all}} = \frac{1}{L \times H} \quad (7)$$

Here we keep only a single head on the GPU, and the fraction of the total KV cache that occupies GPU memory is reduced by $\alpha = 1/(L \times H)$, with a total size of:

$$S_{on-GPU} = 2 \times B \times S \times D_H \times sizeof(datatype) \quad (8)$$

By reducing α, we can preserve GPU memory capacity for extended context inference and make the million-level inferences on consumer GPUs possible.

2.2. Granularity: Sequences, Layers, and Heads

HEADINFER differs from traditional offload in terms of granularity. When deploying large language models, each dimension of the model can become a potential bottleneck in GPU memory. Naively offloading the entire KV cache or entire layers can lead to suboptimal memory usage.

HEADINFER addresses this challenge by introducing a hierarchical set of techniques, including *chunked-prefill* (Agrawal et al., 2024), *layer-wise offload* (Xiong et al., 2024) , and *head-wise offload* that operate at increasingly fine-grained levels of sequence (S), layers (L), and heads (H).

- **Chunked-Prefill (S)**: Rather than processing all sequences of input tokens at once, HEADINFER divides the sequence into smaller chunks, each processed separately during the prefill stage. This partition helps reduce the activation memory usage.
- **Layer-Wise Offload (L)**: Instead of storing the entire KV cache on the GPU, HEADINFER offloads it to the CPU and fetches the KV cache of specific layers on demand. Consequently, only the relevant portion of the KV cache resides on the GPU at any given time.
- **Head-Wise Offload (H)**: Within each layer, HEADINFER can selectively offload the KV cache for all attention heads, fetching certain attention heads on demand. This offers the finest granularity by focusing on a subset of heads within each layer.

By combining these techniques, HEADINFER allows precise control over which parts of the activation and KV cache remain on the GPU. Because these dimensions nest naturally as chunks are subsets of the sequence, layers are subsets of the model, and heads are subsets of the attention layer.

3. Performance Evaluation

3.1. Experimental Setup

We conduct our experiments on RTX-4090 GPUs, which are configured with 24.5 GB of HBM3, 4× AMD EPYC 7B13 with 64 CPUs each (a total of 256 cores), and 2× Gen4 NVMe of 512 GB each and 1 TB of DDR5 RAM in total. The GPUs are independently connected to the host with 16 PCIe 4.0 interfaces, providing 25.0 GB/s unidirectional D2H and H2D throughput for pinned host memory.

3.2. Long-Context Benchmarks

We evaluate HEADINFER using the LongBench v2 (Bai et al., 2024) benchmarks, and other long-context benchmark results such as SCBench (Li et al., 2024), Needle-in-a-Haystack (NIAH) (Kamradt, 2023) and Ruler (Hsieh et al., 2024) are shown in the Appendix A.2. We use the Llama-3-8B-Instruct-Gradient-1024k model with 1 million context lengths.

LongBench v2 is a comprehensive suite of long-context datasets designed to assess long-context problems that require deep understanding and reasoning. It comprises 503

Table 1. Performance(benchmark score) of different methods on **LongBench v2** on a single RTX-4090 GPU, under different task difficulties (Easy/Hard) and context lengths (Short/Medium/Long). Overall performance is average scores on all questions.

LongBench V2	Overall	Easy	Hard	Short	Medium	Long	Contex Length
Standard	28.4	30.2	27.3	33.9	25.1	25.9	25K
Chunked Prefill	28.2	27.1	28.9	32.8	25.6	25.9	30K
Layer-wise offload	29.0	29.2	28.9	**36.1**	24.2	26.9	45K
HEADINFER	**30.2**	**31.2**	**29.6**	33.9	**27.0**	**30.6**	**1024K**

Table 2. Comparison of maximum context length with system optimization methods.

Context Length	Llama-3-8B	Llama-2-7B	Mistral-7B	Qwen2-7B	Gemma-2-9b
Standard	25K	10K	30K	35K	10K
Chunked Prefill	30K	20K	40K	70K	10K
4-bit KV-quant	45K	30K	40K	50K	20K
Layer-wise offload	45K	60K	45K	50K	35K
HEADINFER	**4096K**	**1024K**	**4096K**	**4200K**	**1300K**

Table 3. Long-context video generation on a single RTX-4090.

MAGI-1-4.5B	**Frames**	**Video Size**	**Context (tokens)**	**Generate Latency**
Non-offload	–	-	OOM	–
Layer-offload	768	32s	691200	9h 27m
HEADINFER	**768**	**32s**	**691200**	**9h 46m**
HEADINFER	**3072**	**2m8s**	**2764800**	**43h 9m**

difficult multiple-choice questions within two difficulties, including "Easy/Hard" and word lengths, including "Short" of 0-32k words, "Medium" of 32k-128k words, and "Long" of 128k-2M words. As shown in Table 1, we measure their overall scores for different task categories. HEADINFER achieves the highest benchmark score for "Medium" (27.70) and "Long" (30.6). Layer-wise offload performs best on "Short" (36.1) for truncation side effects.

3.3. Long-Context Video Generation

We evaluated HEADINFER using the MAGI-1 4.5B model (Sand-AI, 2025) on a single RTX-4090. MAGI-1 generates videos by autoregressively predicting a sequence of video chunks, defined as fixed-length segments of consecutive frames. The generation process is equivalent to chunked prefill, where each contains 24 raw frames and 21.6K context tokens. The chunk size is larger than 2k; therefore, offload and HEADINFER can effectively extend the maximal context length without any slowdown.

We evaluate MAGI-1 (4.5 B) by generating "good boy" videos with 720x1280 px, 24 FPS, and 64 denoising steps. Table 3 reports the maximum video length supported on one RTX-4090 and the wall-clock time to complete generation. HEADINFER provides a 4x context length extension with only a 3% slowdown compared with layer-offload baseline. More importantly, only HEADINFER breaks the 1 million token barrier on consumer GPUs, enabling 2-minute videos generation without approximation methods. The details of generation can be found in appendix A.3.

3.4. Efficiency Results of Memory and Throughput

We evaluate the maximum context length supported on RTX-4090, as well as prefill/decoding throughput. Our experiments use the Llama-3-8B, Llama-3-70B, Mistral (Jiang et al., 2023), Qwen (Bai et al., 2023a), and Gemma-2 (Team et al., 2024) models. The default number format for weights and activations is BFloat16, and 4-bit KV-quant is deployed with KIVI (Liu et al., 2024b). The chunk size is set to 10K based on our roofline analysis.

LLM Inference on Consumer GPUs with 24G memory. We measure the GPU memory consumption of Llama3-8B inference with HEADINFER and 1 million context length. HEADINFER uses 17GB during prefill and 16.4GB during decoding; in contrast, other methods are unable to run at this context scale using 24GB RTX-4090. Accordingly, we measure the maximum achievable context length to assess memory efficiency. As shown in Table 2, HEADINFER outperforms other system optimization approaches, scaling from thousand-level contexts (10K-70K) to million-level contexts (1,024K–4,200K). Note that with HEADINFER , the maximum context lengths for Llama3, Llama2, and Mistral are constrained by CPU RAM (512GB for KV cache), while the other methods are limited by GPU memory. This means we can use larger CPU RAM or offload to disk for a more extended context. We leave this exploration for future work.

