Flux Attention: Context-Aware Hybrid Attention for Efficient LLMs Inference

Quantong Qiu[1] **Zhiyi Hong**[1] **Yi Yang**[1] **Haitian Wang**[1]
Kebin Liu[2] **Qingqing Dang**[2] **Juntao Li**[1,†] **Min Zhang**[1]

[1]Soochow University, China [2]Baidu Inc., China
[†]Corresponding author: ljt@suda.edu.cn

Code: https://github.com/qqtang-code/FluxAttention

Abstract

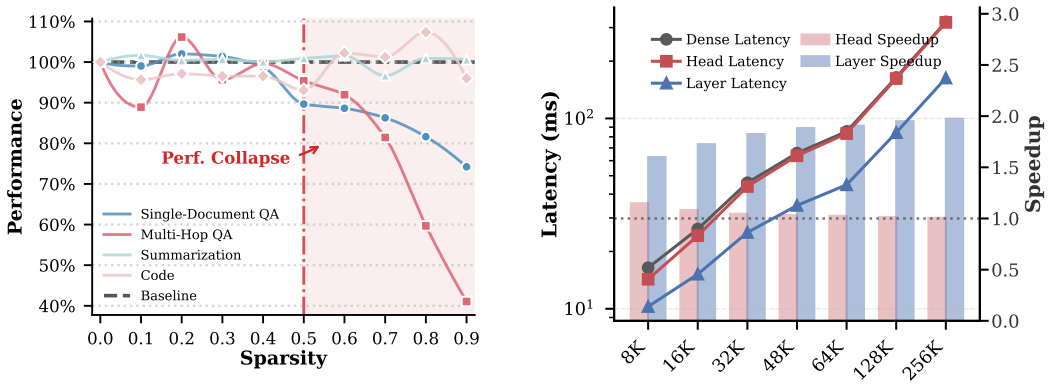
The quadratic computational complexity of standard attention mechanisms presents a severe scalability bottleneck for LLMs in long-context scenarios. While hybrid attention mechanisms combining Full Attention (FA) and Sparse Attention (SA) offer a potential solution, existing methods typically rely on static allocation ratios that fail to accommodate the variable retrieval demands of different tasks. Furthermore, head-level dynamic sparsity often introduces severe computational load imbalance and synchronization long-tails, which hinder hardware acceleration during autoregressive decoding. To bridge this gap, we introduce ***Flux Attention***, a context-aware framework that dynamically optimizes attention computation at the layer level. By integrating a lightweight ***Layer Router*** into frozen pretrained LLMs, the proposed method adaptively routes each layer to FA or SA based on the input context. This layer-wise routing preserves high-fidelity information retrieval while ensuring contiguous memory access, translating theoretical computational reductions into practical wall-clock speedups. As a parameter-efficient approach, our framework requires only 12 hours of training on 8×A800 GPUs. Extensive experiments across multiple long-context and mathematical reasoning benchmarks demonstrate that Flux Attention achieves a superior trade-off between performance and inference speed compared with baseline models, with speed improvements of up to $2.8\times$ and $2.0\times$ in the prefill and decode stages.

1 Introduction

Large Language Models (LLMs) have demonstrated strong capabilities in handling extended context windows for tasks such as document analysis, long-form reasoning, and question answering [29, 34]. However, the standard Full Attention (FA) mechanism [46] scales quadratically with sequence length, creating severe memory and computational bottlenecks during prefilling and autoregressive decoding. Sparse Attention (SA) mechanism addresses this by restricting computations to a subset of tokens to reduce the memory footprint [5, 57].

Modern architectures frequently employ hybrid attention mechanisms that integrate both FA and SA within a single network to balance inference efficiency and generation quality [58]. Conventional hybrid models typically rely on a static allocation of dense and sparse computation. However, downstream applications exhibit highly varied computational demands, as detailed in our preliminary study (Section 2.3). Retrieval-intensive tasks require dense token interactions to locate specific information, whereas context-holistic tasks focus on overarching semantics and remain stable under high sparsity [36]. Consequently, a static configuration risks performance degradation on retrieval tasks and wastes valuable computational resources on holistic tasks.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Performance vs. sparsity.

(b) Decoding latency and speedup.

Figure 1: Impact of sparsity on performance and decoding efficiency. (a) Certain tasks suffer performance collapse beyond a specific threshold; this sweep uses Streaming Sparse Attention (SSA) with the configuration of Appendix F.4. (b) Layer-level sparsity achieves substantial decoding speedup, while head-level sparsity yields marginal speedup.

To achieve dynamic allocation, recent works [43] have explored fine-grained routing at the head level by assigning varying sparsity ratios to individual attention heads based on the input. While algorithmically flexible, this fine-grained routing introduces severe hardware inefficiencies during the memory-bandwidth-bound decode phase. Varying context lengths across heads lead to heterogeneous computational workloads within the same layer. This forces thread blocks executing sparse heads to idle while waiting for retrieval heads, creating a synchronization long-tail that prevents theoretical FLOP reductions from translating into actual wall-clock decoding speedups.

To overcome these challenges, we propose Flux Attention, a context-aware framework that dynamically optimizes attention computation at the layer level. Instead of managing individual heads, we introduce a lightweight Layer Router. By evaluating the semantic context of the input prompt, the router infers the underlying task demands and adaptively assigns each layer to FA or SA mode. This coarse granularity inherently preserves contiguous memory access, enabling the GPU to completely bypass the memory-intensive loading of historical KV tensors when SA is selected.

During training, we freeze all backbone LLM parameters and update only the lightweight Layer Router module. We employ a Gumbel-Softmax [18] relaxation for differentiable soft routing, allowing the model to smoothly learn the correlation between context complexity and computational budget. During inference, this soft formulation is discretized into deterministic hard routing, successfully translating theoretical computational savings into substantial wall-clock speedups.

Extensive evaluations on models such as Qwen-3 [55] and Llama-3.1 [13] demonstrate that Flux Attention successfully adapts sparsity levels across diverse tasks. Our parameter-efficient training converges in just 12 hours on an 8-GPU A800 node. Flux Attention achieves a superior performance-efficiency trade-off compared to existing baselines, delivering up to a 2.7x speedup during the prefill phase and a 2.0x acceleration during autoregressive decoding.

2 Preliminary

2.1 Functional Heterogeneity in Attention Mechanisms

During long-context inference, attention mechanisms in Large Language Models (LLMs) specialize functionally based on their sensitivity to historical context and computational demands. Specialized retrieval heads are essential for high-fidelity information recovery, as they precisely locate relevant tokens across extensive sequences [48]. UnComp [52] observe that heads with abnormally high entropy tend to aggregate at specific model depths to capture long-range dependencies. Layers dominated by these heads function as retrieval layers. To ensure precise retrieval, they require a Full Attention (FA) mode, where the Query (Q) interacts with all historical states Key (K) and Value (V):

$$\mathcal{O}_r = \text{Softmax}\left(QK^\top\right)V, \tag{1}$$

where the scaling factor is omitted for clarity. While FA preserves the complete context, its computational complexity is quadratic with sequence length N, posing challenges for efficient inference.

A substantial portion of heads instead focus on local semantic structures and are robust to context truncation. Layers predominantly composed of these sparse heads operate as sparse layers. Sparse layers employ a Sparse Attention (SA) mechanism to reduce computational overhead in long-sequence processing. SA optimizes efficiency by performing attention operations on a condensed subset of the most critical historical elements ($\tilde{K}$ and $\tilde{V}$):

$$\mathcal{O}_s = \text{Softmax}\left(Q\tilde{K}^\top\right)\tilde{V}. \tag{2}$$

2.2 Rethinking Hybrid Attention Mechanisms

To balance generation quality and inference efficiency, various hybrid attention mechanisms have been proposed. Existing methods, such as PruLong [4], DuoAttention [49], and LycheeDecode [28], adopt a static allocation strategy. They identify retrieval heads offline and permanently assign them full historical states, while uniformly sparsifying the context for the remaining heads across all tasks.

However, the demand for precise information retrieval varies depending on the specific task and input prompt. Elastic Attention [43] suggests dynamic, context-aware sparsity at the head level, which adjusts the retention of historical states dynamically. Although this fine-grained allocation optimizes the theoretical efficiency-performance trade-off, it yields limited actual decoding acceleration. The dynamic adjustment at the head level introduces significant system-level overhead and irregular memory access patterns during deployment, limiting the achievable speedup during the decode phase.

2.3 Motivational Observations

To investigate the limitations of existing sparsity mechanisms, we formalize the quantification of model-level sparsity. The Model Sparsity Ratio (Ω_{MSR}) quantifies the overall proportion of sparse attention mechanisms applied across the model:

$$\Omega_{\text{MSR}} = \frac{1}{H \times L}\sum_{\ell=1}^{L}\sum_{h=1}^{H}\mathbb{I}\left[\pi^{(\ell,h)} = \text{SA}\right], \tag{3}$$

where $\pi^{(\ell,h)}$ denotes the assigned attention mode (FA or SA) for head h in layer ℓ, and $\mathbb{I}[\cdot]$ is the indicator function.

Settings To investigate the impact of varying sparsity ratios (Ω_{MSR}) on long-context LLMs, we profile task accuracy and decode latency. For the accuracy evaluation in Figure 1(a), we use a matrix entropy metric based on UnComp [52] to quantify the information density of individual layers. We rank the layers using these calculated entropy scores and progressively replace the lowest-scoring ones with SA. Model performance is then evaluated across real-world tasks from LongBench [1]. For hardware efficiency (Figure 1(b)), we compare the decode latency and achievable speedup of our layer-level sparsity against a static head-level sparsity baseline. Appendix D provides details on the entropy scoring formulation and latency measurement implementation.

Results Our analysis reveals two bottlenecks in current hybrid attention mechanisms. First, as shown in Figure 1(a), model performance does not degrade linearly with increasing Ω_{MSR}. Instead, accuracy drops sharply for retrieval-intensive tasks once a specific sparsity threshold is exceeded. This indicates that static sparsity assignments do not adapt to varying contextual demands, necessitating a context-aware dynamic retention strategy for historical states. Second, Figure 1(b) demonstrates a distinct discrepancy in hardware efficiency. While head-level sparsity provides algorithmic flexibility, it introduces severe hardware bottlenecks during the memory-bandwidth-bound decode phase. It creates a severe synchronization long-tail effect. Thread blocks executing sparse heads finish quickly but must idle while waiting for memory-intensive retrieval heads within the same layer. This intra-layer load imbalance yields only marginal wall-clock speedups. In contrast, layer-level sparsity ensures uniform computational workloads across all thread blocks. By completely bypassing historical KV loading for designated layers, it eliminates synchronization stalls, effectively translating theoretical FLOP reductions into substantial decode acceleration.

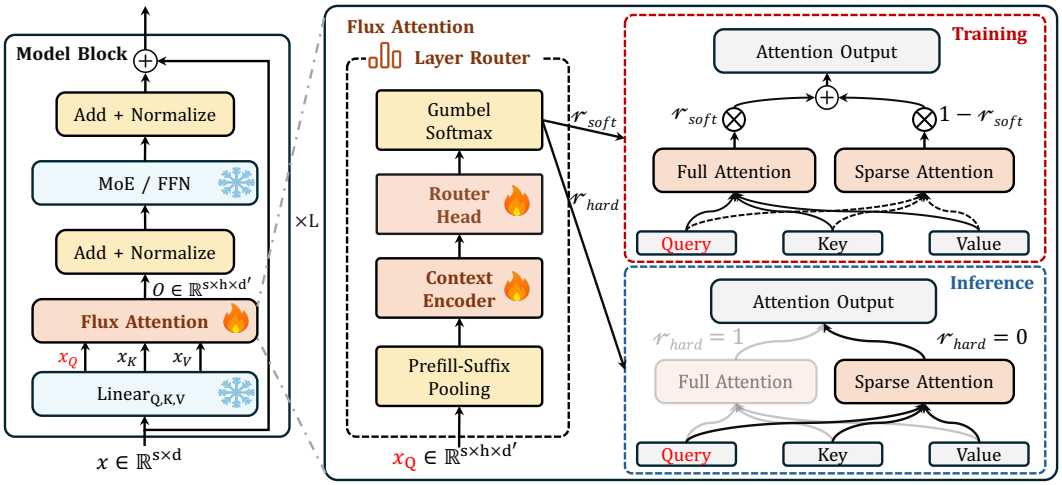


Figure 2: Overview of our dynamic layer-level routing architecture. The model incorporates a Layer Router that assigns each layer to either FA or SA based on the input query x_Q.

These observations present a fundamental dilemma. *Fine-grained head-level sparsity is hardware-unfriendly during decode, whereas static sparsity risks performance collapse*. To address this, we propose a dynamic, context-aware hybrid attention mechanism operating at the layer level to balance model performance with inference efficiency.

3 Methodology

We introduce a Flux Attention mechanism to address the hardware inefficiencies of fine-grained sparsity and the rigidity of static allocations. As illustrated in Figure 2, our architecture relies on a dynamic Layer Router that adaptively assigns each layer to either FA or SA based on the input query. This approach is parameter-efficient: the original LLM backbone parameters remain strictly frozen during training. Optimization only updates the lightweight components of the Layer Router, ensuring rapid convergence while preserving pre-trained weights.

3.1 Context-Aware Layer Router Design

Within the Flux Attention module, a lightweight Layer Router determines the optimal attention mechanism for a given context.

Architecture and Feature Extraction As shown in Figure 2, the router receives the incoming query tensor $x_Q \in \mathbb{R}^{s\times h\times d'}$ as input, where s represents the sequence length, h denotes the number of heads, and d' indicates the head dimension. To efficiently extract semantic context, we apply a Prefill-Suffix Pooling operation to x_Q to extract representations of the initial and final prompt tokens. This operation efficiently aggregates the token-level features into a single sequence-level descriptor. Subsequently, a Context Encoder (MLP) processes this pooled representation to capture contextual dependencies, after which a Router Head (MLP) projects these features into unnormalized routing logits, denoted as π_{FA} and π_{SA}.

Differentiable Training via Soft Routing Optimizing the Layer Router is challenging because the binary routing decisions are discrete and non-differentiable.To address this, we apply the Gumbel-Softmax relaxation [18] to enable end-to-end backpropagation. During training, we sample continuous routing weights $r_{\text{soft}} \in (0, 1)$, which represent the probability of selecting the FA mechanism. This computation is defined as follows:

$$r_{\text{soft}} = \frac{\exp((\pi_{\text{FA}} + g_{\text{FA}})/\tau)}{\exp((\pi_{\text{FA}} + g_{\text{FA}})/\tau) + \exp((\pi_{\text{SA}} + g_{\text{SA}})/\tau)} \tag{4}$$

where $g_{\text{FA}}, g_{\text{SA}} \sim \text{Gumbel}(0,1)$ are independent and identically distributed samples drawn from the Gumbel distribution, and $\tau > 0$ denotes the temperature parameter. The output of the Flux Attention layer is then computed as a convex combination:

$$\mathcal{O}_{\text{train}} = r_{\text{soft}} \cdot \text{FA}(Q, K, V) + (1 - r_{\text{soft}}) \cdot \text{SA}(Q, \tilde{K}, \tilde{V}). \tag{5}$$

The temperature τ controls the smoothness of the routing distribution. We employ a temperature annealing schedule to minimize the train-test discrepancy. Initially, τ is set to a high value to encourage exploration and ensure smooth gradient flow. As training progresses, τ linearly decays towards a small value.

Deterministic Inference via Hard Routing During the inference phase, the router outputs a binary decision $r_{\text{hard}} \in \{0, 1\}$ using an $\arg\max$ operation over the generated logits. When $r_{\text{hard}} = 0$, the layer executes the SA mechanism.

3.2 Training Objective and Sparsity Constraint

We formulate the training objective as a constrained optimization problem to balance generation quality and computational efficiency. Without intervention, the router tends to degenerate by sending all queries to the FA mode, which trivially minimizes the language modeling loss.

