FROM RIGID TO DYNAMIC: ENTROPY-GUIDED ADAPTIVE INFERENCE FOR ACCELERATING LONG-CONTEXT LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing sparse attention and Key-Value (KV) cache compression techniques for long-context Large Language Model (LLM) inference predominantly rely on fixed sparsity patterns or uniform compute budgets across attention heads, failing to accommodate the fine-grained variations in attention behavior across different heads and contexts. In this work, we reveal that attention heads exhibit two distinct entropy dynamics: *Rigid Heads*, whose row-wise entropy remains near zero across input segments, indicating near-deterministic attention; and *Dynamic Heads*, whose entropy fluctuates substantially, reflecting context-sensitive semantic processing. Crucially, the categorization of these heads is context-dependent and cannot be reliably profiled offline. Building on these observations, we propose **EntropyInfer**, a training-free framework that leverages online attention entropy to adaptively allocate compute resources at the granularity of individual heads and segments during prefilling. For the decoding stage, EntropyInfer introduces a latent KV cache compression mechanism that incorporates newly generated output tokens into an observation window to accurately re-rank and retain key-value entries, resolving the prefill-to-decode attention distribution mismatch. Extensive experiments across the Llama, Qwen, and openPangu model families demonstrate that EntropyInfer consistently outperforms state-of-the-art baselines (including SnapKV, AdaKV, and CritiPrefill), achieving up to a $2.39\times$ end-to-end speedup for sequence lengths exceeding 100k tokens with negligible quality degradation compared to full attention. The code is available at `https://anonymous.4open.science/r/EntropyInfer`.

1 INTRODUCTION

Long-context inference has emerged as a primary efficiency bottleneck for modern Large Language Model (LLM) deployment. As input contexts scale to hundreds of thousands of tokens, two major cost centers emerge: quadratic attention computation during prefilling becomes a major contributor to end-to-end latency, while the linearly growing KV cache dominates memory consumption during decoding. (LI et al., 2025; Liu et al., 2025; Shi et al., 2024; Wang et al., 2024; Zhen et al., 2025). Traditional techniques for mitigating these costs prune information from either the attention matrix or the KV cache, creating a persistent trade-off between inference speed and generation fidelity. In this paper, we show that attention entropy evaluated per head and per segment during runtime exposes rich, fine-grained structural properties that prior methods neglect, enabling substantial inference speedups with high generation quality.

The prefilling compute bottleneck stems from the quadratic scaling of attention matrices, which severely inflates Time-To-First-Token (TTFT) in long-context applications. Existing sparse prefilling paradigms accelerate this stage by computing only a selected subset of attention entries. However, prior strategies suffer from inherent structural limitations. MInference (Jiang et al., 2024) enforces fixed geometric topologies that fail to adapt to input-dependent attention shifts; FlexPrefill (Lai et al., 2025) applies a uniform sparsity threshold layer-wide, ignoring head-level functional diversity; CritiPrefill (Lv et al., 2025) allocates a fixed block budget per segment, treating all heads as equally informative. Crucially, these methods fail to capture the diverse semantic information shift across positions, heads, or contexts.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

During the decoding stage, the linearly growing KV cache imposes severe GPU memory pressure. Popular eviction-based techniques, such as SnapKV (Li et al., 2024) and AdaKV (Feng et al., 2026), prune the KV cache at the prefill-to-decode boundary by scoring token importance exclusively under prefill attention signals. However, recent studies (Li et al., 2025b; Wu et al., 2026) reveal a fundamental mismatch: attention distributions shift substantially once generation begins, meaning tokens deemed critical during prefilling are not necessarily those driving subsequent token generation. Committing to permanent cache eviction prior to decoding thus risks discarding context entries essential for accurate generation.

While head-aware approaches like RazorAttention (Tang et al., 2025) recognize head heterogeneity, they rely on offline profiling over static calibration datasets. Through a per-head entropy analysis across Llama-3.1-8B-Instruct and Qwen-2.5-7B-Instruct, we uncover two phenomena that reveal the fundamental limits of offline profiling: First, attention heads cleanly separate into two operational regimes: *Rigid Heads*, whose row-wise entropy remains near zero ($\leq 10^{-5}$) across segments within a given input, indicating near-deterministic attention; and *Dynamic Heads*, whose entropy fluctuates significantly across query positions, capturing context-sensitive semantic dependencies. Second, the categorization of a head into Rigid or Dynamic is itself context-dependent—a single head may act as Rigid under one prompt and Dynamic under another. Consequently, head categorization cannot be fixed offline and must be dynamically evaluated online, where entropy fluctuations provide a computationally lightweight signal for adaptive resource allocation.

Driven by these insights, we introduce **EntropyInfer**, a training-free framework that leverages online attention entropy to adaptively allocate compute across both prefilling and decoding stages:

- **Entropy-Guided Sparse Prefilling**: We estimate per-head, per-segment entropy using a lightweight observation attention matrix. Rigid heads are assigned a minimal, constant compute budget, whereas Dynamic heads receive budgets that scale dynamically with inter-segment entropy variations, concentrating computation on highly uncertain attention regions.
- **Latent KV Cache Compression**: Rather than evicting KV cache entries immediately after prefilling, we defer compression until a small initial set of output tokens is generated. Including these early generated tokens in the observation window allows us to re-rank cache entries based on actual decoding signals, effectively resolving the prefill-to-decode mismatch.

EntropyInfer operates as a drop-in replacement for standard attention mechanisms and requires no fine-tuning or training overhead.

Our main contributions are summarized as follows:

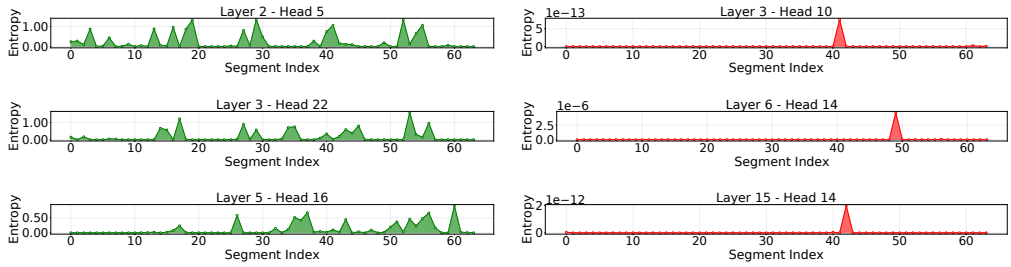
- We discover that attention heads divide into Rigid and Dynamic regimes characterized by segment-wise entropy dynamics, and prove that their categorization is context-dependent, necessitating online estimation.
- We propose EntropyInfer, a unified, training-free framework featuring entropy-guided prefill budget allocation and latent output-token-assisted KV cache compression.
- Extensive evaluations on LongBench, InfiniteBench, and LoCoMo across Llama, Qwen, and openPangu model families show that EntropyInfer delivers up to a $2.39\times$ end-to-end speedup beyond 100k tokens while maintaining generation quality comparable to full dense attention.

2 Related Work

We review methods that reduce the two main costs targeted by EntropyInfer: attention computation during prefilling and KV-cache overhead during decoding.

2.1 Prefilling Acceleration

Long-context prefilling has been accelerated through fixed or blockwise sparse patterns (Beltagy et al., 2020; Zaheer et al., 2020; Acharya et al., 2025), context-aware or head-aware sparse attention (Jiang et al., 2024; Lai et al., 2025; Lv et al., 2025), and model-level computation reuse (Qiao et al., 2025). These approaches reduce prefill cost, but do not jointly allocate computation budgets



(a) Dynamic heads. Max segment entropy $> 10^{-5}$. (b) Rigid heads. Max segment entropy $\leq 10^{-5}$.

Figure 1: Entropy curves exhibit two distinct operational regimes across different attention heads.

online across heads, sequence segments, and input contexts according to current attention dispersion. EntropyInfer addresses this gap with per-head, per-segment entropy-guided budgeting.

2.2 KV Cache Compression

KV-cache research spans eviction and budget allocation (Li et al., 2025a; Zhang et al., 2023; Li et al., 2024; Feng et al., 2026; Yang et al., 2024; Xiao et al., 2024; Liu et al., 2023; Zhang et al., 2025b), query-aware access and transfer compression (Ribar et al., 2024; Tang et al., 2024; Liu et al., 2024a), and improved token-importance or head-importance estimation (Xiong et al., 2025; Kim & Jung, 2025; Tang et al., 2025; Xiao et al., 2025). Recent studies (Li et al., 2025b; Wu et al., 2026) suggest that most eviction methods nevertheless depend on signals collected during prefilling or on head roles determined before generation, despite evidence that attention distributions shift during decoding. EntropyInfer instead re-ranks KV entries after observing early generated tokens and determines head behavior online for each context. Quantization (Liu et al., 2024b; Hooper et al., 2024) and offloading (Sheng et al., 2023; Zhang et al., 2025a) are complementary approaches that can be combined with cache eviction.