4. Conclusion

In this paper, we introduced HEADINFER , a novel head-wise KV cache management framework designed to enable efficient long-context large language model (LLM) inference on consumer GPUs. HEADINFER dynamically offloads KV cache components to CPU memory, employing head-wise and asynchronous offloading strategies. Our Roofline analysis highlights HEADINFER 's ability to preserve GPU computational efficiency while reducing memory footprint, making long-context inference feasible on consumer-grade GPUs.

Impact Statement

Artificial Intelligence (AI) has the potential to transform industries, revolutionize education, and empower individuals. However, the deployment of cutting-edge models, particularly large language models (LLMs), is often hindered by significant resource requirements, creating barriers to entry for smaller organizations and underserved communities.

In this work, we introduce HEADINFER , a memory-efficient inference framework designed to bridge this gap. By leveraging head-wise offloading strategies, HEADINFER enables resource-constrained devices to process unprecedentedly long contexts, achieving capabilities typically reserved for high-performance systems. For instance, HEADINFER allows a consumer-grade GPU to handle over 1 million context tokens, democratizing access to advanced LLM functionalities.

References

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B., Tumanov, A., and Ramjee, R. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, 2024.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

Anthropic, A. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 1, 2024.

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023a.

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023b.

Bai, Y., Tu, S., Zhang, J., Peng, H., Wang, X., Lv, X., Cao, S., Xu, J., Hou, L., Dong, Y., Tang, J., and Li, J. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *arXiv preprint arXiv:2412.15204*, 2024.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Ding, J., Ma, S., Dong, L., Zhang, X., Huang, S., Wang, W., Zheng, N., and Wei, F. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023.

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.

Fu, Y., Cai, Z., Asi, A., Xiong, W., Dong, Y., and Xiao, W. Not all heads matter: A head-level kv cache compression method with integrated retrieval and reasoning. *arXiv preprint arXiv:2410.19258*, 2024.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 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.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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.

Jiang, H., LI, Y., Zhang, C., Wu, Q., Luo, X., Ahn, S., Han, Z., Abdi, A. H., Li, D., Lin, C.-Y., et al. Minference 1.0:

Accelerating pre-filling for long-context llms via dynamic sparse attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Kamradt, G. Needle in a haystack-pressure testing llms. *Github Repository*, pp. 28, 2023.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lee, W., Lee, J., Seo, J., and Sim, J. 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)*, pp. 155–172, 2024.

Li, Y., Jiang, H., Wu, Q., Luo, X., Ahn, S., Zhang, C., Abdi, A. H., Li, D., Gao, J., Yang, Y., et al. Scbench: A kv cache-centric analysis of long-context methods. *arXiv preprint arXiv:2412.10319*, 2024.

Liu, H., Yan, W., Zaharia, M., and Abbeel, P. World model on million-length video and language with blockwise ringattention. *CoRR*, 2024a.

Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., and Hu, X. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *Forty-first International Conference on Machine Learning*, 2024b.

Loeschcke, S. B., Toftrup, M., Kastoryano, M., Belongie, S., and Snæbjarnarson, V. Loqt: Low-rank adapters for quantized pretraining. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Luo, C., Zhao, J., Chen, Z., Chen, B., and Anandkumar, A. Mini-sequence transformers: Optimizing intermediate memory for long sequences training. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Pal, A., Karkhanis, D., Roberts, M., Dooley, S., Sundararajan, A., and Naidu, S. Giraffe: Adventures in expanding context lengths in llms. *arXiv preprint arXiv:2308.10882*, 2023.

Pekelis, L., Feil, M., Moret, F., Huang, M., and Peng, T. Llama 3 gradient: A series of long context models, 2024.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Grella, M., et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Sand-AI. Magi-1: Autoregressive video generation at scale, 2025. URL https://static.magi.world/static/files/MAGI_1.pdf.

Shah, J., Bikshandi, G., Zhang, Y., Thakkar, V., Ramani, P., and Dao, T. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 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.

Team, G., Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Williams, S., Waterman, A., and Patterson, D. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Wu, W., Wang, Y., Xiao, G., Peng, H., and Fu, Y. Retrieval head mechanistically explains long-context factuality. *arXiv preprint arXiv:2404.15574*, 2024.

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*, 2024a.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024b.

Xiong, Y., Wu, H., Shao, C., Wang, Z., Zhang, R., Guo, Y., Zhao, J., Zhang, K., and Pan, Z. Layerkv: Optimizing large language model serving with layer-wise kv cache management. *arXiv preprint arXiv:2410.00428*, 2024.

Ye, Z., Chen, L., Lai, R., Lin, W., Zhang, Y., Wang, S., Chen, T., Kasikci, B., Grover, V., Krishnamurthy, A., et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025.

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, 2023.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. Galore: Memory-efficient llm training by gradient low-rank projection. In *Forty-first International Conference on Machine Learning*, 2024.

Zhou, Z., Ning, X., Hong, K., Fu, T., Xu, J., Li, S., Lou, Y., Wang, L., Yuan, Z., Li, X., et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*, 2024.

A. Experiment Details

A.1. Prefill and Decoding Overhead of HEADINFER

We evaluate the computational overhead of HEADINFER compared to baseline approaches across different context lengths using the Llama-3-8B model. Our analysis focuses on two key phases: prefill and decoding. Here we provide additional experiments between vllm (Kwon et al., 2023) and HEADINFER +flashinfer(Ye et al., 2025).

Table 4. Prefill speed (tokens / s) of Llama3-8B under different context lengths.

Prefill Latency(s)	**10K**	**20K**	**40K**	**100K**	**200K**	**400K**	**1M**	**2M**	**4M**
VLLM	9151	7847	-	-	-	-	-	-	-
HEADINFER group=1	8982	7781	6060	3476	2121	1196	-	-	-
HEADINFER group=2	8910	7646	5784	3455	2100	1190	516	-	-
HEADINFER group=4	8910	7516	5570	3442	2100	1180	516	264	-
HEADINFER group=8	8700	7221	5405	3365	2079	1176	511	263	155
HEADINFER Adaptive	8982	7781	5906	3476	2121	1196	516	264	155

Table 5. Decoding speed (tokens / s) of Llama3-8B under different context sizes.

Decoding Latency(s)	**10K**	**20K**	**40K**	**100K**	**200K**	**400K**	**1M**	**2M**	**4M**
VLLM	52	49	-	-	-	-	-	-	-
HEADINFER group=1	12	7	3.7	1.6	0.79	0.40	-	-	-
HEADINFER group=2	11	6	3.7	1.5	0.78	0.40	0.16	-	-
HEADINFER group=4	10	6	3.4	1.5	0.78	0.40	0.16	0.07	-
HEADINFER group=8	8	5	3.1	1.4	0.77	0.39	0.16	0.07	0.04
HEADINFER Adaptive	12	7	3.7	1.6	0.79	0.40	0.16	0.07	0.04

For prefill operations, HEADINFER demonstrates similar performance to standard approaches for shorter context lengths (up to 20K tokens). Beyond this range, HEADINFER scales efficiently with longer contexts, outperforming Layer-offload due to its fine-grained head-wise KV cache management. Notably, HEADINFER enables inference for 4M tokens on a single GPU, which is otherwise infeasible.

In decoding, HEADINFER maintains low latency at short and medium context lengths. For extended contexts (e.g., 1M and beyond), HEADINFER introduces manageable latency while supporting unprecedented context lengths, with adaptive configurations optimizing performance further.

The performance of HEADINFER relies on its ability to dynamically adapt to varying hardware constraints and workload requirements by controlling the granularity of offloading. Specifically, HEADINFER Adaptive achieves optimal performance by selectively choosing the most suitable head group size based on the context length and memory limitations.

