

LongFlow: Efficient KV Cache Compression for Reasoning Models

Yi Su^{1*}, Zhenxu Tian^{1*}, Dan Qiao², Yuechi Zhou¹

Zheming Yang², Juntao Li^{1†}, Min Zhang¹

¹School of Computer Science and Technology, Soochow University, China

²ByteDance

yisunlp@outlook.com

Abstract

Recent reasoning models such as OpenAI-o1 and DeepSeek-R1 have shown strong performance on complex tasks including mathematical reasoning and code generation. However, this performance gain comes with substantially longer output sequences, leading to significantly increased deployment costs. In particular, long outputs require large KV caches, resulting in high memory consumption and severe bandwidth pressure during attention computation. Most existing KV cache optimization methods are designed for long-input, short-output scenarios and are ineffective for the long-output setting of reasoning models. Moreover, importance estimation in prior work is computationally expensive and becomes prohibitive when continuous re-evaluation is required during long generation. To address these challenges, we propose **LongFlow**, a KV cache compression method with an efficient importance estimation metric derived from an intermediate result of attention computation using only the current query. This design introduces negligible computational overhead and requires no auxiliary storage. We further develop a custom kernel that fuses FlashAttention, importance estimation, and token eviction into a single optimized operator, improving system-level efficiency. Experiments show that LongFlow achieves up to an 11.8× throughput improvement with 80% KV cache compression with minimal impact on model accuracy.¹

1 Introduction

The rapid advancement of Large Language Models (LLMs) (Yang et al., 2024; Grattafiori et al., 2024; Hurst et al., 2024; Liu et al., 2024a) has marked a pivotal stage in artificial intelligence, achieving state-of-the-art performance across a wide range of

tasks. Recently, a new class of **reasoning models** has emerged to explicitly tackle complex reasoning problems (Guo et al., 2025; Jaech et al., 2024; Team et al., 2025; Yang et al., 2025; Meta, 2025). These models rely on long Chain-of-Thought (CoT) reasoning and generate extensive intermediate steps during inference. While highly effective in domains such as mathematical reasoning and code generation, they introduce new challenges for efficient training and deployment.

A key drawback of reasoning models is their tendency to generate a large number of output tokens (Chen et al., 2024; Feng et al., 2025; Wang et al., 2025), which substantially inflates the KV cache and leads to severe memory and bandwidth bottlenecks during attention computation. Although prior work has explored KV cache compression, most methods are designed for long-input, short-output scenarios and are ill-suited for long-output generation. Some approaches only compress the KV cache during the prefill stage (Li et al., 2024; Cai et al., 2024; Su et al., 2025), while others incur significant computational overhead and require additional auxiliary storage (Zhang et al., 2023; Guo et al., 2024). These limitations are particularly pronounced in long-output settings, highlighting the need for more efficient KV cache compression methods for reasoning models.

To address this gap, we propose **LongFlow**, an efficient KV cache compression method tailored for long-output generation. The core idea of LongFlow is a fast yet accurate importance estimation metric for historical tokens, derived directly from an intermediate value of the standard attention computation using only the current query. This design requires no auxiliary storage and introduces negligible computational overhead. We provide both theoretical justification, showing that the metric approximates attention output loss, and empirical validation through extensive experiments.

Beyond the algorithmic design, we introduce sev-

* Equal Contribution.

† Corresponding author.

¹Code is available at <https://github.com/yisunlp/LongFlow>.

eral system-level optimizations to maximize practical efficiency. We adopt a static KV cache that pre-allocates memory to avoid fragmentation and dynamic allocation overhead. To further improve hardware utilization, we implement a custom Triton kernel that fuses FlashAttention, importance estimation, and token eviction into a single optimized operator, reducing attention latency from 47 ms to 8 ms (Figure 1). Overall, LongFlow achieves an $11.8\times$ throughput improvement with an 80% KV cache compression ratio, while preserving most of the model accuracy.

Our contributions are summarized as follows:

- **A lightweight KV cache compression algorithm for long-output generation.** We propose LongFlow, which introduces an accurate importance metric computed efficiently from the current query and intermediate attention results, with negligible overhead.
- **A fused, high-performance attention kernel.** We design a custom Triton kernel that integrates attention computation, importance estimation, and token eviction into a single optimized operator.
- **State-of-the-art efficiency for reasoning models.** Experiments demonstrate up to an $11.8\times$ throughput improvement and an 80% reduction in KV cache size, with minimal impact on model accuracy.

2 Preliminary

2.1 Background: Attention, KV cache, and Hardware-Aware Optimization

The attention mechanism (Vaswani, 2017) is a core component of LLMs. Given an input sequence $\mathbf{X} \in \mathbb{R}^{b \times s \times d}$, where b is the batch size, s is the sequence length, and d is the hidden dimension, it is first projected into $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ by learnable weight matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d \times d}$. For simplicity, we omit the number of the attention heads. The attention output is computed using the Scaled Dot-Product Attention mechanism as follows:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\text{Mask} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \right) \mathbf{V} \quad (1)$$

where d_k is the dimension of the key vectors. However, this attention mechanism relies on all past $\mathbf{K}$ and $\mathbf{V}$ when calculating the output of the current step. Therefore, LLMs typically utilize KV cache to accelerate auto-regressive decoding. The generation process of LLMs with KV cache is divided

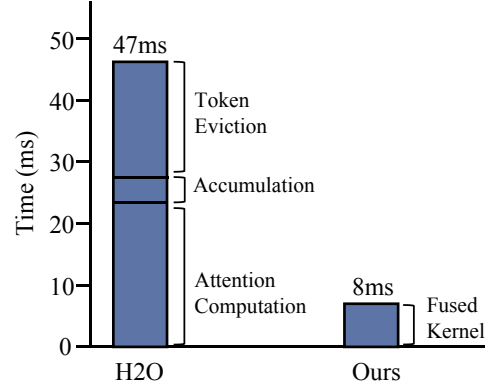


Figure 1: Attention module latency of H2O and our kernel on Qwen3-8B with batch size 128 and sequence length 3200. Both methods evict one token after each attention computation.

into the prefill phase and the decoding phase (Patel et al., 2024).

i) During the prefill phase, given input $\mathbf{X}$, the Keys $\mathbf{K}_{<n}$ and Values $\mathbf{V}_{<n}$ are computed and cached. ii) During the decoding phase, only the Keys and Values of the new token x_n need to be calculated, which are then combined with the cached Keys and Values to execute attention computation.

From Equation 1, we can find that the classical attention computation is a memory-bound operation, as it requires multiple intermediate value accesses in HBM. Fortunately, Flash-Attention (Dao et al., 2022; Dao, 2024) uses IO-aware operations to minimize the number of read and write operations to HBM. By processing the computation in blocks (tiling) and leveraging on-chip SRAM to store intermediate results, Flash-Attention avoids materializing the full $N \times N$ attention score matrix in HBM. This dramatically reduces memory traffic, leading to significant speedups and a lower memory footprint. Inspired by this IO-aware approach, our custom kernel also employs a block-wise computation strategy to ensure maximum efficiency, as illustrated in Figure 2.

2.2 Revisiting KV cache Compression in the Era of Long Reasoning Models

While prior KV cache compression techniques have demonstrated considerable success in the traditional paradigm of long-input, short-output tasks, their core design assumptions are fundamentally challenged by the rise of long reasoning models. The shift to a long-output paradigm exposes several critical limitations in these established methods:

Prefill-Only Compression. Many methods are

confined to the prefill stage, compressing only the initial prompt (Cai et al., 2024). In long-output scenarios, where most of the tokens are generated during the decoding phase, this approach results in a drastically diminished compression ratio.

High Computational and Memory Overhead.

Existing methods rely on token importance for eviction. The importance evaluation process is computationally expensive, adding significant latency to each compression step (Guo et al., 2024; Cai et al., 2025). Moreover, these methods often require storing auxiliary metadata, such as attention scores (Zhang et al., 2023) or past queries (Li et al., 2024), which consumes extra memory and partially cancels out the benefits of compression.

Poor Compatibility with Modern Fused Kernels.

The performance of modern inference systems heavily relies on operator fusion like Flash-Attention (Dao et al., 2022; Dao, 2024) or Paged-Attention (Kwon et al., 2023). However, many compression algorithms are not compatible with these kernels. Their logic can interrupt the fusion process, forcing data to be moved between SRAM and HBM, or they may require recalculating intermediate results that were already available during the attention step, which can make the overall inference process even slower.

These challenges require a complete redesign of KV cache compression based on the principles of dynamic applicability, lower overhead, and system-level implementation.