3 Motivation

3.1 Motivation 1: Head-Level Attention Entropy Heterogeneity

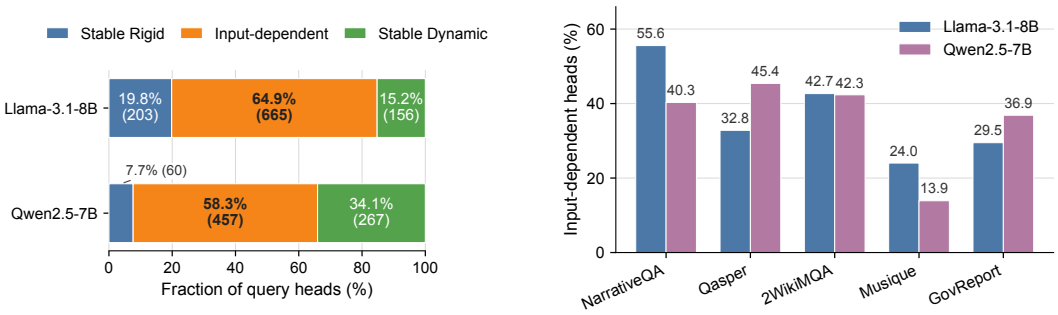
Prior studies (Kim & Jung, 2025; Xiong et al., 2025) demonstrate the utility of Shannon Entropy (Shannon, 1948) for quantifying the dispersion of attention distributions. Formally, for query and key projection matrices $Q_h, K_h \in \mathbb{R}^{N\times d}$ corresponding to attention head h, sequence length N, and head dimension d, the attention weight is computed as $A_h = \mathrm{softmax}(Q_h K_h^\top/\sqrt{d}) \in \mathbb{R}^{N\times N}$. For the i-th query token, the row-wise attention entropy $H(a_{h,i:})$ measures the dispersion of attention across all N keys:

$$H(a_{h,i:}) = -\sum_{j=1}^{N} a_{h,i,j} \log a_{h,i,j} \tag{1}$$

where $a_{h,i,j}$ represents the attention probability assigned by the i-th query to the j-th key. A high entropy value $H(a_{h,i:})$ indicates that attention is broadly dispersed over keys, whereas a near-zero value indicates extreme concentration on a small subset of tokens.

To analyze how attention concentration evolves across query positions, we track per-head entropy trajectories across sequence positions. As depicted in Figure 1, attention heads cleanly exhibit two contrasting behaviors:

- **Rigid Heads**: The row-wise entropy remains consistently below a tiny threshold (e.g., 10^{-5}) across query positions. This indicates near-deterministic, sharp attention distributions that focus rigidly on specific tokens.
- **Dynamic Heads**: The row-wise entropy fluctuates dynamically across query positions, reflecting input-sensitive semantic reasoning where attention dispersion varies substantially.



(a) Head Status Composition.

(b) Variability within each task.

Figure 2: Head type analysis across inputs. Head types vary substantially across documents and within tasks, motivating online classification.

This sharp dichotomy proves that attention heads exhibit structural heterogeneity, confirming that allocating identical compute budgets to all heads is inherently inefficient.

3.2 Motivation 2: Context-Dependent Dynamic Shifts of Attention Heads

To examine whether head classification remains static across different inputs, we evaluate the entropy distributions of attention heads under varying context prompts on Llama-3.1-8B-Instruct. Following the representation vector and observation attention construction in Algorithms 2 and 3, we partition query and key sequences into segments and monitor each head's maximum segment entropy across different documents.

As the input context changes, approximately 64.9% of Llama heads and 58.3% of Qwen heads switch between the Dynamic and Rigid categories, as shown in Figure 2(a). The within-task variability, computed across different input instances, is also substantial, as shown in Figure 2(b). This context-dependent behavior demonstrates that offline profiling cannot reliably classify head roles, highlighting the crucial need to determine head behavior online during inference.

We also provide detailed heatmaps for different inputs in Figure 10 in the appendix.

4 Methodology

In this section, we present the core design of EntropyInfer, encompassing entropy-guided sparse prefilling and latent decoding-stage KV cache compression. Figure 3 illustrates the overall architecture.

4.1 Key Notations

We formalize our framework with the following notation: N denotes the total input context length; L_S and C_S represent the segment length and total segment count, respectively; L_B specifies the block size. Each segment $i \in [1, C_S]$ is assigned a block count C_{B_i} and a compute budget B_i. B_h designates the base budget for sparse prefilling, while B_d represents the target KV cache budget during decoding eviction.

4.2 Entropy-Guided Sparse Prefilling

To dynamically allocate compute resources across attention matrix blocks without incurring the prohibitive $\mathcal{O}(N^2)$ cost of full attention computation, we construct a lightweight "observation attention matrix", denoted as A^{obs}, following block-level aggregation strategies (Lv et al., 2025). Algorithms 2 and 3 in the appendix outline the process for extracting representative query/key vectors $(Q_{\text{Rep}}, K_{\text{Rep}})$ and computing A^{obs}.

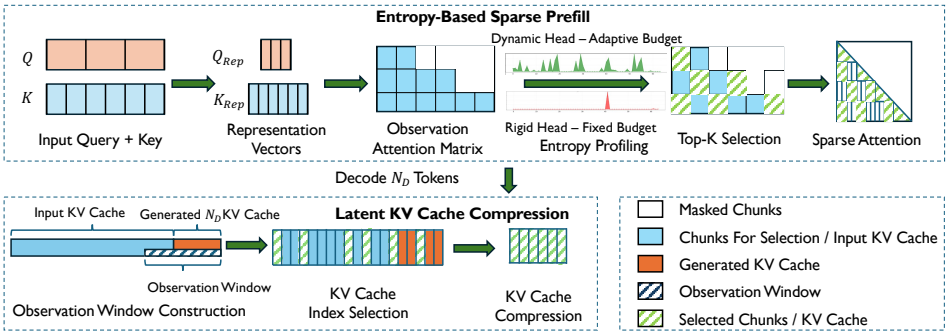


Figure 3: Overview of the EntropyInfer framework.

Once A^{obs} is obtained, we evaluate the row-wise entropy $H(a_{i:})$ for each segment, as detailed in Algorithm 1. To determine the optimal number of key blocks allocated to each query segment, we track entropy variations across consecutive segments. We classify each attention head online as either Rigid or Dynamic based on whether its maximum segment entropy exceeds a predefined threshold e_t.

For **Rigid Heads** ($\max(\{e_i\}) < e_t$), attention dispersion remains consistently negligible; hence, allocating a fixed minimal budget B_h per segment suffices. For **Dynamic Heads**, entropy variations capture fluctuating semantic information. We set an initial budget $B_0 = B_h$ for the first segment and dynamically adjust subsequent segment budgets B_i according to the relative entropy fluctuation $(e_i - e_{i-1})/e_{i-1}$. Higher fluctuation indicates rapid semantic shifts requiring higher attention resolution (larger budget), while lower fluctuation suggests stable semantic focus, allowing budget preservation.

Algorithm 1 Entropy-Guided Prefill Budget Allocation

Require: Segment Entropy Sequence $\{e_i\}$ for $i \in [0, C_S - 1]$, Base Budget B_h
Ensure: Segment Budget Allocation $\{B_i\}$

$e_t \leftarrow 10^{-5}$ ▷ Entropy threshold for head classification
$\alpha \leftarrow 0.5$ ▷ Budget scaling coefficient
$\Delta_t \leftarrow 0.4$ ▷ Variance threshold ratio
if $\max(\{e_i\}) < e_t$ **then** ▷ Rigid Head behavior
  $B_i \leftarrow B_h, \quad \forall i \in [0, C_S - 1]$
else ▷ Dynamic Head behavior
  $B_0 \leftarrow B_h$
  for $i = 1$ **to** $C_S - 1$ **do**
    if $i = 1$ **then** $r_i \leftarrow \Delta_t$
    else $r_i \leftarrow \text{clip}\left(\frac{e_i - e_{i-1}}{e_{i-1}}, 0,\ 1 + \Delta_t\right)$
    end if
    $\Delta_{\text{budget}} \leftarrow \alpha \cdot B_h \cdot (r_i - \Delta_t)$
    $B_i \leftarrow B_{i-1} + \Delta_{\text{budget}}$
    $B_i \leftarrow \min(\max(B_i, B_0), 3 \cdot B_0)$ ▷ Clamp budget range
  end for
  end if
  return $\{B_i\}$

4.3 Latent Decoding-Stage KV Cache Compression

Compressing the KV cache during generation is vital for sustaining long-sequence decoding. Conventional techniques (Li et al., 2024; Feng et al., 2026) compress the cache immediately at the conclusion of the prefilling stage using prompt-tail observation windows. However, this strategy is suboptimal, as output tokens exhibit distinct attention distributions from prompt tokens and provide stronger guidance for token generation (Li et al., 2025b; Wu et al., 2026).

