ICML 2026

Elastic Attention:

Test-time Adaptive Sparsity Ratios for Efficient Transformers

Zecheng Tang, Quantong Qiu, Yi Yang, Zhiyi Hong, Haiya Xiang,

Kebin Liu, Qingqing Dang, Juntao Li, Min Zhang

Soochow University, China | LCM Laboratory | Baidu Inc, China

Correspondence: Juntao Li <ljt@suda.edu.cn>

OVERVIEW

Contents

01

Motivation

Why do we need adaptive sparsity in attention mechanisms?

The Quadratic Complexity Bottleneck

The Problem

Standard **Full Attention (FA)** has **O(n^2)** complexity, posing a significant scalability scalability bottleneck as context windows expand.

Sparse Attention (SA) reduces cost by attending to subsets of tokens, but may degrade performance.

Full Attention (FA)

$$o_r = Softmax(QK^\top)V$$

Sparse Attention (SA)

$$o_s = Softmax(Q\tilde{K}^\top)\tilde{V}$$

Hybrid strategies combine SA and FA but use **static ratios**, failing to adapt to varying task sparsity sensitivities during inference.

Hybrid Head Mechanism

$$o^{(\ell)} = Concat[o^{(\ell,1)}, \dots, o^{(\ell,H)}]$$

Each head is assigned a computation type: $\pi^{(\ell,h)} \in \{FA, SA\}$

Output depends on the assigned mode:

$o^{(\ell,h)} = o_r$ if $\pi^{(\ell,h)} = FA$

$o^{(\ell,h)} = o_s$ if $\pi^{(\ell,h)} = SA$

Key Limitation: The sparsity ratio is **fixed at inference time**, and its relationship with downstream task performance remains unexplored.

Key Observation: Two Task Regimes

Sparsity-Robust Tasks

e.g., Summarization, Code Completion

Performance remains **stable** across a wide range of sparsity levels. These tasks can be solved using only **coarse-grained** contextual information.

Performance curve: nearly flat as Ω_{MSR} increases

Sparsity-Sensitive Tasks

e.g., Question Answering, Multi-hop Reasoning

Shows **sharp decline** once sparsity exceeds a threshold. Require retrieving **fine-grained evidence** from context for accurate outputs.

Performance curve: steep drop beyond sparsity threshold

Core Insight

A model should only **differentiate between these two task regimes** and allocate appropriate sparsity, instead of painstakingly learning distinct sparsity configurations for each individual task.

Experimental basis: Llama3.1-8B-Instruct on LongBench (6 tasks). Retrieval heads ranked by activation frequency, progressively replaced with sparse heads to increase Ω_{MSR}. Results show clear task categorization.

Definitions: Sparsity Ratios

Definition 2.1

Model Sparsity Ratio (Ω_{MSR})

$$\Omega_{MSR}(f_\theta) = \frac{1}{H \cdot L} \sum_{h=1}^{H} \sum_{\ell=1}^{L} I[\pi^{(\ell,h)} = SA]$$

Measures the **fraction of sparse attention heads** in the model. $I[\cdot]$ is the indicator function. Ranges from 0 (all FA) to 1 (all SA).

Definition 2.2

Effective Sparsity Ratio (Ω_{ESR})

$$\Omega_{ESR}(f_\theta) = \frac{1}{H \cdot L} \sum_{h=1}^{H} \sum_{\ell=1}^{L} \rho^{(\ell,h)}$$

Measures **effective sparsity** considering both sparse heads and their pruning ratios $\rho^{(\ell,h)} \in [0,1)$. $\rho = 0$ for FA heads, $\rho = \rho_{SA}$ for SA heads.

Research Gap

Existing hybrid head mechanisms fix Ω_{MSR} at inference time. The relationship between sparsity ratio and downstream task performance was **previously unexplored**.