3 Method

In this section, we introduce LongFlow, a novel KV cache compression method designed to overcome the limitations of prior work. Our approach is built on three pillars: a lightweight design philosophy, a rigorous theoretical derivation, and a high-performance system implementation.

3.1 A Lightweight Design Philosophy

The design of LongFlow is guided by a lightweight philosophy, standing in stark contrast to the conventional "look-back" approach of many prior methods. These methods assume that an accurate importance estimation requires aggregating historical information. As previously analyzed, this inherent dependency on historical data leads to prohibitive computational and memory overheads. To break from this costly paradigm, we built LongFlow upon two core principles:

The Principle of Zero-History Estimation. Our central hypothesis is that the current query, $\mathbf{q}_t$ contains sufficient information to effectively estimate the importance of all historical tokens. To validate this premise, we conduct a preliminary analysis of SnapKV (Li et al., 2024). We evaluate its performance on the LongBench benchmark (Bai et al., 2023) while reducing the size of its query observation window. The results in Figure 5a in Appendix show that the performance of SnapKV degrades only slightly as the query window gets smaller, which empirically demonstrates its remarkable robustness. This evidence highlights the validity of the single-query method for importance estimation, forming the empirical foundation for our lightweight design.

The Principle of Zero-Cost Estimation The second principle guiding LongFlow’s design is Zero-Cost Estimation. This principle posits that the compression mechanism should not be an expensive, standalone process, but rather an intrinsic and nearly-free byproduct of the main attention computation. This stands in contrast to prior methods that often treat the importance estimation as a distinct step performed after the attention calculation. Our method implements this principle by deriving the importance metric directly from an intermediate value of the standard attention forward pass which allows us to reuse the values that must be computed for the attention output. The reuse eliminates the need for auxiliary storage and ensures that the compression step adds no overhead.

3.2 Derivation of the Importance Metric

Our derivation of the importance metric begins with a formal definition of the eviction objective. Ideally, at any given step, we should evict the token whose removal has the minimal impact on the final logits across all future generation steps. This globally optimal objective is computationally intractable. To form a tractable objective, we first simplify our goal to minimizing the impact on the immediate next step’s attention output, $\mathbf{o}_{t+1}$:

$$\arg \min_i \left\| \mathbf{o}_{t+1} - \mathbf{o}_{t+1}^{(i)} \right\|^2, \quad (2)$$

where $\mathbf{o}_{t+1}^{(i)}$ is the new attention output computed without the key-value pair of token t_i .

Although this objective is greatly simpler, it is still an unrealizable objective as it depends on the future query $\mathbf{q}_{t+1}$. Fortunately, we find a good

approximation for this objective. In our preliminary exploration, we find that the adjacent queries $(\mathbf{q}_t, \mathbf{q}_{t+1})$ have a high similarity (see Figure 5b in Appendix, similar observation can be found in Su et al. (2025)). Intuitively, since the attention output is a weighted sum of Value vectors driven by the attention scores $\mathbf{q}_t^T \mathbf{k}$, the strong similarity between adjacent queries implies that the overall attention distribution will be stable. This stability suggests that the impact of evicting a token at step t is an excellent proxy for its impact at step $t + 1$ if we omit the softmax denominator. We thus formalize our practical approximation as follows:

$$\left\| \mathbf{o}_{t+1} - \mathbf{o}_{t+1}^{(\setminus i)} \right\|^2 \approx \left\| \mathbf{o}_t - \mathbf{o}_t^{(\setminus i)} \right\|^2, \quad \text{where } i < t. \quad (3)$$

A formal error analysis of this approximation is presented in Section 3.3.

Directly computing this objective for all candidate tokens remains prohibitively expensive. To derive an efficient proxy, we analyze the structure of the attention output. The exact change in the output $\mathbf{o}_t$ after evicting a historical token t_i is:

$$\Delta \mathbf{o}_t = \mathbf{o}_t - \mathbf{o}_t^{(\setminus i)} = \sum_{j=0}^t \frac{\exp(s_t^j)}{Z} \mathbf{v}^j - \sum_{j \neq i} \frac{\exp(s_t^j)}{Z^{(\setminus i)}} \mathbf{v}^j, \quad (4)$$

where the unnormalized attention score of token j is $s_t^j = \mathbf{q}_t^T \mathbf{k}^j / \sqrt{d_k}$, and $Z = \sum_{l=0}^t \exp(s_t^l)$ and $Z^{(\setminus i)} = \sum_{l \neq i} \exp(s_t^l)$ are the softmax denominators before and after the eviction.

The main complexity arises from the change in this denominator from Z to $Z^{(\setminus i)}$. A further approximation is to omit the impact of eviction on the softmax denominator, assuming $Z \approx Z^{(\setminus i)}$. This is a simple and reasonable assumption when the number of tokens is large (we discuss the error bound of this approximation in Section 3.3). Under this approximation, the change in the output simplifies dramatically to the contribution vector of the evicted token itself:

$$\begin{aligned} \Delta \mathbf{o}_t &\approx \sum_{j=0}^t \frac{\exp(s_t^j)}{Z} \mathbf{v}^j - \sum_{j \neq i} \frac{\exp(s_t^j)}{Z} \mathbf{v}^j \\ &= \frac{\exp(s_t^i)}{Z} \mathbf{v}^i = \alpha_t^i \mathbf{v}^i, \end{aligned} \quad (5)$$

where α_t^i is the attention weight of token i .

Thus, the objective is simplified to finding the minimum $\alpha_t^i \mathbf{v}^i$. We define the final importance score using the following equation for its computa-

tional simplicity and empirical effectiveness:

$$\text{LongFlowScore}(t_i) = \left\| \alpha_t^i \mathbf{v}^i \right\|_1 = \alpha_t^i \sum_{l=1}^d |(\mathbf{v}^i)_l|. \quad (6)$$

The token with the minimum LongFlowScore is selected for eviction. This method is exceptionally efficient. The contribution vectors $\mathbf{c}_t^i = \alpha_t^i \mathbf{v}^i$ are a necessary intermediate result in the standard attention computation. Therefore, calculating the LongFlow score only requires performing an additional, lightweight reduction operation on this existing intermediate tensor. This perfectly realizes our Zero-Cost Estimation principle.

3.3 Theoretical Justification of the Approximations

To derive the final LongFlow score from the ideal objective (from Equation 2 to Equation 6), we introduce two primary approximations: (1) using the contribution vector $\mathbf{c}_t^i$ as a proxy for the true attention output change $\Delta \mathbf{o}_t$ (the *denominator approximation*), and (2) using the current query $\mathbf{q}_t$ as a proxy for the next-step query $\mathbf{q}_{t+1}$ (the *query approximation*). We now provide a brief analysis of the error bounds for each. The detailed proofs are available in Appendix A.

The total error $\mathcal{E}_i$ for a given token t_i is the difference between the ideal next-step objective and our practical score's squared magnitude:

$$\mathcal{E}_i = \left| \left\| \mathbf{o}_{t+1} - \mathbf{o}_{t+1}^{(\setminus i)} \right\|^2 - \left\| \alpha_t^i \mathbf{v}^i \right\|^2 \right|. \quad (7)$$

Using the triangle inequality, we can decompose this error into two distinct sources:

$$\mathcal{E}_i \leq \underbrace{\left| \left\| \mathbf{o}_{t+1} - \mathbf{o}_{t+1}^{(\setminus i)} \right\|^2 - \left\| \mathbf{c}_{t+1}^i \right\|^2 \right|}_{\text{Error from Denominator Approx.}} + \underbrace{\left| \left\| \mathbf{c}_{t+1}^i \right\|^2 - \left\| \mathbf{c}_t^i \right\|^2 \right|}_{\text{Error from Query Drift}}. \quad (8)$$

Error from the Denominator Approximation.

This approximation replaces the true output change with the contribution vector of the evicted token. A rigorous algebraic manipulation shows that the remainder term of this approximation, $\mathbf{R}_{t+1}^i = \Delta \mathbf{o}_{t+1} - \mathbf{c}_{t+1}^i$, is bounded by $\|\mathbf{R}_{t+1}^i\| \leq \frac{2V\alpha_{t+1}^i}{1-\alpha_{t+1}^i}$, where $V = \max_j \|\mathbf{v}^j\|$. This bound confirms that the approximation is highly accurate when the attention weight of the evicted token α_t^i is small. Since our method is designed to evict tokens with low attention, this approximation is well-justified.