To eliminate this prefill-to-decode mismatch, we introduce **latent decoding-stage KV cache compression**, detailed in Algorithm 4 in the appendix. Rather than evicting tokens at step 0 of decoding, we allow the LLM to generate N_D output tokens first. We then construct an expanded observation window incorporating both prompt-tail queries and newly generated output queries Q_W, compute their cross-attention scores against historical keys K, and select top-K cache entries according to cumulative weight vector $\mathbf{w}$.

4.4 Complexity Analysis

The overall prefilling computational complexity for each attention head under EntropyInfer is upper-bounded by:

$$\mathcal{O}\left(\left(\frac{1}{L_S L_B}+\frac{3}{2L_S L_B}\log\frac{B_h}{L_B}\right)N^2+3B_h N\right) \tag{2}$$

where L_S and L_B refer to the segment and block sizes, respectively. B_h refers to the base budget for sparse prefilling and N denotes the sequence length. Because the coefficient associated with N^2 is orders of magnitude smaller than that of standard dense attention (where $L_S L_B \gg 1$), the effective compute complexity scales nearly linearly with context length N, as visually validated in Figure 4 (a). Detailed complexity derivation is provided in the Appendix A.8.

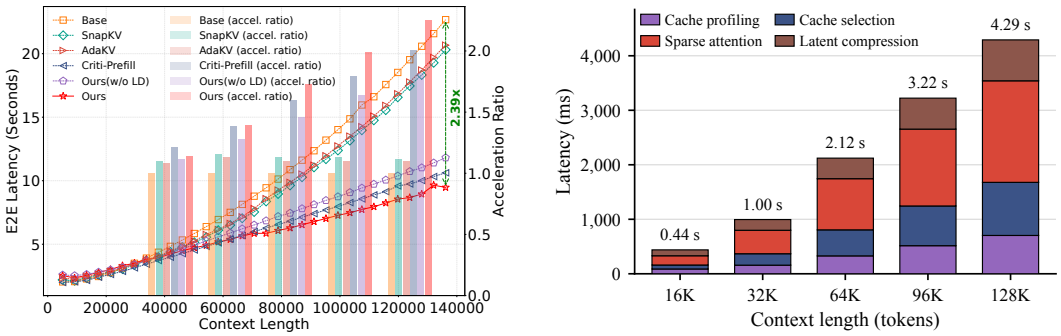
5 Experiments

5.1 Experimental Settings

We evaluate EntropyInfer on (1) InfiniteBench (Zhang et al., 2024), an ultra-long-context benchmark comprising real-world and synthetic tasks with input lengths exceeding 100K tokens; (2) LongBench (Bai et al., 2024), a widely adopted dataset covering question answering, summarization, few-shot learning, and code completion; and (3) RULER (Hsieh et al., 2024), a challenging benchmark that evaluates model performance across configurable context lengths using question-answering and needle-in-a-haystack tasks. We conduct comprehensive evaluations using Llama, Qwen, and openPangu backbones. We compare against (1) **SnapKV** (Li et al., 2024), which compresses the KV cache using a snapshot mechanism, reducing inference latency and improving memory utilization; (2) **AdaKV** (Feng et al., 2026), which reallocates the cache budget across attention heads, assigning more capacity to heads with more dispersed attention patterns; (3) **CritiPrefill** (Lv et al., 2025), which uses criticality-based sparse attention to accelerate the prefill stage; and (4) **CritiPrefill + SnapKV / AdaKV** for the RULER experiments, which combine CritiPrefill's sparse-prefill component with the corresponding KV-cache compression method. Unless stated otherwise, the sparse-prefill and decoding-cache budgets are 2048 and 1024 tokens, respectively. Efficiency is measured on NVIDIA H100 GPUs while generating 100 tokens. Full dataset statistics, metric definitions, model details, hyperparameters, and hardware configurations are provided in Appendix A.1.

Table 1: Results on InfiniteBench. R stands for Retrieve, M stands for math, C stands for Code. CPrefill stands for CritiPrefill. Bold entries indicate the best results, while underlined entries indicate the second-best results.

Model	Llama-3.1-8B-Instruct						Qwen2.5-7B-Instruct					
Method	Base	SnapKV	AdaKV	CPrefill	**Ours**	**Ours w/o LD**	Base	SnapKV	AdaKV	CPrefill	**Ours**	**Ours w/o LD**
R.PassKey	100.00	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	100.00	85.42	83.56	**99.15**	96.44	96.44
R.Number	99.49	85.59	82.37	**99.15**	95.42	95.42	93.22	4.41	4.07	**89.49**	86.78	86.78
En.Dia	21.00	11.50	15.00	11.00	16.50	**19.00**	16.00	11.00	12.00	10.50	12.00	**15.00**
En.Sum	26.32	22.35	22.00	24.97	22.02	**26.48**	21.95	19.95	**20.62**	19.84	19.75	19.40
En.MC	66.81	**66.81**	66.38	56.77	62.01	62.01	46.72	**46.72**	46.29	37.99	44.10	44.98
En.QA	14.44	11.77	12.21	12.92	14.02	**14.14**	4.81	4.60	4.71	4.43	**5.26**	4.84
Zh.QA	13.22	12.03	12.05	11.03	12.46	**12.70**	9.12	**9.32**	9.18	8.39	9.02	8.64
M.Find	33.14	33.14	32.86	34.86	**35.14**	**35.14**	38.57	35.14	35.71	34.57	**44.57**	**44.57**
C.Debug	22.08	22.08	22.08	**26.65**	25.38	25.38	26.14	**25.63**	**25.63**	24.11	23.35	23.35
Avg	44.06	40.59	40.55	41.93	42.55	**43.36**	39.62	26.91	26.86	36.50	37.92	**38.22**



(a) End-to-end latency.

(b) Latency breakdown.

Figure 4: End-to-end latency and latency breakdown using Llama-3.1-8B-Instruct. Lower latency and higher speedup are better.

5.2 Main Results

InfiniteBench Ultra-Long Context Results. Table 1 shows that EntropyInfer achieves higher average scores than all efficiency-oriented baselines. On Llama-3.1-8B-Instruct, Ours and Ours w/o LD obtain 42.55 and 43.36, respectively, compared with 41.93 for CritiPrefill, the strongest baseline; on Qwen2.5-7B-Instruct, the corresponding scores are 37.92 and 38.22, versus 36.50. Moreover, Ours w/o LD remains within only 0.70 and 1.40 points of dense inference on the two models, while enabling latent decoding introduces modest additional drops of 0.81 and 0.30 points. These results demonstrate that entropy-guided sparse prefilling effectively preserves task-relevant information in ultra-long contexts, while latent decoding provides further compression with limited quality degradation.

LongBench Results. The LongBench results in Appendices A.4 and A.9 further demonstrate the consistent effectiveness of EntropyInfer across Llama, Qwen, and openPangu backbones. The complete method improves the average score from 48.32 to 48.55 on Llama-3.1-8B-Instruct and from 47.51 to 47.68 on Qwen2.5-7B-Instruct, while Ours w/o LD further reaches 48.91 and 47.98, respectively. EntropyInfer also closely tracks dense inference on Qwen3 and openPangu. Ours w/o LD achieves average differences of at most 0.53 points, and even improves the openPangu-7B result with chain-of-thought from 38.40 to 38.89. Together, these results show that EntropyInfer generalizes across model families and different long-context tasks, retaining most of the dense model's generation quality while supporting efficient inference.

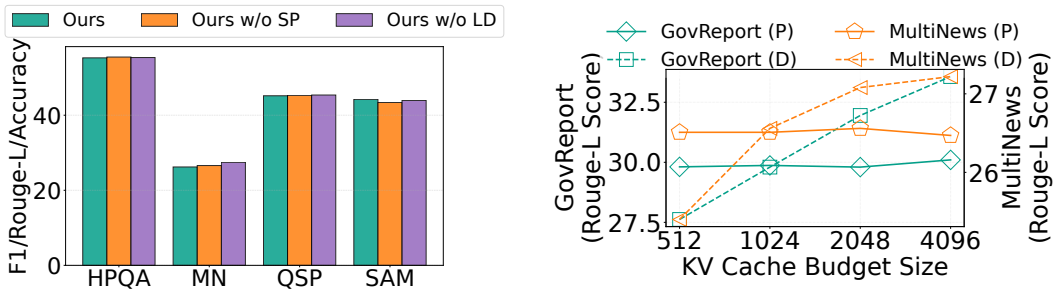
5.3 Efficiency Evaluation

End-to-end Latency Evaluation. Figure 4 (a) illustrates end-to-end latency benchmarks on Llama-3.1-8B-Instruct across context lengths ranging from 4k to 140k tokens. EntropyInfer delivers substantial runtime speedups over dense inference, scaling up to a 2.39× **acceleration ratio** at 140k context length. When using sparse prefilling alone (Ours w/o LD), prefill computation incurs minor per-head indexing overhead compared to CritiPrefill. However, adding latent decoding compression (Ours) resolves this overhead by accelerating token generation steps, making EntropyInfer the fastest method overall in long-context regimes.