A dynamic penalty mechanism controls the inference budget. Let $\boldsymbol{t} \in (0, 1)$ denote the target computational budget for sparse computation (i.e., the permissible fraction of SA layers, corresponding to Ω_{MSR}). Notably, instead of enforcing a rigidly fixed $\boldsymbol{t}$ for each task, we impose *task-dependent non-tight constraints* with predefined lower and upper bounds, since the optimal sparsity for a given task is inherently unknown. We therefore define the sparsity deviation as $L_{\text{diff}}(\mathcal{X}) = \mathbb{E}_{\mathcal{X}}[1 - r_{\text{soft}}] - \boldsymbol{t}$, which represents the gap between the expected sparse routing probability across all layers and the allocated budget. We solve the overall optimization objective via Lagrangian relaxation:

$$\max_{\lambda_1, \lambda_2 \geq 0} \min_{\theta} \underbrace{L_{\text{language}}(\mathcal{X})}_{\text{language modeling}} + \underbrace{\lambda_1 L_{\text{diff}}(\mathcal{X}) + \lambda_2 L_{\text{diff}}^2(\mathcal{X})}_{\text{sparsity regularization}}, \tag{6}$$

where θ represents the trainable parameters of the Layer Router, and $L_{\text{language}}(\mathcal{X})$ is the standard cross-entropy loss. The Lagrangian multipliers λ_1 and λ_2 are task-specific trainable Lagrange multipliers optimized via gradient ascent [4], which decouple the sparsity–performance trade-offs across tasks and mitigate optimization conflicts.

3.3 Efficient Deployment

To translate theoretical sparsity gains into real-world inference acceleration and memory savings, Flux Attention decouples routing computation between the prefill and decode phases, with a sparse-decode implementation aligned with our experimental settings. The Layer Router infers only once during the prefill phase, generating a deterministic hard routing decision ($r_{\text{hard}} \in \{0, 1\}$) per layer based on the input context. This decision is cached and reused across all decoding steps, eliminating per-token routing overhead. The realized model-level sparsity of a request is therefore measured directly from the cached decisions as $\Omega_{\text{MSR}} = \frac{1}{L} \sum_{\ell=1}^{L} \left(1 - r_{\text{hard}}^{(\ell)}\right)$; we report the realized value throughout rather than the target $\boldsymbol{t}$. Routing is thus context-aware rather than token-wise: recomputing it at every generated token would add control overhead and break the homogeneous per-layer execution that produces the systems benefit, while the underlying task intent usually remains stable within a single response. In multi-turn or agentic use the route is refreshed at the natural turn boundary, where the extended dialogue is re-prefilled and the router runs again on the enlarged prompt (Appendix H.4); the cached assignment is fixed only within one decode trajectory, not globally. Our sparse-decode configuration further optimizes efficiency: for sparse layers, we only maintain the minimal KV cache required by the sparse kernel, fully bypassing full historical KV access and storage; for retrieval layers, complete KV cache is retained to preserve retrieval performance. This design delivers significant decoding speedups and KV cache reduction in long-context scenarios.The detailed execution flow of the inference is summarized in Algorithm 1.

Table 1: Performance on LongBench-E [1]. We report average performance (Perf.) and Ω_{MSR} per task category. The 1st and the 2nd performance in each comparison group are highlighted with **bold font** and underlined, respectively. Gray-shaded rows denote the sparse-decode configuration.

Method	S-Doc QA		M-Doc QA		Summ		In-Context			Synthetic		Code		Avg.	
	Qasper	MF-en	HotQA	2Wiki	Gov.	M.News	TREC	TQA	SAMS	PCount	PRe	RB-P	Lcc	**Perf.**	Ω_{MSR}
Qwen3-4B backbone model															
Qwen3-4B	35.21	52.16	44.81	32.15	33.47	23.45	70.67	88.22	39.74	2.33	96.84	50.84	57.93	48.45	-
+ DuoAttention	**35.83**	49.84	47.09	32.24	33.32	**23.70**	**69.33**	85.87	39.75	**4.50**	94.57	50.56	57.43	48.22	0.50
+ PruLong	34.15	50.78	44.48	32.89	32.96	23.53	67.67	**88.69**	39.55	3.17	90.17	49.00	54.07	47.16	0.50
+ TriangleMix	35.55	**52.02**	45.37	31.76	33.32	**23.70**	69.00	88.20	39.74	3.83	91.51	48.58	56.38	47.72	0.50
+ FluxAttn (FA-SSA)	35.02	49.44	49.64	32.27	33.26	23.48	**69.33**	88.29	39.78	1.50	94.56	**53.44**	59.69	48.72	0.44
+ FluxAttn (FA-XA)	35.74	51.70	45.83	32.34	**33.57**	23.66	69.00	87.23	39.81	3.50	93.74	50.81	59.28	48.32	0.53
+ FluxAttn (FA-TA)	35.02	50.89	45.17	**34.24**	33.02	23.53	69.00	88.08	**40.38**	3.94	**96.06**	51.68	**60.00**	**48.76**	0.47
+ FluxAttn (FA-SSA)	35.10	51.68	**49.65**	32.86	33.04	23.42	**69.33**	88.00	40.00	1.67	94.47	51.40	58.68	48.59	0.44
Qwen3-8B backbone model															
Qwen3-8B	41.22	49.92	58.98	44.21	33.27	23.42	71.33	86.77	41.83	2.00	98.33	56.08	66.31	52.16	-
+ DuoAttention	**41.78**	**51.55**	55.96	41.70	33.24	23.34	69.33	**89.35**	41.62	0.50	98.93	57.54	69.39	52.13	0.50
+ PruLong	37.95	51.20	51.94	36.48	33.11	23.36	69.00	87.90	42.11	1.00	98.00	57.05	67.66	50.80	0.50
+ TriangleMix	40.82	51.31	57.57	**44.51**	**33.32**	23.35	71.33	86.73	41.79	**2.00**	94.33	55.04	65.89	51.65	0.50
+ FluxAttn (FA-SSA)	40.30	50.49	56.02	40.90	33.01	**23.55**	71.67	88.31	41.61	0.33	**100.00**	**59.46**	68.27	52.18	0.46
+ FluxAttn (FA-XA)	40.41	50.26	57.78	40.57	33.27	23.51	69.67	87.19	**42.12**	1.33	99.33	55.41	65.51	51.57	0.51
+ FluxAttn (FA-TA)	41.00	49.76	**58.19**	44.36	**33.32**	23.35	70.00	88.77	41.70	1.33	99.67	55.60	67.22	**52.22**	0.47
+ FluxAttn (FA-SSA)	39.92	50.04	55.72	40.81	33.03	23.50	**72.00**	88.48	40.96	0.33	99.22	58.57	**69.46**	52.05	0.46
Llama-3.1-8B-Instruct backbone model															
Llama-3.1-8B-Instruct	44.06	53.44	59.62	44.08	34.50	26.02	71.00	90.54	42.94	12.67	99.33	47.78	63.85	53.28	-
+ DuoAttention	34.63	50.74	49.70	36.41	34.25	25.78	70.00	91.45	42.13	9.80	97.33	53.59	**68.55**	52.11	0.50
+ PruLong	41.51	52.36	50.46	37.57	34.25	25.86	66.33	89.93	41.72	9.07	97.00	**56.84**	66.23	51.68	0.50
+ TriangleMix	45.10	**54.60**	56.67	41.88	34.09	25.51	71.33	90.93	42.63	10.62	94.67	43.64	59.35	51.67	0.50
+ FluxAttn (FA-SSA)	**45.25**	54.42	54.54	41.34	34.54	**26.16**	68.33	**91.91**	42.17	9.00	97.67	47.74	65.35	52.28	0.51
+ FluxAttn (FA-XA)	42.14	53.13	**58.53**	**43.50**	**34.66**	26.06	70.67	91.46	**43.13**	8.00	**99.67**	50.91	64.78	**53.07**	0.72
+ FluxAttn (FA-TA)	44.77	54.12	57.35	43.43	34.31	25.80	**72.33**	91.32	42.62	9.33	98.33	45.48	60.70	52.42	0.62
+ FluxAttn (FA-SSA)	43.76	53.41	57.36	39.43	32.96	25.63	70.33	91.27	42.20	**11.00**	98.67	45.60	66.17	52.30	0.51

4 Experiments

4.1 Settings

Training and Data We select Qwen3 (4B and 8B) [55] and Llama-3.1-8B-Instruct [13] as the backbone LLMs. We construct the training dataset by combining five sources: ChatQA2-Long-SFT-data [53], MuSiQue [45], CoLT-132K [25], GovReport [17], and XSum [35]. This dataset covers both retrieval-intensive tasks (Single-Doc QA and Multihop QA) and context-holistic tasks (code completion, summarization, and in-context learning). The resulting dataset spans sequence lengths ranging from 1K to 64K tokens, and contains approximately 0.74B tokens in total. For the context-holistic and retrieval-intensive task categories, we empirically set $\boldsymbol{t} = 1.0$ and $\boldsymbol{t} = 0.45$, respectively, as motivated by Section 2.3. We conduct the training process using eight A800 GPUs, and each run completes within 12 hours. We provide additional training details in the Appendix F and list the hyperparameters in the table 7.

Evaluation We compare our method with representative sparsity approaches: DuoAttention [50], PruLong [4], and TriangleMix [15]. The computation modes for sparse layer attention include Streaming Sparse Attention (SSA) [51], XAttention (XA) [54], and Triangle Attention (TA). The configurations for layer computation follow the format of "{Retrieval Layer mode}-{Sparse Layer mode}" (e.g., FA-SSA denotes the use of FA for retrieval layers and SSA for sparse layers). All evaluations are conducted using the `LOOM-Eval` framework [44]. Due to space limitations, additional results are provided in the Appendix G.

4.2 Evaluation Results

Real-world Long-context Tasks Table 1 presents the evaluation results on LongBench-E [1], a real-world long-context benchmark that comprises 13 tasks across 6 categories with varying context lengths. FluxAttn maintains the performance of the model on long-context tasks while achieving substantial context compression. Across the Qwen3 series, variants of FluxAttn frequently match or slightly exceed the average performance of the full attention baselines. We further evaluate the effect of applying sparse attention during the decode phase, as indicated in the shaded rows. The method remains competitive under sparse decode. On Qwen3-4B, the sparse-decode configuration achieves an average score of 48.59, which remains above the full attention baseline. For Qwen3-8B

Table 2: Model performance on RULER [16], LongBench-v2 [2] and some Math tasks [6, 33]

Models	RULER							LongBench-v2			Math		
	8K	16K	32K	64K	128K	256K	Perf.	Easy	Hard	Perf.	GSM8K	AIME24	Perf.
Qwen3-4B backbone model													
Qwen3-4B	87.49	86.82	60.05	70.98	53.19	43.27	66.00	32.67	22.18	25.96	39.70	30.35	35.03
+ DuoAttention	79.38	76.08	52.91	69.02	43.28	44.96	60.67	**31.33**	24.06	26.68	39.70	37.05	38.38
+ PruLong	74.21	75.72	47.88	59.27	47.10	45.69	60.25	28.00	25.56	26.44	39.70	30.35	35.03
+ TriangleMix	**87.42**	**85.10**	58.73	67.94	50.97	44.47	63.74	**31.33**	22.18	25.48	40.30	37.25	38.78
+ FluxAttn (FA-SSA)	81.58	82.11	58.73	**72.89**	52.81	**56.91**	66.95	29.33	**28.57**	**28.85**	40.30	37.05	38.68
+ FluxAttn (FA-XA)	86.79	84.94	59.52	68.82	51.77	43.43	63.67	30.00	24.06	26.20	42.20	**40.35**	41.28
+ FluxAttn (FA-TA)	84.28	84.53	**60.58**	68.60	51.91	51.64	65.55	**31.33**	26.32	28.12	**45.00**	**40.35**	**42.68**
+ FluxAttn (FA-SSA)	80.36	80.75	56.08	71.49	**59.17**	56.00	**67.19**	28.00	28.20	28.12	39.90	37.25	38.58
Qwen3-8B backbone model													
Qwen3-8B	89.69	85.62	63.23	82.39	65.84	66.71	75.74	39.33	27.82	31.97	40.60	32.35	36.48
+ DuoAttention	86.68	86.01	63.23	77.52	61.50	61.95	72.41	**40.67**	25.56	31.01	41.20	35.65	38.43
+ PruLong	83.85	80.86	60.05	77.25	62.54	61.49	70.97	36.00	28.20	31.01	40.40	32.35	36.38
+ TriangleMix	81.01	75.67	63.49	73.76	61.54	**66.84**	70.47	36.00	27.44	30.53	41.20	**44.15**	42.68
+ FluxAttn (FA-SSA)	84.09	81.90	60.58	79.30	64.74	65.27	73.03	36.67	29.32	31.97	**46.90**	42.35	**44.63**
+ FluxAttn (FA-XA)	85.88	85.54	**65.08**	**81.95**	**65.09**	65.38	**74.65**	32.67	**32.71**	**32.69**	43.20	35.65	39.43
+ FluxAttn (FA-TA)	**87.49**	**86.17**	60.85	78.72	60.75	63.03	73.51	37.33	27.44	31.01	43.00	39.05	41.03
+ FluxAttn (FA-SSA)	83.54	81.00	59.79	77.93	64.83	65.12	72.51	39.33	28.20	32.21	45.30	43.20	44.25
Llama-3.1-8B-Instruct backbone model													
Llama-3.1-8B-Instruct	92.88	92.83	89.46	70.79	80.12	72.34	83.47	32.00	33.08	32.69	42.30	30.35	36.33
+ DuoAttention	91.71	86.35	85.65	62.65	62.30	38.69	70.33	26.67	28.57	27.88	44.40	33.65	39.03
+ PruLong	86.96	76.55	70.65	54.52	48.18	30.00	59.87	30.00	24.44	26.44	41.30	29.85	35.58
+ TriangleMix	92.44	90.76	86.75	68.00	78.25	64.39	80.46	29.33	25.56	26.92	46.30	37.05	41.68
+ FluxAttn (FA-SSA)	82.88	78.09	70.39	52.29	62.20	50.73	76.75	34.00	28.95	30.77	45.30	37.05	41.18
+ FluxAttn (FA-XA)	92.43	**90.85**	**88.23**	**68.56**	75.86	60.80	79.51	**36.00**	**31.95**	**33.41**	44.40	33.65	39.03
+ FluxAttn (FA-TA)	**92.72**	90.53	86.45	67.78	**80.63**	**67.09**	**81.50**	34.67	28.95	31.01	**46.90**	**38.30**	**42.60**
+ FluxAttn (FA-SSA)	90.11	79.39	79.22	56.08	62.94	59.39	73.67	34.67	30.08	31.73	45.90	37.35	41.63

and Llama-3.1-8B-Instruct, the average scores (52.05 and 52.30, respectively) demonstrate only a slight degradation compared to the standard dense decoding approach.

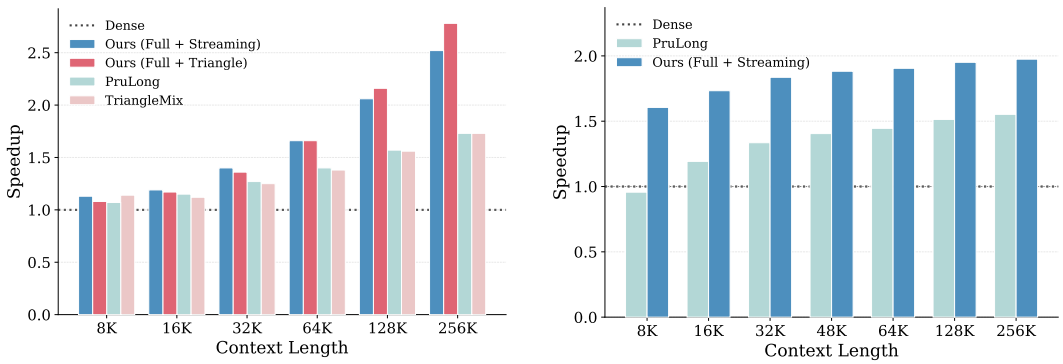
Length Extrapolation Capability Testing To further assess the ability of the models to handle extreme context lengths, we evaluated our method on the RULER benchmark [16], which tests length extrapolation capabilities from 8K to 256K tokens. The results are summarized in Table 2. Overall, FluxAttn demonstrates robust length extrapolation, maintaining information retrieval and reasoning capabilities even at the 256K context boundary, where many existing sparse attention baselines experience severe performance degradation. Consistent with our findings in real-world tasks, we also observe that extending sparsity to the decode phase (shaded rows) preserves the extrapolation capabilities. The sparse-decode configuration of FluxAttn on Qwen3-4B achieves an average score of 67.19 (the highest among all methods in the comparison group) and a score of 56.00 at 256K. This result further validates that our method can achieve comprehensive efficiency gains without compromising ultra-long context understanding.