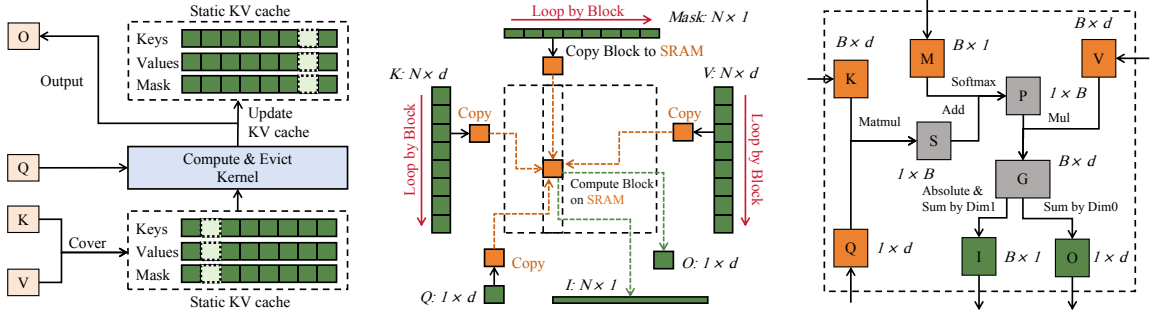


Figure 2: The data and computation flow of our method. O: attention output; I: LongFlowScore; S, P and G are intermediate states in kernel forward pass. **(Left)**: The process of a decoding step. The current KV will cover a slot selected in the previous step, and then the static KV and Mask will be sent to the kernel together with Q for calculation to obtain the current attention output and the slot to be covered in the next step. **(Middle)**: The data flow between HBM and SRAM in the kernel. KV and Mask will enter SRAM by block and perform fused attention calculation. **(Right)**: The computational flow on chip. Unlike standard flash attention calculations, we split the matrix multiplication of P and V into two steps and derive LongFlowScore from the intermediate result G.

Error from the Query Approximation. Query approximation uses the score computed with the current query as a proxy for the ideal score that would be computed with the future query, $\mathbf{q}_{t+1}$. The error introduced by this temporal approximation originates from the difference in attention weights, $|\alpha_{t+1}^i - \alpha_t^i|$. Due to the Lipschitz continuity of the softmax function, the change in output weights is bounded by the maximum change in the input pre-softmax scores, s_t^j :

$$|\alpha_{t+1}^i - \alpha_t^i| \leq \max_j |s_{t+1}^j - s_t^j|. \quad (9)$$

The error in the scores, $\Delta s^j = s_{t+1}^j - s_t^j$, arises from the query shift. By the Cauchy-Schwarz inequality, this error is bounded by $|\Delta s^j| \leq \frac{\|\mathbf{q}_{t+1} - \mathbf{q}_t\| \cdot \|\mathbf{k}^j\|}{\sqrt{d_k}}$. Assuming queries are normalized to unit vectors, we can relate the query difference to their cosine similarity: $\|\mathbf{q}_{t+1} - \mathbf{q}_t\|^2 = 2(1 - \cos(\mathbf{q}_t, \mathbf{q}_{t+1}))$. This gives a final bound on the maximum score error:

$$\max_j |\Delta s^j| \leq \frac{\sqrt{2(1 - \cos(\mathbf{q}_t, \mathbf{q}_{t+1}))} \cdot \max_j \|\mathbf{k}^j\|}{\sqrt{d_k}}. \quad (10)$$

This bound provides a strong theoretical guarantee. It shows that as the similarity between consecutive queries approaches 1, the error from our temporal approximation approaches 0.

3.4 High-Performance Implementation

To realize the theoretical efficiency of LongFlow as practical speedups, we implement a high-performance system centered around a custom

fused kernel. This section details our key system-level optimizations, including the memory management strategy and the fused attention & eviction kernel. Figure 2 shows the data and computation flow of our method.

Static Memory and Consistent Workload. As established in our design philosophy, we employ a static KV cache and a consistent per-step eviction policy. The entire memory for the KV cache is pre-allocated to eliminate dynamic allocation overheads and memory fragmentation. By overwriting a single token at every decoding step, we ensure a consistent computational workload, which is particularly beneficial in distributed systems, as it guarantees a balanced workload and predictable overhead across all parallel workers.

Fused Attention and Eviction Kernel. The core of our implementation is a custom Triton kernel that fuses the entire attention and eviction process. While our kernel is designed based on the I/O-aware principles of FlashAttention (Dao et al., 2022), it incorporates three critical modifications to specifically and efficiently handle the auto-regressive decoding process with LongFlow: (1) Given that the query sequence length is always 1 during decoding, we eliminate the outer loop over the query dimension in standard FlashAttention. (2) To seamlessly integrate the online calculation of the LongFlowScore, we omit the running maximum used for numerical stability in safe softmax. To compensate and prevent potential overflow, we perform the calculation of softmax in FP32. (3) We restructure the computation to maximize data reuse for both tasks. Within the single loop over

KV cache blocks, we compute the un-normalized contribution vectors $\exp(\text{scores})V$ as a key intermediate result. This tensor is then summed along the sequence dimension to update the final attention output accumulator, and its L1-norm is calculated to serve as the un-normalized LongFlowScore for that block. These modifications result in a single, highly-optimized operator that computes both the attention output and the token to evict in one pass, as detailed in Algorithm 1 in Appendix.

4 Experiments

We conduct a series of experiments to comprehensively evaluate the effectiveness of LongFlow. Our evaluation focuses on three key aspects: model quality, inference speed, and memory efficiency.

4.1 Experimental Setup

Models. We evaluate LongFlow on different reasoning models to demonstrate its applicability. Our experiments are primarily conducted on **DeepSeek-R1-Distill-Llama-8B** (Guo et al., 2025) and the **Qwen3** (Yang et al., 2025) series, including its 0.6B, 1.7B, 4B, and 8B variants.

Datasets. Our evaluation spans a wide spectrum of challenging reasoning benchmarks to ensure a comprehensive assessment of model accuracy. For competition-level mathematics, we use problems from MATH-500 (Hendrycks et al., 2021), AMC-23 (MAA, 2023), AIME-24 (MAA, 2024) and AIME-25 (MAA, 2025). To test advanced scientific reasoning, we employ the graduate-level expert QA from GPQA (Rein et al., 2024), university-level problems from Minerva (Lewkowycz et al., 2022), and Olympiad-level questions from Olympiad-Bench (He et al., 2024). Finally, we include the widely-used GSM8K benchmark (Cobbe et al., 2021) for grade-school math word problems.

Baselines. We compare LongFlow against several representative KV cache compression methods:

- **Vanilla:** The standard attention without any KV cache management. It serves as the accuracy upper bound but represents the worst scenario for memory consumption and latency.
- **H2O**(Zhang et al., 2023): A classic importance-based method that prioritizes tokens based on accumulated attention scores.
- **VATP**(Guo et al., 2024): An advanced method that integrates both attention scores and Value vector information into its importance metric.
- **R-KV**(Cai et al., 2025): A recent method also

designed for reasoning models. It incorporates information about repetitive patterns in the generation to guide its eviction policy.

Evaluation Metrics. We evaluate all methods across two primary dimensions. For system performance, we measure the inference throughput, peak GPU memory usage, and memory fragments. For model accuracy, we report the task-specific accuracy on each of the benchmarks.

Implementation Details. For our main experiments, the generation output length is set to 16,000 tokens. For all compression methods, we set the KV cache budget to be either 3,200 or 2,400 tokens. For H2O and VATP, this budget is evenly split between heavy-hitter tokens and recent tokens. For LongFlow, if the number of tokens in the prefill stage exceeds the budget, we first use SnapKV (Li et al., 2024) to compress the tokens to the budget size. For all the baselines, we set the compression interval to achieve a balance between compression ratio and inference speed. All the experiments are conducted on NVIDIA A100 40GB GPUs.

4.2 Main Results on Model Accuracy

We present our main results on model accuracy in Table 1. The evaluation across two distinct model families and two KV cache budget settings demonstrates that LongFlow effectively preserves the reasoning capabilities of the base models, achieving performance that is highly competitive with state-of-the-art baselines and close to the uncompressed Vanilla model.

Comparison With the Baselines. Across both models and budget settings, our proposed LongFlow demonstrates highly competitive performance against state-of-the-art baselines. R-KV generally achieves the highest average accuracy (with more memory for redundant token identification), establishing itself as a strong baseline focused on quality preservation. LongFlow consistently performs on par with or better than other methods like VATP and H2O.