Note: Different SA strategies correspond to specific ρ_{SA}. For example, if SA attends to only 10% of tokens, then $\rho_{SA} = 0.9$ (90% of tokens are pruned). The gap between Ω_{MSR} and Ω_{ESR} reveals that head-level sparsity does not directly translate to token-level computational savings.

02

Method

Elastic Attention – dynamically adjusting sparsity via Attention Router

Elastic Attention Architecture

Three core components: (a) adapted model block with **frozen backbone**, (b) dynamic head assignment via Attention Router, (c) lightweight router design. The router operates analogously to **MoE gating**.

(a) Adapted Model Block

- Backbone parameters are **frozen**
- Only router parameters trained
- Elastic Attention module inserted
- Feed Forward layers unchanged
- Add + Norm preserved

(b) Dynamic Head Assignment

- Input hidden states → Router
- Head-wise routing decision
- Each head assigned FA or SA
- Outputs concatenated
- **Fused kernel** for parallel exec

(c) Attention Router Design

- Task MLP + Router MLP
- Boundary pooling strategy
- Gumbel-Softmax routing
- STE for gradients
- Only **0.27M params/layer**

Information Flow

1. Key hidden states $x_K \in R^{s \times H \times d'}$ fed into router

→ **2.** Pooling + Task MLP → Router MLP produces logits $Z \in R^{H \times 2}$

→ **3.** Gumbel-Softmax → hard binary decision $r_{hard}^{(\ell,h)} \in \{0,1\}$

→ **4.** Head executes FA (r=1) or SA (r=0) → Fused output

Key advantage: Training completes in only **12 hours on 8xA800 GPUs** without modifying backbone model parameters

Attention Router Design

Two-Component Architecture

Task MLP

Infers **task-specific characteristics** from input hidden states. Produces task-aware representations that disentangle features from different task regimes.

Router MLP

Leverages task-aware representations to **determine attention computation mode** (FA or SA) for each head independently.

Processing Pipeline

1. Key states $x_K \in R^{s \times H \times d'}$

2. Boundary pooling: $x'_K \in R^{H \times d'}$ (first 100 + last 100 tokens)

3. Task MLP → Router MLP → logit matrix $z \in R^{H \times 2}$

4. Softmax classifier → binary decision $r_{hard}^{(\ell,h)}$

Why Boundary Pooling?

Full-sequence pooling introduces substantial **redundancy** that degrades task representation quality. Boundary tokens encapsulate critical **system instructions** and **user queries**.

Task Representation Similarity

Before Task MLP: High pairwise cosine similarity across tasks

After Task MLP: Inter-task similarity significantly decreases

---

The Task MLP **effectively disentangles** task-specific features and produces more discriminative representations for routing decisions.

Parameter overhead: Only 0.27M parameters per layer (assuming head dimension of 128), preserving both inference efficiency and computational cost.

Optimization with Gumbel-Softmax

Two Key Challenges

(i) How to make softmax-based routing approximate **hard routing decisions** used at inference?

(ii) How to handle **non-differentiability** introduced by hard (argmax) routing decisions?

Solutions

Gumbel-Softmax

Continuous relaxation with **annealing** gradually sharpens routing distribution toward discrete assignments during training.

Straight-Through Estimator

$r_{hard} = r_{soft} + detach(r_{hard} - r_{soft})$

Gradients flow through soft distribution while forward pass uses hard decisions.

Training Objective

Hard routing: $r_{hard}^{(\ell,h)} = arg\ max_{c\in\{0,1\}} r_{soft}^{(h,c)}$

STE trick: $r_{hard} = r_{soft} + stop\_grad(r_{hard} - r_{soft})$

Min-max optimization:

$$max_{\lambda_1,\lambda_2}\ min_\theta\ L_{language} + \lambda_1 L_{diff} + \lambda_2 L_{diff}^2$$

$$L_{language} = CE\_Loss(Y \mid f_\theta(X))$$

$$L_{diff} = \Omega_{MSR}(f_\theta(X)) - t$$

Task-Dependent Target Sparsity

Instead of enforcing fixed **t** per task, we impose **task-dependent non-tight constraints** with predefined bounds. Lagrange multipliers λ_1, λ_2 are trainable and optimized via gradient ascent, decoupling sparsity-performance trade-offs across tasks.