Latency Breakdown. We report EntropyInfer's latency breakdown across context lengths in Figure 4(b), with detailed values in Table 14. Sparse attention dominates the overhead but scales near-linearly with context length, highlighting EntropyInfer's advantage over $\mathcal{O}(N^2)$ full attention.

5.4 Further Analysis

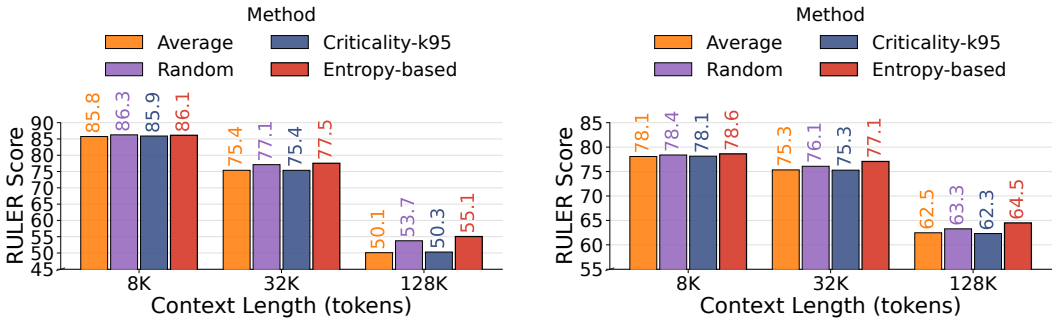
Module ablation. We isolate the contribution of each EntropyInfer component on LongBench using Llama-3.1-8B-Instruct. We compare the complete method against two variants: (1) **Ours**, which enables entropy-based sparse prefill and latent decode; (2) **Ours w/o SP**, which disables entropy-



(a) Module ablation.

(b) Sensitivity to budgets.

Figure 5: Module ablation and sensitivity to prefill (P) and decoding (D) budgets.



(a) Llama-3.1-8B-Instruct

(b) Qwen3-4B

Figure 6: Entropy-allocation comparison on Llama and Qwen backbones. EntropyInfer achieves the best overall generation quality among the evaluated allocation schemes.

based sparse attention, enables latent decode; and (3) **Ours w/o LD**, which enables entropy-based sparse attention, and disables latent decode.

Figure 5 (a) shows that removing either sparse prefill or latent decoding changes LongBench quality only marginally. Together, however, the two modules produce the largest end-to-end speedup (Figure 4(a)), confirming that they provide complementary efficiency gains without materially degrading generation quality.

Effectiveness Analysis of Entropy-based Allocation Scheme. To further investigate the effectiveness of our entropy-based budget allocation scheme, we compare our budget allocation scheme against the following budget allocation methods under the same prefill budget setting: (1) **Averaged**, which evenly distributes the budget across all query-head-segment units; (2) **Random**, which samples each unit's allocation uniformly within its feasible range using a fixed random seed. The sampled allocations are projected to match Entropy-based method's total budget; (3) **Criticality-k95**, which allocates blocks according to the minimum number of top-ranked historical blocks required to cover 95% of the estimated attention mass. The resulting demands are projected to the same total budget as Entropy-based.

As shown in Figure 6, EntropyInfer's prefill budget allocation scheme (Entropy-based) demonstrates consistent superiority against other schemes, demonstrating its ability to capture semantic information shift and maintain high generation quality. The result is not directly comparable to the RULER results in Table 2, since the latter profiles entropy information using averaged query value within each GQA group to ensure efficiency, whereas the results in Figure 6 profile each head independently to better reveal the differences among the budget allocation schemes.

Parameter Sensitivity. We assess EntropyInfer's sensitivity to prefill and decode budgets, entropy threshold and segment/block sizes.

Table 2: RULER results on Llama-3.1-8B-Instruct and Qwen3-8B. The methods are grouped by the inference stages they optimize.

Methods	Llama3.1-8B-Instruct						Qwen3-8B					
	4K	8K	16K	32K	64K	128K	4K	8K	16K	32K	64K	128K
Ours w/o LD	**86.57**	**85.9**	**79.82**	**74.32**	**71.69**	**51.24**	**83.8**	**80.07**	**80.46**	**77.93**	**70.23**	**57.01**
CritiPrefill	**86.57**	85.68	78.55	66.67	59.95	43.43	**83.8**	79.37	78.7	75.26	64.98	55.06
Ours w/o SP	**77.42**	**74.92**	**71.9**	**70.49**	**70.08**	**62.86**	69.96	64.21	**63.02**	**57.53**	**48.7**	**41.11**
SnapKV	71.92	66.42	60.53	57.7	53.96	41.15	**72.29**	**66.5**	62.17	57.3	43.61	25.89
AdaKV	71.25	65.57	60.03	56.84	53.32	39.56	70.63	65.25	60.55	56.38	42.74	24.81
Ours	**77.4**	**74.84**	**69.35**	**66.85**	**64.4**	**46.45**	69.71	64.52	**62.32**	**53.17**	**42.89**	**31.45**
CritiPrefill + SnapKV	71.96	65.62	57.74	48.17	39.23	21.34	**72.29**	**65.71**	59.86	49.59	32.65	19.28
CritiPrefill + AdaKV	71.3	65.09	58.16	47.72	38.18	19.33	70.63	64.33	58.56	48.23	30.64	18.24

Prefill Base Budget and Decode Cache Budget. Figure 5 (b) evaluates sensitivity to prefilling and decoding budgets. Prefill performance remains exceptionally stable across base budgets, reflecting the robust adaptivity of entropy-guided allocation. Increasing decoding cache budget yields marginal gains for specific tasks while increasing memory footprint, validating our default budget selection ($B_d = 1024$).

Entropy Threshold. We study the robustness of entropy threshold e_t. The study is detailed in Appendix A.2.1, showing that e_t is robust under different values, and therefore can effectively categorize the two types of heads proposed in EntropyInfer.

Segment and Block Sizes. We study EntropyInfer's sensitivity to segment and block sizes. The result and analysis are detailed in Appendix A.2.2. The overall result suggests that EntropyInfer is robust to these parameters, while adjusting the segment size according to task types can further enhance effectiveness.

Results on RULER.

RULER Hsieh et al. (2024) is a challenging benchmark with configurable context lengths up to 128K. To compare EntropyInfer with methods that accelerate the same inference stages, we divide the evaluation into three groups: {Ours, CritiPrefill+SnapKV, CritiPrefill+AdaKV} apply both sparse prefilling and decoding-stage KV cache compression; {Ours w/o LD, CritiPrefill} apply sparse prefilling only; and {Ours w/o SP, SnapKV, AdaKV} apply KV cache compression only.

As shown in Table 2, EntropyInfer has a significant advantage at context lengths above 32K. On Llama-3.1-8B-Instruct, all three EntropyInfer variants (Ours, Ours w/o LD, and Ours w/o SP) outperform the other baselines at context lengths above 32K and remain comparable at shorter lengths. Ours scores lower than its two single-stage variants because each component trades some generation quality for lower inference latency. On Qwen3-8B, the corresponding EntropyInfer variant achieves the best result within each of the three comparison groups at context lengths above 16K. We also report results on Qwen3-32B at a context length of 128K in Table 12 in the appendix. EntropyInfer outperforms the other methods within each comparison group, further demonstrating the scalability of our method.

6 Conclusion

In this paper, we introduced **EntropyInfer**, a training-free framework that accelerates long-context LLM inference through entropy-guided sparse prefilling and latent decoding-stage KV-cache compression. It allocates prefill computation online at head-and-segment granularity and re-ranks cached tokens after observing early generated outputs, addressing both context-dependent head behavior and prefill-to-decode attention shifts. Across Llama, Qwen, and openPangu models, EntropyInfer retains high generation quality while achieving up to a $2.39\times$ end-to-end speedup beyond 100k tokens, offering a practical drop-in solution for long-context inference.

AI USE STATEMENT

Generative AI tools were used to assist with language editing and document formatting. All AI-assisted content was reviewed and verified by the authors, who take full responsibility for the paper.

ETHICS STATEMENT

This work does not raise specific ethical concerns beyond the general environmental and resource costs associated with large-model inference.

REPRODUCIBILITY STATEMENT

We have released our code at `https://anonymous.4open.science/r/EntropyInfer`. Experimental settings are described in Section 5.1 and detailed in Appendix A.1, including datasets, backbone models, evaluation metrics, hyperparameters and hardware configurations.

REFERENCES

Shantanu Acharya et al. Star attention: Efficient LLM inference over long sequences. In Aarti Singh et al. (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 356–371. PMLR, July 2025.

Yushi Bai et al. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku et al. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172.

Iz Beltagy et al. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

Hanting Chen et al. Pangu Embedded: An Efficient Dual-system LLM Reasoner with Metacognition, May 2025a.

Jinhan Chen et al. SABlock: Semantic-Aware KV Cache Eviction with Adaptive Compression Block Size, October 2025b.