Comparison with Full KV and Different Cache Budgets. A key finding is that all compression methods, including LongFlow, successfully preserve the vast majority of the original model’s capabilities while operating on a significantly reduced KV cache (e.g., a 3.2k budget representing an 80% compression ratio for a 16k generation). As shown in Table 1, the average performance degradation compared to the uncompressed Vanilla baseline is negligible to minor—as low as 0.08% for

Model	Method	AIME24 (30)	AIME25 (30)	AMC (40)	GPQA (198)	GSM8K (1318)	MATH (500)	Minerva (272)	Olympiad (675)
DeepSeek-R1-Distill-Llama-8B	Budget = 2400								
	Vanilla	30.00	20.00	77.50	41.41	77.48	83.40	20.59	44.30
	H2O	43.33	20.00	65.00	41.41	77.71	79.80	19.12	44.00
	R-KV	36.67	20.00	75.00	38.38	77.41	80.80	20.22	45.93
	VATP	33.33	13.33	72.50	33.84	77.48	79.80	20.59	43.56
	Ours	33.33	16.67	70.00	40.40	77.71	79.80	20.22	43.85
	Budget = 3200								
	Vanilla	30.00	20.00	77.50	41.41	77.47	83.40	20.59	44.30
	H2O	36.67	23.33	72.50	38.38	77.69	81.60	21.32	46.22
	R-KV	30.00	23.33	77.50	42.93	77.62	82.40	19.49	46.81
Qwen3-8B	VATP	40.00	16.67	72.50	42.42	77.39	81.60	19.12	46.67
	Ours	33.33	26.67	82.50	40.91	77.62	79.80	20.22	45.93
	Budget = 2400								
	Vanilla	60.00	46.67	90.00	34.85	95.68	92.60	33.46	58.22
	H2O	40.00	20.00	65.00	26.77	95.98	85.20	32.35	49.04
	R-KV	33.33	23.33	85.00	28.79	96.13	89.00	32.72	53.33
	VATP	40.00	26.67	77.50	23.23	96.29	87.60	31.99	49.33
	Ours	36.67	26.67	80.00	30.81	95.83	88.00	31.25	51.11
	Budget = 3200								
	Vanilla	60.00	46.67	90.00	34.85	95.68	92.60	33.46	58.22
	H2O	46.67	26.67	80.00	27.78	96.36	90.40	35.29	53.78
	R-KV	50.00	33.33	82.50	32.32	96.36	91.60	33.46	55.56
	VATP	43.33	33.33	82.50	30.30	96.43	91.00	34.19	55.41
	Ours	50.00	26.67	85.00	30.30	96.36	89.80	34.19	55.56

Table 1: Model Performance on different models across different datasets. Numbers in parentheses indicate dataset sizes. **Bold** indicates the best performance.

DeepSeek-R1-Distill-Llama-8B and approximately 1.3% for Qwen3-8B. When the KV cache budget is tightened from 3.2k to 2.4k, a graceful degradation is observed across all methods. LongFlow performs better under this pressure, showing less degradation than both H2O and VATP.

4.3 Performance on Different Model Sizes

To assess the performance of LongFlow across different model sizes, we conduct a comparative analysis using the Qwen3 model family, including the 0.6B, 1.7B, 4B, and 8B variants. For this specific analysis, the AIME-24 and AIME-25 datasets were excluded due to their small sample sizes and for brevity. The results in Figure 4 (in Appendix C) show that LongFlow maintains a highly competitive performance, achieving results that are comparable or superior to the baselines across all tested model sizes and benchmarks, which demonstrates the effectiveness and robustness of our method.

4.4 Throughput and Memory Analysis

Finally, we evaluate the system-level performance of LongFlow in terms of throughput and memory

efficiency. For this analysis, we use the Qwen3-1.7B on a single NVIDIA A100 40GB GPU. We configure a scenario with an input length of 512 and an output length of 16,000. We set the KV cache budget to 3,200 for all compression methods. To measure the maximum performance, we progressively increase the batch size for each method until an Out-of-Memory (OOM) error occurs. We report the throughput, peak memory footprint, and memory fragments in Figure 3.

The results clearly demonstrate LongFlow’s substantial advantages in both throughput and memory management. In terms of throughput, LongFlow outperforms FullKV by a remarkable 11.8 times and is approximately 4.0 times faster than other methods. Although LongFlow performs KV cache compression at every decoding step, its algorithmic efficiency and fused kernel implementation substantially reduce per-step overhead, enabling remarkably higher throughput than competing methods. Regarding peak memory usage, LongFlow’s footprint is comparable to other methods under the same budget, while R-KV consumes slightly more due to its more complex importance estimation

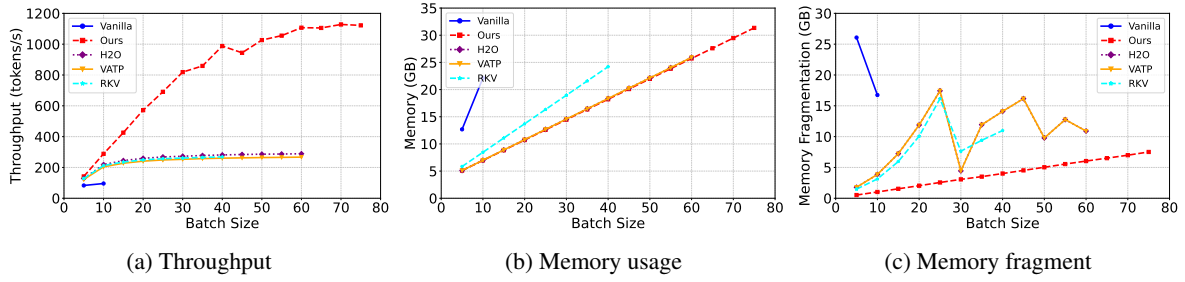


Figure 3: Performance comparison of LongFlow against baselines. The compression is conducted every step for LongFlow and every 128 steps for other methods. LongFlow achieves higher throughput and supports a larger maximum batch size due to superior memory management.

logic. Notably, LongFlow’s advantage in memory management lies in its superior handling of fragmentation. Its static memory scheme and consistent per-step eviction policy result in a memory layout that is both less fragmented and highly predictable. This less fragmented memory layout allows LongFlow to support a larger maximum batch size than competing methods. The predictability makes LongFlow more stable and reliable in real-world deployment by eliminating runtime memory-related uncertainties.

5 Related Work

Reasoning Models While traditional LLMs perform well on general tasks, achieving breakthroughs in complex logical reasoning has remained a significant challenge. This bottleneck has recently been addressed by a new class of models named reasoning models designed for long CoT generation. Pioneered by OpenAI-o1 (Jaeche et al., 2024), DeepSeek-R1 (Guo et al., 2025) leverages Reinforcement Learning with Verifiable Rewards (RLVR) to unlock the model’s latent abilities for long CoT generation and high-level reasoning. These breakthroughs catalyze a subsequent wave of increasingly powerful reasoning models from numerous research labs, including Qwen3 (Yang et al., 2025), Gemini-2.5 (Comanici et al., 2025), and GPT-5 (OpenAI, 2025). Researchers have employed advanced training strategies to develop increasingly powerful models that mark new milestones. However, the long-CoT reasoning paradigm introduces significant deployment challenges. Generating lengthy sequences inflates the KV cache, leading to severe memory and computational bottlenecks. This problem is further exacerbated by the models’ tendency to overthink (Chen et al., 2024). While numerous approaches have been proposed to develop more efficient reasoning models (Feng et al., 2025; Wang et al., 2025; Jiang

et al., 2025), they often focus on shortening the CoT, which risks constraining the model’s reasoning capability. Developing more effective methods remains an open challenge.

KV Cache Compression KV cache compression is a widely used technique for improving the efficiency of LLM inference by retaining essential contextual information under limited memory budgets. Existing methods typically exploit attention patterns to identify and evict less important tokens, or adopt more explicit objectives that approximate attention output similarity (Feng et al., 2024). Despite their effectiveness in long-input scenarios, these eviction-based methods suffer from irreversible information loss and often incur non-trivial computational overhead, making them less suitable for long-output generation. To address this, more recent approaches construct compact representations instead of directly discarding tokens, for example through semantic chunking (Liu et al., 2025) or cross-layer merging (Liu et al., 2024b). Other methods periodically refresh cached representations to mitigate permanent information loss (Xu et al., 2024). While promising, these techniques typically introduce additional computation or architectural complexity. In contrast, our work focuses on a fundamentally lightweight KV cache compression method specifically tailored to the long-output setting of reasoning models.