Long-form Reasoning and Math Tasks We further evaluate our models on the long-context reasoning benchmark LongBench-V2 [2], as well as the mathematical reasoning tasks GSM8K [6] and AIME24 [33]. Table 2 demonstrates that FluxAttn exhibits strong performance across both domains. On LongBench-V2, the proposed method attains the highest scores on both the easy and hard subsets among all baselines. Furthermore, our approach improves the performance on the mathematical benchmarks, yielding the best results on GSM8K and AIME24. This proves that FluxAttn robustly preserves complex logical reasoning capabilities.

4.3 Overall Inference Efficiency

To evaluate the hardware acceleration of our method, we benchmark the inference speedup of FluxAttn against the standard dense baseline and existing sparse methods across varying context lengths. Figure 3 presents the speedup metrics for both the prefill and decode phases.

End-to-End Prefill Acceleration Figure 3(a) shows the end-to-end latency reduction during the compute-bound prefill phase, i.e., time-to-first-token. As the context window expands, the quadratic complexity of standard attention becomes a bottleneck, allowing our dynamic routing mechanism



(a) End-to-end speedup in the prefill phase.

(b) Kernel speedup in the decode phase.

Figure 3: Speedup comparison across different context lengths. The dotted line represents the dense baseline performance (1.0x).

to demonstrate substantial gains. At a 256K context length, our method (configured with FA-TA, i.e., full attention in retrieval layers and Triangle Attention in sparse layers) achieves up to a 2.8× end-to-end speedup, outperforming static baselines such as PruLong and TriangleMix.

Kernel-Level Decode Acceleration The advantage of layer-level routing is evident during the memory-bandwidth-bound decode phase, as shown in Figure 3(b). While prior approaches like PruLong struggle to translate theoretical sparsity into proportional wall-clock speedups due to fragmented memory access, FluxAttn solves this bottleneck by operating at the layer level. Our method achieves a scalable kernel speedup, approaching 2.0× at a 256K context length. This result empirically shows that our context-aware, layer-wise routing aligns with modern GPU execution patterns to deliver improved inference efficiency. Note that the two panels report different phases and units—prefill latency and per-token decode latency—and must not be compared against each other; in particular, head-level kernels remain competitive on prefill, and our claim concerns per-token decode.

Quality–Latency–Memory Trade-off at Matched Sparsity The comparisons above vary the sparsity each method selects, which confounds routing granularity with the amount of computation retained. We therefore compare dense FA2, PruLong, and Flux Attention on Llama-3.1-8B-Instruct at approximately matched model sparsity ($\Omega_{\text{MSR}} \approx 0.50$), jointly reporting accuracy, prefill and decode speedups, and persistent KV memory in a single table (Tables 15 and 16, Appendix G.6). At 128K and 256K, where Ω_{MSR} is exactly matched, Flux improves accuracy over PruLong by 14.76 and 29.39 points while reducing decode latency by 19.2% and 32.0%; persistent KV memory falls by 16.1–39.3%, because a routed sparse layer need not retain the complete historical KV state whereas mixed-head execution does. Since a "sparse unit" is a head for PruLong and a layer for Flux, we report realized Ω_{MSR} together with measured latency and memory rather than treating the ratio as a proxy for systems cost, and we note that this reflects the optimized head-wise implementation we evaluate rather than every possible head-wise kernel.

Batched Serving Single-request profiling can overstate the benefit of a sparsity scheme, because heterogeneous per-head workloads may be co-scheduled once the server batches requests. At a 32K context on 8×NVIDIA H20 GPUs, Flux stays faster at every feasible batch size (1.32× at batch 8, 2.07× at batch 16) and reaches batch 24 while FA2 exhausts memory (Appendix G.5).

Router Overhead Analysis A critical requirement for dynamic routing is minimizing its own computational cost. As illustrated in Figure 9, our router incurs a negligible overhead, averaging only 0.20 ms per layer. Notably, the design exhibits length-invariant stability, maintaining a constant execution speed across sequence lengths ranging from 512 to 1M tokens. This ensures that the routing mechanism itself does not become a bottleneck at extreme context lengths, thereby preserving the substantial speedups achieved in the prefill phase.

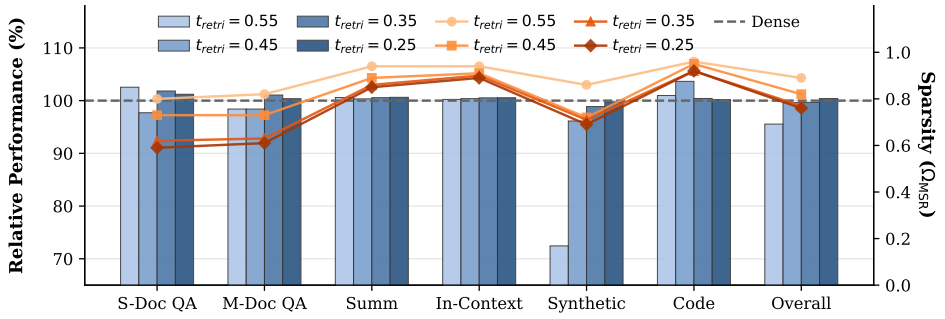


Figure 5: Comparison of performance and test-time Ω_{MSR} among different training sparsity target $\boldsymbol{t}$ settings. The bar chart denotes the performance and the line chart denotes Ω_{MSR} in each task.

5 Analysis

5.1 Dynamic Allocation Strategy of the Layer Router

Task-Level Dynamic Sparsity Different downstream tasks impose inherently distinct requirements on attention sparsity. As shown in the upper region of Figure 4, retrieval-intensive tasks frequently activate FA (dark blue) to support the dense token interactions required for fact-finding. Conversely, context-holistic tasks predominantly route the mid-to-high layers to SA, which validates that high-level holistic semantic understanding is highly robust to attention sparsification. This demonstrates that Flux Attention replaces static allocations with task-aware dynamic sparsity.

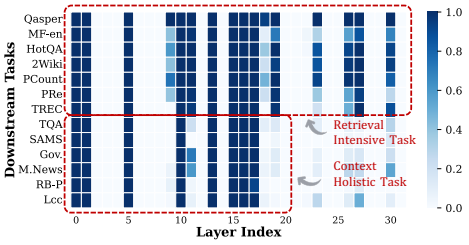


Figure 4: Overview of the layer-wise routing activation frequencies in Llama-3.1-8B-Instruct. Dark blue indicates layers consistently routed to FA across all six tasks in LongBench-E, whereas light blue denotes layers consistently routed to SA.

Context-Aware Intra-Task Sparsity Beyond cross-task adaptation, the router further captures the intrinsic sparsity requirements of individual input contexts, rather than merely memorizing coarse-grained task-level patterns. This instance-level variance is evident where intermediate activation frequencies ($\sim 0.4 - 0.6$, light blue) within a single task show the router adjusting to the complexity of different inputs. We also find that specific layers (e.g., layers 0, 1, 5, 13, and 15–17) are consistently routed to FA across all tasks. This indicates the router preserves the universal architectural properties of the backbone while allocating the remaining computational budget based on specific task and context demands.

Notably, the emergence of this fine-grained, task-aware routing relies on a well-balanced training curriculum. An unbalanced data distribution can cause the router to collapse into a homogenized routing strategy, as extensively analyzed in Appendix H.1. Furthermore, we find that a prefill-suffix pooling operation on the boundary 100 tokens is highly effective in driving this context-aware routing, as it isolates essential instruction signals from sequence noise (detailed in Appendix H.2).

5.2 Impact of Target Sparsity Allocation

We study the impact of target sparsity $\boldsymbol{t}$ on model performance. Specifically, we fix the target sparsity of context-holistic tasks to 1, while progressively decreasing target sparsity for retrieval-intensive tasks ($\boldsymbol{t}_{\text{retri}}$) from 0.55 to 0.25. Since $\boldsymbol{t}$ is a target for Ω_{MSR} (Section 3), a lower $\boldsymbol{t}_{\text{retri}}$ means *less* Sparse Attention and *more* Full Attention on retrieval-intensive tasks, and vice versa. As shown in Figure 5, decreasing $\boldsymbol{t}_{\text{retri}}$ causes the resulting (Ω_{MSR}) allocated by the model to exhibit slightly greater task-level differentiation across different tasks. However, Ω_{MSR} does not strictly match the target $\boldsymbol{t}$. This discrepancy arises because we use *task-dependent and non-tight constraints*, which do not force the model to exactly satisfy the prescribed sparsity. We provide full training curves and further explanations in Appendix H.6. Additionally, when $\boldsymbol{t}_{\text{retri}}$ is set too low (e.g., 0.25) to allocate a higher proportion of FA computation, the overall performance can even surpass that of the backbone

model. Conversely, setting $\boldsymbol{t}_{\text{retri}}$ too high causes the performance on retrieval-intensive tasks to drop sharply, consistent with the observations in Section 2.3. To optimize inference efficiency, we adopt $\boldsymbol{t}_{\text{retri}} = 0.45$ in our main experiments, which achieves a favorable balance between strong overall performance and computational cost.

5.3 Scalability via Backbone Adaptation

To evaluate the flexibility of Flux Attention, we investigate how well the method supports continued training. A critical question for dynamic sparsity methods is whether the routing mechanism can be decoupled from the backbone for subsequent model adaptation. To test this, we freeze the weights of the trained Layer Router, which fixes its learned dynamic allocation strategy, and continue training the model backbone using the data mixture from Section 4.1.

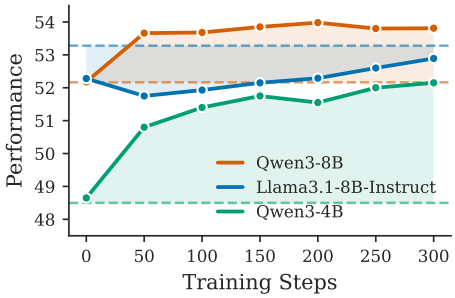


Figure 6: Performance trajectories during continued training with a frozen Layer Router. The backbone effectively adapts its representations to the established sparse pathways, demonstrating steady improvement over time.

As Figure 6 illustrates, continued training yields steady performance improvements across different models. Notably, both Qwen3-8B and Qwen3-4B rapidly surpass their original backbone performance (dashed lines) within just 50 steps and maintain a significant gain. While Llama3.1-8B-Instruct initially falls below its baseline, it demonstrates strong and continuous recovery throughout the training process, steadily closing the performance gap. We attribute this delayed convergence to the heightened sensitivity of instruction-tuned models, which require additional steps to realign their complex representations under forced sparsity constraints. These trends indicate that the backbone can effectively adapt its representations to the prescribed sparse pathways. Flux Attention thus offers practical post-training flexibility, allowing users to lock in an efficiency budget and fine-tune for downstream applications without disrupting the routing dynamics.

6 Conclusion

We introduce *Flux Attention*, a context-aware dynamic routing framework mitigating the quadratic computational bottleneck of Large Language Models in long-context scenarios. Unlike existing hybrid attention mechanisms relying on rigid static allocations or hardware-inefficient head-level routing, our approach employs a lightweight Layer Router adaptively assigning each transformer layer to full or sparse Attention based on task and input demands. Extensive evaluations demonstrate our parameter-efficient method, requiring only 12 hours of training, achieves speedups up to 2.8× during prefilling and 2.0× during autoregressive decoding. Crucially, it preserves high-fidelity information recovery across diverse long-context benchmarks, establishing a superior and scalable trade-off between generation quality and inference efficiency for modern LLMs.

Acknowledgments and Disclosure of Funding

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. 62622608 and 62576232) and sponsored by CCF-Baidu Open Fund.

References

[1] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[2] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic

long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, 2025.

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

[4] Adithya Bhaskar, Alexander Wettig, Tianyu Gao, Yihe Dong, and Danqi Chen. Cache me if you can: How many kvs do you need for effective long-context lms? *arXiv preprint arXiv:2506.17121*, 2025.

[5] Rewon Child. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[6] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

[7] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024.

[8] DeepSeek-AI. Deepseek-v3.2-exp: Boosting long-context efficiency with deepseek sparse attention, 2025.

[9] Aditya Desai, Kumar Agrawal, Shuo Yang, Alejandro Cuadron Lafuente, Luis Gaspar Schroeder, Matei Zaharia, Joseph E Gonzalez, and Ion Stoica. vattention: Verified sparse attention via sampling. In *International Conference on Learning Representations*, volume 2026, pages 52253–52284, 2026.

[10] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

[11] Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

[12] Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. Zamba: A compact 7b ssm hybrid model, 2024.

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

[14] Junxian Guo, Haotian Tang, Shang Yang, Zhekai Zhang, Zhijian Liu, and Song Han. Block Sparse Attention. `https://github.com/mit-han-lab/Block-Sparse-Attention`, 2024.

[15] Zhiyuan He, Yike Zhang, Chengruidong Zhang, Huiqiang Jiang, Yuqing Yang, and Lili Qiu. Trianglemix: Accelerating prefilling via decoding-time contribution sparsity, 2025.

[16] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[17] Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online, June 2021. Association for Computational Linguistics.

[18] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[19] Xiaodong Ji, Hailin Zhang, Fangcheng Fu, and Bin Cui. Sale : Low-bit estimation for efficient sparse attention in long-context llm prefilling, 2025.

[20] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

[21] Jerome Ku, Eric Nguyen, David W. Romero, Garyk Brixi, Brandon Yang, Anton Vorontsov, Ali Taghibakhshi, Amy X. Lu, Dave P. Burke, Greg Brockman, Stefano Massaroli, Christopher Ré, Patrick D. Hsu, Brian L. Hie, Stefano Ermon, and Michael Poli. Systems and algorithms for convolutional multi-hybrid language models at scale, 2025.

[22] Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *The Thirteenth International Conference on Learning Representations*, 2025.

[23] Adrian Łańcucki, Konrad Staniszewski, Piotr Nawrot, and Edoardo Maria Ponti. Inference-time hyper-scaling with kv cache compression. *Advances in Neural Information Processing Systems*, 38:9365–9397, 2026.

[24] Hoang Anh Duy Le, Sahil Joshi, Zeyu Yang, Zhaozhuo Xu, and Anshumali Shrivastava. Scout before you attend: Sketch-and-walk sparse attention for efficient llm inference. *arXiv preprint arXiv:2602.07397*, 2026.

[25] Jia Li, Hao Zhu, Huanyu Liu, Xianjie Shi, He Zong, Yihong Dong, Kechi Zhang, Siyuan Jiang, Zhi Jin, and Ge Li. aixcoder-7b-v2: Training llms to fully utilize the long context in repository-level code completion. *arXiv preprint arXiv:2503.15301*, 2025.

[26] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.

[27] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model, 2024.

[28] Gang Lin, Dongfang Li, Zhuoen Chen, Yukun Shi, Xuhui Chen, Baotian Hu, and Min Zhang. Lycheedecode: Accelerating long-context LLM inference via hybrid-head sparse decoding. In *The Fourteenth International Conference on Learning Representations*, 2026.

[29] Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, et al. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*, 2025.

[30] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024.

[31] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[32] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms, 2025.

[33] MAA. American invitational mathematics examination (aime). *URL https://maa.org/math-competitions/aime*, 2024.

[34] Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, et al. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*, 2025.

[35] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

[36] Dan Peng, Zhihui Fu, Zewen Ye, Zhuoran Song, and Jun Wang. Accelerating prefilling for long-context llms via sparse pattern sharing. *arXiv preprint arXiv:2505.19578*, 2025.

