LoGo: Token-Level Dynamic Local-Global Attention

Anonymous authors
Paper under double-blind review

Abstract

As context lengths scale, attention increasingly becomes a primary computational bottleneck in large language models. Standard Transformers remain powerful but computationally inefficient, as they allocate the same attention budget to every token regardless of its contextual demand. Existing local-global hybrids provide a more efficient alternative by mixing restricted- and full-context attention, but they typically allocate span statically across layers or heads. To address these limitations, we propose LoGo, a token-level dynamic local-global attention mechanism that uses attention span as a direct proxy for attention budget allocation. Each LoGo layer contains coupled local and global branches: all tokens receive efficient local attention over a restricted context window, while a learned gate activates global attention with full-context access only for tokens requiring long-range information. A threshold-based budget controller maintains a target global ratio without auxiliary losses, and a progressive masking schedule stabilizes training before sparse routing takes effect. We further implement query-sparse Triton kernels that convert reduced global-attention computation into practical speedups. Extensive experiments validate LoGo's effectiveness, showing that it preserves the scaling behavior of full-attention Transformers across model sizes. In controlled comparisons, LoGo improves over the full-attention Transformer and matched-budget static local-global hybrids, with clear gains on long-range retrieval. Analysis further shows that LoGo learns interpretable span allocation patterns. These results suggest that learned token-level span allocation is an effective and scalable way to improve the long-context performance-compute trade-off.

1 Introduction

Large language models (LLMs) have advanced rapidly through continual improvements in performance under fixed compute budgets. As context lengths scale, attention becomes an increasingly dominant source of computational cost. Yet standard Transformers (Vaswani et al., 2017) allocate this computation uniformly across tokens, despite substantial variation in prediction difficulty and contextual demand. Conditional computation (Bengio et al., 2013) offers a natural way to address this mismatch by allocating compute on demand. In long-context modeling, this motivates dynamic attention-budget allocation across tokens as a path toward more efficient long-context scaling.

The key challenge is how to estimate the attention budget to assign to each token before attention is computed. We argue that attention span is a natural proxy for this budget: it controls both accessible context and cost, and its demand can often be anticipated from linguistic context. Tokens in fixed phrases may be predictable from local evidence, whereas tokens involved in long-range retrieval require global access. However, existing local-global hybrids allocate span statically, either across layers (inter-layer hybrids; (Team et al., 2024; Agarwal et al., 2025)) or across heads (intra-layer hybrids; (Dong et al., 2025)). In both cases, all tokens follow the same span pathway regardless of their contextual needs. Such static layouts impose preset per-token budgets, often wasting global computation on locally predictable tokens. Existing dynamic variants remain limited, as they typically switch among attention mechanisms rather than allocate span directly (Ren et al., 2023), rely on handcrafted routing (Li et al., 2026b;a), or lack matched-budget evaluation at scale (Ainslie et al., 2023). These gaps motivate three questions:

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

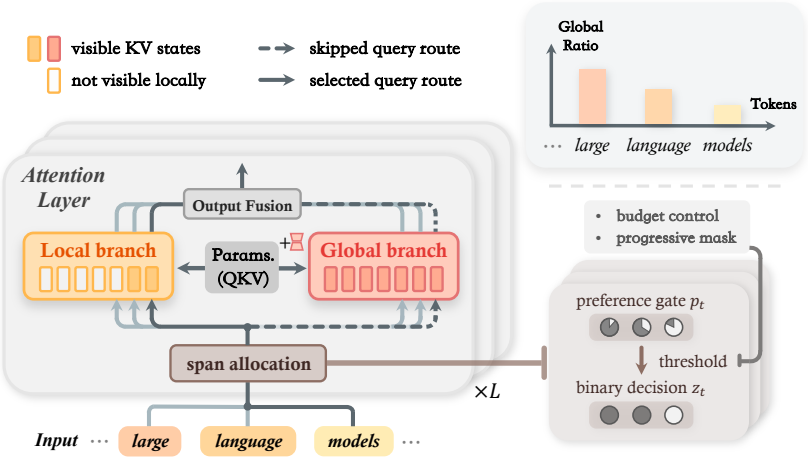

Figure 1: **Overview of LoGo.** Each layer applies local attention to all tokens while activating the global branch only for selected query routes. A token-level gate predicts global-span preference, an adaptive threshold converts this preference into binary routing decisions under a target global ratio, and P-mask gradually introduces sparse routing during training.

RQ1. Can a model learn token-level local/global span allocation, spending global attention on tokens that require long-range context?

RQ2. As a pretrained backbone, can such a model preserve the scaling behavior of standard Transformers?