6 Conclusion

This paper introduces LongFlow, a novel and lightweight KV cache compression method designed to mitigate the significant inference overhead in long-output reasoning models. Grounded in a zero-history and zero-cost design philosophy, LongFlow employs a highly efficient, theoretically-justified importance metric derived directly from intermediate attention values, incurring negligible

computational cost. With our co-designed system, LongFlow increase throughput by up to 11.8x and reduce the KV cache footprint by 80%, while maintaining high model accuracy. By striking an effective balance between performance and accuracy, LongFlow presents a practical pathway toward the efficient deployment of reasoning models.

Limitations

Despite strong empirical and system-level efficiency, LongFlow has several limitations.

Dependence on query stability. LongFlow estimates token importance using only the current-step query and intermediate attention results. This design is most reliable when consecutive decoding queries are similar, which often holds in long chain-of-thought generation. However, in settings with abrupt distribution shifts (e.g., topic switches, tool-use interleaving, or highly stochastic decoding), the current query may be a weaker proxy for near-future behavior, potentially making eviction decisions less optimal.

Limited to long-output decoding. LongFlow is optimized for autoregressive decoding with a query length of one and frequent per-step cache updates. It is not directly tailored to regimes dominated by long-input prefill, bidirectional attention, or non-autoregressive generation. A unified approach that jointly optimizes prefill and long-output decoding remains future work.

Acknowledgments

We want to thank all the anonymous reviewers for their valuable comments. This work was supported by the National Natural Science Foundation of China (NSFC No. 62576232) and Key Laboratory of Data Intelligence and Advanced Computing, Soochow University.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, and 1 others. 2025. R-kv: Redundancy-aware kv cache compression for

training-free reasoning models acceleration. *arXiv preprint arXiv:2505.24133*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and 1 others. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, and 1 others. 2024. Do not think that much for $2+3=?$ on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

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

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Sicheng Feng, Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Efficient reasoning models: A survey. *arXiv preprint arXiv:2504.10903*.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2024. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

- Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. 2024. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters. *arXiv preprint arXiv:2406.12335*.
- Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.
- Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.
- Lingjie Jiang, Xun Wu, Shaohan Huang, Qingxiu Dong, Zewen Chi, Li Dong, Xingxing Zhang, Tengchao Lv, Lei Cui, and Furu Wei. 2025. Think only when you need with large hybrid-reasoning models. *arXiv preprint arXiv:2505.14631*.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.
- Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857.
- Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.
- Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.
- Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024b. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031.
- Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. 2025. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *arXiv preprint arXiv:2502.00299*.
- MAA. 2023. American mathematics competitions (amc 10/12). URL <https://maa.org/math-competitions/amc>.
- MAA. 2024. American invitational mathematics examination (aime). URL <https://maa.org/math-competitions/aime>.
- MAA. 2025. American invitational mathematics examination (aime). URL <https://maa.org/math-competitions/aime>.
- AI Meta. 2025. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>, checked on, 4(7):2025.
- OpenAI. 2025. Gpt-5 system card.
- Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132. IEEE.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.
- Yi Su, Quantong Qiu, Yuechi Zhou, Juntao Li, Qingrong Xia, Ping Li, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2025. Calidrop: Kv cache compression with calibration. *arXiv preprint arXiv:2507.19906*.
- Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.
- A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.
- Jikai Wang, Juntao Li, Jianye Hou, Bowen Yan, Lijun Wu, and Min Zhang. 2025. Efficient reasoning for llms through speculative chain-of-thought. *arXiv preprint arXiv:2504.19095*.
- Fangyuan Xu, Tanya Goyal, and Eunsol Choi. 2024. Refreshkv: Updating small kv cache during long-form generation. *arXiv preprint arXiv:2411.05787*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

A Detailed Error Bound Derivation

In this section, we provide the detailed proofs for the two key approximations that underpin the LongFlow method, justifying the claims made in Section 3.3. Our goal is to bound the total error $\mathcal{E}^i = \left| \left\| \mathbf{o}_{t+1} - \mathbf{o}_{t+1}^{(\setminus i)} \right\|^2 - \left\| \mathbf{c}_t^i \right\|^2 \right|$, which we decompose using the triangle inequality:

$$\mathcal{E}^i \leq \underbrace{\left| \left\| \mathbf{o}_{t+1} - \mathbf{o}_{t+1}^{(\setminus i)} \right\|^2 - \left\| \mathbf{c}_{t+1}^i \right\|^2 \right|}_{\text{Error from Denominator Approx.}} + \underbrace{\left| \left\| \mathbf{c}_{t+1}^i \right\|^2 - \left\| \mathbf{c}_t^i \right\|^2 \right|}_{\text{Error from Query Drift}}. \quad (11)$$

We now derive explicit bounds for each of these two error terms.

A.1 Bounding the Error from the Denominator Approximation

This error arises from approximating the true change in attention output, $\Delta \mathbf{o}_{t+1} = \mathbf{o}_{t+1} - \mathbf{o}_{t+1}^{(\setminus i)}$, with the contribution vector $\mathbf{c}_{t+1}^i = \alpha_{t+1}^i \mathbf{v}^i$. We now derive an exact expression for the remainder term $\mathbf{R}_{t+1}^i = \Delta \mathbf{o}_{t+1} - \mathbf{c}_{t+1}^i$.

First, let us expand the terms. The original attention output is a sum over all t historical tokens, while the output after eviction is a sum over the remaining $t-1$ tokens with re-normalized attention weights:

$$\begin{aligned} \mathbf{o}_{t+1} &= \sum_{j=1}^t \alpha_{t+1}^j \mathbf{v}^j = \alpha_{t+1}^i \mathbf{v}^i + \sum_{j \neq i} \alpha_{t+1}^j \mathbf{v}^j \\ \mathbf{o}_{t+1}^{(\setminus i)} &= \sum_{j \neq i} \alpha_{t+1}^{j, (\setminus i)} \mathbf{v}^j \end{aligned}$$

Substituting these into the definition of the remainder term:

$$\begin{aligned} \mathbf{R}_{t+1}^i &= \Delta \mathbf{o}_{t+1} - \mathbf{c}_{t+1}^i \\ &= \left(\mathbf{o}_{t+1} - \mathbf{o}_{t+1}^{(\setminus i)} \right) - \mathbf{c}_{t+1}^i \\ &= \left(\alpha_{t+1}^i \mathbf{v}^i + \sum_{j \neq i} \alpha_{t+1}^j \mathbf{v}^j \right) \\ &\quad - \left(\sum_{j \neq i} \alpha_{t+1}^{j, (\setminus i)} \mathbf{v}^j \right) - \alpha_{t+1}^i \mathbf{v}^i \\ &= \sum_{j \neq i} \left(\alpha_{t+1}^j - \alpha_{t+1}^{j, (\setminus i)} \right) \mathbf{v}^j. \end{aligned}$$

The key is to relate the new weights $\alpha_{t+1}^{j, (\setminus i)}$ to the old weights α_{t+1}^j . The new weights are re-

normalized over a smaller set of tokens:

$$\begin{aligned} \alpha_{t+1}^{j, (\setminus i)} &= \frac{\exp(s_{t+1}^j)}{\sum_{l \neq i} \exp(s_{t+1}^l)} \\ &= \frac{\exp(s_{t+1}^j)}{Z_{t+1} - \exp(s_{t+1}^i)} \\ &= \frac{\exp(s_{t+1}^j)/Z_{t+1}}{(Z_{t+1} - \exp(s_{t+1}^i))/Z_{t+1}} \\ &= \frac{\alpha_{t+1}^j}{1 - \alpha_{t+1}^i}. \end{aligned} \quad (12)$$

Now, we substitute this identity back into the expression for the remainder term:

$$\begin{aligned} \mathbf{R}_{t+1}^i &= \sum_{j \neq i} \left(\alpha_{t+1}^j - \frac{\alpha_{t+1}^j}{1 - \alpha_{t+1}^i} \right) \mathbf{v}^j \\ &= \sum_{j \neq i} \left(\frac{-\alpha_{t+1}^j \alpha_{t+1}^i}{1 - \alpha_{t+1}^i} \right) \mathbf{v}^j \\ &= -\frac{\alpha_{t+1}^i}{1 - \alpha_{t+1}^i} \sum_{j \neq i} \alpha_{t+1}^j \mathbf{v}^j. \end{aligned} \quad (13)$$

Finally, we recognize that the remaining summation is simply the original attention output minus the contribution of the i -th token: $\sum_{j \neq i} \alpha_{t+1}^j \mathbf{v}^j = \mathbf{o}_{t+1} - \mathbf{c}_{t+1}^i$. This gives the final exact expression for the remainder term:

$$\mathbf{R}_{t+1}^i = -\frac{\alpha_{t+1}^i}{1 - \alpha_{t+1}^i} (\mathbf{o}_{t+1} - \mathbf{c}_{t+1}^i). \quad (14)$$

Taking the norm and assuming $\|\mathbf{v}^j\| \leq V$, we have $\|\mathbf{o}_{t+1}\| \leq V$. The norm of the remainder is then bounded:

$$\begin{aligned} \|\mathbf{R}_{t+1}^i\| &\leq \frac{\alpha_{t+1}^i}{1 - \alpha_{t+1}^i} (\|\mathbf{o}_{t+1}\| + \|\mathbf{c}_{t+1}^i\|) \\ &\leq \frac{\alpha_{t+1}^i (1 + \alpha_{t+1}^i)}{1 - \alpha_{t+1}^i} V \\ &\leq \frac{2V \alpha_{t+1}^i}{1 - \alpha_{t+1}^i}. \end{aligned} \quad (15)$$

A.2 Bounding the Error from the Query Approximation

This error arises from query drift, captured by the term $|\|\mathbf{c}_{t+1}^i\|^2 - \|\mathbf{c}_t^i\|^2| = |(\alpha_{t+1}^i)^2 - (\alpha_t^i)^2| \cdot \|\mathbf{v}^i\|^2$. The core task is to bound $|\alpha_{t+1}^i - \alpha_t^i|$.