HEADINFER (Head = 8 / Group = 1). This configuration aggregates all attention heads within a layer into a single group for offloading, effectively making the KV cache management layer-wise. As such, **HEADINFER (Head Group = 8)** is functionally equivalent to **Layer-Offload**, where all KV cache for a layer is stored on either the GPU or offloaded to the CPU in a single operation. However, HEADINFER provides the flexibility to adjust the granularity of KV cache management beyond this layer-wise approach.

HEADINFER (Head = 1 / Group = 8). At the other extreme, this configuration offloads each attention head individually, offering the finest level of granularity for KV cache control. While this achieves the highest memory savings, frequent PCIe transfers and kernel launches introduce overhead, especially for shorter context lengths, which can impact throughput.

HEADINFER Adaptive dynamically selects the optimal head group size (h) to balance memory efficiency and computational throughput. This adaptability allows it to achieve superior performance across diverse hardware and context length requirements:

This adaptability positions **HEADINFER Adaptive** as an essential method for large-scale language model inference in memory-constrained environments.

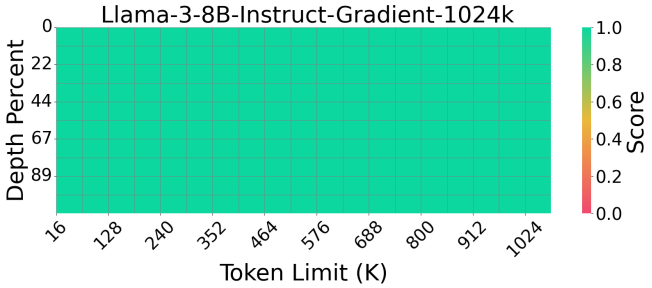

Figure 3. HEADINFER provides equal accuracy as standard inference on the Needle-in-a-Haystack benchmark

A.2. Long-Context Benchmarks Details

SCBench (Li et al., 2024) is a comprehensive suite of datasets encompassing multiple tasks, designed to assess long-context understanding capabilities. It contains subsets with different prompt lengths up to 500K tokens and 227K on average. As shown in Table 6, we measure their overall scores for different task categories. HEADINFER outperforms all other methods on all 9 tasks, with superior 1M context length.

Table 6. Performance(benchmark score) of different methods on **SCBench** on a single RTX-4090 GPU. kv and ps are string retrieval in key-value and prefix-suffix scenarios. vt is variable tracking. qa-chn, qa-eng are English/Chinese question answering. mf is finding the math answer. many-shot is finding multiple shots in context. summary is document summarization.

SCBench	kv	ps	vt	qa-chn	qa-eng	mf	many-shot	summary
Standard 25K	15.8	9.6	4.6	9.4	13.3	2.6	16.3	32.3
Chunked Prefill 30K	21.4	10.4	6.9	9.4	15.5	2.2	25.2	33.5
Layer-wise offload 45K	22.6	12.8	8.4	10.4	15.7	2.2	25.9	33.6
HEADINFER 1024K	**28**	**17.2**	**42**	**11.9**	**23.0**	**9.4**	**25.9**	**37.1**

Needle-in-a-Haystack (Kamradt, 2023) is a challenging pressure test designed to assess the ability of models to accurately identify and retrieve relevant information from a lengthy context. Figure 3 presents the results on Needle In A Haystack. We use 1024K for Llama-3-8B-Instruct-Gradient-1024k. HEADINFER can accurately recall the information queried within the context in question.

Ruler (Hsieh et al., 2024). Designed for a comprehensive evaluation of long context, the Ruler benchmark is a recent synthetic benchmark suite that includes 13 complex tasks in four main categories. Each context length variation includes 2,600 examples, with tests conducted at 4K, 8K, 16K, 32K, 64K, and 128K tokens. The benchmark comprises four key categories. The retrieval category includes various Needle-in-a-Haystack tasks: Single (S-NIAH) for finding individual key-value pairs in noisy text, Multi-keys (MK-NIAH) for retrieving specific values among hard distractors, Multi-values (MV-NIAH) for finding all values linked to a single key, and Multi-queries (MQ-NIAH) for retrieving values across multiple keys. The Multi-hop Tracing category features Variable Tracking (VT), requiring models to trace and return all variable names pointing to the same value through variable bindings. For aggregation, the benchmark includes Common Words Extraction (CWE) for identifying top-K common words from mixed sets and Frequent Words Extraction (FWE) for finding the most frequent words from a Zeta distribution. The Question Answering category extends traditional QA datasets by adding distracting paragraphs, testing models' ability to locate and utilize relevant information amid distractors.

Table 7. Maximum achievable sequence lengths for different inference methods

Method	Supported Sequence Length within Ruler
Standard Inference	16K
Chunked-Prefill	32K
Layer-wise Offload	32K
HEADINFER	128K

Table 7 demonstrates the maximum achievable sequence lengths for different inference methods on the Ruler benchmark. Standard inference, while straightforward, is limited to 16K tokens due to memory constraints. Both Chunked-Prefill and Layer-wise Offload methods double this capacity to 32K tokens through their respective optimization strategies. HEADINFER shows a significant advancement by enabling processing of sequences up to 128K tokens - a 4x improvement over other

offloading methods and a 8x improvement over standard inference. This extension in sequence length is achieved through HEADINFER 's novel head-wise offloading strategy, which more efficiently manages GPU memory utilization.

Table 8. Performance on Ruler benchmark tasks across different context lengths

Context Length	NIAH (%)	MK-2 (%)	MK-3 (%)	VT (%)	CWE (%)	FWE (%)	QA-1 (%)	QA-2 (%)
4K	100.0	99.6	100.0	99.20	99.38	94.53	84.6	59.8
8K	100.0	99.8	99.6	99.08	94.68	84.93	79.2	56.2
16K	100.0	100.0	99.4	98.72	56.90	90.60	79.6	53.2
32K	100.0	99.6	99.8	97.32	2.78	93.20	77.2	50.4
64K	100.0	97.4	97.8	92.48	0.10	84.27	76.0	49.4
128K	100.0	75.2	56.6	54.68	0.10	74.8	71.8	41.2

HEADINFER only

Table 8 presents a comprehensive analysis of performance across different tasks and context lengths. Since the ruler designs different tasks for different context lengths, the ruler's Multi-hop Tracing, Aggregation, and Question Answering become more difficult to complete in long sequence scenarios. This is different from scbench, which directly truncates long sequence tasks. However, even in this case, HEADINFER can still show the same performance as standard inference, which confirms its mathematical consistency.

A.3. Long-Context Video Generation Details

The video generation process with MAGI-1 4.5B follows an autoregressive approach where each video chunk is predicted sequentially. Our configuration includes:

- Resolution: 720x1280 pixels
- Frame Rate: 24 FPS
- Denoising Steps: 64 steps per frame for time experiment of sec 3.3.
- Prompt: "good boy" (consistent across all experiments)
- Hardware: Single NVIDIA RTX 4090 (24GB VRAM)

Figure 4. Generating 768 frames using MAGI-4.5B. The step of denoising is set as 64.

Figure 5. Generating 3072 frames using MAGI-4.5B. The step of denoising is set as 64.