Yuan Feng et al. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *The Thirty-Ninth Annual Conference on Neural Information Processing Systems*, 2026.

Aaron Grattafiori et al. The Llama 3 Herd of Models, November 2024.

Coleman Hooper et al. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, Nips '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 979-8-3313-1438-5.

Cheng-Ping Hsieh et al. RULER: What is the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

Huiqiang Jiang et al. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, Nips '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 979-8-3313-1438-5.

Heekyum Kim et al. Entropy-guided KV caching for efficient LLM inference. *Mathematics*, 13 (15):1–14, July 2025. doi: None.

Xunhao Lai et al. FlexPrefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *The Thirteenth International Conference on Learning Representations*, 2025.

Haoyang LI et al. A survey on large language model acceleration based on KV cache management. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856.

Haoyang Li et al. A survey on large language model acceleration based on kv cache management. *Transactions on Machine Learning Research*, 2025a.

Haoyang Li et al. LoopServe: An Adaptive Dual-phase LLM Inference Acceleration System for Multi-Turn Dialogues, July 2025b.

Mo Li et al. NeedleBench: Evaluating LLM retrieval and reasoning across varying information densities. *Transactions on Machine Learning Research*, 2025c. ISSN 2835-8856.

Yuhong Li et al. SnapKV: LLM knows what you are looking for before generation. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, Nips '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 979-8-3313-1438-5.

Jiaheng Liu et al. A comprehensive survey on long context language modeling, 2025.

Yuhan Liu et al. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, Acm Sigcomm '24, pp. 38–56, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 979-8-4007-0614-1. doi: 10.1145/3651890.3672274.

Zichang Liu et al. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Nips '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Zirui Liu et al. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. *arXiv preprint arXiv:2402.02750*, 2024b.

Junlin Lv et al. CritiPrefill: A segment-wise criticality-based approach for prefilling acceleration in llms. In *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, 2025. doi: 10.1109/ICASSP49660.2025.10887916.

Adyasha Maharana et al. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024.

Aurick Qiao et al. SwiftKV: Fast prefill-optimized inference with knowledge-preserving model transformation. In Christos Christodoulopoulos et al. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25734–25753, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1306.

Qwen et al. Qwen2.5 Technical Report, January 2025.

Luka Ribar et al. SparQ attention: Bandwidth-efficient LLM inference. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024.

C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3): 379–423, July 1948. ISSN 00058580. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Ying Sheng et al. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, Honolulu, Hawaii, USA, 2023. JMLR.org.

Luohe Shi et al. Keep the cost down: A review on methods to optimize LLM' s KV-cache consumption, 2024.

Alexey Svyatkovskiy et al. IntelliCode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1433–1443, Virtual Event USA, November 2020. ACM. ISBN 978-1-4503-7043-1. doi: 10.1145/3368089.3417058.

Hanlin Tang et al. RazorAttention: Efficient KV cache compression through retrieval heads. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jiaming Tang et al. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, Vienna, Austria, 2024. JMLR.org.

Xindi Wang et al. Beyond the limits: A survey of techniques to extend the context length in large language models. In Kate Larson (ed.), *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pp. 8299–8307. International Joint Conferences on Artificial Intelligence Organization, August 2024. doi: 10.24963/ijcai.2024/917.

Wenbo Wu et al. LouisKV: Efficient KV cache retrieval for long input-output sequences. In *The Fourteenth International Conference on Learning Representations*, 2026.

Guangxuan Xiao et al. Efficient streaming language models with attention sinks. In B. Kim et al. (eds.), *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Guangxuan Xiao et al. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jing Xiong et al. UNComp: Can Matrix Entropy Uncover Sparsity? — A Compressor Design from an Uncertainty-Aware Perspective. In Christos Christodoulopoulos et al. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 4179–4199, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.209.

An Yang et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Dongjie Yang et al. PyramidInfer: Pyramid KV cache compression for high-throughput LLM inference. In Lun-Wei Ku et al. (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3258–3270, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.195.

Manzil Zaheer et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33, 2020.

Huawei Zhang et al. KVSwap: Disk-aware KV Cache Offloading for Long-Context On-device Inference, December 2025a.

Xinrong Zhang et al. ∞Bench: Extending Long Context Evaluation Beyond 100K Tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.814.

Yanqi Zhang et al. DiffKV: Differentiated memory management for large language models with parallel KV compaction. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, Sosp '25, pp. 431–445, New York, NY, USA, 2025b. Association for Computing Machinery. ISBN 979-8-4007-1870-0. doi: 10.1145/3731569.3764810.

Zhenyu Zhang et al. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Nips '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Ranran Zhen et al. Taming the titans: A survey of efficient LLM inference serving. In Lucie Flek et al. (eds.), *Proceedings of the 18th International Natural Language Generation Conference*, pp. 522–541, Hanoi, Vietnam, October 2025. Association for Computational Linguistics.

A APPENDIX

The appendix is organized as follows. Appendix A.1 details the datasets, backbone models, evaluation metrics, hyperparameters, and hardware configurations. Appendix A.2 studies sensitivity to the entropy threshold, segment size, and block size. Appendix A.3 ablates the Dynamic–Rigid head distinction. Appendix A.4 reports the full LongBench results and additional RULER results. Appendix A.5 quantifies the average compute budgets assigned by the dynamic allocation mechanism. Appendix A.6 provides a detailed latency breakdown and memory-efficiency analysis. Appendix A.7 presents the algorithms for representation-vector construction, observation-attention computation, and latent-decoding KV-cache compression. Appendix A.8 derives the computational complexity of sparse prefilling. Appendix A.9 evaluates generalization to the openPangu model family through latency, LongBench, and LoCoMo results. Finally, Appendix A.10 visualizes how the Rigid–Dynamic head partition varies across inputs and task domains.

A.1 EXPERIMENTAL DETAILS

A.1.1 DATASETS AND BACKBONE MODELS

We evaluate task quality on RULER (Hsieh et al., 2024), LongBench (Bai et al., 2024), and InfiniteBench (Zhang et al., 2024). These benchmarks cover synthetic and real-world tasks including question answering, summarization, document retrieval, and code generation. InfiniteBench contains sequences beyond 100K tokens, whose detailed dataset lengths are reported in Table 3. For efficiency evaluation, we adapt the Needle-in-a-Haystack prompt configuration (Li et al., 2025c) and generate 100 output tokens.

Our evaluation covers the following open-source model families:

- Llama (Grattafiori et al., 2024): Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct
- Qwen (Qwen et al., 2025; Yang et al., 2025): Qwen2.5-7B-Instruct, Qwen3-4B, Qwen3-8B, and Qwen3-32B
- openPangu (Chen et al., 2025a): openPangu-Embedded-1B and openPangu-Embedded-7B.

Table 3: InfiniteBench dataset lengths.

Datasets	# Examples	Avg Input Tokens
R.PassKey	590	122.4k
R.Number	590	122.4k
En.Dia	200	103.6k
En.Sum	103	**171.5k**
En.MC	229	**184.4k**
En.QA	351	**192.6k**
Zh.QA	175	**2068.6k**
M.Find	350	87.9k
C.Debug	394	114.7k

A.1.2 EVALUATION METRICS

The metrics used for InfiniteBench, LongBench and RULER are detailed below.

- For LongBench, NarrativeQA, Qasper, MultifieldQA-en, HotpotQA, 2WikiMQA, TriviaQA, and Musique use F1 score; MultiNews, GovReport, QMSum, and SAMSum use ROUGE-L; TREC, Passage Count, and Passage Retrieval use accuracy; and LCC and RepoBench-P use edit similarity.
- For InfiniteBench, En.QA and Zh.QA use F1 score; En.Sum uses ROUGE-L-Sum; En.MC, En.Dia, Code.Debug, and Retrieve.PassKey use accuracy; and Retrieve.Number and Math.Find use exact match.
- For RULER, we use recall-based accuracy (Hsieh et al., 2024).

The core metrics are defined as follows:

- **F1 Score**: The harmonic mean of precision and recall over token overlaps.
- **Accuracy**: The percentage of correctly predicted instances.
- **ROUGE-L**: Measures the Longest Common Subsequence (LCS) ratio between candidate C and reference R: ROUGE-L $= \frac{\text{LCS}(C,R)}{|R|}$.
- **Edit Similarity**: Evaluates normalized Levenshtein distance between code sequences (Svyatkovskiy et al., 2020).
- **Exact Match**: Measures the ratio of model outputs matching the exact ground truth string.
- **Recall-based Accuracy**: Given a set of target answer items $\mathcal{A}$ and a model response $\hat{y}$, the per-example score is $\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} \mathbb{I}[a \text{ appears in } \hat{y}]$.

A.1.3 Hyperparameters and Hardware

Hyperparameters.