Lemma 1 (Lipschitz Property of Softmax). *The softmax function $\sigma(\mathbf{s})_i = \exp(s^i) / \sum_j \exp(s^j)$ is 1-Lipschitz continuous with respect to the L1-norm on its output and the L-infinity norm on its input.*

Proof. By the Mean Value Theorem for vector-valued functions, for any two score vectors $\mathbf{s}$ and $\mathbf{s}'$, there exists a point $\mathbf{c}$ on the line segment between them such that:

$$\sigma(\mathbf{s}') - \sigma(\mathbf{s}) = J_\sigma(\mathbf{c})(\mathbf{s}' - \mathbf{s})$$

where J_σ is the Jacobian matrix of the softmax function. Our goal is to bound the norm of this Jacobian. The entries of the Jacobian, $J_{ij} = \frac{\partial \sigma_i}{\partial s^j}$, are computed as follows:

For the diagonal entries ($i = j$):

$$\begin{aligned} \frac{\partial \sigma_i}{\partial s^i} &= \frac{\partial}{\partial s^i} \left(\frac{\exp(s^i)}{\sum_k \exp(s^k)} \right) \\ &= \frac{\exp(s^i) (\sum_k \exp(s^k)) - \exp(s^i) \exp(s^i)}{(\sum_k \exp(s^k))^2} \\ &= \frac{\exp(s^i)}{\sum_k \exp(s^k)} \left(1 - \frac{\exp(s^i)}{\sum_k \exp(s^k)} \right) \\ &= \sigma_i(1 - \sigma_i). \end{aligned}$$

For the off-diagonal entries ($i \neq j$):

$$\begin{aligned} \frac{\partial \sigma_i}{\partial s^j} &= \frac{\partial}{\partial s^j} \left(\frac{\exp(s^i)}{\sum_k \exp(s^k)} \right) \\ &= \frac{0 - \exp(s^i) \exp(s^j)}{(\sum_k \exp(s^k))^2} \\ &= -\frac{\exp(s^i)}{\sum_k \exp(s^k)} \frac{\exp(s^j)}{\sum_k \exp(s^k)} \\ &= -\sigma_i \sigma_j. \end{aligned}$$

Thus, the Jacobian can be succinctly written as $J_{ij} = \sigma_i(\delta_{ij} - \sigma_j)$, where δ_{ij} is the Kronecker delta.

To obtain the bound used in the main text, we need to bound the induced L1/L ∞ norm. It is a known property of the softmax function that $\|\sigma(\mathbf{s}') - \sigma(\mathbf{s})\|_1 \leq \|\mathbf{s}' - \mathbf{s}\|_\infty$, which means the Lipschitz constant for this norm pairing is 1. While the full proof of this specific norm bound is lengthy and typically presented in optimization literature, we can gain insight by analyzing the simpler induced L1-norm, $\|J\|_1 = \max_j \sum_i |J_{ij}|$. The sum of the absolute values of the j -th column is:

$$\begin{aligned} \sum_i |J_{ij}| &= |\sigma_j(1 - \sigma_j)| + \sum_{i \neq j} |-\sigma_i \sigma_j| \\ &= \sigma_j(1 - \sigma_j) + \sum_{i \neq j} \sigma_i \sigma_j \\ &= \sigma_j - \sigma_j^2 + \sigma_j \sum_{i \neq j} \sigma_i \\ &= \sigma_j - \sigma_j^2 + \sigma_j(1 - \sigma_j) = 2\sigma_j(1 - \sigma_j). \end{aligned}$$

Since the maximum value of the function $f(x) = 2x(1 - x)$ for $x \in [0, 1]$ is $1/2$ (at $x = 1/2$), the induced L1-norm is bounded by $\|J\|_1 \leq 1/2$. The bound of 1 for the L1/L ∞ norm pairing used in our main analysis is a tighter, more specific result. $\square$

Applying the Lemma, the change in a single attention weight is bounded by the maximum change in any pre-softmax score:

$$|\alpha_{t+1}^i - \alpha_t^i| \leq \max_j |s_{t+1}^j - s_t^j|. \quad (16)$$

The error in the scores, $s_{t+1}^j - s_t^j = (\mathbf{q}_{t+1} - \mathbf{q}_t)^T \mathbf{k}^j / \sqrt{d_k}$, is bounded by the Cauchy-Schwarz inequality. Assuming unit-normalized queries, we have $\|\mathbf{q}_{t+1} - \mathbf{q}_t\|^2 = 2(1 - \cos(\mathbf{q}_t, \mathbf{q}_{t+1}))$. This gives the final bound on the maximum score error:

$$|\Delta s^j| \leq \frac{\sqrt{2(1 - \cos(\mathbf{q}_t, \mathbf{q}_{t+1}))} \cdot \max_j \|\mathbf{k}^j\|}{\sqrt{d_k}}. \quad (17)$$

A.3 Combined Error Bound

By combining the bounds for both error sources, we can conclude that the total error $\mathcal{E}^i$ is small when our method evicts a token with a small attention weight α_{t+1}^i and when the query similarity is high. This provides a rigorous theoretical foundation for the effectiveness of the LongFlow method.

B Algorithm of LongFlow Kernel

Algorithm 1 shows the workflow of our kernel.

C Additional Figures

Figure 4 shows the results of scaling model size. Figure 5 shows the figures we used in our preliminary experiments.

D Additional Results and Implementation Details

This appendix provides additional experimental results and implementation details for completeness.

D.1 Additional Baselines: Scope and TOVA

We additionally compare LongFlow with two recent baselines, **Scope** and **TOVA**, on Qwen3-8B under both Budget=2400 and Budget=3200. For both methods, we follow their standard periodic compression setting (every 128 decoding steps). For Scope, half of the KV budget is allocated to the prompt and the other half to the decode stage, with the total budget not exceeding the target budget.

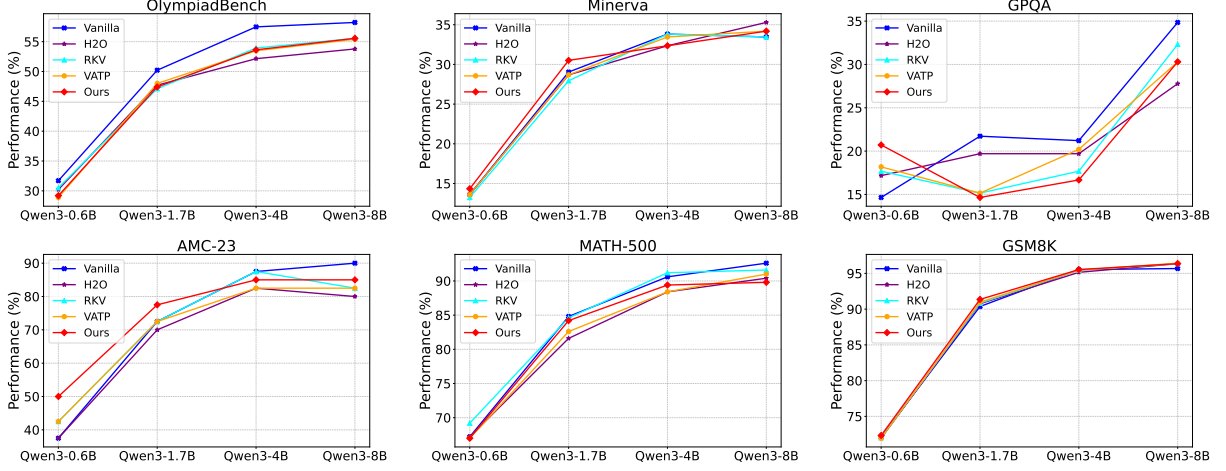
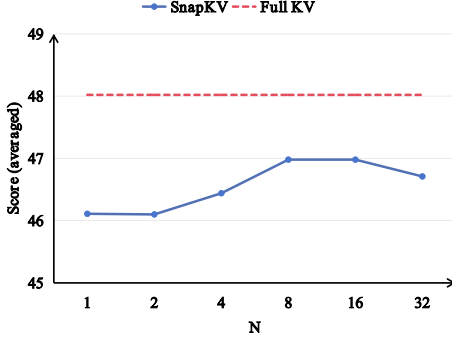
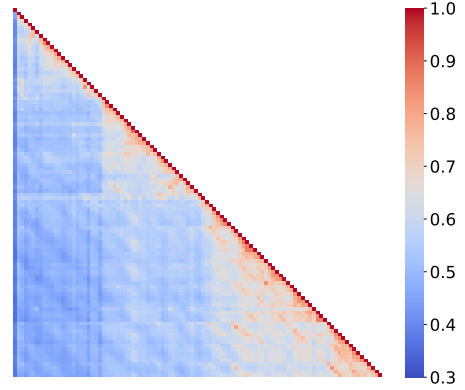