Figures 4 and 5 showcase sample frames from our generated videos at different timestamps. Figure 4 displays frames from a 32-second video (768 frames), while Figure 5 demonstrates our extended generation capability with a 2-minute 8-second video (3072 frames). All videos are generated by HEADINFER implementation of MAGI-1 4.5B. We observe from Figures 4 the visual quality remains consistent throughout the generation process, with no noticeable degradation in later frames of 768 frames. However, with longer context up to 2.7 millions and 3072 frame, noticeable degradation is observed (good boy out of screen and transition exposure). It shows that HEADINFER provides a 4× context length extension. This enables the generation of videos exceeding 2 minutes in length (2,764,800 tokens) on a single consumer-grade RTX-4090 GPU.

A.4. Inference Memory Analysis

In this work, we compare several inference configurations, each targeting different trade-offs between memory efficiency and inference speed. While Figure 1 gives the theoretical benefit of HEADINFER compared to other methods in terms of memory usage, we also give how the theoretical memory is calculated here. Table 9 provides a comparative breakdown of memory usage across different inference strategies of Llama3-8B. Table 10 summarizes the memory usage comparison on the MAGI-4.5B model under various inference strategies. Larger models (e.g. Llama2-13B, Qwen2.5-32B) on larger GPU (e.g. A100 40B/8B) are presented on Table 11 and Table 12. These methods balance GPU memory consumption for weights, KV-cache, and activations while addressing the challenges of scaling to long-context inputs. Each strategy is outlined below:

Table 9. Memory usage comparison for Llama3-8B with 1 million context length

Method	Weight	KV-cache	Activation	Total	Total KV cache *
Standard	15.08	128	64	207	128
Chunked Prefill	15.08	128	0.625	143	128
4bit-KV-quant	15.08	32	64	111	32
layer-wise-offload	15.08	8	64	87	128
HEADINFER	15.08	1	0.625	16.7	128

*Total KV cache includes both GPU and CPU memory for offloading methods

Table 10. Memory usage comparison for Magi-4.5B with 1-min video and 1.4 million context length

Method	Weight	KV-cache	Activation	Total	Total KV-cache *
Standard	16.73	158.20	4.94	179.88	158.20
layer_wise-offload	16.73	9,89	4.94	31.56	158.20
HEADINFER	16.73	1.24	4.94	22.91	158.20

*Total KV cache includes both GPU and CPU memory for offloading methods

- **Standard**: The standard inference method keeps all weights, KV-cache, and activations entirely in GPU memory. With a context length of S, the KV-cache scales as $S \times L \times D_h \times H$, where L is the number of layers, D_h is the hidden dimension per head, and H is the number of attention heads. Activations require additional memory of $S \times D + 2 \times S \times I$, where I is the intermediate MLP dimension. While this approach achieves baseline performance, it is constrained by GPU memory limits (e.g., 128 GB for the KV-cache and 207GB for toal).

- **Chunked-Prefill**: By dividing the input into smaller chunks of size $\texttt{chunk}$, this method reduces activation memory from S to $\texttt{chunk}$. The memory footprint for activations is $\texttt{chunk} \times D + 2 \times \texttt{chunk} \times I$, where only part of the sequence resides in GPU memory during processing. Although the KV-cache size remains $S \times L \times D_h \times H$, this technique significantly lowers activation memory requirements, reduce total memory from 207GB to 143GB.

- **4-bit KV-Quant**: This method compresses the KV-cache from fp16/bf16 (16-bit floating point) to a 4-bit representation, reducing its size by a factor of 4. The memory usage becomes $S \times L \times D_h \times H/4$, while activations remain $S \times D + 2 \times S \times I$.

- **Layer-wise Offload**: This strategy offloads the KV-cache for entire layers to CPU memory as soon as computation for that layer is complete. On GPU, the memory required for KV-cache is reduced to $S \times D_h \times H \times 2$, the final 2 due to the ping-pong memory mechanism. However, offloading incurs communication overhead, making this approach more suitable for scenarios with sufficient PCIe bandwidth.

- **HEADINFER** : Our proposed approach achieves fine-grained control over memory by offloading KV-cache at the attention head level. With a ping-pong memory mechanism, the on-GPU KV-cache is reduced to $S \times D_h \times 2$, significantly lowering memory requirements. Activations are further minimized by combining chunked prefill with selective offloading. HEADINFER enables unprecedented scaling to 1M tokens and beyond, allowing context lengths of up to 4 million tokens with minimal GPU memory usage. Also, we observe the total estimated memory of 16.7 GB is close to the real measured memory of 17.0 GB, demonstrating the effectiveness of our estimation.

Table 11. Memory usage comparison for Llama2-13B with 1 million context length

Method	Weight	KV-cache	Activation	Total	A100-40GB
Standard	24.37	800	128	952.37	40
chunk-prefill	24.37	800	1.25	825.62	40
4bit-KV-quant	24.37	200	128	352.37	40
layer_wise-offload	24.37	40	128	192.37	40
HEADINFER	24.37	1	1.25	**26.62**	40

Table 12. Memory usage comparison for Qwen2.5-32B with 1 million context length

Method	Weight	KV-cache	Activation	Total	A100-80GB
Standard	61.03	256	236	553.03	80
chunk-prefill	61.03	256	2.33	319.36	80
4bit-KV-quant	61.03	64	236	361.03	80
layer_wise-offload	61.03	8	236	305.03	80
HEADINFER	61.03	1	2.33	**64.36**	80

Table 13 summarizes the GPU memory usage for weights, KV cache, and activations, as well as the total usage for in-device experiment. A detailed inference setting for each model and a comprehensive memory estimation of each method (including key hyperparameters, chunk sizes, and offloading policies) are provided here. As shown in table 13, we focus on five representative strategies for Llama-3-8B inference under varying context lengths:

Table 13. Memory consumption analysis for different inference methods (in GB) on Llama-3-8B

Method	Weights	KV cache	Activation	Total	Total KV cache *
Standard-25K	15.08	3.13	1.56	19.77	3.13
Chunked-Prefill-30K	15.08	3.75	0.63	19.46	3.75
4bit-KV-Quant-45K	15.08	1.41	2.81	19.30	1.41
Layer-Wise-Offload-45K	15.08	0.35	2.81	18.25	5.63
HEADINFER -4000K	15.08	3.91	0.63	19.61	500

*Total KV cache includes both GPU and CPU memory for offloading methods

- **Standard-25K:** Baseline approach with unmodified inference. All model weights, activations, and the entire KV cache stay on the GPU. Context length is limited by GPU memory.
- **Chunked-Prefill-30K:** Splits the prompt into sequential chunks to reduce activation overhead. KV cache remains on GPU. We use a chunk size (e.g., 4K or 8K tokens) such that each partial forward pass does not exceed GPU capacity.
- **4bit-KV-Quant-45K:** Applies a 4-bit quantization technique to KV cache, shrinking cache size by approximately 4× compared to FP16/BF16. However, the method can introduce additional on-GPU overhead in activation or conversion operations. In our tests, we adopt a standard quantization library (Hooper et al., 2024) for uniform 4-bit KV representation.
- **Layer-Wise-Offload-45K:** Offloads each layer's KV cache to CPU memory as soon as possible. During inference, only the KV cache for the currently processing layer is kept on the GPU. Once attention computation for that layer completes, its KV cache is swapped out to the CPU. This approach significantly lowers on-GPU KV cache usage but may incur additional offload overhead at each layer boundary.
- **HEADINFER -4000K:** Our proposed head-wise offloading approach (HEADINFER) that partitions the KV cache by attention heads. While some heads remain fully on GPU, the cache for other heads (or tokens) are immediately offloaded to CPU memory. Despite a very large total KV cache (reported as "500 GB" in Table 13), only a small fraction resides on the GPU at any time. This enables context lengths in the order of millions of tokens if sufficient CPU RAM is available.