RQ3. Under matched parameters and FLOPs, can token-level span allocation improve the performance-compute trade-off over static local-global hybrids?

To answer these questions, we propose LoGo (Looking far, Glancing only), a token-level dynamic local-global attention mechanism. Figure 1 illustrates the overall design. Each LoGo layer contains local and global branches with the same attention form but different accessible spans, sharing the main attention parameters while being decoupled by lightweight transformations. A learned scalar gate predicts each token's preference for global span: selected tokens combine local and global outputs, whereas unselected tokens bypass the global branch and remain purely local. The gate is compared with an adaptive threshold that maintains a target global activation ratio, and a progressive masking schedule delays sparse routing until the gate has learned meaningful preferences. Because LoGo sparsifies only global queries while preserving all key-value states, selected tokens retain exact full-context access. We further implement query-sparse Triton kernels that translate reduced global-attention computation into practical speedups.

Extensive experiments validate LoGo's effectiveness. First, scaling runs from 200M to 3.3B parameters show that LoGo consistently matches or improves standard Transformers on language modeling, indicating that token-level span allocation preserves backbone scalability. Second, in controlled 1.5B comparisons under matched parameters and attention FLOPs, LoGo improves the performance-compute trade-off over the standard Transformer and static inter- and intra-layer hybrids: it achieves the best language-modeling results, remains competitive on short-context recall, and provides clear gains on long-range retrieval after context extension. Third, kernel benchmarks show that our implementation converts reduced FLOPs into practical speedups, with near-proportional gains at long sequence lengths. Additional ablations validate the main design choices, show robustness across global ratios, and demonstrate compatibility with static hybrids for reducing both memory and computation. Finally, analysis shows that LoGo learns interpretable allocation patterns, assigning more global span to tokens requiring long-range access.

2 Related Work

Conditional Computation in Transformers. Conditional computation improves the performance-compute trade-off through token-adaptive compute budgets (Bengio et al., 2013; 2015). Existing methods dynamically allocate compute along several axes: experts (Huang et al.,

2024; Yang et al., 2024; Jin et al., 2025), depth (Elbayad et al., 2020; Raposo et al., 2024), and attention via adaptive head selection (Zhang et al., 2022) or token-wise sparse attention budgets (Lin et al., 2026). The closest line dynamically activates attention modules. CoLT5 (Ainslie et al., 2023) routes tokens through light and heavy branches for feed-forward and attention computation, but focuses on encoder-side routing. SMA (Ren et al., 2023) activates a GAU module based on SSM states, but activated tokens still share the same restricted span. Moreover, both are evaluated at limited scale without matched-budget comparisons. LoGo instead targets decoder-only backbones and learns token-level span allocation as a direct proxy for dynamic attention budgeting. We evaluate it at pretrained scale under matched parameters and FLOPs, with emphasis on long-context capability. A broader discussion is provided in Appendix A.1.

Local-Global Hybrid Attention. Local-global hybrid attention reduces long-context cost by combining restricted-span computation with full-context access. Existing designs typically apply the mixture statically across layers, as in inter-layer hybrids (Team et al., 2024; Agarwal et al., 2025; Xiao et al., 2026), or across heads, as in intra-layer hybrids (Dong et al., 2025; Zuo et al., 2025). Some sequence-adaptive hybrids vary attention pathways across positions, but most use fixed predefined patterns (McDermott et al., 2025), while closer methods such as TransMamba (Li et al., 2026b) and Oryx (Li et al., 2026a) rely on handcrafted per-token budgets rather than learned routing. In contrast, LoGo keeps the attention form fixed and learns span allocation end-to-end: each token decides whether to activate full-context attention under a controlled global budget. This enables matched-budget comparison with static hybrids while avoiding unnecessary global computation on locally predictable tokens. A broader discussion is provided in Appendix A.2.

3 METHODS

We present LoGo in three parts. Section 3.1 formalizes global/local attention and the token-level span-allocation objective. Section 3.2 introduces the LoGo attention layer and its three core components. Section 3.3 describes the query-sparse implementation for reduced computation.