Figure 4: Accuracy of LongFlow and the baselines across different model sizes on different datasets.



(a) SnapKV performance on LongBench as the query observation window shrinks. We use LLaMA-3-8B and an context length of 8192.



(b) Cosine similarity between queries. Results are shown for the first 100 tokens, averaged over 100 samples from MATH500 using Qwen3-8B (Layer 10, Head 10). The high and stable similarity between adjacent queries supports our approximation.

Figure 5: Empirical motivation for our single-query hypothesis.

Overall, LongFlow remains competitive with the additional baselines while maintaining its primary advantage in efficiency. In particular, under the 80% compression setting (Budget=3200), LongFlow continues to preserve most of the original model capability.

D.2 Ablation on Norm Choice in LongFlowScore

Our default LongFlowScore is defined as:

$$\text{LongFlowScore}(t_i) = \alpha_i^t \|v_i\|_1.$$

We choose the L1 norm mainly for computational efficiency and ease of fusion inside the attention kernel. To evaluate whether the choice of norm matters, we compare L1 and L2 under the same budget on Qwen3-8B.

The differences are minor and do not change the overall ranking behavior. This is consistent with

our intuition that token ordering is largely dominated by the attention weight α_i^t , while value magnitudes within the same layer/head are relatively stable. We therefore use L1 as the default choice due to its lower reduction cost and better suitability for fused implementation.

D.3 Numerical Error of the Blockwise Fused Implementation

Our fused kernel follows the same IO-aware blockwise computation strategy as FlashAttention. To quantify the numerical error introduced by this implementation, we compare operator-level outputs in FP16 against a PyTorch reference implementation.

The deviation introduced by our fused blockwise implementation is on the same scale as, or smaller than, widely adopted FlashAttention kernels. This suggests that the blockwise strategy introduces only negligible numerical differences in practice.

Table 2: Additional comparison on Qwen3-8B under different KV budgets.

Budget	Method	AIME24	AIME25	AMC	GPQA	GSM8K	MATH	Minerva	Olympiad
2400	Vanilla	60.00	46.67	90.00	34.85	95.68	92.60	33.46	58.22
	H2O	40.00	20.00	65.00	26.77	95.98	85.20	32.35	49.04
	R-KV	33.33	23.33	85.00	28.79	96.13	89.00	32.72	53.33
	VATP	40.00	26.67	77.50	23.23	96.29	87.60	31.99	49.33
	TOVA	36.67	20.00	77.50	34.34	95.60	86.20	31.25	50.07
	Scope	43.33	30.00	82.50	37.88	95.75	89.40	32.35	52.15
	Ours	36.67	26.67	80.00	30.81	95.83	88.00	31.25	51.11
3200	Vanilla	60.00	46.67	90.00	34.85	95.68	92.60	33.46	58.22
	H2O	46.67	26.67	80.00	27.78	96.36	90.40	35.29	53.78
	R-KV	50.00	33.33	82.50	32.32	96.36	91.60	33.46	55.56
	VATP	43.33	33.33	82.50	30.30	96.43	91.00	34.19	55.41
	TOVA	46.67	33.33	82.50	37.37	96.06	90.60	33.46	53.19
	Scope	53.33	50.00	87.50	42.42	96.13	91.40	31.99	53.78
	Ours	50.00	26.67	85.00	30.30	96.36	89.80	34.19	55.56

Table 3: Ablation of norm choice in LongFlowScore on Qwen3-8B with Budget=3200.

Norm	AIME24	AIME25	AMC	GPQA	GSM8K	MATH	Minerva	Olympiad
L1	50.00	26.67	85.00	30.30	96.36	89.80	34.19	55.56
L2	50.00	30.00	82.50	30.80	96.21	90.20	34.19	55.11

Table 4: Operator-level numerical error compared with PyTorch reference.

Implementation	Attention Output Error	Importance Score Error
torch (reference)	0	0
flash-attention2	5.5×10^{-4}	/
ours	1.4×10^{-6}	4.3×10^{-5}

D.4 Actual Generation Length Statistics

In the main experiments, 16k denotes the *maximum* allowed output length, rather than the minimum or average generation length. Table 5 reports the actual generation statistics across datasets.

These results show that many reasoning benchmarks naturally exhibit long-form generation, and the 95th percentile lengths for several datasets are close to the 16k limit. Therefore, setting the maximum decoding length to 16k is reasonable for evaluating long-form reasoning scenarios.

We also checked whether pruning causes systematic length inflation. On the Olympiad benchmark, the average total generation length is 8488.62 for Vanilla and 8523.06 for LongFlow. The difference is only about 34 tokens ($\sim 0.4\%$), which falls within normal variation and does not indicate systematic “overthinking” caused by pruning.

D.5 Clarifications on Compression Interval and Memory Fragmentation

Compression interval. The compression interval refers to how frequently KV cache eviction is

performed during decoding. For example, a compression interval of 128 means that token eviction is applied once every 128 decoding steps, rather than at every step. In prior KV compression methods such as H2O, R-KV, and GEAR, periodic compression is commonly used to reduce the overhead of importance estimation. In contrast, LongFlow performs eviction at every decoding step because its importance metric is derived directly from intermediate attention values and fused into the attention kernel.

Memory fragmentation. We measure memory fragmentation using PyTorch CUDA memory statistics, defined as:

$$\text{Reserved Memory} - \text{Allocated Memory},$$

where Allocated Memory is the memory occupied by active tensors and Reserved Memory is the total memory reserved by the CUDA caching allocator. In our experiments, these values are obtained via `torch.cuda.memory_allocated()` and `torch.cuda.memory_reserved()`, and we report the peak fragmentation during decoding.

Batch-size search granularity. The maximum batch size reported in Figure 3 is obtained by increasing the batch size with a step size of 5 until out-of-memory occurs. Therefore, the reported peak batch size for Vanilla (around 10) is approximate; the actual feasible peak lies roughly in the

Table 5: Generation statistics of the evaluation datasets. We report the number of samples, average question length, average answer length, average total length, and the 95th percentile of total length.