Input Context(This is a very long story book):
...[repeating of long paragraph]...
HeadInfer's favorite number is 42
...[repeating of long paragraph]...

Question:	**What is HeadInfer's favorite number?**	
Original model:	HeadInfer's favorite number is 42	☺
StreamLLM:	HeadInfer's favorite number is not provided	☹
H2O:	zzz...(repeating meaningless sign)	☹
HeadInfer:	**HeadInfer's favorite number is 42**	☺
HeadInfer + 50% Sparsity:	**HeadInfer's favorite number is 42**	☺

Figure 6. Token eviction methods cannot work when querying the less relevant information to the main theme. Here, we use a 10K document from LongBench (Bai et al., 2023b) and add one sentence that is not relevant to the main theme. In this case, H2O discards tokens less relevant to the main theme, leading to error generation. StreamingLLM discards tokens based on the query but remaining question tokens, making it Hallucinations. HEADINFER can successfully output the exact information from the lengthy input, even when we compress 75% of the KV cache

For total memory we define:

$$M_{\text{total}}; =; M_{\text{weights}} + M_{\text{KV cache}} + M_{\text{activation}}.$$

Weights (*e.g.*, 15.08GB in Llama-3-8B) remain constant for all methods. The **KV cache** size grows with the number of tokens processed (sequence length) but can be reduced or offloaded depending on the method. **Activation** memory arises from forward-pass intermediate tensors (attention blocks, MLP layers, etc.) and is partially minimized by chunking or parallelization strategies.

In Table 13, *KV cache* and *Activation* columns refer to the approximate GPU memory usage during inference; any data offloaded to CPU memory (or disk) is not included in these columns but is counted in *Total KV cache* if applicable. For instance, **HEADINFER** has a *GPU-side* KV cache usage of about 3.9GB at any instant, whereas the *overall* KV cache across CPU and GPU is up to 500GB, enabling very long context windows.

B. Motivation and Related Work

In this section, we first explain that the KV cache size becomes a critical issue for long-text generation in LLM inference, and it becomes more problematic when deploying modern offloading-based inference systems. We then discuss why the existing KV cache management methods cannot fundamentally address the problem in an offloading-based system.

B.1. Memory Requirements for Inference

This section characterizes the memory requirements for transformer inference. It can be categorized into two components: i) Model Parameters and ii) Activation memory primarily referring to KV cache. The memory requirement for model parameters primarily depends on the hidden dimension D, the number of attention heads, and the number of Transformer layers L. Nearly all the parameters in a Transformer block come from linear layers within L attention blocks, L multilayer perceptron (MLP) blocks, and one language modeling head (LM-head) block. Take Llama-3-8B as an example; the total parameters in a transformer-based model can be approximated as $14 \times L \times HD^2$ with 15GB of memory. The memory required for activation memory primarily consists of the KV cache, which depends on the model architecture, batch size B, and sequence length S, and it can be pretty significant. The memory can be estimated as $2 \times B \times S \times H \times D$. For instance, in the Llama-3-8B (Dubey et al., 2024) model architecture, serving with FP16 KV cache for 1 million tokens would require at least 207 GB of memory—exceeding the capacity of a single 80GB GPU.

B.2. KV Cache in LLM Inference Systems

As discussed in the previous section, today's LLM serving systems exploit KV caching to avoid redundant computation of key and value projections during the chunked-prefill decoding stage. While this is an effective solution for short sequence generation with a single client request, the KV cache quickly becomes a key memory consumer when we generate long sequences or employ modern request batching techniques (Sheng et al., 2023).

In the Llama-3-8B (Dubey et al., 2024) model architecture, serving with FP16 KV cache for 1 million tokens would require at least 256 GB of memory—exceeding the capacity of a single 80GB GPU. Additionally, the latencies of pre-filling and decoding with such large contexts are significant, posing substantial challenges to the effective use of LLMs in long-context scenarios.

The rapidly expanding KV cache leads to an urgent need and numerous efforts for KV cache compression, particularly in scenarios with limited GPU memory. Architectural modifications, such as Grouped-Query Attention (Ainslie et al., 2023), Rope (Su et al., 2024), LongNet (Ding et al., 2023), MST (Luo et al., 2024), LoQT (Loeschcke et al., 2024), Lora (Hu et al., 2021) and Galore (Zhao et al., 2024) require expensive model pre-training. One direction is non-Transformer architecture design, such as Mamba (Gu & Dao, 2023), Linear Attention (Katharopoulos et al., 2020), RWKV (Peng et al., 2023). However, the transformer is still the most widely used model structure, and in this paper, we focus on KV cache reduction for typical transformers. One direction is to identify and retain only the most 'valuable' tokens within the KV cache. Representative methods include Sliding Window Attention (Beltagy et al., 2020), Heavy Hitter (Zhang et al., 2023), and StreamingLLM (Xiao et al., 2024b). Another direction is to identify and retain the attention heads. Wu et al. (Wu et al., 2024) find a way to evaluate the importance of attention heads. Head-wise sparsity such as duo-attention (Xiao et al., 2024a), HeadKV (Fu et al., 2024), and Razorattention (Tang et al., 2024) start to divide up KV cache budgets based on the importance of each head, which is usually determined by the need for retrieval or reasoning. Minference (Jiang et al., 2024) takes this idea further by applying distinct sparse patterns to different heads.

LLM Inference Systems with Offloading. In modern GPU systems, there is a significant disparity between CPU and GPU memory capacities and costs. CPU RAM typically offers much larger capacity at a lower cost than GPU memory. For example, modern server-grade systems can easily accommodate 1-2TB of CPU RAM, while even high-end GPU cards like the NVIDIA A100 are limited to 80GB of memory. The cost difference is also substantial—server-grade DDR4/DDR5 RAM typically costs a fraction per gigabyte compared to specialized GPU memory.

This observation is supported by several prominent works offloading the KV cache to the CPU memory: FlexGen (Sheng et al., 2023) leverages this hardware characteristic by utilizing CPU memory as an extension of GPU memory, allowing for efficient LLM inference even with limited GPU resources. DeepSpeed (Aminabadi et al., 2022) implements sophisticated offloading strategies that take advantage of the larger CPU memory capacity to handle model weights and KV cache that would not fit in GPU memory alone. Infinitigen (Lee et al., 2024) builds on these foundations by introducing dynamic KV cache management that works synergistically with offloading systems, but its efficiency is highly related to token eviction.

However, this memory hierarchy presents a trade-off: while CPU memory provides larger capacity at lower cost, data transfer between CPU and GPU over PCIe becomes a potential bottleneck due to limited bandwidth compared to GPU memory access speeds. This necessitates careful data movement management and strategic offloading decisions to maintain efficient inference performance.

B.3. Challenges in KV Cache Management

In this context, several recent works propose reducing the KV cache size through retrieval head evictions. However, all the prior works assume the persistence of attention patterns across layers, that is, if a head is deemed a retrieval head.

KV token eviction affects long-context performance. Figure 6 shows significant performance degradation since the actual information required by the query might be discarded if considered unimportant, which uses the KV cache of all prior tokens for computing attention results, and the KV cache management method of H2O with a KV cache budget of 2000 tokens. H2O (Zhang et al., 2023) is a state-of-the-art technique that retains only a small percentage of important tokens in the KV cache to reduce its size. It assesses the importance of each token in every iteration and removes unimportant ones before the next iteration to keep the KV cache size in check.