[37] Dan Peng, Zhihui Fu, Zewen Ye, Zhuoran Song, and Jun Wang. Accelerating prefilling for long-context llms via sparse pattern sharing, 2025.

[38] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024.

[39] Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint*, 2024.

[40] Liliang Ren, Yang Liu, Shuohang Wang, Yichong Xu, Chenguang Zhu, and Cheng Xiang Zhai. Sparse modular activation for efficient sequence modeling. *Advances in Neural Information Processing Systems*, 36:19799–19822, 2023.

[41] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

[42] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: query-aware sparsity for efficient long-context LLM inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 47901–47911. PMLR / OpenReview.net, 2024.

[43] Zecheng Tang, Quantong Qiu, Yi Yang, Zhiyi Hong, Haiya Xiang, Kebin Liu, Qingqing Dang, Juntao Li, and Min Zhang. Elastic attention: Test-time adaptive sparsity ratios for efficient transformers, 2026.

[44] Zecheng Tang, Haitian Wang, Quantong Qiu, Baibei Ji, Ruoxi Sun, Keyan Zhou, Juntao Li, and Min Zhang. Loom-scope: a comprehensive and efficient long-context model evaluation framework. *arXiv preprint arXiv:2507.04723*, 2025.

[45] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[47] Xinghao Wang, Pengyu Wang, Dong Zhang, Chenkun Tan, Shaojun Zhou, Zhaoxiang Liu, Shiguo Lian, Fangxu Liu, Kai Song, and Xipeng Qiu. Sparser block-sparse attention via token permutation. *arXiv preprint arXiv:2510.21270*, 2025.

[48] Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. *arXiv preprint arXiv:2404.15574*, 2024.

[49] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv*, 2024.

[50] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Shang Yang, Haotian Tang, Yao Fu, Song Han, et al. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations*, 2025.

[51] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024.

[52] Jing Xiong, Jianghan Shen, Fanghua Ye, Chaofan Tao, Zhongwei Wan, Jianqiao Lu, Xun Wu, Chuanyang Zheng, Zhijiang Guo, Min Yang, Lingpeng Kong, and Ngai Wong. UNComp: Can matrix entropy uncover sparsity? — a compressor design from an uncertainty-aware perspective. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4179–4199, Suzhou, China, November 2025. Association for Computational Linguistics.

[53] Peng Xu, Wei Ping, Xianchao Wu, Zihan Liu, Mohammad Shoeybi, and Bryan Catanzaro. Chatqa 2: Bridging the gap to proprietary llms in long context and rag capabilities. *arXiv preprint arXiv:2407.14482*, 2024.

[54] Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[55] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

[56] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, Vienna, Austria, July 2025. Association for Computational Linguistics.

[57] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

[58] Chen Zhang, Yang Bai, Jiahuan Li, Anchun Gui, Keheng Wang, Feifan Liu, Guanyu Wu, Yuwei Jiang, Defei Bu, Li Wei, et al. Efficient context scaling with longcat zigzag attention. *arXiv preprint arXiv:2512.23966*, 2025.

[59] Jintao Zhang, Haoxu Wang, Kai Jiang, Kaiwen Zheng, Youhe Jiang, Ion Stoica, Jianfei Chen, Jun Zhu, and Joseph E Gonzalez. Sla2: Sparse-linear attention with learnable routing and qat. *arXiv preprint arXiv:2602.12675*, 2026.

[60] Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattn: Accurate sparse attention accelerating any model inference. In *International Conference on Machine Learning (ICML)*, 2025.

[61] Xuan Zhang, Fengzhuo Zhang, Cunxiao Du, Chao Du, Tianyu Pang, Wei Gao, and Min Lin. Lightransfer: Your long-context llm is secretly a hybrid model with effortless adaptation. *arXiv preprint arXiv:2410.13846*, 2024.

[62] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. H2O: heavy-hitter oracle for efficient generative inference of large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[63] Weilin Zhao, Zihan Zhou, Zhou Su, Chaojun Xiao, Yuxuan Li, Yanghao Li, Yudi Zhang, Weilun Zhao, Zhen Li, Yuxiang Huang, Ao Sun, Xu Han, and Zhiyuan Liu. Infllm-v2: Dense-sparse switchable attention for seamless short-to-long adaptation, 2025.

A Code Release and Supplementary Material

Our code is available at `https://github.com/qqtang-code/FluxAttention`.

B Related Work

B.1 Sparse and Selective Attention

To mitigate the quadratic complexity of standard attention mechanisms, existing research has broadly followed two main directions: inference-time heuristics and training-aware sparsification. Inference-time heuristics typically employ static patterns, such as fixed sliding windows or strides [51, 15, 3], to restrict the receptive field. To capture dynamic dependencies more effectively, content-aware approaches discard uninformative tokens based on accumulated importance scores [62, 26, 30], or use kernel-based estimators to identify important blocks and bypass redundant computations [20]. Additionally, prefill optimizers apply importance-driven selection to accelerate the processing of long contexts [22, 54, 60, 37, 19]. Recent block-level methods refine this further: Sketch-and-Walk [24] uses lightweight sketches and cross-layer walk scores to select top-k blocks for both prefill and decode, while PBS-Attn [47] permutes tokens to expose greater block sparsity and accelerates prefill with customized kernels. Verified sparsity takes a different route: vAttention [9] samples attention weights to certify that a per-head, per-token sparsity pattern preserves the full-attention output. Despite their effectiveness, these heuristic methods frequently rely on sensitive hyperparameters, limiting their robustness across different tasks.

In contrast, training-aware sparsification incorporates sparsity into the optimization objective to align the training process with sparse inference. A prominent direction in this area involves learnable selection. For instance, SeerAttention [11], NSA [56], and MoBA [32] employ learnable gates and hierarchical constraints to approximate ground-truth attention patterns. To connect dense pre-training with sparse adaptation, InfLLM-v2 [63] introduces a dense-sparse switchable mechanism via parameter-free pooling, whereas DSA [8] utilizes a lightning indexer alongside a two-stage training strategy to efficiently filter the top-k key-value pairs. However, most of these methods focus on fine-grained block-level or token-level selection within a fixed attention framework, rather than dynamically adapting the overall attention mode based on input complexity.

The operators above determine *which tokens or blocks* are attended. Flux Attention is orthogonal: it does not introduce a new sparse operator, and the results in this paper cover three materially different backends—SSA, XAttention, and Triangle Attention. Any of the block-level or fine-grained operators above could in principle serve as the sparse branch of a routed layer; we describe this as architectural compatibility rather than measured integration, and we restrict performance claims to the backends we evaluate.

B.2 Hybrid Architectures and Dynamic Allocation

To balance computational efficiency and model performance, hybrid architectures strategically integrate Full Attention (FA) with linear-complexity operators. The dominant paradigm, inter-layer hybridization, interleaves linear layers with standard attention layers to recover associative recall capabilities [21, 7]. Notable large-scale implementations, such as Jamba [27], utilize fixed block-wise ratios, whereas variants optimize memory utilization through shared global blocks [12] or sliding windows [39]. Extending this paradigm to adapt existing dense models, LightTransfer [61] identifies specific "lazy layers" that primarily focus on recent or initial tokens, statically replacing their full attention with streaming attention to convert long-context LLMs into hybrid variants with effortless adaptation. More recently, intra-layer hybridization has emerged as a strategy to refine structural granularity. For example, PruLong [4] and DuoAttention [49] combine FA and Sparse Attention (SA) within individual layers by assigning different attention heads to different computational modes. Furthermore, LongCat [58] proposes the LoZA mechanism, constructing a static ZigZag topology by replacing low-sensitivity Multi-head Latent Attention (MLA) modules with linear-complexity SA. A critical limitation of these approaches is their reliance on static topologies or pre-defined ratios established prior to inference, lacking the flexibility required to dynamically distinguish diverse tasks.

Dynamic allocation does not make a method novel by itself. SeqBoat [40] learns to activate a sparse attention module inside an otherwise recurrent model; Elastic Attention [43] allocates varying sparsity at the head level based on contextual importance; and SLA2 [59] learns a router that chooses between sparse and linear attention. Decode-time cache management is a parallel line of work, including inference-time hyper-scaling, which trades KV cache capacity for a longer generation budget [23]. What distinguishes Flux Attention is the *granularity and the deployment objective*: it conditions a whole-layer FA/SA decision on the input, keeps the decision homogeneous inside each layer so that the savings survive into measured decode latency and KV memory, and evaluates that quality–decode-latency–memory trade-off rather than FLOP reduction alone. We therefore do not claim to be the first dynamic or learned sparse-attention method; Table 3 positions our contribution along the unit of selection, training, inference phase, and objective.

Table 3: Positioning of Flux Attention against representative dynamic and fine-grained attention-selection methods. The axes are the unit at which computation is selected, whether the selection is learned or fixed, the inference phase the method primarily targets, and the deployment objective. Flux Attention is complementary to fine-grained operators: it chooses the coarse per-layer execution mode, while a method such as vAttention can supply fine-grained sparsity inside the routed sparse branch.

Method	Selection unit	Selection	Primary phase	Deployment objective
Flux Attention (ours)	Whole layer	Learned, input-conditioned FA/SA	Prefill routing, decode reuse	Decode-regular execution, KV memory
Elastic Attention [43]	Head	Learned, test-time adaptive	Prefill/TTFT	Adaptive sparsity ratio
DuoAttention [49]	Head	Offline static allocation	Decode	Hybrid long-context attention
PruLong [4]	Head	Offline static allocation	Decode	Pruned long-context attention
TriangleMix [15]	Block pattern	Fixed triangle topology	Prefill	Training-free sparse backend
XAttention [54]	Block	Fixed block selection score	Prefill	Prefill acceleration
SeqBoat [40]	Token / segment	Learned module activation	Decode	Sparse modular recurrence
SLA2 [59]	Attention computation	Learnable router	Training and decode	Sparse–linear hybridization
KV hyper-scaling [23]	KV cache entries	Compression schedule	Decode	Cache capacity vs. output length
Sketch-and-Walk [24]	Block	Learned walk scores over sketches	Prefill and decode	Top-k block selection
PBS-Attn [47]	Block, after permutation	Fixed permutation plus block sparsity	Prefill	Custom prefill kernels

One apparent conflict in the literature is worth resolving explicitly. Head-level dynamic sparsity has been reported to deliver substantial speedups, whereas our measurements show only marginal gains for it. The two results concern different inference phases: reported head-level fused-kernel gains are prefill/TTFT accelerations, while our Figure 3(b) and Appendix D.3 measure per-token autoregressive decode latency, where historical-KV access and memory bandwidth dominate. For decode, we additionally compare against PruLong's optimized block-sparse decode path in Table 15. Specialized head-wise kernels can therefore accelerate prefill, while the evaluated optimized head-wise implementation provides a smaller and less scalable decode benefit than whole-layer routing.

To address the rigidity of static designs, recent studies have explored dynamic allocation strategies. For instance, Elastic Attention [43] dynamically allocates varying sparsity at the head level based on contextual importance. While offering algorithmic flexibility, such head-level dynamic sparsity introduces hardware inefficiencies in the memory-bandwidth-bound decode phase. Specifically, varying context lengths across different attention heads lead to synchronization bottlenecks, as fast-executing sparse heads must wait for memory-intensive retrieval heads within the same layer. This creates significant memory bandwidth bottlenecks, hindering hardware acceleration and limiting practical speedups, especially during the autoregressive decoding phase.

B.3 Dynamic Routing in Neural Networks

Dynamic routing and conditional computation have been studied to decouple model capacity from inference cost. Traditional approaches, such as Mixture-of-Experts (MoE) [41, 10], route tokens to specialized Feed-Forward Network (FFN) experts. Recent methods like Mixture-of-Depths (MoD) [38] apply this idea by dynamically skipping specific layers for uninformative tokens to optimize compute allocation. While these methods successfully route computation, they primarily focus on FFNs or complete layer skipping, leaving the dynamic optimization of the attention mechanism less explored. Flux Attention instead routes the attention *mode* of an entire layer, which keeps intra-layer work homogeneous and memory access contiguous so that the theoretical reduction in attention compute is realized as wall-clock speedup.

C Limitation & Broader Impacts

Limitations. Although Flux Attention effectively balances long-context inference efficiency and hardware utilization, our approach has two main limitations. The first involves a granularity trade-off in routing. To achieve end-to-end wall-clock speedups, Flux Attention uses layer-level routing instead of finer-grained head-level or token-level sparsity. This design eliminates the synchronization overhead caused by computational imbalance but inevitably reduces algorithmic expressiveness. In complex reasoning tasks where only specific attention heads require full contextual information, the binary nature of layer-level routing can lead to suboptimal performance.

A second limitation relates to the locality of the prefill-suffix pooling strategy. Our Layer Router extracts features from the prefix and suffix of the prompt to evaluate the context, based on the heuristic assumption that long-context tasks typically define their primary intent through instructions at the beginning or questions at the end. While highly effective for most standard scenarios, this heuristic might occasionally miss scattered intents within the middle context. However, we consider this an optimizable constraint rather than a fundamental flaw. In future work, introducing finer-grained task representations could provide the router with a more comprehensive global context, helping it learn and route more accurately. Furthermore, we hypothesize that scaling up the training data diversity and extending the training duration could effectively alleviate this limitation and further improve the router's generalization robustness.

A third limitation concerns scope of validation rather than the design itself. Flux Attention is a routing framework, so its measured benefit depends on the sparse branch it is paired with. We validate three materially different backends—SSA, XAttention, and Triangle Attention—and report wall-clock results only for those; finer-grained operators such as vAttention are shown to compose at the attention interface (Appendix H.5), but a fully optimized integration of that kind remains future work, and we do not claim universal backend compatibility. Similarly, the accuracy comparisons against the dense backbone are close enough on several benchmarks that they should be read as parity rather than improvement, which is why we report router-training variance alongside them (Appendix G.4), evaluate batched serving rather than only single-request profiling (Table 14), and restrict the systems claim to the tested serving setup.

Broader Impacts. Beyond the practical efficiency gains, our research on Flux Attention offers fundamental scientific insights into the internal mechanics of Attention. The success of our routing mechanism empirically demonstrates that the attention allocation in LLMs is inherently sparse. More importantly, it highlights that different transformer layers capture highly distinct semantic representations and exhibit strong task heterogeneity. By revealing that uniform, dense computation across all layers is often redundant, our work encourages the research community to rethink standard scaling laws. We hope these insights will pave the way for the development of more dynamic, layer-adaptive architectures that align with the intrinsic sparsity of language models, ultimately promoting greener and more sustainable AI deployments.

D Sparsification Setup and Latency Profiling Implementation

This section details the layer importance identification, the progressive sparsification strategy, and the hardware latency measurement protocol mentioned in Section 2.3.

D.1 Layer Entropy Score Calculation

Following the methodology proposed by UnComp [52], we identify and rank Transformer layers based on their informational density and uncertainty when processing long contexts. We use a matrix entropy-based profiling method, quantifying the information content of each layer over long-context validation datasets to estimate its inherent structural sparsity.

For a given layer ℓ, we calculate its Entropy Score (E_ℓ) by measuring the truncated matrix entropy of its hidden representations. Formally, let s be the input sequence length, d be the hidden dimension, and $\mathcal{X}^{(\ell)} \in \mathbb{R}^{s\times d}$ be the hidden states matrix of layer ℓ. We first derive the trace-normalized covariance matrix $\Sigma^{(\ell)} = \frac{\mathcal{X}^{(\ell)}(\mathcal{X}^{(\ell)})^\top}{\mathrm{Tr}(\mathcal{X}^{(\ell)}(\mathcal{X}^{(\ell)})^\top)}$. The score is computed as the von Neumann entropy over its top-K eigenvalues:

$$E_\ell = -\sum_{i=1}^{K} \lambda_i^{(\ell)} \log \lambda_i^{(\ell)} \tag{7}$$

where $\lambda_i^{(\ell)}$ denotes the i-th largest eigenvalue of $\Sigma^{(\ell)}$, and K is the truncation threshold used to filter out noise. A lower E_ℓ indicates lower information density (i.e., lower uncertainty) and higher redundancy, making the layer a suitable candidate for sparsification. Intuitively, the spectrum of a trace-normalized hidden-state Gram matrix measures how concentrated the layer's representation is: a few dominant eigenvalues mean the layer's information lies in a low-dimensional subspace and is redundant, whereas a flat spectrum spreads information across many independent directions and is more likely to be damaged by aggressive sparsification. This is a profiling heuristic that ranks layers offline, not a guarantee that a given layer can be sparsified without loss; the learned router is free to override the ranking per input, and Section 5 plus Appendix H.3 quantify how far it departs from it.