For SnapKV and AdaKV, the KV cache budget is configured to 1024 tokens. For CritiPrefill, the segment prefilling budget is set to 2048 tokens. For EntropyInfer, we set the sparse prefill base budget to $B_h = 2048$ tokens and the decoding eviction budget to $B_d = 1024$ tokens. For the fused methods including CritiPrefill+SnapKV and CritiPrefill+AdaKV, we set the sparse prefill budget to $B_h = 2048$ tokens, and the decoding eviction budget to $B_d = 1024$ tokens.

Hardware. For Llama-3.1-8B-Instruct, Qwen2.5-7B-Instruct, Qwen3-4B, Qwen3-8B, and the openPangu model series, we use a single NVIDIA H100 80 GB GPU with 192 GB of CPU memory and 8 CPU cores to conduct experiments. For larger models, we use 2 NVIDIA H100 80 GB GPUs for Qwen3-32B and 4 NVIDIA H100 80 GB GPUs for Llama-3.1-70B-Instruct. The CPU and memory resources scale by the same factor as the number of GPUs.

A.2 Additional Parameter Sensitivity Experiments

A.2.1 Entropy Threshold

To study the impact of the threshold e_t, we conducted parameter sensitivity experiments under different e_t values using four datasets (GovReport, Musique, NarrativeQA, and SAMSum) covering different prompt types on Llama-3.1-8B-Instruct (1024 heads) and Qwen2.5-7B-Instruct (784 heads). The results are shown in Table 4. Varying e_t from 10^{-6} to 10^{-4} only shifts a small fraction of heads between dynamic and rigid types (at most 8.89% for Llama and 5.10% for Qwen), demonstrating categorization robustness.

We also profiled dynamic heads under different context lengths using the Needle-in-a-Haystack dataset. Table 5 shows that the categorization remains highly consistent across lengths (diff rate $\leq 4.79\%$).

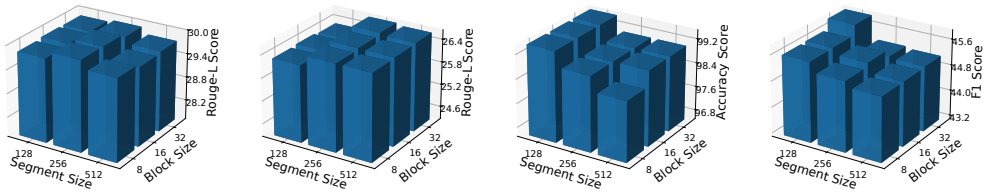
To check the distribution of categorization changes, we analyzed the per-layer dynamic head counts on Musique (Table 6). The differences are minimal and evenly distributed across layers (at most 4

Table 4: Number of dynamic heads under different e_t values (10^{-6} / 10^{-5} / 10^{-4}) on the test datasets. Llama and Qwen denote Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct, respectively. Diff is the percentage difference in dynamic-head count between $e_t = 10^{-4}$ and $e_t = 10^{-6}$.

Dataset v.s. Dynamic Heads	e_t = [10^{-6}/10^{-5}/10^{-4}] (Llama)	Diff (%)	e_t = [10^{-6}/10^{-5}/10^{-4}] (Qwen)	Diff (%)
GovReport (Summarization)	766 / 726 / 689	7.52%	682 / 667 / 646	4.59%
Musique (Multi-Document QA)	953 / 935 / 914	3.81%	756 / 752 / 744	1.53%
NarrativeQA (Single Document QA)	940 / 925 / 891	4.79%	761 / 757 / 755	0.77%
SAMSum (Few-shot Learning)	686 / 643 / 595	8.89%	640 / 626 / 600	5.10%

Table 5: Llama-3.1-8B dynamic heads under different context lengths. Diff Rate is the difference rate between 10^{-4} and 10^{-6}.

Context Length v.s. Dynamic Heads	e_t = [1e-6 / 1e-5 / 1e-4]	**Diff(%)**
12286	829 / 800 / 780	4.79%
36862	924 / 916 / 901	2.25%
69630	953 / 932 / 921	3.12%
102396	994 / 986 / 976	1.76%
131068	999 / 991 / 983	1.56%



(a) GovReport. (b) MultiNews. (c) Passage-Retrieval. (d) Qasper.

Figure 7: Parameter sensitivity across segment and block sizes.

heads changed per layer). In summary, our method is highly robust to e_t, and $e_t = 10^{-5}$ effectively categorizes attention heads.

Table 6: Llama per-layer analysis on Musique. Diff Rate is the difference rate between 10^{-4} and 10^{-6}.

Layer v.s. Dynamic Heads	e_t **= [1e-6 / 1e-5 / 1e-4]**	**Diff(%)**
Layer 0	32 / 32 / 31	3.12%
Layer 4	27 / 26 / 26	3.12%
Layer 8	32 / 32 / 31	3.12%
Layer 12	32 / 32 / 32	0.00%
Layer 16	32 / 32 / 32	0.00%
Layer 20	28 / 25 / 24	12.50%
Layer 24	30 / 29 / 28	6.25%
Layer 28	25 / 25 / 25	0.00%
Layer 31	29 / 29 / 25	12.50%

A.2.2 Segment and Block Size

We perform a grid search over segment length L_S and block size L_B, as shown in Figure 7. Summarization tasks (GovReport, MultiNews) benefit from larger segment sizes due to broader context aggregation, whereas retrieval and QA tasks prefer finer-grained segment sizes. Overall, block size variation remains stable, with larger blocks occasionally enhancing semantic continuity as also observed in block-sparse attention studies (Chen et al., 2025b).

A.3 Additional Ablation Experiment

A.3.1 Head Type Ablation

We conduct an ablation in which all attention heads are treated as either Dynamic or Rigid. The results are shown in Table 7. Ours denotes the complete EntropyInfer method; Ours - All Dynamic treats every head as Dynamic and performs dynamic budget allocation; and Ours - All Rigid treats every head as Rigid. The results indicate that dynamic allocation is important for model performance, whereas treating every Rigid Head as Dynamic does not provide an additional gain, supporting EntropyInfer's distinction between the two head types.

Table 7: Ablation study on head types using Llama-3.1-8B-Instruct.

Method	NarrativeQA	2WikiMQA
Ours	**29.86**	**45.88**
Ours - All Dynamic	29.56	45.62
Ours - All Rigid	28.19	43.49

A.4 Additional Effectiveness Results

A.4.1 LongBench Results

Table 8 reports the full comparison on Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct. Results on Llama-3.1-70B-Instruct and the Qwen3 model series are shown in Tables 9 and 10.

Table 8: Performance comparison on LongBench datasets. CPrefill denotes CritiPrefill. Bold text indicates the best performance, while underlined text indicates the second-best.

Model	Llama-3.1-8B-Instruct						Qwen2.5-7B-Instruct					
Method	Base	AdaKV	CPrefill	SnapKV	**Ours**	**Ours w/o LD**	Base	AdaKV	CPrefill	SnapKV	**Ours**	**Ours w/o LD**
NrtvQA	29.71	29.53	27.29	29.64	29.86	**30.20**	28.73	28.43	25.44	**28.57**	26.81	26.81
Qsp	45.30	42.91	**45.43**	43.89	45.18	45.38	44.21	42.20	**44.20**	42.61	43.98	44.14
MFQA_en	55.13	53.70	**55.46**	54.12	54.74	55.20	53.05	51.86	51.19	51.34	51.73	**52.13**
HotpotQA	55.50	55.23	**56.54**	54.72	55.28	55.37	57.98	56.95	54.65	57.16	57.20	**57.30**
2WikiMQA	44.30	44.84	43.62	44.81	**45.88**	44.25	46.33	44.54	45.92	44.54	47.09	**47.16**
Musique	31.32	30.62	29.38	29.99	**31.74**	31.68	30.03	29.77	27.81	29.54	30.85	**30.92**
GovReport	35.16	28.36	35.04	28.62	30.02	**35.18**	31.86	26.60	31.45	27.05	28.74	**31.48**
QMSum	25.31	24.11	24.96	24.20	24.97	**25.27**	23.23	23.11	23.19	23.11	23.18	**23.60**
MultiNews	27.30	26.00	27.26	26.05	26.21	**27.45**	23.96	22.39	23.76	22.75	23.38	**23.82**
TREC	72.50	68.50	72.00	67.50	**73.50**	73.00	72.00	68.50	70.00	67.50	**71.00**	**71.00**
TriviaQA	91.64	91.48	90.98	**91.73**	91.18	91.49	88.81	87.52	**89.09**	86.69	**89.09**	**89.09**
SamSUM	43.66	42.40	44.18	42.67	**44.23**	43.94	45.64	44.91	**46.76**	45.03	46.22	46.23
PsgCount	6.96	**7.01**	4.64	6.82	6.49	5.97	8.00	**8.00**	7.00	**8.00**	6.50	6.50
PsgRetrieval_en	99.50	**99.50**	97.50	**99.50**	98.50	98.50	100.00	**100.00**	91.00	**100.00**	91.50	91.50
LCC	63.04	62.33	**63.24**	62.13	62.77	63.23	60.07	59.55	60.27	59.75	60.30	**60.40**
Repobench-P	56.67	56.35	55.55	56.24	56.20	**56.47**	67.26	**65.75**	65.63	65.19	65.26	65.53
Avg	48.94	47.68	48.32	47.66	48.55	**48.91**	48.82	47.51	47.34	47.43	47.68	**47.98**

Table 9: LongBench results on Llama-3.1-70B-Instruct.