The figure indicates that this is not the practice case, despite H2O-like approaches assuming that the attention pattern does not change across heads. The tokens deemed unimportant in the one-head iteration could become important in other heads.

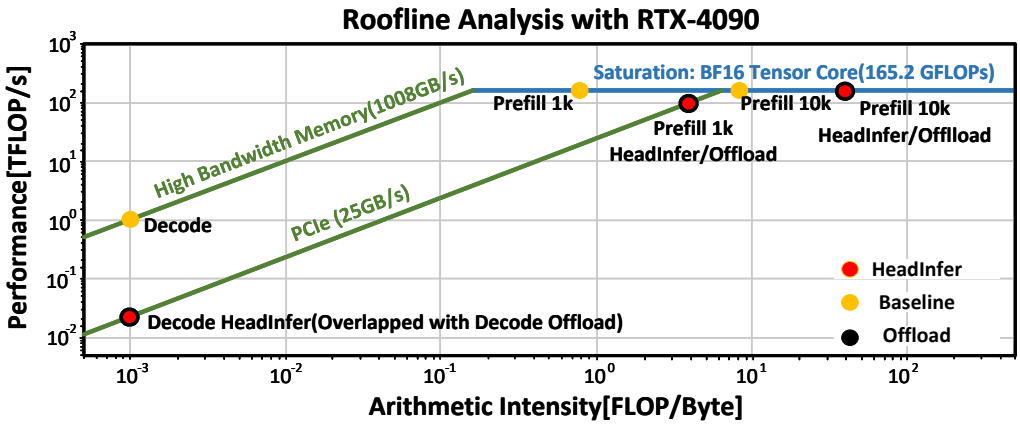

Figure 7. Flashattention in the roofline plot analysis using the RTX-4090 device setting.

Consequently, H2O exhibits high similarity until around 200 iterations (i.e., within the KV cache budget). However, as the sequence length extends beyond the KV cache budget, it struggles with the attention pattern's dynamic nature, resulting in more error generation than the optimal case. Note that while we only show the scenario of a KV cache budget of 2000 out of a total sequence length of 10000 tokens for brevity, this issue would become more pronounced as the sequence length surpasses it.

Prior works aiming to reduce the KV cache size through token eviction inherently have some challenges. Given the dynamic attention pattern across iterations, permanently excluding evicted tokens from retinal head token generation can result in a non-negligible drop in accuracy. Instead, we must keep the full attention tokens from the retrieval head while selectively evicting less important heads. Furthermore, prior works' iterative allocation of KV cache memory leads to inefficient KV cache management. The number of key/value tokens required increases during chunked-prefill, and each extended context inference demands effective memory management. Failing to account for this allocation may result in ineffective KV cache management. Thus, we need to adjust the memory of key/value token pre-allocation while considering the variances between retrieval and full head.

C. Roofline Model for head-wise flash attention

Although KV cache offload is proposed to reduce memory usage, it remains an open question whether offloading harms overall performance, especially when the context length S is large and works with chunked-prefill. This section analyzes the theoretical peak performance for a given GPU under constrained high-bandwidth memory (HBM) and peripheral component interconnect express (PCIe).

Performance Model. We consider a GPU characterized by HBM capacity M_{HBM}, memory bandwidth B_{mem}, and compute throughput F_{GPU} (measured in FLOPS). We also incorporate the slower PCIe bandwidth B_{pcie} into the performance model to account for the offload.

Memory Bound vs. Compute Bound. GPU operators can be classified as compute bound or memory bound, which is determined by the time spent in arithmetic operations and the time spent accessing HBM. Two primary bottlenecks define the system's performance regime:

- Memory-Bound: When B_{mem} (memory bandwidth) is insufficient to transfer the KV cache quickly enough, inference operates below the GPU's peak FLOPS capacity.
- Compute-Bound: When B_{mem} is sufficient, high compute efficiency is achieved and the throughput is determined by the GPU's peak computation rate F_{GPU}.

Roofline Model. The roofline plot typically displays a kernel's computation throughput and arithmetic intensity, providing a visual representation of its hardware performance. We present the roofline analysis for the FlashAttention kernel (Dao et al., 2022; Dao, 2023; Shah et al., 2024) executed on an RTX-4090 (Figure 7), with details in Appendix C. We believe that the roofline model can analyze both GPU and heterogeneous systems performance.

Our key observations from roofline analysis are:

- **Prefill (compute-bound behavior).** "Prefill HEADINFER /Offload" has higher arithmetic intensity than "Prefill" as it only offloads KV cache, and the arithmetic intensity grows as context length increases. For context lengths $S \geq 10\text{k}$, prefill remains compute-bound, even when offloading via slower PCIe bandwidth. In contrast, for short context lengths ($S \leq 1\text{k}$), "HEADINFER /Offload" shifts to the memory-bound regime, leading to offload-induced slowdowns. The turning point is achieved at 2K, and 10K can ensure high computational efficiency.

- **Decoding (memory-bound behavior).** Decoding performance is primarily memory-bound. Consequently, relying on PCIe bandwidth during offloading substantially degrades overall throughput.

- **HEADINFER (unchanged behavior).** HEADINFER 's head-wise offloading during both chunked-prefill and decoding does not alter the position of the roofline plot due to the independence of the attention heads. Although head-wise computation impacts GPU utilization, the adaptive head strategy can preserve performance.

The Roofline model (Williams et al., 2009) serves as an effective theoretical framework to assess the potential performance of deploying a model on particular hardware. Here we evaluate hardware performance of memory access and processing unit capabilities.

Table 14. Performance comparison of different attention mechanisms under RTX-4090 setting

Operator	Regular				Offload			
	Memory	Arithmetic Intensity	FLOP	Bound	Memory	Arithmetic Intensity	FLOP	Bound
Prefill								
flash-attn (1k)	21M	820	165T	comp	4.2M	4100	102T	mem
flash-attn (10k)	209M	8200	165T	comp	42M	41000	165T	comp
flash-attn (100k)	2.1G	82000	165T	comp	419M	410000	165T	comp
head-wise (1k)	2.6M	820	165T	comp	0.5M	4100	102T	mem
head-wise (10k)	26M	8200	165T	comp	5.2M	41000	312T	comp
head-wise (100k)	262M	82000	165T	comp	52M	410000	312T	comp
Decode								
flash-attn (1k)	17M	1	1T	mem	17M	1	13G	mem
flash-attn (10k)	168M	1	1T	mem	168M	1	13G	mem
flash-attn (100k)	1.7G	1	1T	mem	1.7G	1	13G	mem
head-wise (1k)	2.1M	1	1T	mem	2.1M	1	13G	mem
head-wise (10k	21M	1	1T	mem	21M	1	13G	mem
head-wise (100k)	210M	1	1T	mem	210M	1	13G	mem

Table 14 presents the analysis of layers in Llama-3-8b. From the table, we observe that during the prefill stage, the majority of computations are compute-bound, leading to high performance. Conversely, in the decode stage, all computations are memory-bound, resulting in performance significantly below the computational capacity of the GPU's computation units. Moreover, offload would make small context prefill memory-bound. Head-wise, the roofline model performs the same arithmetic intensity and peak performance as standard FlashAttention.