3.1 PRELIMINARIES AND PROBLEM FORMULATION

Global and Local Attention. Let $\mathbf{X}^{(\ell)} = [\mathbf{x}_1^{(\ell)}; \ldots; \mathbf{x}_T^{(\ell)}] \in \mathbb{R}^{T\times d}$ denote the input hidden states to the ℓ-th attention layer, and omit ℓ when clear. Standard multi-head attention projects $\mathbf{X}$ into $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ and computes the attention output for each head as:

$$\mathbf{o}_t = \sum_{s\in\mathcal{C}_t} a_{t,s}\mathbf{v}_s = \sum_{s\in\mathcal{C}_t} \frac{\exp(\mathbf{q}_t\mathbf{k}_s^\top/\sqrt{d_h})}{\sum_{j\in\mathcal{C}_t}\exp(\mathbf{q}_t\mathbf{k}_j^\top/\sqrt{d_h})}\mathbf{v}_s, \tag{1}$$

where $\mathcal{C}_t$ denotes the causal context accessible to query position t, and d_h is the head dimension. Global attention (Vaswani et al., 2017) uses the full causal prefix $\mathcal{C}_t^\infty = \{1, \ldots, t\}$, while local attention (Beltagy et al., 2020) restricts context to a fixed window $\mathcal{C}_t^w = \{\max(1, t-w+1), \ldots, t\}$. We denote the corresponding operators by $\mathrm{SoftmaxAttn}_\infty(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ and $\mathrm{SoftmaxAttn}_w(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, respectively. Global attention provides full-context access at $\mathcal{O}(T^2d)$ cost, whereas local attention reduces the cost to $\mathcal{O}(Twd)$ at the price of losing direct access to tokens outside the window.

Static Local-Global Hybrids. Let $z_{t,h}^{(\ell)} \in \{0, 1\}$ indicate whether token t uses global attention at layer ℓ and head h. Static hybrids fix this span decision independently of the token position. Inter-layer hybrids choose $z_{t,h}^{(\ell)}$ by layer, whereas intra-layer hybrids choose it by head. In both cases, $z_{t,h}^{(\ell)} = z_h^{(\ell)}$ for all t. Thus every token follows the same predefined local-global pathway, despite differences in contextual demand.

Token-Level Span Allocation Objective. LoGo learns token-level span allocation jointly with the language-modeling objective, aiming to minimize prediction loss under a prescribed attention-compute budget. At each layer, it learns to predict a binary routing decision:

$$z_t^{(\ell)} = f_\theta^{(\ell)}(\mathbf{x}_t^{(\ell)}), \quad z_t^{(\ell)} \in \{0, 1\}, \tag{2}$$

where $z_t^{(\ell)} = 1$ activates global attention for token t, and $z_t^{(\ell)} = 0$ uses local attention only. All heads in a layer share this token-level decision for efficient implementation. To constrain the attention budget, we define a target global activation ratio $\rho^{(\ell)}$. The realized ratio at layer ℓ is:

$$F^{(\ell)} = \frac{1}{|\mathcal{B}|} \sum_{(b,t)\in\mathcal{B}} z_{b,t}^{(\ell)}, \tag{3}$$

where $\mathcal{B}$ denotes valid token positions in the training batch. LoGo enforces the budget by maintaining $F^{(\ell)} \approx \rho^{(\ell)}$, formulating dynamic attention-budget allocation as a constrained token-level choice between local and global span.

3.2 LoGo Attention

LoGo implements token-level span allocation via three components: coupled local/global attention branches, a token-level global-preference gate, and a budget controller with progressive masking.

Coupled Local and Global Branches. Each LoGo layer contains two branches with the same attention form: a local branch over a short causal window ($w = 128$ in our main experiments) and a global branch over the full causal context. To control parameter size while allowing branch specialization, we first use shared main attention projections to produce the local-branch representations:

$$\mathbf{Q}^L = \mathbf{X}\mathbf{W}_q, \quad \mathbf{K}^L = \mathbf{X}\mathbf{W}_k, \quad \mathbf{V}^L = \mathbf{X}\mathbf{W}_v. \tag{4}$$

For each head h, the global representations are then obtained by applying a lightweight transformation to the corresponding local-branch representations, with $\mathbf{U}_h^G = \phi_{U,h}(\mathbf{U}_h^L)$ for $\mathbf{U} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$. We instantiate $\phi_{U,h}$ with either a linear projection (LP) or a scale-and-offset transformation (SO):

$$\begin{aligned} &\text{LP:} \quad \phi_{U,h}(\mathbf{U}_h^L) = \mathbf{U}_h^L \mathbf{A}_{U,h}, && \mathbf{A}_{U,h} \in \mathbb{R}^{d_h \times d_h}, \\ &\text{SO:} \quad \phi_{U,h}(\mathbf{U}_h^L) = \mathbf{U}_h^L \odot \mathbf{s}_{U,h} + \mathbf{b}_{U,h}, && \mathbf{s}_{U,h}, \mathbf{b}_{U,h} \in \mathbb{R}^{d_h}. \end{aligned} \tag{5}$$

We use LP as the default in LoGo and include SO as an ablation. Both forms are lightweight relative to the main $\mathcal{O}(d^2)$ projections and add roughly 1% parameters in our 1.5B setting. The transformations are initialized as identities, so the branches start from shared representations and specialize during training. The output projection $\mathbf{W}_o$ is shared.