Efficient Fused Kernel Deployment

Challenge: Different attention head types (FA vs. SA) cannot be efficiently parallelized at runtime with standard implementations, causing **memory overhead** and **kernel launch overhead**.

Baseline: Serial Dispatch

Step 1: Split heads by routing decision

$I_{full} = \{h \mid r_h = 0\}, I_{sp} = \{h \mid r_h = 1\}$

Step 2: Separate kernel launches

FlashAttn for FA + SlidingWin for SA

Step 3: Merge results (memory copy)

Ours: Parallel via BSA Kernel

Step 1: Routing metadata passed to kernel

$m = Map(r)$ (lightweight, no tensor split)

Step 2: Unified kernel execution

Thread-block level branching per head type

Result: Single launch, no memory overhead

Key Benefits

No memory overhead from tensor splitting and copying

No kernel launch overhead from multiple invocations

Speedup increases with longer context lengths

GPU scheduler optimally distributes sequence blocks

Focus: Single-GPU deployment without inter-device communication. When sequence length is sufficiently large, parallelism along the sequence dimension dominates execution. The fused kernel yields prefill-time acceleration during inference.

03

Experiments

Evaluation across 3 LLMs and multiple long-context benchmarks

Main Results: Long-context Benchmarks

Models: Qwen3-4B, Qwen3-8B, Llama-3.1-8B-Instruct | **Baselines:** InfLLM-V2, DuoAttention, PruLong | **Configs:** FA-SSA, FA-XA | **Training:** 8xA800 GPUs, 12 hours, frozen backbone

LongBench-E (Qwen3-8B)	Avg Perf.	Ω_MSR	vs Base
Backbone (FA)	48.45	0.00	-
+ InfLLM-V2	42.20	-	-6.25
+ DuoAttention	45.45	0.70	-3.00
+ PruLong	46.05	0.70	-2.40
+ Elastic (FA-SSA)	46.15	0.64	-2.30
+ Elastic (FA-XA)	44.01	0.75	-4.44

RULER (Qwen3-8B)	Avg Perf.	Ω_MSR	vs Base
Backbone (FA)	66.00	0.00	-
+ InfLLM-V2	63.23	-	-2.77
+ DuoAttention	70.10	0.70	+4.10
+ PruLong	73.05	0.70	+7.05
+ Elastic (FA-SSA)	77.41	0.66	+11.41
+ Elastic (FA-XA)	82.34	0.78	+16.34

LongBench-V2 (Qwen3-8B)	Easy	Hard	Ω_MSR
Backbone (FA)	32.67	22.18	0.00
+ DuoAttention	29.32	27.82	0.70
+ PruLong	28.67	25.56	0.70
+ Elastic (FA-SSA)	28.00	31.20	0.70
+ Elastic (FA-XA)	30.77	33.08	0.72

Key Findings

1. Best average performance across all benchmarks
2. Dynamic sparsity allocation: \u03A9_MSR ~0.85 on code, ~0.68 on QA
3. FA-XA excels at extremely long contexts (256K tokens)
4. Only 12h training on 8xA800 GPUs, frozen backbone

Ablation Studies

Task MLP Effectiveness

Task MLP **significantly reduces** inter-task cosine similarity of representations. Before: high pairwise similarity. After: discriminative task features.

MLP Hidden Size

Hidden dims 2x-8x d' tested. **4xd' (default)** provides best parameter-performance trade-off. 8x achieves highest score but with more overhead.

Routing Patterns

Certain heads in **middle-to-higher layers** consistently function as retrieval heads (always FA). Others consistently routed to SA. Some exhibit task-dependent switching.