We also show the roofline analysis on other GPUs, such as the A100, to demonstrate the generality of this analysis. Figure 8 shows that the prefill 1K offload is also positioned on memory-bound while the prefill 10K offload is on the compute-bound. The details data is listed on table 15.

D. Extension: HEADINFER Implementation with Head-wise Sparsity

Figure 9 shows our memory management framework. HEADINFER , which enables offloading the head-wise KV cache with head-wise sparsity. The key design principle behind HEADINFER is to exploit the redundancy of CPU memory capacity to increase the context size after identifying the important heads in the KV cache. As such, most of the heads for the KV cache are kept in the CPU memory as we generate new tokens, not discarding them like previous work. However, we do not bring the entire KV cache to the GPU for attention but load and compute only the retrieval head of keys and values, leaving other non-retrieval ones staying on the GPU without offloading. To do so, we maintain the head-wise cache pool in the CPU memory and iteratively load the necessary data.

Table 15. Performance comparison of different attention mechanisms under A100 setting

Operator	Regular				Offload			
	Memory	Arithmetic Intensity	FLOP	Bound	Memory	Arithmetic Intensity	FLOP	Bound
Prefill								
flash-attn (1k)	21M	820	312T	comp	4.2M	4100	102T	mem
flash-attn (10k)	209M	8200	312T	comp	42M	41000	312T	comp
flash-attn (100k)	2.1G	82000	312T	comp	419M	410000	312T	comp
head-wise (1k)	2.6M	820	312T	comp	0.5M	4100	102T	mem
head-wise (10k)	26M	8200	312T	comp	5.2M	41000	312T	comp
head-wise (100k)	262M	82000	312T	comp	52M	410000	312T	comp
Decode								
flash-attn (1k)	17M	1	1.4T	mem	17M	1	23G	mem
flash-attn (10k)	168M	1	1.4T	mem	168M	1	23G	mem
flash-attn (100k)	1.7G	1	1.4T	mem	1.7G	1	23G	mem
head-wise (1k)	2.1M	1	1.4T	mem	2.1M	1	23G	mem
head-wise (10k)	21M	1	1.4T	mem	21M	1	23G	mem
head-wise (100k)	210M	1	1.4T	mem	210M	1	23G	mem

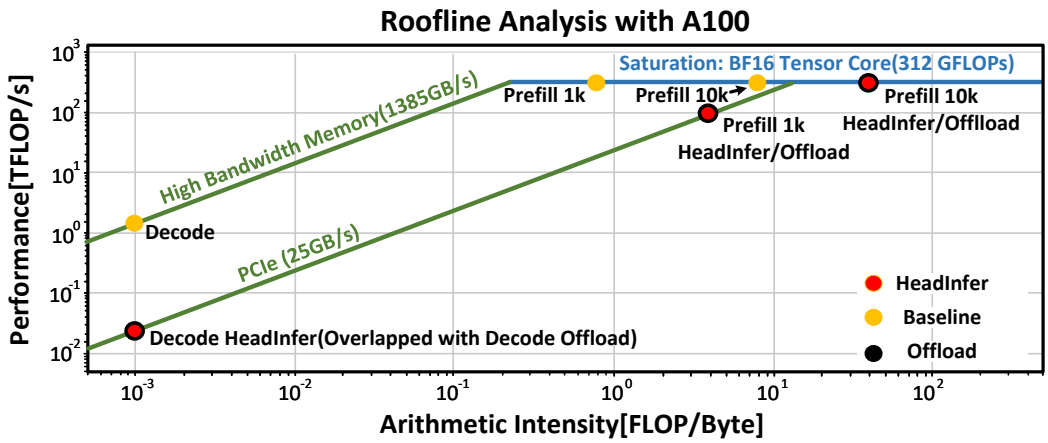

Figure 8. Flashattention in the roofline plot analysis using A100 device setting.

In detail, we use the pre-trained attention input head to speculate the important retrieval head. The speculation is done by processing a customized dataset and analyzing the output. This reduces the waste of PCIe bandwidth by only transferring retrieval heads critical for attention computation. In addition, although the data is offloaded to CPU memory, which is much cheaper and larger than GPU memory, we manage the KV cache pool size so as not to fully utilize the CPU memory.

D.1. Design Principles

Interleaved KV cache Updates Across GPU and CPU. The uneven memory consumption and low PCIe link utilization (studied in section 3) during different attention head generation provide an opportunity to exploit the idle GPU memory and PCIe link during the KV cache update phase. To exploit this opportunity, during the attention processing, a head of the attention KV cache can be dynamically fetched on the GPU to compute the attention weight output in parallel while the CPU prefetches the next head. A key requirement to generate the attention output for a given attention head is to stage its parameters (p), query (q), key (k), and value (v), and the attention weight generation is scheduled for each head. In case the key (k) and value (v) of the head are not present on the GPU, the generation operation will trigger a prefetch read from the CPU memory where the head is offloaded, causing I/O operations in the critical execution path of updates. By leveraging the fact that multiple head attention, such as MHA (Vaswani et al., 2017) and GQA (Ainslie et al., 2023), are embarrassingly parallel, and HEADINFER partitions the attention into smaller subgroups, we can perform fine-grained attention generation scheduling across both GPU and CPU without impacting the consistency of generation or introducing computational dependencies between different subgroups. Furthermore, interleaving does not incur memory allocation and deallocation overheads because on the GPU, memory allocation is handled by PyTorch through lightweight memory pools, and on the host, the memory for all subgroups (except static GPU subgroups) is already pre-allocated and pre-pinned (if

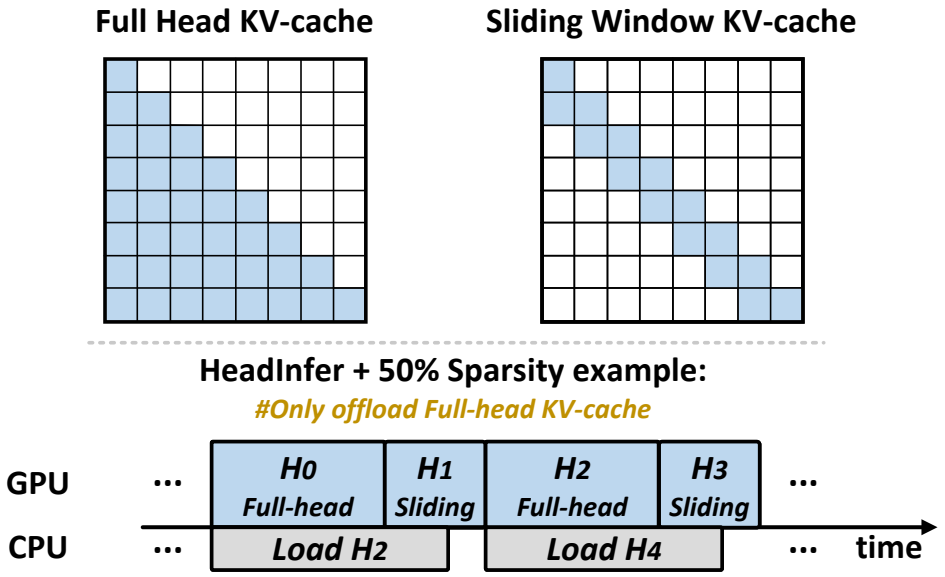

Figure 9. Demonstrations of KV cache policies in inference from the head-wise view. Upper plots illustrate symbolic plots of an attention map deploying different policies in LLM generation. Lower: the overview of HEADINFER .