The two branches then compute span-specific attention:

$$\mathbf{O}^L = \mathrm{SoftmaxAttn}_w(\mathbf{Q}^L, \mathbf{K}^L, \mathbf{V}^L), \quad \mathbf{O}^G = \mathrm{SoftmaxAttn}_\infty(\mathbf{Q}^G, \mathbf{K}^G, \mathbf{V}^G), \tag{6}$$

where the global branch will be computed only for selected query positions, as described next.

Token-Level Span Allocation and Output Fusion. For each token, LoGo predicts a scalar global-context preference $p_t^{(\ell)}$ using a gate vector $\mathbf{w}_g^{(\ell)}$. A layer-specific threshold $b_{\text{th}}^{(\ell)}$ converts this continuous preference into a hard global-activation decision $z_t^{(\ell)}$:

$$p_t^{(\ell)} = \mathrm{sigmoid}\left({\mathbf{x}_t^{(\ell)}}^\top \mathbf{w}_g^{(\ell)} \right), \quad z_t^{(\ell)} = \mathbb{I}\left[p_t^{(\ell)} > b_{\text{th}}^{(\ell)} \right]. \tag{7}$$

With near-zero initialization of the gate projection, $p_t^{(\ell)}$ is initially centered around 0.5. Let $\mathcal{I}^{(\ell)} = \{t : z_t^{(\ell)} = 1\}$ denote the selected global-query positions. The local branch covers all tokens, while the global branch is computed only for queries in $\mathcal{I}^{(\ell)}$:

$$\mathbf{O}^G_{\mathcal{I}^{(\ell)}} = \mathrm{SoftmaxAttn}_\infty\left(\mathbf{Q}^G_{\mathcal{I}^{(\ell)}}, \mathbf{K}^G, \mathbf{V}^G \right). \tag{8}$$

LoGo sparsifies only global queries while keeping $\mathbf{K}^G$ and $\mathbf{V}^G$ dense over all valid tokens, so selected queries retain exact full-context access.

The continuous gate also serves as the fusion weight between local and global outputs. Before fusion, branch-specific ContextNorm operators, implemented as head-wise RMSNorm following Sun et al. (2023), normalize the output scale of each branch:

$$\widetilde{\mathbf{O}}^L = \mathrm{Norm}^L(\mathbf{O}^L), \quad \widetilde{\mathbf{O}}^G = \mathrm{Norm}^G(\mathbf{O}^G). \tag{9}$$

This prevents the gate from compensating for branch-wise magnitude mismatch and keeps it focused on learning span preference. The fused attention output is:

$$\mathbf{o}_t = \begin{cases} (1 - p_t^{(\ell)}) \cdot \widetilde{\mathbf{o}}_t^L + p_t^{(\ell)} \cdot \widetilde{\mathbf{o}}_t^G, & z_t^{(\ell)} = 1, \\ \widetilde{\mathbf{o}}_t^L, & z_t^{(\ell)} = 0. \end{cases} \tag{10}$$

Thus, unselected tokens reduce exactly to the local output, while selected tokens form a weighted sum of local and global context. The paired weights explicitly compete over the two spans, unlike additive fusion (Ainslie et al., 2023; Ren et al., 2023).

Budget Control and Progressive Masking. LoGo controls the global-attention budget by updating the activation threshold from batch-level routing statistics. Given the realized activation ratio $F^{(\ell)}$ and target ratio $\rho^{(\ell)}$ at layer ℓ, we update:

$$b_{\text{th}}^{(\ell)} \leftarrow b_{\text{th}}^{(\ell)} + \gamma \, \text{sign}\Big(F^{(\ell)} - \rho^{(\ell)}\Big), \tag{11}$$

where γ is the threshold update rate. If $F^{(\ell)} > \rho^{(\ell)}$, the threshold increases and fewer tokens activate global attention; otherwise, it decreases. This controller enforces the desired attention budget without auxiliary balancing losses or direct modification of the language-modeling objective (Wang et al., 2024). Although layer-specific targets are possible, we use the same target ratio across layers for controlled comparison. For example, $\rho^{(\ell)} = 0.5$ activates global attention for half of the query tokens in each layer, matching the main global-attention compute of a 1:1 static local-global hybrid.