Target Sparsity t Allocation

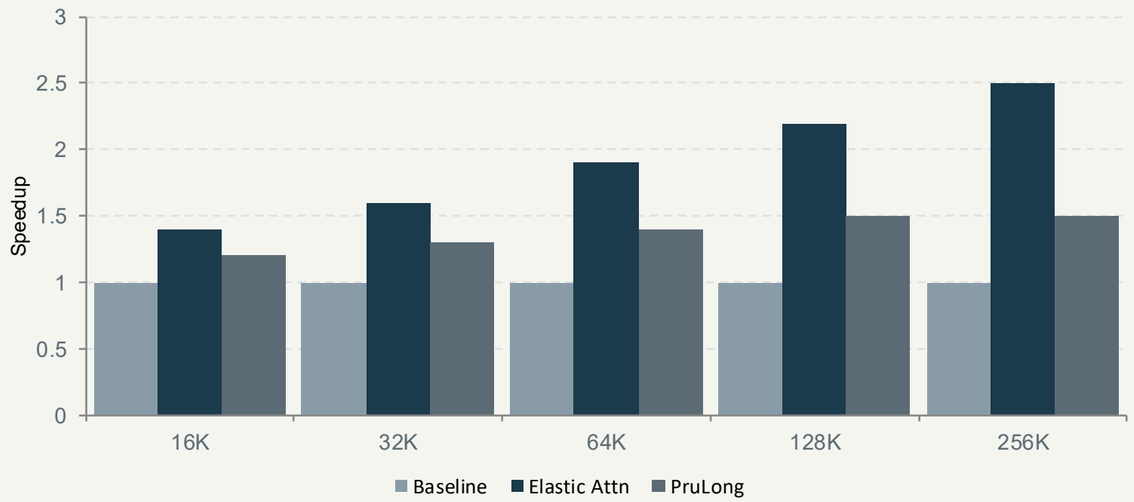
Fixing t_rob = 1.0, varying t_sen from 0.7 to 0.4. Lower t_sen (more FA) can **surpass backbone performance** but with less efficiency. Default t_sen = 0.7 balances performance and efficiency. Non-tight constraints allow model to not strictly match target sparsity.

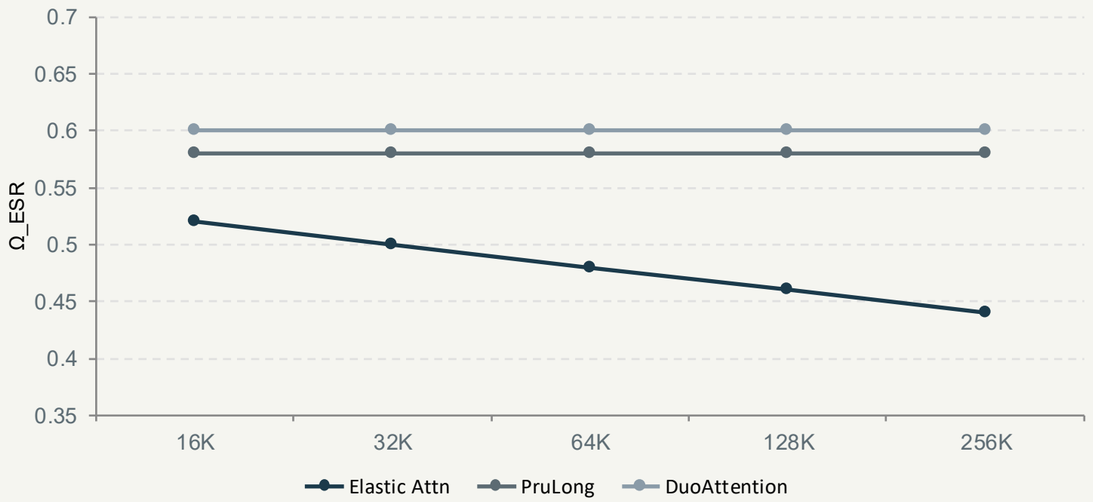
Scalability: XA-SSA Setting

Full SA regime with XA-SSA preserves **strong performance**. Qwen3-4B gap within 1 point across benchmarks. 8B models show degradation but with substantial speedup. Elastic Attention integrates smoothly with LoRA and full fine-tuning.