D.2 Progressive Sparsification Strategy

Based on the computed entropy scores E_ℓ, we evaluate the information density of all L layers across the model. As defined in the main text, the Model Sparsity Ratio (Ω_{MSR}) represents the proportion of layers converted to sparse attention. To simulate the varying levels of sparsity reported in our experiments (e.g., $\Omega_{\mathrm{MSR}} = 20\%$), we use a thresholding mechanism based on these scores. We first determine the number of layers to preserve as full attention via $k = \lfloor (1 - \Omega_{\mathrm{MSR}}) \cdot L \rfloor$. The k layers with the highest entropy scores are retained as retrieval layers to ensure global information integration and preserve complex contextual pathways. The remaining $(L - k)$ layers with the lowest entropy scores are replaced with sparse layers.

D.3 Latency Measurement Implementation

To evaluate the hardware efficiency of different sparsity paradigms, we profile latency during the autoregressive decoding phase. All latency measurements are performed on a single NVIDIA A800 GPU (80GB) using PyTorch with BF16 precision.

To simulate realistic long-context retrieval scenarios while isolating the decoding bottleneck, we fix the batch size to 1 and evaluate across varying prompt sequence lengths. For each configuration, we perform 10 warm-up steps to initialize the CUDA context and stabilize GPU clocks, followed by 50 profiling iterations. The reported latency is the average wall-clock time required to generate a single token.

Implementation of Sparsity Baselines For the head-level sparsity baseline, we retain a subset of attention heads for dense computation while the remaining heads operate sparsely. However, highly optimized attention kernels (e.g., FlashAttention) lack hardware-level support for processing mixed context lengths across different heads within the same layer. Consequently, enforcing head-level sparsity results in fragmented, non-contiguous memory access patterns. The GPU memory bandwidth is still consumed by loading the full historical KV cache into SRAM, leading to only marginal wall-clock speedups despite the theoretical FLOP reduction.

In contrast, our layer-level sparsity implementation avoids this issue. When a layer operates sparsely, the decoding step fetches only the locally required KV states, bypassing the global historical KV tensors. This layer-level routing allows contiguous memory loading, translating theoretical sparsity into proportional decoding acceleration. We calculate the speedup as the ratio of the latency of the full dense model to that of the sparsified model for a given input length.

E Detailed Pseudocode for Inference and Deployment

To maximize hardware efficiency and translate theoretical FLOP reductions into practical wall-clock speedups, the inference process is strictly decoupled into two phases. During the prefill phase, the lightweight Layer Router analyzes the semantic context of the input queries and generates a deterministic hard routing decision for each layer. These decisions are subsequently cached. During the decode phase, the router module is entirely bypassed. The model reuses the cached $O(1)$ routing decisions to dynamically manage the KV cache. Specifically, for layers designated as sparse, the system maintains only a minimal sparse KV cache and completely bypasses the heavy loading of full historical states, thereby alleviating the memory-bandwidth bottleneck.

F Implementation Details

This section details the training configurations, baseline implementations, and system-level optimizations for efficient long-context processing.

F.1 Training Configuration and Hyperparameters

We evaluate the proposed approach on models of various sizes, including Qwen3-4B, Qwen3-8B [55], and Meta-Llama-3.1-Instruct [13]. We freeze the pre-trained backbone and update only the parameters of the Layer Router to maintain the general capabilities of the model. For task representation, we apply a Prefill-Suffix Pooling operation to aggregate the first 100 and the last 100 tokens of the sequence, as these segments typically contain the system instructions and user queries required to identify the task.

We train all models with a sequence length of $L = 65,536$ tokens in bfloat16 precision using the AdamW optimizer [31] ($\beta_1 = 0.9, \beta_2 = 0.95$). Training is conducted on a distributed cluster with Fully Sharded Data Parallel (FSDP) under a hybrid sharding strategy. To balance the convergence of the router and sparsity regularization, we apply a decoupled learning rate schedule. The Layer Router uses a learning rate of 5×10^{-4} for rapid adaptation to retrieval patterns, while the sparsity regularization terms use a higher learning rate of 1×10^{-3}. The dual regularization coefficients λ_1 and λ_2 are randomly initialized and optimized alongside the router parameters. A cosine decay learning rate schedule is applied after a linear warmup phase over the first 20% of the training steps.

F.2 Training Data Provenance

Table 4 lists the five sources of the router training mixture against the task families they cover. The mixture targets task characteristics—multi-hop retrieval, long-form summarization, and repository-level code—rather than importing the evaluation benchmark suites, and the evaluation benchmark files themselves are not used as training inputs.

That statement is about task-level coverage, and it is deliberately weaker than a sample-level non-overlap claim: corpora such as GovReport are public, so a shared passage between the training corpus and a downstream

Algorithm 1 Flux Attention: Context-Aware Routing and Efficient Inference

Input: Layer index l, Input Queries x_Q, Keys x_K, Values x_V
Input: Current Phase $P \in \{\text{Prefill, Decode}\}$
Input: Routing Decision Cache $\mathcal{C}$, Global State Cache $\mathcal{KV}$

if $P ==$ Prefill **then**
  {— *Context-Aware Layer Routing* —}
  $p \leftarrow$ Prefill-Suffix-Pooling(x_Q) {Extract sequence-level descriptor from Query}
  $h \leftarrow$ ContextEncoder(p) {Capture contextual dependencies via MLP}
  $[\pi_{SA}, \pi_{FA}] \leftarrow$ RouterHead$_l(h)$ {Generate unnormalized routing logits}
  $r_{hard}^{(l)} \leftarrow \text{argmax}(\pi_{SA}, \pi_{FA})$ {1 for FA mode, 0 for SA mode}
  $\mathcal{C}[l] \leftarrow r_{hard}^{(l)}$ {Cache routing decision for the Decode phase}
else
  {— *Efficient Routing Reuse* —}
  $r_{hard}^{(l)} \leftarrow \mathcal{C}[l]$ {Bypass Router entirely, reuse $O(1)$ deterministic decision}
end if

{— *Attention Execution & KV Cache Management* —}
if $r_{hard}^{(l)} == 1$ **then**
  if $P ==$ Prefill **then**
    $\mathcal{KV}[l] \leftarrow (x_K, x_V)$ {Retain complete KV cache for precise retrieval}
  end if
  $\mathcal{O} \leftarrow$ FullAttention$(x_Q, x_K, x_V, \mathcal{KV}[l])$
else
  if $P ==$ Prefill **then**
    $\mathcal{KV}_{sparse}[l] \leftarrow$ Condense(x_K, x_V) {Maintain only minimal sparse KV cache}
  end if
  $\mathcal{O} \leftarrow$ SparseAttention$(x_Q, x_K, x_V, \mathcal{KV}_{sparse}[l])$ {Bypass heavy historical KV loading}
end if
return $\mathcal{O}$

Table 4: Training mixture of the Layer Router (approximately 0.74B tokens over 1K–64K sequences). The mixture is chosen to expose the router to both retrieval-intensive and context-holistic regimes; it does not import the evaluation benchmark suites.

Source	Task family it covers	Regime exposed to the router
ChatQA2-Long-SFT-data [53]	Single-document and multi-document QA over long inputs	Retrieval-intensive
MuSiQue [45]	Multi-hop question answering	Retrieval-intensive
CoLT-132K [25]	Repository-level code completion	Context-holistic
GovReport [17]	Long-form summarization of government reports	Context-holistic
XSum [35]	Abstractive summarization	Context-holistic

benchmark cannot be excluded by construction. We therefore specify the matching rule explicitly instead of asserting blanket disjointness.

F.3 Baseline Implementation Details

We compare the proposed approach with several state-of-the-art sparse attention mechanisms, categorizing them into training-free and training-based methods. For training-free baselines, we evaluate TriangleMix [1] [15], which relies on heuristic-based sparsity without parameter updates. For training-based baselines, including

[1] `https://github.com/microsoft/MInference/tree/main/TriangleMix`

Table 5: Baseline comparison controls. All trainable baselines are trained in the same environment and on the same data mixture as Flux Attention while retaining their original method-specific hyperparameters; TriangleMix's Triangle Attention (TA) is training-free and serves as a sparse-attention backend for Flux rather than as a routed competitor.

Method	**Routing / selection unit**	**Training or calibration in our comparison**	**Deployment focus**
FluxAttn (ours)	Whole layer, input-conditioned FA/SA	Router trained on the shared data mixture; backbone frozen	Decode-regular layer execution
PruLong	Offline head selection	Same data and environment; original method hyperparameters	Optimized block-sparse decode
DuoAttention	Head allocation	Same data and environment; original method hyperparameters	Hybrid long-context attention
TriangleMix / TA	Triangle Attention operator	Training-free	Sparse-attention backend

PruLong [2] [4] and DuoAttention [3] [50], we follow a unified fine-tuning protocol. We train all baselines in identical environments and on the same dataset while maintaining their original hyperparameter settings. Table 5 summarizes the resulting calibration and fairness controls, together with each method's routing or selection unit and deployment focus.

F.4 Sparsity and Kernel Configuration

We use Block-Sparse-Attention [14] for efficient streaming inference to control the granularity and retention policy of the attention mechanism. We set the block size to 64 to define the minimum unit of sparsity, and the chunk size to 16,384 to process ultra-long sequences. A sink token size of 128 is maintained to preserve the attention sink phenomenon, ensuring stability during streaming generation. Additional kernel parameters, such as stride, normalization, and selection modes, are detailed in the Sparsity Config section of Table 7.

F.5 Systems Setup and Profiling Protocol

Table 6 documents the hardware and software configuration behind every latency and memory number reported in Figure 3 and Table 15. All compared methods share the GPU, precision, input and output shapes, warm-up, repetition, and timing protocol; the attention execution path is the controlled difference, so the dense baseline uses the standard FlashAttention-2 path in the same PyTorch/CUDA environment as the sparse and hybrid methods. We report prefill and decode quantities separately and label them by phase and unit throughout: prefill latency is the time to first token, and decode latency is the mean wall-clock time per generated token.

G Additional Evaluation Results

In this section, we present detailed performance for the benchmarks discussed in the main paper. Furthermore, to provide a more comprehensive comparison, we have expanded our evaluation by including two additional baselines: XAttention [4] [54] and LycheeDecode [5] [28].

G.1 Evaluation Results on LongBench

Table 8 provides the complete evaluation results on the LongBench-E. We report the performance metrics across all sub-tasks, categorized into 6 categories: Single-Document QA (S-Doc QA), Multi-Document QA (M-Doc QA), Summarization (Summ), In-context Learning (In-Context), Synthetic, and Code Tasks.

Table 6: Systems setup for the latency and memory measurements reported in Figure 3 and Table 15. All compared methods share the hardware, precision, input/output shapes, warm-up, repetition, and timing protocol; the attention execution path is the controlled difference.

Item	Setting
GPU	1×NVIDIA A800 80GB (PCIe); batched-serving results in Table 14 on 8×NVIDIA H20
Precision	BF16
Batch size	1 for profiling; 8/16/24/32 for the serving experiment
Dense attention path	FlashAttention-2 with the same PyTorch/CUDA environment as the sparse and hybrid paths
Sparse attention kernel	Block-Sparse-Attention, block size 64, sink size 128, chunk size 16384 (Section F.4)
Warm-up / profiling	10 warm-up iterations, 50 profiling iterations, one token per step
Reported quantity	Mean wall-clock time per generated token; speedup is the ratio of dense to sparsified latency at the same input length

Table 7: Hyperparameters: General configuration.

Hyperparameter	Value
Model & Training	
Base Model	`Qwen, Llama`
Sequence length	65536
Precision	bfloat16
Global Batch Size	48
Training Steps	300
Mask / Reg. LR	$5e^{-4}$ / $1e^{-3}$
Warmup Ratio	0.2
AdamW Momentum (β_1, β_2)	$(0.9, 0.95)$
Weight Decay	0.1
Learning Rate Schedule	Cosine
Sparsity Config	
Pool Size	100
Sink / Local Size	128 / 2048
Block / Chunk Size	64 / 16384
Stride / Threshold	16 / 0.9
Selection Mode	`Inverse`

G.2 Evaluation Results on RULER

This subsection details the experimental settings and evaluation results for the RULER benchmark. For RULER, as detailed in Table 9, our evaluation on the RULER benchmark covers a wide range of context lengths, ranging from 8K to 256K tokens. The comparative performance results are summarized in Table 10.

Table 10 reports the tasks shared by the compared methods and omits Common Words Extraction (CWE) and Variable Tracking (VT). We complete the comparison for these two tasks in Table 11: on Llama-3.1-8B-Instruct, Flux Attention improves the two-task aggregate over the dense backbone by 2.59 points and over DuoAttention by 0.35 points at approximately matched sparsity, and the conclusion holds across most of the evaluated length bins, with the advantage narrowing beyond 128K.

[2] `https://github.com/princeton-pli/PruLong`
[3] `https://github.com/mit-han-lab/duo-attention`
[4] `https://github.com/mit-han-lab/x-attention`
[5] `https://github.com/HITsz-TMG/TMGNLP/tree/main/LycheeDecode`

Table 8: Performance on LongBench-E [1]. We report average performance (Perf.) and Ω_{MSR} per task category. The 1st and the 2nd performance in each comparison group are highlighted with **bold font** and underlined, respectively. Gray-shaded rows denote the sparse-decode configuration.