Method	Base	CritiPrefill	Ours	Ours w/o LD
NrtvQA	35.17	34.09	34.45	**34.49**
Qasper	49.56	49.03	49.74	**49.88**
MFQA	54.04	53.83	54.27	**54.28**
HotpotQA	65.12	62.33	**64.64**	**64.64**
2WikiMQA	67.18	64.85	**65.38**	**65.38**
Musique	47.78	43.24	**45.48**	45.47
GovReport	34.92	34.85	30.09	**34.95**
QMSum	24.37	23.93	**24.42**	24.16
MultiNews	26.67	26.56	26.04	**26.64**
TREC	76.5	76.5	**77**	**77**
TriviaQA	94.04	93.88	**94.04**	**94.04**
SAMSum	46.92	**47.29**	47.11	46.94
PassageCount	21	**12.5**	11	11
PassageRetrieval	97.5	96.5	**98**	**98**
LCC	35.15	**35.03**	34.59	34.59
RepoBench-P	60.86	57.92	59.5	**59.52**
Avg	52.3	50.77	50.98	**51.31**

Table 10: LongBench results on the Qwen3 model series.

Model	Qwen3-4B-Instruct-2507			Qwen3-8B		
Methods	Base	Ours	Ours-w/o-LD	Base	Ours	Ours-w/o-LD
NrtvQA	27.00	**25.15**	25.11	26.06	24.43	**24.46**
Qasper	45.35	**45.35**	45.30	48.19	48.09	**48.24**
MFQA	50.41	51.14	**51.21**	53.14	53.47	**53.70**
HotpotQA	58.93	58.21	**58.22**	58.98	57.20	**57.22**
2WikiMQA	43.28	42.52	**42.54**	43.37	**39.81**	**39.81**
Musique	24.49	24.31	**24.38**	36.35	**35.52**	35.50
GovReport	30.76	28.43	**30.68**	33.46	30.28	**33.34**
QMSum	22.43	**23.12**	22.26	24.31	24.10	**24.22**
MultiNews	24.13	23.55	**23.89**	24.58	24.03	**24.66**
TREC	73.50	**73.00**	**73.00**	71.50	**72.00**	**72.00**
TriviaQA	87.75	87.82	**87.83**	90.71	**90.86**	**90.86**
SAMSum	45.28	**46.21**	46.20	44.36	44.20	**44.43**
PsgCount	1.00	**0.67**	**0.67**	1.50	**3.50**	**3.50**
PsgRet	100.00	**96.50**	**96.50**	100.00	**98.50**	**98.50**
LCC	62.58	62.94	**62.95**	67.81	67.51	**67.58**
Repobench-P	55.46	**53.96**	53.82	63.40	62.20	**62.21**
Avg	47.02	46.43	**46.54**	49.23	48.48	**48.76**

A.4.2 RULER RESULTS ON QWEN2.5-7B-INSTRUCT AND QWEN3-32B

Tables 11 and 12 show results for Qwen2.5-7B-Instruct and Qwen3-32B, respectively.

Table 11: RULER results on Qwen2.5-7B-Instruct.

Method	4K	8K	16K	32K	64K	128K
Ours w/o LD	**85.04**	**82.74**	**82.65**	**75.25**	**64.25**	**51.7**
CritiPrefill	**85.04**	81.59	79.84	73.88	60.85	48.18
Ours w/o SP	**73.64**	**65.64**	**59.51**	**50.63**	**41.95**	**33.27**
SnapKV	63.02	55.69	49.63	39.32	27.39	17.91
AdaKV	62.76	55.6	49.68	39	27.76	18.18
Ours	**73.58**	**65.26**	**57.9**	**46.41**	**38.04**	**29.82**
CritiPrefill + SnapKV	63.02	54.52	47.23	35.38	22.98	13.73
CritiPrefill + AdaKV	62.76	54.11	46.47	35.38	22.52	13.42

Table 12: RULER results on Qwen3-32B.

Method	RULER 128K
Ours	**28.77**
CritiPrefill + SnapKV	19.75
CritiPrefill + AdaKV	19.44
Ours w/o LD	**58.61**
CritiPrefill	57.46
Ours w/o SP	**39.55**
SnapKV	31.33
AdaKV	31.53

A.5 DYNAMIC BUDGET ANALYSIS

To ensure a fair comparison, we set EntropyInfer's base budget to 2048 (same as CritiPrefill). Each Dynamic Head adjusts its budget per segment based on attention entropy. As shown in Table 13, this dynamic allocation does not significantly exceed the base budget: our default setting (segment size=512, block size=8) averages only 2132.51 tokens per segment.

Table 13: Average budget for different block/segment size settings.

Segment Size v.s. Block Size	8	32
128	2588.40	2763.34
512	2132.51(Experiment Setting)	2303.81

A.6 Additional Efficiency Experiment

A.6.1 Detailed Latency Breakdown.

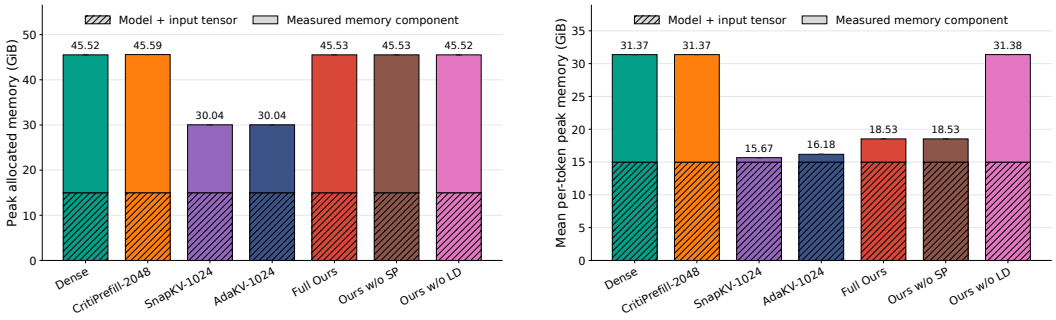
We report the latency breakdown in Table 14

Table 14: Detailed Latency Breakdown. (ms)

Components	16384	32768	65536	98304	131072
obs_query_update	19.22	19.21	20.01	20.25	19.90
layout_cast	15.86	15.80	16.35	16.53	16.27
cache_profiling_total	88.04	157.89	328.29	515.23	703.19
cache_selection_total	72.87	209.88	477.72	728.96	974.36
sparse_attention	**169.69**	**430.56**	**938.64**	**1409.94**	**1863.16**
latent_compress	108.02	196.76	377.16	568.49	750.67

A.6.2 Memory Efficiency Analysis

We study the memory efficiency of different baselines using Llama-3.1-8B-Instruct. To quantify the effect of KV cache compression during inference, we report both peak memory usage and average per-token memory usage when generating 100 tokens from 128K-token inputs, as shown in Figure 8. Results show that EntropyInfer incurs negligible memory overhead (around 0.01 GB) while significantly reducing average memory usage during generation when latent decoding is enabled. Compared with SnapKV and AdaKV, EntropyInfer's average memory usage is around 2 GB higher because it generates N_D tokens before compressing the KV cache to preserve generation quality. However, compared with methods without KV cache compression, EntropyInfer reduces memory usage by approximately 12 GB, accelerating decoding while reducing memory pressure.



(a) Peak memory overhead during generation.

(b) Per-token average overhead.

Figure 8: Memory efficiency analysis on different baselines using Llama-3.1-8B-Instruct.

A.7 Detailed Algorithms

Representation vector construction, observation attention computation and latent decoding algorithms are detailed in Algorithms 2, 3 and 4.

Algorithm 2 Construction of Representation Vector Q_{Rep} (Key Vector K_{Rep} follows symmetrically).