Dataset	Count	Question Avg.	Answer Avg.	Total Avg.	Total P95
AIME24	30	107.07	11755.70	11862.77	16125.85
AIME25	30	158.83	13043.70	13202.53	16467.45
AMC	40	84.60	7221.85	7306.45	16079.05
GPQA	198	212.82	7885.35	8098.17	16174.30
GSM8K	1319	61.31	1828.88	1890.19	3313.50
MATH	500	70.94	4524.57	4595.51	15470.20
Minerva	272	131.17	5588.21	5719.38	15082.50
Olympiad	675	100.05	8423.02	8523.06	16143.30

Algorithm 1 LongFlow Fused Attention and Eviction Kernel

```

1: Input: Query  $\mathbf{q}_t \in \mathbb{R}^{1 \times d}$ , KV cache  $\mathbf{K} \in \mathbb{R}^{(t-1) \times d}$ ,
    $\mathbf{V} \in \mathbb{R}^{(t-1) \times d}$ , Mask  $\mathbf{M} \in \{0, 1\}^{t-1}$ 
2: Output: Attention output  $\mathbf{o}_t \in \mathbb{R}^{1 \times d}$ , Eviction index
    $i_{\text{evict}}$ 
3:
4: // Initialize accumulators
5:  $\mathbf{o}_{\text{acc}} \leftarrow \mathbf{0} \in \mathbb{R}^d$ ;  $\triangleright$  Output accumulator
6:  $l_{\text{acc}} \leftarrow 0 \in \mathbb{R}$   $\triangleright$  Softmax denominator accumulator
7:  $\mathbf{S}_{\text{loss}} \leftarrow \mathbf{0} \in \mathbb{R}^{t-1}$   $\triangleright$  Temporary storage for
   un-normalized scores
8:  $B_k \leftarrow \text{block\_size}$ 
9:
10: // Single pass over the KV cache in blocks
11: for  $j \leftarrow 1, B_k, \dots, t-1$  do
12:   // Load a block of  $\mathbf{K}$ ,  $\mathbf{V}$  from HBM to on-chip SRAM
13:    $\mathbf{K}_j \leftarrow \mathbf{K}[j:j+B_k, :]$ ;  $\mathbf{V}_j \leftarrow \mathbf{V}[j:j+B_k, :]$ 
14:
15:   // Compute scores and apply mask for the current block
16:    $\mathbf{S}_j \leftarrow \mathbf{q}_t \mathbf{K}_j^T / \sqrt{d}$ 
17:    $\mathbf{S}_j[\neg \mathbf{M}_j] \leftarrow -\infty$ 
18:
19:   // Compute softmax numerators and update denominator
   accumulator
20:    $\mathbf{P}_j \leftarrow \exp(\mathbf{S}_j)$ 
21:    $l_{\text{acc}} \leftarrow l_{\text{acc}} + \text{sum}(\mathbf{P}_j)$ 
22:
23:   // Compute un-normalized contribution vectors and
   update output accumulator
24:    $\mathbf{C}_j \leftarrow \mathbf{P}_j \cdot \mathbf{V}_j$   $\triangleright$  Element-wise scaling, shape:
    $B_k \times d$ 
25:    $\mathbf{o}_{\text{acc}} \leftarrow \mathbf{o}_{\text{acc}} + \text{sum}(\mathbf{C}_j, \text{dim} = 0)$ 
26:
27:   // Compute and store un-normalized importance for
   the block
28:    $\mathbf{s}_{\text{loss},j} \leftarrow \text{sum}(|\mathbf{C}_j|, \text{dim} = 1)$   $\triangleright$  L1-norm of
   contribution vectors
29:   Store  $\mathbf{s}_{\text{loss},j}$  at corresponding indices in  $\mathbf{S}_{\text{loss}}$ 
30: end for
31:
32: // Post-processing after the loop
33:  $\mathbf{o}_t \leftarrow \mathbf{o}_{\text{acc}} / l_{\text{acc}}$   $\triangleright$  Normalize the final attention output
34:  $\mathbf{S}_{\text{final\_loss}} \leftarrow \mathbf{S}_{\text{loss}} / l_{\text{acc}}$   $\triangleright$  Normalize the importance scores
35:  $i_{\text{evict}} \leftarrow \text{argmin}(\mathbf{S}_{\text{final\_loss}})$   $\triangleright$  Find the token with the
   minimum score
36:
37: return  $\mathbf{o}_t, i_{\text{evict}}$ 

```

11–14 range. The larger maximum batch size of LongFlow is jointly due to KV compression and the reduction of memory fragmentation through static allocation.

D.6 Static Allocation under Mixed-Length Workloads

The primary focus of this paper is long-form reasoning, where generation lengths frequently approach several thousand tokens and often reach the 16k upper bound. Under this setting, static KV allocation is well aligned with the workload and does not introduce significant waste.

For mixed workloads with many short requests, pre-allocating large KV buffers may lead to underutilized memory. In practical systems, several strategies can mitigate this issue:

- **Workload-aware routing:** short requests can be routed to standard inference nodes, while long-form reasoning requests are routed to LongFlow-enabled nodes.
- **Lazy static allocation:** static allocation can be triggered only when the KV cache grows beyond a certain threshold, rather than being allocated at the beginning of decoding.
- **Hybrid allocation strategies:** dynamic allocation can be used in the early decoding stages, transitioning to static allocation once the KV cache reaches the configured budget.

These deployment strategies are orthogonal to the core LongFlow algorithm and can be incorporated in production systems when handling heterogeneous workloads.

D.7 Full-Trajectory Query-Similarity Analysis

The preliminary query-similarity analysis in Figure 5(b) covers only the first 100 generated tokens from one layer and one head. To examine

Table 6: Adjacent-query cosine similarity over the analyzed Qwen3-8B AIME24 trajectories.

Query space	# Measurements	Mean	Std.	Variance	Median	5th pct.	10th pct.
Pre-RoPE	9,919,872	0.8330	0.1221	0.0149	0.8659	0.5813	0.6693
Post-RoPE	9,919,872	0.8277	0.1219	0.0149	0.8599	0.5773	0.6643

whether the query approximation remains valid over longer trajectories, we analyze two completed AIME24 trajectories generated by Qwen3-8B, covering more than 8,000 output tokens. We compute the cosine similarity between adjacent queries over all generated positions, layers, and query heads, yielding 9,919,872 layer-head-step measurements. Table 6 reports the aggregate statistics before and after rotary positional embeddings (RoPE).

To examine whether similarity degrades as generation proceeds, we further report Post-RoPE statistics at representative generation positions in Table 7.

Table 7: Post-RoPE adjacent-query similarity across representative generation ranges. Low-sim. is the fraction of measurements below 0.8.

Generation range	Mean	Std.	Low-sim. (%)
0–512	0.8153	0.1211	34.04
512–1,024	0.8180	0.1222	32.97
1,024–1,536	0.8203	0.1262	31.98
1,536–2,048	0.8196	0.1247	32.35
3,072–3,584	0.8365	0.1223	27.23
3,584–4,096	0.8359	0.1221	27.32
4,096–4,608	0.8310	0.1170	28.64
4,608–5,120	0.8419	0.1079	24.69

As shown in Tables 6 and 7, adjacent queries remain similar on average, and the reported generation ranges do not exhibit a systematic decrease in similarity. These results support the use of the current query as a practical proxy during long-form generation. Nevertheless, the analysis is based on two trajectories and should be interpreted as empirical evidence rather than a worst-case guarantee.

D.8 Throughput Scalability on Larger Models

We evaluate LongFlow on Qwen3-4B and Qwen3-8B using an NVIDIA H20 96GB GPU. The input length is 512, the KV-cache budget is 3,200, and the output length is 4,096. We compare LongFlow with H2O, VATP, and R-KV at batch sizes 16, 24, and 32. The baselines use their practical periodic-eviction setting, approximately once every 128 decoding steps, whereas LongFlow performs fused eviction at every step. The throughput results are reported in Table 8.

Table 8: Throughput (tokens/s) on Qwen3-4B and Qwen3-8B. All experiments use an NVIDIA H20 96GB GPU, input length 512, KV-cache budget 3,200, and output length 4,096.

Model	Batch size	H2O	VATP	R-KV	LongFlow
Qwen3-4B	16	190.34	217.57	230.14	383.15
Qwen3-4B	24	249.76	232.05	245.96	329.87
Qwen3-4B	32	261.32	225.96	169.68	504.07
Qwen3-8B	16	223.83	208.51	110.86	356.47
Qwen3-8B	24	188.24	225.46	119.30	450.13
Qwen3-8B	32	254.17	238.02	251.83	485.18

As shown in Table 8, LongFlow achieves the highest throughput in every tested configuration. This shows that its efficiency advantage is not limited to Qwen3-1.7B and extends to the 4B and 8B scales under the evaluated setting.

D.9 Operator-Level Kernel Ablation

To disentangle the algorithmic contribution of the LongFlow importance metric from the implementation gain of the fused kernel, we keep the eviction metric fixed and compare two implementations of the same attention-and-eviction computation: a PyTorch implementation and the fused Triton kernel. The experiment uses Qwen3-4B on an NVIDIA H20 96GB GPU with batch size 128 and KV length 3,200. Table 9 reports the operator-level latency while holding the eviction metric fixed.

Table 9: Operator-level latency with the LongFlow eviction metric held fixed.

Batch	KV len.	PyTorch (ms)	Triton (ms)	Speedup
128	3,200	26.34	5.22	5.05×

As shown in Table 9, the fused kernel reduces operator latency from 26.34 ms to 5.22 ms, a 5.05× speedup for the same eviction metric. This result shows that LongFlow’s efficiency comes from two complementary components: a low-cost importance metric derived from attention intermediates and a fused kernel that realizes the metric with reduced memory traffic.