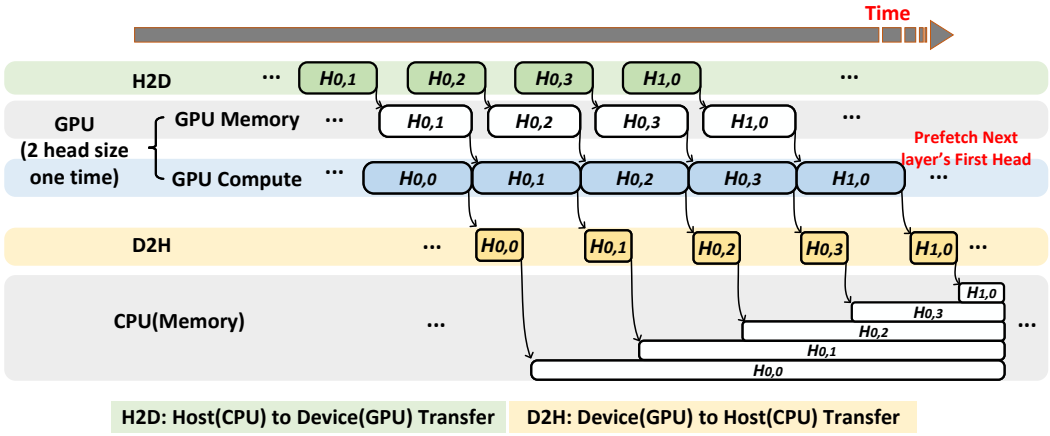

Figure 10. Workflow of HEADINFER generating a model with (n+1) layers and (j+1) attention heads.

enabled) during initialization.

As illustrated in Figure 10, the attention is partitioned into 4 sub-heads, out of which the first head is statically placed on the GPU for the entire inference lifetime; the KV cache corresponding to the head resides in the GPU memory. Therefore, the interleaved offloading adopted by HEADINFER scheduled all the heads to be updated on the GPU, i.e., for every head updated on the GPU, the host-to-device transform for the next head and the device-to-host transform for the previous head would occur in a non-blocking fashion. This interleaved offloading makes sure that only two heads are maintained on the GPU while most KV cache are offloaded on the CPU. This interleave-centric design allows for efficient overlap between GPU computations and asynchronous head-wise KV cache movement across the PCIe link, which we will detail next.

Overlapping Head-wise KV cache Movement and GPU Compute The data movement observed when the state-of-the-art framework enabling layer-wise KV cache offload (e.g., Flexgen(Sheng et al., 2023) and Deepspeed(Aminabadi et al., 2022)) runs an inference process. We observe that after the updates to the KV cache generation to a given layer i are computed on the GPU, the updated KV cache is H2D transferred to the CPU to continue inference in the subsequent chunked-prefill and decoding. Only when all the KV cache of next later are transferred to the GPU can the subsequent iteration begin. Given the parallel nature of attention, the sub-heads can be updated and transferred out of order and do not impact the accuracy of the inference. On contract, using the existing offloading solutions can be slow with head-wise sparsity attributed to (a) the KV

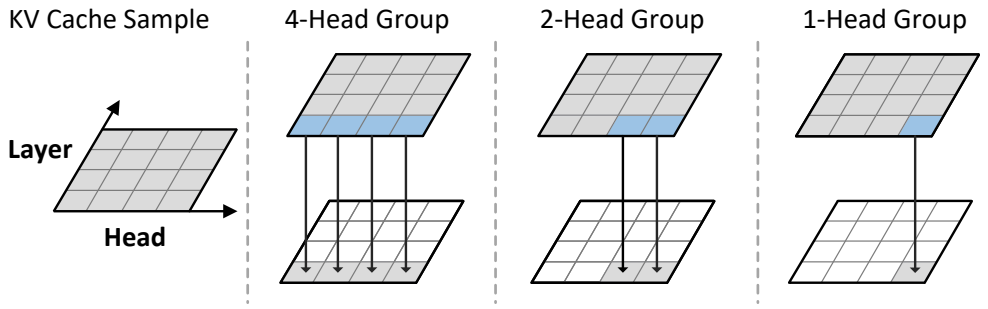

Figure 11. Demonstrations of adaptive head-wise offloading.

cache of all heads within one layer staying on the GPU, which would occupy large memory for long context, and (b) not all attention layers are discretized uniformly, which causes blocking H2D transfer of KV cache.

To mitigate the aforementioned challenges, we propose a head-centric design illustrated in Figure 10 for efficient offloading interleaving of H2D transfor and GPU compute. It works as follows: while the GPU computes the generation of the initial head (H^1), the KV cache corresponding to the next head (H^2), including key (k) and value (v), are being prefetched using asynchronous H2D transfers, thereby overlapping GPU computations with next-head prefetching. Meanwhile, the CPU update for previous H^0 is being uploaded using asynchronous D2H transfers. After this, three operations happen in parallel: (1) H2D transfer of the next head and prefetching of the next KV cache to be updated on the GPU (2) Updating of the previous head with KV cache from GPU; and (3) GPU generation of current head outputs, thereby exploiting full-duplex D2H and H2D transfers and parallel GPU computations.

The attention phase is executed in a parallel fashion by multiple heads of the mechisan. Consequently, our proposed overlapping of GPU computations and PCIe transfers does not incur any blocking overheads, as the computation speed on the GPU is slower than the PCIe throughput to transfer subheads back and forth between the GPU and CPU using high-throughput PCIe.

Efficient Management of KV cache State-of-the-art hybrid KV cache offloading solutions (e.g., FlexFLow) by default retain the layer-wise KV cache corresponding to the statically GPU-resident head (h1 to h4) on the GPU during the attention computation, and for the remainder of the layer, KV cache are offloaded to the host memory. We extend this design with head-wise KV cache management to incorporate head-wise sparsity. Head-wise sparsity divides attention heads into two categories: important retrieval heads retain all tokens, while unimportant The streaming head only retains the most recent token. Since the memory usage of the retrieval head will be much larger than that of the streaming head, especially under long context inference, HEADINFER will selectively offload the retrieval head to the CPU and keep the streaming head in the GPU.

Adaptive selecting head-wise granularity. In transformer-based architectures, attention heads often operate in parallel. However, processing these heads individually—or in small groups—can incur repeated kernel launches and excessive PCIe transfers, particularly if the context size for each head is small. This overhead can quickly dominate total inference time, undermining the benefits of parallelism. Adaptive head-wise offloading addresses these inefficiencies by merging multiple heads into a single "HeadGroup." As shown on Figure 11, with a reduced number of HeadGroups (e.g., 2-HeadGroup or 4-HeadGroup), fewer offloading operations are required, lowering both the latency and the bandwidth usage on the CPU–GPU boundary. Conceptually, this "batching" of heads takes advantage of the fact that many computations within attention heads are structurally similar, thus allowing shared memory transfers and kernel calls. Layer-offload can be viewed as the extreme case of adaptive head-wise offloading. Instead of grouping only a fraction of heads at a time, layer-offload groups every head in a given layer into one offload operation. While it can drastically reduce overhead further, the trade-off is less granularity and potentially higher intermediate memory requirements. In practice, the decision to use 2-head, 4-head, or full-layer offloading depends on available hardware resources, batch size, and the typical context length being processed.

By carefully tuning the grouping strategy—ranging from small, flexible head groups to large, layer-wide groups—adaptive head-wise offloading makes it possible to significantly optimize inference time in transformer-based models, particularly in latency-sensitive scenarios involving small context.