Method	S-Doc QA		M-Doc QA		Summ		In-Context			Synthetic		Code		Avg.	
	Qasper	MF-en	HotQA	2Wiki	Gov.	M.News	TREC	TQA	SAMS	PCount	PRe	RB-P	Lcc	**Perf.**	Ω_{MSR}
Qwen3-4B backbone model															
Qwen3-4B	35.21	52.16	44.81	32.15	33.47	23.45	**70.67**	88.22	39.74	2.33	96.84	50.84	57.93	48.45	-
+ XAttention	32.80	50.36	44.54	33.17	**33.79**	**23.73**	69.00	87.44	39.90	3.42	74.59	49.42	59.62	46.44	-
+ LycheeDecode	34.12	**52.63**	44.74	31.53	31.16	23.24	68.67	87.69	39.12	4.00	**96.87**	49.17	57.29	47.83	0.50
+ DuoAttention	**35.83**	49.84	47.09	32.24	33.32	23.70	69.33	85.87	39.75	**4.50**	94.57	50.56	57.43	48.22	0.50
+ PruLong	34.15	50.78	44.48	32.89	32.96	23.53	67.67	**88.69**	39.55	3.17	90.17	49.00	54.07	47.16	0.50
+ TriangleMix	35.55	52.02	45.37	31.76	33.32	23.70	69.00	88.20	39.74	3.83	91.51	48.58	56.38	47.72	0.50
+ FluxAttn (FA-SSA)	35.02	49.44	49.64	32.27	33.26	23.48	69.33	88.29	39.78	1.50	94.56	**53.44**	59.69	48.72	0.44
+ FluxAttn (FA-XA)	35.74	51.70	45.83	32.34	33.57	23.66	69.00	87.23	39.81	3.50	93.74	50.81	59.28	48.32	0.53
+ FluxAttn (FA-TA)	35.02	50.89	45.17	**34.24**	33.02	23.53	69.00	88.08	**40.38**	3.94	96.06	51.68	**60.00**	**48.76**	0.47
+ FluxAttn (FA-SSA)	35.10	51.68	**49.65**	32.86	33.04	23.42	69.33	88.00	40.00	1.67	94.47	51.40	58.68	48.59	0.44
Qwen3-8B backbone model															
Qwen3-8B	41.22	49.92	**58.98**	44.21	33.27	23.42	71.33	86.77	41.83	**2.00**	98.33	56.08	66.31	52.16	-
+ XAttention	38.03	48.93	57.32	40.65	**33.41**	23.37	69.33	87.69	41.77	**2.00**	83.72	55.53	65.71	50.13	-
+ LycheeDecode	38.81	49.34	58.39	43.13	31.80	22.73	71.33	87.13	40.13	**2.00**	96.33	54.72	65.46	51.22	0.50
+ DuoAttention	**41.78**	**51.55**	55.96	41.70	33.24	23.34	69.33	**89.35**	41.62	0.50	98.93	57.54	69.39	52.13	0.50
+ PruLong	37.95	51.20	51.94	36.48	33.11	23.36	69.00	87.90	42.11	1.00	98.00	57.05	67.66	50.80	0.50
+ TriangleMix	40.82	51.31	57.57	**44.51**	33.32	23.35	71.33	86.73	41.79	**2.00**	94.33	55.04	65.89	51.65	0.50
+ FluxAttn (FA-SSA)	40.30	50.49	56.02	40.90	33.01	**23.55**	71.67	88.31	41.61	0.33	**100.00**	**59.46**	68.27	52.18	0.46
+ FluxAttn (FA-XA)	40.41	50.26	57.78	40.57	33.27	23.51	69.67	87.19	**42.12**	1.33	99.33	55.41	65.51	51.57	0.51
+ FluxAttn (FA-TA)	41.00	49.76	58.19	44.36	33.32	23.35	70.00	88.77	41.70	1.33	99.67	55.60	67.22	**52.22**	0.47
+ FluxAttn (FA-SSA)	39.92	50.04	55.72	40.81	33.03	23.50	**72.00**	88.48	40.96	0.33	99.22	58.57	**69.46**	52.05	0.46
Llama-3.1-8B-Instruct backbone model															
Llama-3.1-8B-Instruct	44.06	53.44	59.62	44.08	34.50	26.02	71.00	90.54	42.94	**12.67**	99.33	47.78	63.85	**53.28**	-
+ XAttention	43.84	53.80	**59.98**	43.47	34.47	26.05	**72.33**	90.39	42.99	8.33	73.33	50.13	62.92	51.00	-
+ LycheeDecode	42.51	54.33	58.29	**45.22**	30.38	25.07	70.00	91.26	41.58	**12.67**	96.00	46.84	63.22	52.25	0.50
+ DuoAttention	34.63	50.74	49.70	36.41	34.25	25.78	70.00	91.45	42.13	9.80	97.33	53.59	**68.55**	52.11	0.50
+ PruLong	41.51	52.36	50.46	37.57	34.25	25.86	66.33	89.93	41.72	9.07	97.00	**56.84**	66.23	51.68	0.50
+ TriangleMix	45.10	**54.60**	56.67	41.88	34.09	25.51	71.33	90.93	42.63	10.62	94.67	43.64	59.35	51.67	0.50
+ FluxAttn (FA-SSA)	**45.25**	54.42	54.54	41.34	34.54	**26.16**	68.33	**91.91**	42.17	9.00	97.67	47.74	65.35	52.28	0.51
+ FluxAttn (FA-XA)	42.14	53.13	58.53	43.50	**34.66**	26.06	70.67	91.46	**43.13**	8.00	**99.67**	50.91	64.78	53.07	0.72
+ FluxAttn (FA-TA)	44.77	54.12	57.35	43.43	34.31	25.80	**72.33**	91.32	42.62	9.33	98.33	45.48	60.70	52.42	0.62
+ FluxAttn (FA-SSA)	43.76	53.41	57.36	39.43	32.96	25.63	70.33	91.27	42.20	11.00	98.67	45.60	66.17	52.30	0.51

Table 9: Detailed configuration for the RULER benchmark evaluation. We evaluate across exponentially increasing context windows up to 256k tokens.

PARAMETER	CONFIGURATION DETAILS
Context Windows	8k, 16k, 32k, 64k, 128k, 256k
Sample Size	50 samples per task-length pair
Evaluation Tasks	**Retrieval (NIAH):** `Single_{1-3}`, `MultiKey_{1-3}`, `MultiQuery`, `MultiValue` **QA & Extraction:** `QA_{1,2}`, `FWE`

G.3 Sensitivity to the Sparse Attention Backend

Figure 1(a) profiles sparsity sensitivity with Streaming Sparse Attention (SSA) under the configuration of Section F.4. Because the observed degradation could in principle be an artifact of that backend, we repeat the sweep with XAttention (XA) and Triangle Attention (TA) on Llama-3.1-8B-Instruct under the same evaluation setting; Table 12 reports the per-category results normalized to the dense configuration. Both backends reproduce the qualitative pattern of Figure 1(a): retrieval-heavy document QA and multi-hop QA degrade sharply at high sparsity, while summarization and code tolerate aggressive sparsification. The onset and magnitude differ across backends, which is precisely why the Layer Router conditions its allocation on the chosen sparse branch instead of assuming a universal sparsity threshold. This is also the sense in which Flux Attention is backend-agnostic but not backend-indifferent: the framework routes at the layer level, and its measured benefits are reported for SSA, XA, and TA only. Methods such as Quest [42], MoBA, and DSA are natural candidates for the sparse branch, but we make no performance claim about them without an implementation and a matched experiment.

G.4 Router-Training Stochasticity

Because the retained router is trained rather than calibrated, the headline LongBench-E comparison could in principle reflect a lucky initialization. Table 13 reports two independently trained FA-SSA routers on Llama-3.1-8B-Instruct, the originally reported checkpoint and an additional run (seed 456). The aggregate difference is 0.08 points and no single task differs by more than 0.5 points, so the router-training noise is small relative to the

Table 10: Model performance on RULER tasks. **Bold** and underline indicate the best and second-best performance within each backbone group, respectively.NS denotes Single-needle in a haystack (NIAH) tasks, NMK stands for Multi-key NIAH, NMV is Multi-value NIAH, and NMQ refers to Multi-query NIAH. QA represents standard Question Answering tasks, and FWE stands for Frequent Words Extraction. Numeric suffixes (e.g., -1, -2, -3) indicate different subsets or difficulty levels within the corresponding task categories.

Models	NS-3	QA-2	NMK-3	NS-2	FWE	QA-1	NMQ	NMV	NMK-2	NS-1	NMK-1	Avg.
Backbone: Qwen3-4B												
Qwen3-4B	95.02	35.59	**37.54**	80.00	84.47	40.14	66.19	**65.91**	47.39	**100.00**	73.05	66.00
+ LycheeDecode	72.33	30.67	5.00	64.67	71.33	27.33	55.58	50.58	25.00	**100.00**	57.67	50.92
+ TriangleMix	95.00	34.67	34.33	82.67	83.11	35.67	60.42	60.00	43.67	99.33	72.33	63.74
+ DuoAttention	97.08	32.12	21.45	84.75	81.38	27.17	66.67	63.41	30.66	99.28	63.50	60.67
+ PruLong	96.34	33.80	22.44	81.51	84.87	22.93	64.87	57.87	27.04	**100.00**	71.03	60.25
+ XAttention	90.00	35.67	34.00	76.33	82.33	36.33	57.00	59.25	34.00	**100.00**	68.67	61.23
+ FA-SSA	**97.67**	**38.67**	33.00	93.67	85.33	34.33	72.58	59.17	**51.33**	**100.00**	70.67	66.95
+ FA-XA	95.00	38.00	32.00	78.33	85.56	38.00	61.58	62.50	40.00	**100.00**	69.33	63.67
+ FA-TA	96.00	35.67	30.33	91.67	**85.79**	**43.00**	65.08	64.17	40.33	99.67	69.33	65.55
+ FA-SSA Decode	97.00	37.67	34.00	**96.67**	85.44	33.33	**76.92**	55.67	50.33	**100.00**	72.00	**67.19**
Backbone: Qwen3-8B												
Qwen3-8B	98.00	39.33	**44.67**	96.67	84.78	**56.33**	90.92	78.08	**63.33**	**100.00**	81.00	**75.74**
+ LycheeDecode	75.00	35.33	8.33	79.33	73.44	45.33	65.25	54.83	24.33	**100.00**	64.33	56.86
+ TriangleMIX	98.00	40.13	16.33	97.33	86.18	40.33	91.17	79.10	49.33	**100.00**	77.33	70.47
+ DuoAttention	98.67	38.67	37.33	94.67	84.67	50.33	88.83	77.42	55.33	**100.00**	70.67	72.41
+ PruLong	98.33	36.33	33.33	94.00	84.22	46.67	82.50	76.33	58.67	**100.00**	70.33	70.97
+ XAttention	97.33	38.67	35.33	96.67	85.11	47.33	85.75	76.92	51.33	**100.00**	**85.00**	72.68
+ FA-SSA	98.67	42.00	20.00	97.00	85.89	50.67	90.58	80.17	59.00	**100.00**	79.33	73.03
+ FA-XA	**99.00**	39.00	39.00	96.67	85.78	52.00	87.92	78.75	58.67	**100.00**	84.33	74.65
+ FA-TA	97.33	39.33	33.33	97.33	85.56	52.67	**91.42**	74.58	56.33	**100.00**	80.67	73.51
+ FA-SSA Decode	97.33	42.00	20.00	97.67	84.33	51.00	88.92	**82.00**	56.00	**100.00**	78.33	72.51
Backbone: Llama-3.1-8B-Instruct												
Llama-3.1-8B-Instruct	**99.00**	**48.67**	**71.33**	94.00	82.11	**70.00**	**96.50**	**95.17**	69.00	99.67	**92.67**	**83.47**
+ LycheeDecode	34.67	43.67	20.00	77.33	69.78	56.33	67.50	66.08	38.67	98.67	77.00	59.06
+ TriangleMIX	97.33	48.33	62.00	97.67	80.33	67.33	93.08	79.42	**75.00**	94.33	90.33	80.46
+ DuoAttention	89.33	39.67	60.67	77.67	84.89	52.67	71.08	78.33	62.00	93.67	63.67	70.33
+ PruLong	76.33	33.00	44.00	72.33	85.11	37.67	68.83	68.25	53.00	72.00	48.00	59.87
+ XAttention	89.33	48.00	65.00	78.00	81.44	58.33	82.33	86.58	62.00	**100.00**	78.00	75.36
+ FA-SSA	98.33	42.67	59.33	95.00	84.44	41.33	93.17	82.25	67.67	95.33	84.67	76.75
+ FA-XA	97.33	**48.67**	68.33	88.00	82.33	60.33	88.75	89.17	66.00	**100.00**	85.67	79.51
+ FA-TA	**99.00**	48.00	62.00	**99.00**	80.56	69.00	93.50	84.00	**75.00**	94.33	92.00	81.50
+ FA-SSA Decode	90.00	44.40	70.80	86.40	**86.27**	47.20	90.30	71.00	73.20	70.40	73.67	73.67

reported gaps. Where the margin over the dense backbone is smaller than this band, we read the result as parity rather than as a reliable improvement, and we state the comparison accordingly.

G.5 Batched Serving Throughput

Single-request profiling can overstate the benefit of a sparsity scheme, because heterogeneous per-head workloads may be co-scheduled across requests once the server batches them. Table 14 therefore measures aggregate decode throughput with concurrent requests at a 32K context. Flux Attention remains faster at every feasible batch size and reaches a larger feasible batch before running out of memory; we report the infeasible configurations as OOM rather than extrapolating throughput, and we state the claim for the tested serving setup rather than universally. At batch 32 both methods exhaust memory on this hardware.

G.6 Full Quality–Latency–Memory Table

Table 15 gives the compact matched-sparsity comparison summarized in the main text, and Table 16 the complete version with raw prefill and decode latencies, persistent KV memory, and peak memory per configuration, so that every reported speedup can be traced back to the measured quantities.

Table 11: Common Words Extraction (CWE) and Variable Tracking (VT), the two RULER tasks omitted from Table 10, evaluated on Llama-3.1-8B-Instruct under the same six context lengths $\{8\text{K}, 16\text{K}, 32\text{K}, 64\text{K}, 128\text{K}, 256\text{K}\}$. Including them preserves the main conclusion: FluxAttn remains ahead of both the dense backbone and the static head-level baseline at approximately matched sparsity. This added evaluation covers Llama-3.1-8B-Instruct only.

Method	**Sparsity**	**CWE**	**VT**	**Mean**
Backbone (FA)	0%	39.90	80.60	60.25
FluxAttn	≈51% dynamic	43.07	82.60	**62.84**
DuoAttention	50% static	42.10	82.87	62.49

Length	**Backbone**	**FluxAttn**	**DuoAttention**
0–8K	58.0	64.8	**66.0**
16K–32K	61.6	**81.2**	78.8
32K–64K	85.2	**86.0**	77.2
64K–128K	**96.0**	94.0	93.2
>128K	91.4	84.8	**91.0**

Table 12: Sensitivity of the Figure 1(a) observation to the sparse operator, on Llama-3.1-8B-Instruct. Values are performance normalized to the dense configuration (100.00) and are averaged over the LongBench sub-tasks grouped below. Both XAttention (XA) and Triangle Attention (TA) reproduce the qualitative trend of Figure 1(a)—retrieval-heavy QA degrades sharply at high sparsity while summarization and code remain tolerant—but the onset and magnitude of the degradation are backend-dependent, which is why the router conditions its allocation on the chosen sparse branch instead of assuming a universal threshold.

Sparsity	`XAttention (XA)`				`Triangle Attention (TA)`			
	Doc QA	Multi-Hop	Summ.	Code	Doc QA	Multi-Hop	Summ.	Code
0.0	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
0.2	102.78	96.26	99.97	98.54	101.76	99.10	94.86	95.63
0.4	101.70	92.01	101.14	102.91	100.16	95.26	92.84	97.13
0.6	96.87	92.37	101.04	104.54	97.98	93.21	92.32	97.01
0.7	95.44	90.16	95.85	104.02	93.21	81.86	90.57	101.11
0.8	89.71	88.33	96.02	102.00	87.20	83.65	88.05	96.14
0.9	77.89	87.90	96.36	104.23	75.51	81.14	88.58	96.00

G.7 Comparison with Elastic Attention

To ensure a fair evaluation, we establish a consistent baseline setting: both methods employ full attention for the retrieval mode and streaming attention for the sparse mode. For Elastic Attention [43], we use its official implementation [6].

While Elastic Attention dynamically introduces sparsity at the head level to reduce theoretical FLOPs, this fine-grained approach often encounters hardware bottlenecks. Head-wise sparsity creates fragmented memory access patterns and computational load imbalance across GPU threads, limiting the actual wall-clock speedup. Flux Attention routes at the layer level, maintaining contiguous memory access for better hardware efficiency.

The two methods are also measured on different inference phases, and the numbers in Table 17 are prefill/TTFT speedups. Elastic Attention's fused-kernel result likewise concerns prefill acceleration; our decode-phase measurements (Figure 3(b) and Table 15) show that the same granularity is markedly less effective per generated token, where historical-KV access and memory bandwidth dominate. The distinction matters when reading speedup claims: prefill speedup and per-token decode speedup are not interchangeable, and we annotate every system number in this paper by phase.

This efficiency is empirically validated in Table 17. Evaluated on Llama-3.1-8B-Instruct, Elastic Attention requires an aggressive sparsity of 68% to achieve a 2.6× prefill speedup. Dropping this amount of context impairs its ability to process complex dependencies, reducing its RULER score to 72.85. Flux Attention operates

[6] `https://github.com/LCM-Lab/Elastic-Attention`

Table 13: LongBench-E per-task results for two independently trained FA-SSA routers on Llama-3.1-8B-Instruct. Router-training stochasticity shifts the task averages by at most 0.5 points, and the aggregate difference between the two runs is 0.08 points.