Tuning Method (Qwen3-4B FA-SSA)	LongBench	RULER	Note
Frozen Backbone (default)	48.45	66.00	Only router trained
+ LoRA	48.08	61.81	Slight decrease
+ Full Fine-tuning	49.05	61.62	Improved LongBench
Backbone (FA only)	50.40	61.97	Upper bound reference

Performance vs. Efficiency Trade-off





Key Observations

1. Inference speedup **increases with longer contexts** as model dynamically allocates higher \u03A9_MSR for longer inputs.
2. Elastic Attention achieves **consistently lower \u03A9_ESR** than training-based hybrid models, indicating superior scalability in effective sparsification.

Comparison with other methods: NSA and InfLLM-V2 impose strict divisibility requirements on KV heads (multiples of 16), incompatible with some models. MoBA and InfLLM-V2 must reserve computation for sequence-level feature pre-computation, causing OOM at 256K context. In contrast, Elastic Attention has **no architectural constraints** and scales gracefully to 256K+ tokens.

04

Conclusion & Future Work

Summary of contributions and research directions

Conclusion

Key Contributions

1 **Discovers two task regimes** - sparsity-robust tasks (summarization) vs. sparsity-sensitive tasks (QA) - enabling principled sparsity allocation.

2 Introduces **head-wise dynamic routing** between FA and SA via a lightweight Attention Router, operating like MoE gating with only 0.27M parameters per layer.

3 Achieves **strong performance** across 3 LLMs and multiple benchmarks with only **12 hours training on 8xA800 GPUs**, without modifying backbone parameters.

4 Brings **negligible overhead** with efficient fused kernel deployment, enabling simultaneous processing of FA and SA heads in a single forward pass.

Core Idea

Elastic Attention enables models to **automatically adjust overall sparsity during the prefill stage** based on input characteristics, accommodating both sparsity-robust and sparsity-sensitive task regimes through data-dependent computation.

Training Cost: 12 hours on 8xA800 GPUs | **Parameters:** 0.27M per layer | **Backbone:** Frozen | **Speedup:** Up to 2.5x at 256K context

Future Directions

On-Device Deployment

Well-suited for **small-to-medium scale LLMs** on single GPU or limited devices. Applications include:

- On-device agent models with extended reasoning
- Document understanding workloads
- Real-time long-context applications

Dynamic Architectures

Motivates **next-generation architectures** beyond sparse attention that elevate dynamic, input-dependent computation to a first-class design principle.

Examples: Titans (Behrouz et al., 2024) and other test-time adaptive mechanisms.

Input-Adaptive Computation

Future models should **adapt computational behavior to individual inputs** rather than adhering to rigid, fully static execution paradigms.

As task complexity and contextual demands vary widely, dynamic allocation becomes essential.

Broader Vision

As LLMs continue to scale in context length and task diversity, the ability to **dynamically allocate computation based on input characteristics** will become a fundamental requirement. Elastic Attention represents a step toward this vision of truly adaptive, efficient, and intelligent language models.

Thank You

Questions & Discussion

Elastic Attention: Test-time Adaptive Sparsity Ratios for Efficient Transformers

ICML 2026, Seoul, South Korea

github.com/LCM-Lab/Elastic-Attention

Zecheng Tang, Quantong Qiu, Yi Yang, Zhiyi Hong, Haiya Xiang, Kebin Liu, Qingqing Dang, Juntao Li, Min Zhang

Soochow University | LCM Laboratory | Baidu Inc