Require: Query States Q, Segment Size L_S
Ensure: Representation Vector Q_{Rep}
$Q_{Max} \leftarrow Q.\,\text{reshape}(N_S, L_S, d).\,\max(\text{dim} = 1)$
$Q_{Min} \leftarrow Q.\,\text{reshape}(N_S, L_S, d).\,\min(\text{dim} = 1)$
$Q_{Rep} \leftarrow \{Q_{Max}, Q_{Min}\}$
return Q_{Rep}

Algorithm 3 Observation Attention Computation

Require: Representation Vectors $Q_{\text{Rep}}, K_{\text{Rep}}$, Causal Mask M
Ensure: Observation Attention Matrix A^{obs}
$Q_{\text{max}}, Q_{\text{min}} \leftarrow Q_{\text{Rep}}$
$K_{\text{max}}, K_{\text{min}} \leftarrow K_{\text{Rep}}$
$Z_1 \leftarrow Q_{\text{max}} K_{\text{max}}^{\top}$, $Z_2 \leftarrow Q_{\text{max}} K_{\text{min}}^{\top}$
$Z_3 \leftarrow Q_{\text{min}} K_{\text{max}}^{\top}$, $Z_4 \leftarrow Q_{\text{min}} K_{\text{min}}^{\top}$
$Z \leftarrow \max\left(\frac{Z_1+Z_3}{2}, \frac{Z_2+Z_4}{2}\right)$
$A^{\text{obs}} \leftarrow \text{Softmax}(Z + M)$ ▷ Row-wise normalization
return A^{obs}

Algorithm 4 Latent Decoding KV Cache Compression via Output-Guided Scoring

Require: Key matrix $K \in \mathbb{R}^{L\times d}$, Value matrix $V \in \mathbb{R}^{L\times d}$, Query matrix $Q \in \mathbb{R}^{L\times d}$, Observation window size N_W, Generated tokens N_D, Cache budget B_d
Ensure: Compressed KV Cache $\text{KV}_{\text{compressed}}$
Step 1: Observation window construction
Identify window index set $\mathcal{I}_W \subset \{1, \ldots, L\}$ combining prompt-tail and newly generated query positions ($|\mathcal{I}_W| = N_W$).
Extract observation queries: $Q_W \leftarrow Q[\mathcal{I}_W, :]$ ▷ $Q_W \in \mathbb{R}^{N_W \times d}$
Step 2: Importance score aggregation
Compute interaction matrix: $S \leftarrow Q_W K^{\top}$ ▷ $S \in \mathbb{R}^{N_W \times L}$
Aggregate token weight vector: $\mathbf{w} \leftarrow \sum_{i=1}^{N_W} S_{i,:}$ ▷ $\mathbf{w} \in \mathbb{R}^{1\times L}$
Step 3: Top-K cache selection
Select top B_d token indices: $\mathcal{I}_{\text{idx}} \leftarrow \text{TopK}(\mathbf{w}, B_d)$
Step 4: Cache compression
$\text{KV}_{\text{compressed}} \leftarrow \{(K_{j,:}, V_{j,:}) \mid j \in \mathcal{I}_{\text{idx}}\}$
return $\text{KV}_{\text{compressed}}$

A.8 Complexity Analysis

Below we provide the detailed complexity derivation.

Let $T(N)$ denote the prefilling compute overhead for an input of length N. The sparse attention workflow consists of three primary components: observation matrix computation $T_{\text{obs}}(N)$, Top-K block selection $T_{\text{topk}}(N)$, and sparse block attention execution $T_{\text{spa}}(N)$:

$$T_{obs}(N) = C_S \times C_B = \frac{N}{L_S} \times \frac{N}{L_B} = \frac{N^2}{L_S L_B} \tag{3}$$

$$\begin{aligned} T_{topk}(N) &= \sum_{i=0}^{C_S} C_{B_i} \log \frac{B_i}{L_B} \\ &= \frac{3}{2} \sum_{i=0}^{C_S} C_B \log \frac{B_h}{L_B} \\ &= \frac{3}{2} C_S C_B \log \frac{B_h}{L_B} = \frac{3N^2}{2L_S L_B} \log \frac{B_h}{L_B} \end{aligned} \tag{4}$$

$$T_{spa}(N) = \sum_{i=0}^{C_S} B_i \times L_S \leq 3 \sum_{i=0}^{C_S} B_h L_S = 3 C_S B_h L_S = 3 B_h N \tag{5}$$

Therefore, the total complexity $T_{total}(N)$ can be expressed as Equation (6).

$$\begin{aligned} T_{total}(N) &= T_{obs}(N) + T_{topk}(N) + T_{spa}(N) \\ &= \mathcal{O}((\frac{1}{L_S L_B} + \frac{3}{2 L_S L_B} \log \frac{B_h}{L_B}) N^2 + 3 B_h N) \end{aligned} \tag{6}$$

A.9 Evaluation on the openPangu Model Family

To evaluate the method's generalizability, we integrate EntropyInfer into openPangu-Embedded-1B-v1.1 and openPangu-Embedded-7B-v1.1 (Chen et al., 2025a). End-to-end latency results (Figure 9) confirm consistent inference speedups, with acceleration scaling more prominently for longer contexts. Results on LongBench (Table 15) and the multi-turn dialogue benchmark LoCoMo (Maharana et al., 2024) (Table 16) demonstrate that EntropyInfer reduces latency while retaining output quality across model families.

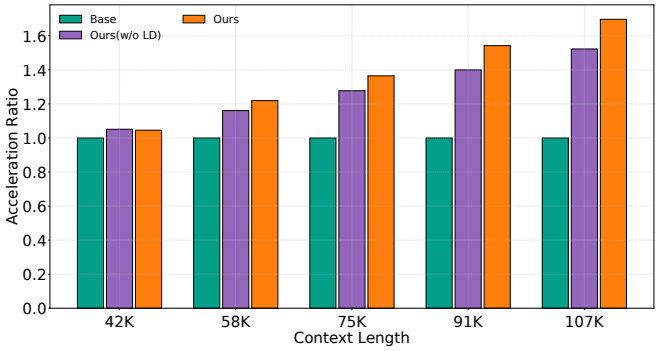


Figure 9: End-to-end acceleration on the openPangu model family across context lengths.

Table 15: LongBench results on openPangu-Embedded-1B and openPangu-Embedded-7B.

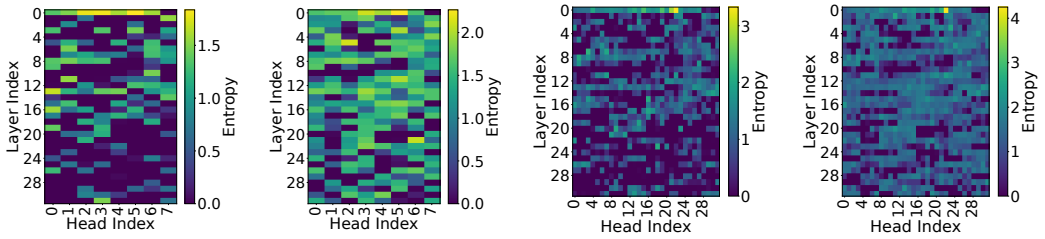
Model	Method	NrtvQA	Qasper	MFQA-en	HotpotQA	2WikiMQA	Musique	Avg
openPangu-7B	**Base**	14.82	37.46	44.37	35.64	29.58	20.08	30.33
	Ours	**15.12**	35.53	43.28	34.91	**31.06**	18.04	29.66
	Ours w/o LD	14.86	**36.94**	**44.44**	**35.23**	31.00	**18.41**	**30.15**
openPangu-7B w/ CoT	**Base**	15.07	37.75	37.21	47.79	61.24	31.31	38.40
	Ours	**16.10**	37.39	**36.84**	50.84	53.91	31.94	37.84
	Ours w/o LD	13.83	**38.01**	36.43	**51.53**	**61.29**	**32.26**	**38.89**
openPangu-1B	**Base**	10.36	28.75	43.66	34.69	36.05	19.63	28.86
	Ours	10.45	28.86	42.75	32.90	**35.17**	18.11	28.04
	Ours w/o LD	**10.46**	**29.38**	**43.95**	**32.92**	35.11	**18.17**	**28.33**

Table 16: LoCoMo results on openPangu-Embedded-1B and openPangu-Embedded-7B.

Model	Method	Multi-Hop	Temporal	Open-domain	Single-Hop	Adversarial	Overall
openPangu-7B	**Base**	27.84	16.04	10.73	31.53	3.14	21.12
	Ours	**27.41**	**14.66**	10.55	**31.63**	2.91	**20.82**
	Ours w/o LD	26.40	14.14	**10.70**	31.51	**3.36**	20.65
openPangu-7B w/ CoT	**Base**	27.16	19.07	15.52	27.50	29.82	26.03
	Ours	**25.61**	**22.58**	11.21	**27.30**	21.08	24.12
	Ours w/o LD	24.14	19.93	**13.64**	25.10	**33.86**	**25.54**
openPangu-1B	**Base**	20.08	14.16	11.69	22.32	9.64	17.32
	Ours	18.88	13.45	10.77	21.99	7.40	16.35
	Ours w/o LD	**19.16**	**13.53**	**10.95**	**22.12**	**7.85**	**16.56**

A.10 Entropy Distribution Heatmap

As shown in Figure 10, the partition of heads into Rigid vs. Dynamic categories fluctuates significantly across different inputs and task domains (e.g., GovReport vs. Musique). A head that exhibits low, rigid entropy on one document may display high, fluctuating entropy on another.



(a) GovReport (Grouped). (b) Musique (Grouped). (c) GovReport (Per-head). (d) Musique (Per-head).

Figure 10: Entropy patterns vary dynamically across attention heads and contextual domains. "Grouped" indicates aggregating attention entropies across GQA key-value head groups.