Task	Original	Seed 456	Mean $\pm$ Std
Qasper	45.25	45.38	45.32 ± 0.07
MF-en	54.42	54.42	54.42 ± 0.00
HotQA	54.54	54.54	54.54 ± 0.00
2Wiki	41.34	41.34	41.34 ± 0.00
Gov.	34.54	34.71	34.63 ± 0.09
M.News	26.16	26.22	26.19 ± 0.03
TREC	68.33	68.74	68.54 ± 0.21
TQA	91.91	91.92	91.92 ± 0.01
SAMS	42.17	42.07	42.12 ± 0.05
PCount	9.00	9.09	9.05 ± 0.05
PRe	97.67	97.67	97.67 ± 0.00
RB-P	47.74	48.24	47.99 ± 0.25
Lcc	65.35	65.12	65.24 ± 0.12
Avg.	52.19	52.27	52.23 ± 0.04

Table 14: Decode throughput under batched serving on Llama-3.1-8B-Instruct at a 32K context, measured on a node of 8$\times$NVIDIA H20 GPUs. Throughput is aggregate tokens/s across the served batch. OOM means the configuration exhausts GPU memory on this hardware; we report it rather than extrapolating throughput. The relative advantage does not disappear once heterogeneous requests are co-scheduled, and FluxAttn reaches a larger feasible batch before running out of memory.

Batch size	FluxAttn (tok/s)	FA2 (tok/s)	FluxAttn / FA2
8	194	147	1.32$\times$
16	196	95	2.07$\times$
24	280	OOM	—
32	OOM	OOM	—

at a lower sparsity of 51%, i.e., it retains more attention computation. Due to its layer-wise efficiency, it still achieves a 2.5$\times$ prefill speedup. By preserving more context, Flux Attention improves the RULER score to 76.75 (+3.9) while maintaining comparable performance on LongBench.

To verify the generalizability of our method, we extend the comparison across varying model architectures and scales, including Qwen3-4B, Qwen3-8B, and Llama-3.1-8B-Instruct. As shown in Table 18, although Elastic Attention matches the dense performance on LongBench, it degrades significantly on RULER. For example, its Llama-3.1-8B-Instruct RULER score drops from 83.47 (dense baseline) to 72.85. Flux Attention consistently outperforms Elastic Attention in RULER across all evaluated models. In Qwen3-4B, our method surpasses Elastic Attention (66.95 vs. 61.81) and slightly exceeds the dense baseline.

H Analysis

H.1 Impact of Data Composition on Task Differentiation

In previous sections, we have established that Flux Attention dynamically tailors sparsity to specific task demands. To fully unleash this capability, we discover that a well-balanced training curriculum acts as a crucial catalyst. To empirically validate this, we analyze the routing dynamics—specifically, the evolution of sparsity levels across training steps—under different data distribution settings.

Figure 7 (Left) illustrates the sparsity trajectories when the router is trained on a well-balanced dataset. Driven by this diverse curriculum, the router successfully disentangles the underlying task demands and exhibits a clear divergence in its routing behavior. Notably, after an initial shared exploration phase, retrieval-intensive tasks converge to a lower sparsity level to preserve critical historical keys and values. In contrast, context-holistic tasks confidently sparsify the context, diverging toward higher sparsity levels. This demonstrates that a balanced mixture effectively teaches the router to establish robust, task-specific boundaries.

Conversely, Figure 7 (Right) demonstrates the routing behavior when the training data is heavily skewed (e.g., dominated by context-holistic tasks). Under this setting, the router faithfully optimizes for the predominant data

Table 15: Quality–latency–memory trade-off at matched sparsity on Llama-3.1-8B-Instruct. Accuracy is the RULER score per context length; speedups are relative to the dense FA2 baseline at the same context length, with prefill and decode reported separately because they are different phases. FluxAttn and PruLong operate at approximately matched model sparsity ($\Omega_{\mathrm{MSR}} \approx 0.50$), so the comparison isolates routing granularity rather than the amount of computation retained. "KV saving" is the reduction in persistent decode-time KV memory relative to dense. The accuracy entries coincide with the corresponding rows of Table 2, where the FluxAttn configuration is the sparse-decode one. Raw latencies and peak memory are given in Table 16, and the systems setup in Table 6.

Metric	**8K**	**16K**	**32K**	**64K**	**128K**	**256K**
Acc. Dense / full	92.88	92.83	89.46	70.79	80.12	72.34
Acc. PruLong / head	86.96	76.55	70.65	54.52	48.18	30.00
Acc. FluxAttn / layer	**90.11**	**79.39**	**79.22**	**56.08**	**62.94**	**59.39**
Ω_{MSR} (PruLong / FluxAttn)	.50 / .53	.50 / .50	.50 / .53	.50 / .53	.50 / .50	.50 / .50
Prefill speedup, PruLong	1.06×	1.14×	1.25×	1.42×	1.60×	1.76×
Prefill speedup, FluxAttn	1.22×	1.18×	1.36×	1.59×	1.67×	1.77×
Decode speedup, PruLong	1.15×	1.23×	1.27×	1.36×	1.43×	1.51×
Decode speedup, FluxAttn	1.19×	1.30×	1.41×	1.53×	**1.77**×	**2.22**×
KV saving, FluxAttn	16.1%	28.9%	32.1%	37.8%	36.2%	39.3%

Table 16: Full quality–latency–memory Pareto comparison at matched granularity controls on Llama-3.1-8B-Instruct; the compact view is Table 15. All methods are measured under the same precision and profiling procedure (Appendix F.5); speedups are relative to the dense FA2 baseline at the same context length. "Persistent KV" is the retained decode-time KV state and "Peak" the peak GPU memory. At 128K and 256K, where Ω_{MSR} is exactly matched at 0.50, FluxAtt improves accuracy over PruLong by 14.76 and 29.39 points, respectively, while reducing decode latency by 19.2% and 32.0% relative to PruLong. Prefill and decode columns report different inference phases and must not be conflated: head-level sparsity is competitive on prefill while remaining behind on per-token decode.

Ctx	**Method / granularity**	**Acc.**	Ω_{MSR}	**Prefill ms (speedup)**	**Decode ms/tok (speedup)**	**Persistent KV GiB (saving)**	**Peak GiB**
8K	Dense / full	92.88	0	985.0 (1.00×)	25.17 (1.00×)	0.89 (0.0%)	16.95
	PruLong / head	86.96	0.500	926.2 (1.06×)	21.88 (1.15×)	0.89 (0.0%)	17.03
	FluxAtt / layer	90.11	0.531	807.4 (1.22×)	21.15 (1.19×)	0.75 (16.1%)	16.89
16K	Dense / full	92.83	0	2280.1 (1.00×)	16.31 (1.00×)	1.81 (0.0%)	18.96
	PruLong / head	76.55	0.500	2004.3 (1.14×)	13.26 (1.23×)	1.81 (0.0%)	19.09
	FluxAtt / layer	79.39	0.500	1932.3 (1.18×)	12.55 (1.30×)	1.28 (28.9%)	18.29
32K	Dense / full	89.46	0	5918.8 (1.00×)	19.97 (1.00×)	3.66 (0.0%)	23.04
	PruLong / head	70.65	0.500	4736.6 (1.25×)	15.72 (1.27×)	3.66 (0.0%)	23.27
	FluxAtt / layer	79.22	0.531	4352.1 (1.36×)	14.16 (1.41×)	2.49 (32.1%)	21.46
64K	Dense / full	70.79	0	17201.4 (1.00×)	20.82 (1.00×)	7.39 (0.0%)	31.07
	PruLong / head	54.52	0.500	12146.8 (1.42×)	15.31 (1.36×)	7.39 (0.0%)	31.51
	FluxAtt / layer	56.08	0.531	10818.5 (1.59×)	13.61 (1.53×)	4.60 (37.8%)	27.69
128K	Dense / full	80.12	0	54991.2 (1.00×)	32.42 (1.00×)	14.69 (0.0%)	47.21
	PruLong / head	48.18	0.500	34321.8 (1.60×)	22.67 (1.43×)	14.69 (0.0%)	48.08
	FluxAtt / layer	62.94	0.500	32928.9 (1.67×)	18.32 (1.77×)	9.38 (36.2%)	40.22
256K	Dense / full	72.34	0	190731.7 (1.00×)	54.57 (1.00×)	28.79 (0.0%)	79.49
	PruLong / head	30.00	0.500	108592.4 (1.76×)	36.14 (1.51×)	28.79 (0.0%)	81.21
	FluxAtt / layer	59.39	0.500	107758.0 (1.77×)	24.58 (2.22×)	17.47 (39.3%)	65.28

distribution. Rather than maintaining distinct task boundaries, the sparsity trajectories fail to clearly diverge after the initial phase, naturally converging toward a shared target sparsity. This results in a more homogenized routing strategy tailored to the specific domain it was exposed to.

This analysis yields an important insight into the training dynamics of the Layer Router: the router intrinsically aligns its allocation strategy with the global optimization landscape provided by the training data. Therefore, to train a general-purpose model capable of fine-grained, context-aware sparsification across diverse tasks, constructing a balanced task mixture during training is the optimal and highly effective practice.

Table 17: Performance and efficiency comparison between Elastic Attention and Flux Attention on Llama-3.1-8B-Instruct. Flux Attention operates at a substantially lower sparsity level (51% vs. 68%), i.e., it retains more attention computation, and still achieves a comparable prefill/TTFT speedup (2.5× vs. 2.6×) thanks to its hardware-friendly layer-wise routing. The reported speedups are prefill speedups; the decode-phase comparison is given in Table 15. Retaining more attention computation allows Flux Attention to deliver superior performance on the rigorous RULER benchmark while maintaining competitive results on LongBench.

Method	**Granularity**	**Sparsity**	**Performance**		**Speedup**
			LongBench	**RULER**	
Elastic Attention	Head-wise	68%	**53.35**	72.85	**2.6**×
Flux Attention (Ours)	**Layer-wise**	**51%**	52.28	**76.75**	2.5×

Table 18: Comprehensive performance comparison among Dense Attention, Elastic Attention, and our Flux Attention across various foundation models. Flux Attention effectively mitigates the severe performance degradation seen in Elastic Attention on the RULER benchmark while maintaining highly competitive results on LongBench.

Model	**Method**	**Performance**	
		LongBench	**RULER**
Qwen3-4B	Dense (Full Attention)	48.45	66.00
	Elastic Attention	48.08	61.81
	Flux Attention (Ours)	**48.72**	**66.95**
Qwen3-8B	Dense (Full Attention)	52.16	75.74
	Elastic Attention	51.51	71.74
	Flux Attention (Ours)	**52.18**	**73.03**
Llama-3.1-8B-Instruct	Dense (Full Attention)	53.28	83.47
	Elastic Attention	**53.35**	72.85
	Flux Attention (Ours)	52.28	**76.75**

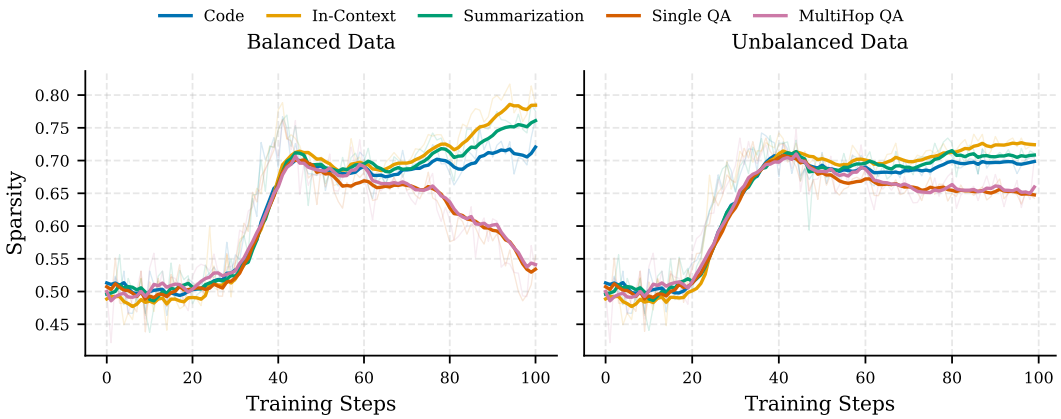


Figure 7: Evolution of sparsity levels across training steps under different data distributions. **Left:** Training on a well-balanced dataset, where the router successfully disentangles tasks into distinct sparsity levels. **Right:** Training on an unbalanced dataset dominated by context-holistic tasks, leading to homogenized routing.

H.2 Impact of Input Truncation on Task Identification

To optimize the trade-off between routing efficiency and accuracy, we investigate the sensitivity of the layer router to the input sequence length. Specifically, we analyze how varying the truncation budget influences the capacity of the router to distinguish between task types and allocate appropriate sparsity patterns. Figure 8 illustrates the performance and sparsity trends as the pooling window expands from 50 tokens (boundary-only) to the full sequence.

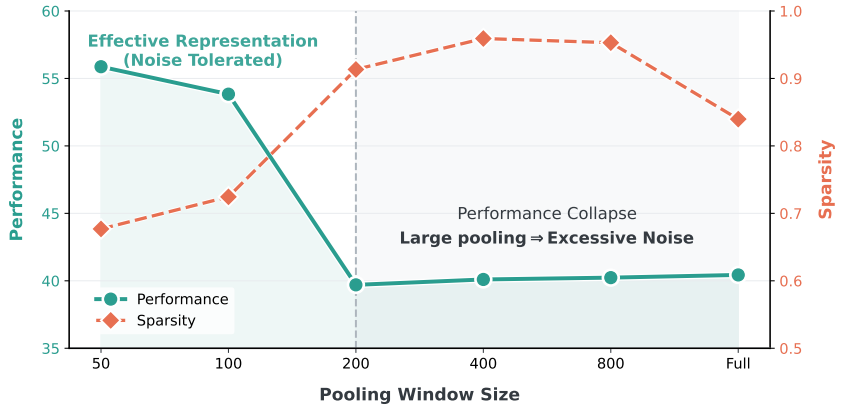


Figure 8: Impact of pooling window size on downstream performance and routing sparsity (Ω_{MSR}). We evaluate varying truncation budgets ($L \in \{50, 100, 200, 400, 800, \text{Full}\}$), retaining only the sequence boundaries (prefix and suffix). Increasing the pooling size beyond 100 tokens introduces context noise, which disrupts the routing mechanism. Consequently, the router misclassifies task features and assigns excessive sparsity to retrieval-intensive tasks, thereby degrading the overall performance.

Our default strategy extracts only the first and last 100 tokens. This design leverages the structure of long-context prompts, where task-defining instructions typically appear at the beginning of the sequence, and specific user queries are appended at the end. The intermediate content primarily consists of raw context. Although this context is necessary for generation, it acts as noise during the routing process, which focuses on macro-level task identification.

Contrary to the assumption that additional context improves routing, Figure 8 demonstrates a drop in performance when the pooling size exceeds 100 tokens. We attribute this phenomenon to the limited capacity of the lightweight MLP within the routing module. As the pooling window expands, the task identification signals are diluted by the document tokens. The MLP struggles to filter out this noise and fails to capture the semantic features necessary for classification. Consequently, the router makes suboptimal decisions, such as assigning high sparsity levels (> 0.9) to retrieval-intensive tasks that require denser attention. This misallocation causes the observed decrease in the quality of generation. These findings support the choice of a 100-token boundary window to maintain an optimal signal-to-noise ratio and facilitate accurate feature extraction.

H.3 Relation Between Offline Entropy and Learned Routing

Section 2.3 motivates layer-level routing with an offline entropy score E_ℓ (Section D.1), which raises a natural question: does the trained router simply reproduce that static ranking? We measure this directly rather than assuming it.

E_ℓ is task-independent and computed once from a trace-normalized hidden-state Gram matrix using its top-128 eigenvalues on a profiling split of 26 disjoint, task-balanced LongBench-E samples. We then collect both soft and hard routing decisions over 20 samples from each of 13 LongBench-E tasks (260 samples in total, 32 layers) and correlate E_ℓ with each task's mean soft FA probability across layers. Table 19 reports the result. The correlations are weak to moderate and task-dependent, ranging from $+0.027$ (Qasper) to -0.314 (MultiNews), with a macro-average Spearman correlation of -0.215 and a macro-average Pearson correlation of -0.108. The macro-average Spearman correlation is stable to the eigenvalue-truncation threshold (-0.231, -0.234, -0.284, -0.215 for $K = 16, 32, 64, 128$). The interpretation we adopt is deliberately limited: entropy is an offline indicator that layer sparsifiability is heterogeneous, not a routing label and not a supervision target. The router is trained from contextual representations and the task objective through the differentiable routing mechanism, so it remains free to depart from the static ordering, and it does so. We considered instead supervising the router directly with the static entropy ranking and chose not to: doing so would force a fixed layer ordering and could not express the input-dependent behavior observed above, where the same layer carries a near-zero FA probability on one task and a near-one probability on another.

The modest aggregate correlation is also not evidence that the router ignores the backbone's structure. Spearman correlation measures whether the layer *ordering* is preserved, not the magnitude of the routing change: the mean cross-task correlation between routing patterns is 0.918, so a shared architectural prior is retained, while a subset of layers moves substantially with the task. Layer 11, for instance, has a soft FA probability of 0.000 on Qasper

Table 19: Corrected correlation between the task-independent offline entropy score E_ℓ (Appendix D.1) and the learned soft FA probability of each layer, on Llama-3.1-8B-Instruct. E_ℓ is computed once from a trace-normalized hidden-state Gram matrix using its top-128 eigenvalues on a profiling split of 26 disjoint task-balanced LongBench-E samples; routing is collected over 20 samples from each of 13 LongBench-E tasks (260 samples total) across 32 layers. Correlations are weak to moderate and task-dependent, so the learned router departs from a static entropy ranking rather than reproducing it; entropy is not used as a routing label or supervision target. The macro-average Pearson correlation is $r = -0.108$, and the macro-average Spearman correlation is stable to the eigenvalue-truncation threshold (-0.231, -0.234, -0.284, -0.215 for $K = 16, 32, 64, 128$).

Task	**Spearman ρ**	**Mean FA**
Qasper	$+0.027$	35.2%
MultiFieldQA	-0.250	44.2%
HotpotQA	-0.205	38.4%
2WikiMQA	-0.167	39.6%
GovReport	-0.293	57.5%
MultiNews	-0.314	57.7%
TREC	-0.129	41.7%
TriviaQA	-0.175	50.0%
SAMSum	-0.237	61.9%
PassageCount	-0.278	38.6%
PassageRetrieval	-0.239	46.8%
LCC	-0.285	65.4%
RepoBench-P	-0.246	65.8%
Macro average	-0.215	49.4%

but 0.979 on SAMSum and 0.984 on RepoBench-P, and the overall FA share ranges from 35.2% to 65.8% across the 13 tasks. Large behavioral adaptation can therefore occur without completely reordering the layers.

H.4 Routing Across Conversation Turns

Flux Attention routes once per prefill and reuses the cached decision for the whole decode trajectory, so a natural concern is whether the assignment becomes stale in agentic or multi-turn use, where the context can shift considerably during generation. Our design refreshes the route at the natural turn boundary: when the dialogue is extended, the enlarged history is re-prefilled and the router runs again, so the cached assignment is fixed only within one decode trajectory rather than globally. Within a response we deliberately do not reroute per token, because per-token rerouting would add control overhead and would break the homogeneous per-layer execution that produces the systems benefit, while the underlying task intent is usually stable over a single response.

To test that the refresh is substantive rather than nominal, we evaluated 80 two-turn MT-Bench conversations. After the first-turn interaction is appended to the context, 92.5% of conversations changed at least one routing decision, and the average sparse-layer ratio moved from 45.8% to 59.4% (+13.6 points). The cached route is therefore trajectory-local: stable within a response, adaptive at turn boundaries.

H.5 Composability with Fine-Grained Sparse Branches

Flux Attention selects the execution mode of a whole layer and does not constrain how the sparse branch is implemented. To check that this abstraction is usable rather than merely nominal, we integrated a vAttention-style [9] per-head sparse branch with Llama-3.1-8B-Instruct on an H20 GPU without modifying the router interface. The routed output stays close to full attention, with a cosine similarity of 0.984.

This result establishes functional composability at the attention interface, and we deliberately do not overstate it: it verifies integration and output agreement, not the wall-clock benefit of a fully optimized vAttention kernel under layer routing, because vAttention itself operates at a finer granularity than the layer. The relationship is complementary—Flux Attention chooses whether a layer runs FA or SA, while a fine-grained operator such as vAttention can then allocate sparsity inside the selected sparse branch. The same argument applies to the block-level methods we discuss in Section B: Sketch-and-Walk [24] and PBS-Attn [47] determine which token blocks are attended within a layer and could serve as the sparse branch in principle, which we describe as architectural compatibility rather than measured integration. Our measured wall-clock results cover SSA, XAttention, and Triangle Attention.

H.6 Loss Curves and Performance Metrics

We examine the training stability and dynamic routing behavior of Flux Attention by visualizing the optimization dynamics in Figure 10. This analysis decomposes the training process into the primary language modeling loss, the sparsity regularization loss, the evolution of the routed sparsity metric (Ω_{MSR}), and the adaptive coefficients (λ).

Optimization Stability. As shown in Figures 10a and 10b, the joint optimization of the language modeling objective and Layer Router parameters remains stable. The LM loss decreases rapidly and plateaus around 1.8, suggesting that the lightweight Layer Router and the introduced sparsity do not impede convergence. Meanwhile, the sparsity regularization loss drops significantly within the first 100 steps. This indicates that the continuous relaxation scheme via Gumbel-Softmax effectively guides the router toward the specified sparsity constraints.

Differentiation in Flux Attention Allocation. Figure 10c provides empirical support for our motivation in Section 1, showing that downstream tasks exhibit varying sensitivities to attention sparsity. Starting from a neutral initialization, the Layer Router learns to differentiate between task types automatically. Retrieval-intensive tasks converge to lower Ω_{MSR} values, i.e., they retain a larger allocation of Full Attention to preserve performance. In contrast, context-holistic tasks stabilize at higher values near the target threshold. Recall that under the convention of Equation (3), a higher Ω_{MSR} means more Sparse Attention and less Full Attention. This confirms that Flux Attention identifies tasks capable of tolerating higher sparsity, thereby improving inference throughput without redundant computation.

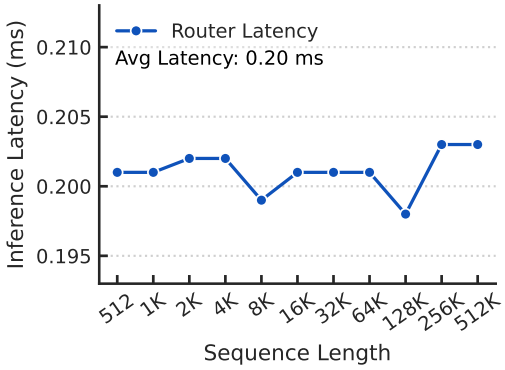
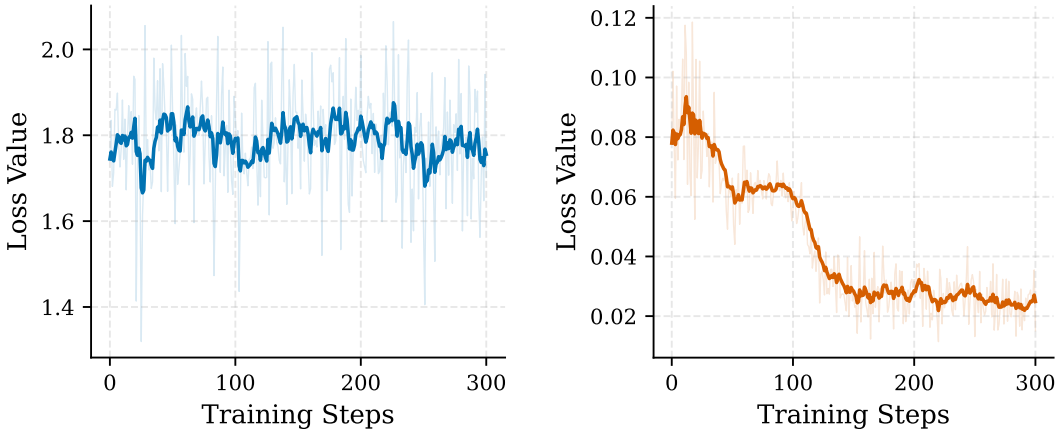


Figure 9: **Router latency analysis.** The router incurs negligible overhead (avg. 0.20 ms). Our design ensures length-invariant stability, maintaining constant speed from 512 to 1M tokens.

Adaptive Coefficients. Figure 10d tracks the evolution of the Lagrangian multipliers (λ), which dynamically scale the penalty for sparsity violations. We observe that λ increases most aggressively for certain tasks, suggesting the model prioritizes meeting density requirements where necessary. This adaptive mechanism balances the trade-off between computational cost and model quality automatically, eliminating the need for manual, task-specific tuning.

I Error Analysis

In Table 11, 12, and 13, we present representative model outputs comparing our method with other baselines. Due to the extensive length of the contexts, only a partial input context is shown. We observe that the primary source of performance improvement stems from our method's ability to accurately identify and respond to the key contextual segments relevant to the query.

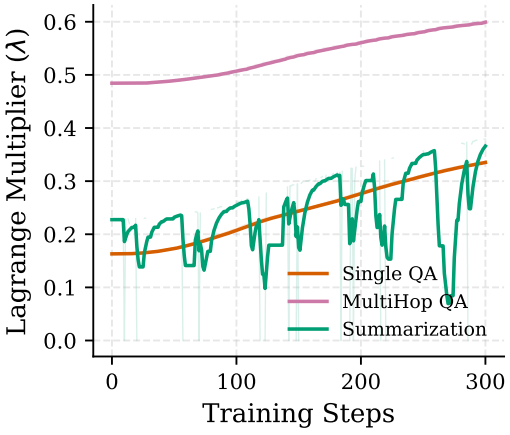


(a) Language Modeling Loss

(b) Sparsity Regularization Loss



(c) Ω_{MSR} during the training process



(d) Adaptive Coefficients (λ)

Figure 10: Decomposition of Training Objectives for Flux Attention. We visualize the training dynamics of the Layer Router, separating the total loss into (a) the primary language modeling objective and (b) the sparsity regularization term. Subfigures (c) and (d) illustrate the task-level differentiation in sparsity allocation (Ω_{MSR}) and adaptive coefficients (λ), demonstrating how the model automatically distinguishes between context-holistic and retrieval-intensive tasks.

Case 1: Reading Comprehension & Information Extraction

Context:

"Low-cost solutions can give billions access to modern cooking by 2030, but the world is failing to deliver... Nearly one in three people around the world still cook their meals over open fires or on basic stoves, resulting in significant damage to health, living standards and gender equality – and yet this challenge can be overcome this decade through a relatively modest amount of investment... Today, 2.3 billion people rely on charcoal, firewood, coal, agricultural waste and animal dung as fuel to prepare meals, causing them to breathe in harmful smoke in the process..."

Question:

Based on the information provided in these news articles, which of the following conclusions regarding African clean cooking is the most reliable?

Correct Prediction (Ours / Ground Truth): ✓

Option B

Content: **In comparison to other regions of the world, the issue of outdated cooking methods in Africa is particularly severe**: nearly four-fifths of the African population still rely on traditional stoves or open fires for cooking, a figure that stands at less than one half globally.

Incorrect Baselines:

Base Model (Qwen3-4B), PruLong, DuoAttention, Triangle:

Pred: Option A

Content: **African nations broadly benefit from carbon markets, particularly in addressing financing issues for clean cooking**: Carbon credits can bridge the funding gap for clean cooking investments in Africa, while also enhancing the affordability of clean cookstoves and fuels.

(Error: Baselines hallucinate or improperly infer information about carbon markets, failing to ground their conclusion strictly in the provided text's severity statistics.)

Figure 11: **Comparison on a long-context reading comprehension task.** Our model accurately extracts and verifies the severity statistics of outdated cooking methods in Africa compared to global figures, while all baselines consistently fall for the same unsupported distractor regarding carbon markets.

Case 2: Core Argument Identification in Abstract Text

Context:
"The Lawyer as Friend: The Moral Foundations of the Lawyer-Client Relation' Charles Friedt Advocatus sed non ladro, Res miranda populo .... Medieval anthem honoring St. Ives
Can a good lawyer be a good person? The question troubles lawyers and law students alike. They are troubled by the demands of loyalty to one's client and by the fact that one can win approval as a good, maybe even great, lawyer even though that loyalty is engrossed by over-privileged or positively distasteful clients. How, they ask, is such loyalty compatible with that devotion to the common good characteristic of high moral principles? And whatever their views of the common good, they are troubled because the willingness of lawyers to help their clients use the law to the prejudice of the weak or the innocent seems morally corrupt..."

Question:
What is the core argument of this article?

Correct Prediction (Ours / Ground Truth): ✓
Option C
Content: **Refuting the social doubts about lawyers' professional ethics through analogy.**

Incorrect Baselines:

Base Model (Llama3.1-8B), DuoAttention, Triangle:
Pred: Option B Content: **A good lawyer can be a good person.**
(Error: Baselines fail to abstract the overarching argumentative framework, instead parroting the literal opening rhetorical question as the core argument itself.)

PruLong:
Pred: Option A Content: **Lawyers should be regarded as friends of clients.**
(Error: The model superficially extracts the title text ("The Lawyer as Friend") without understanding that the "friend" concept is merely the analogy used to resolve the broader ethical doubts.)

Figure 12: **Qualitative comparison on identifying the core argument in a philosophical legal text.** Our model successfully synthesizes the text to identify the underlying argumentative strategy (refutation via analogy), whereas baselines are easily distracted by literal sentences from the title and opening hook.

Case 3: Methodology Extraction from Academic Paper

Context:
"Published as a conference paper at ICLR 2024 MAGICDRIVE: STREET VIEW GENERATION WITH DIVERSE 3D GEOMETRY CONTROL ABSTRACT Recent advancements in diffusion models have significantly enhanced the data synthesis with 2D control. Yet, precise 3D control in street view generation, crucial for 3D perception tasks, remains elusive. Specifically, utilizing Bird's-Eye View (BEV) as the primary condition often leads to challenges in geometry control (e.g., height), affecting the representation of object shapes, occlusion patterns, and road surface elevations... In this paper, we introduce MAGICDRIVE, a novel street view generation framework, offering diverse 3D geometry controls including camera poses, road maps, and 3D bounding boxes, together with textual descriptions, achieved through tailored encoding strategies..."

Question:
How does the MagicDrive encode the bounding box in training step?

Correct Prediction (Ours / Ground Truth): ✓
Option D
Content: **MagicDrive encode the class labels and the corner points separately, and then uses an MLP to compress them into a vector.**

Incorrect Baselines:

Base Model (Qwen3-8B), PruLong, DuoAttention, Triangle:
Pred: Option B Content: **MagicDrive uses Fourier embedding for 4 corner point and passes them through an MLP for encoding in training step. MagicDrive then uses an MLP to compress both the class label and position embeddings into a single hidden vector for each bounding box.**
(Error: Baselines hallucinate or misattribute specific technical details, such as "Fourier embedding for 4 corner points," failing to accurately extract the exact encoding mechanism described in the paper.)

Figure 13: **Qualitative comparison on extracting technical methodology from a machine learning paper.** Our model accurately identifies the specific bounding box encoding strategy, whereas all baselines suffer from hallucination, confidently generating plausible but incorrect architectural details (Fourier embeddings) not supported by the text.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have properly made the main claims in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Yes, we include the limitations in the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: See sections in the Method and Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.

- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide detailed experiments setup in the Experiments section, we also have detailed algorithm in Appendix for Reproducibility.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code is provided in the supplementary material.

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide the details of the implementation.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Inference with greedy decoding experiments do not have any randomness.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We give the computing requirements in the Experiment Setup and Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes, we do.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We have included the broader impacts in the Conclusion.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not release models or datasets with a high risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Yes, we provide proper citations for all resources used and strictly adhere to their respective licenses.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide comprehensive documentation for our new assets in the Appendix and supplementary materials.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This paper does not study human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: We are not subject to the approval of IRB.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: We do not use LLMs.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.