Because an untrained gate cannot reliably estimate span demand, hard routing from the start may block global-branch learning signals for arbitrary tokens. We therefore use a progressive masking schedule (P-mask): $b_{\text{th}}^{(\ell)}$ is initialized to a negative value so that all tokens initially activate global attention since $p_t^{(\ell)} \in (0, 1)$. We set this initial value based on γ and the planned P-mask duration, so that $b_{\text{th}}^{(\ell)}$ reaches the initial gate center (0.5) when P-mask ends. As the threshold moves toward the target operating point, global masking is introduced gradually, allowing the gate and both branches to learn from dense supervision before sparse routing takes effect.

3.3 Efficient Implementation

LoGo reduces attention FLOPs only when token-level sparsity is exposed to the attention implementation. This section summarizes our parallel kernels, decoding strategy, and complexity analysis, with additional details in Appendix B. A naive implementation that computes dense global attention for all tokens and masks outputs would preserve the interface but lose the intended compute benefit. We instead implement the global branch as query-sparse attention: unselected tokens remain in the dense key-value memory but do not instantiate global-query rows, so global-attention cost scales with the number of selected queries.

Query-Sparse Prefill and Training. During prefill and training, the local branch uses standard sliding-window attention. For the global branch, our Triton kernel packs the selected queries into a compacted tensor and computes Eq. 8, skipping pruned query rows. The kernel stores the original sequence positions of compacted queries together with compacted `cu_seqlens`, which are used to enforce causal visibility and partition KV blocks in the forward pass. The backward pass mirrors the same row-sparse structure, writing query gradients only for selected rows and accumulating KV gradients over causally valid selected-query blocks. This preserves dense-attention semantics for selected rows while avoiding computation for pruned ones.

Autoregressive Decoding. During decoding, since LoGo computes the gate before launching global attention, unselected tokens skip the global call, while selected tokens attend over the full global cache and fuse both branches as in Section 3.2. For batched decoding, we use a conservative policy that reuses existing dense decode kernels: the global call is skipped only when the micro-batch contains no selected rows; otherwise, we launch a dense call and mask inactive outputs. A specialized serving kernel could further compact active decode rows as in prefill.

Complexity and Compatibility. Ignoring projections and MLPs, dense causal attention costs $\mathcal{O}(T^2Hd_h)$ per layer, while the local branch costs $\mathcal{O}(TwHd_h)$. Since LoGo computes global attention only for selected query rows, its global-branch cost $\mathcal{O}(\sum_{t\in\mathcal{I}^{(\ell)}} t\,Hd_h)$ scales as $\mathcal{O}(\rho^{(\ell)}T^2Hd_h)$ under target global ratio $\rho^{(\ell)}$. The total attention cost is therefore $\mathcal{O}(TwHd_h+\rho^{(\ell)}T^2Hd_h)$, matching the dominant compute of a static hybrid with the same global fraction, while allocating global computation dynamically across tokens. Memory-wise, LoGo keeps a dense KV cache with $\mathcal{O}(T)$ memory, as in standard global attention, but can be combined with memory-saving hybrids by replacing their global-attention layers with LoGo. Finally, LoGo uses query-row sparsity shared across heads and preserves the original key-value layout, avoiding head-wise load imbalance and remaining compatible with tensor- and sequence-parallel implementations.

4 EXPERIMENTS

Our experiments evaluate LoGo along five axes. First, we test whether token-level span allocation preserves Transformer scaling behavior. Second, in a controlled setting, we compare LoGo with the standard Transformer and matched-budget static local-global hybrids. Third, we verify that the query-sparse implementation translates reduced attention FLOPs into operator-level speedups. Fourth, we ablate the main design choices and test robustness across global ratios and hybrid combinations. Finally, we analyze whether LoGo learns meaningful token-level allocation patterns. Training and evaluation details are provided in Appendix C.1 and Appendix C.2.

4.1 SCALING PROPERTY

We first test whether LoGo preserves Transformer scaling as a pretrained backbone. We train LoGo with target global ratio $\rho^{(\ell)} = 0.5$ and a full-attention Transformer baseline across five model sizes, from 200M to 3.3B parameters. Both families use the same MHA-SwiGLU architecture (Grattafiori et al., 2024) and identical training hyperparameters, with a fixed tokens-per-parameter ratio following the Chinchilla compute-optimal principle (Hoffmann et al., 2022). As shown in Table 1, LoGo preserves Transformer scaling behavior and consistently matches or improves the full-attention baseline across scales. Detailed training setup is provided in Appendix C.1.

Table 1: **Scaling comparison between LoGo and the Transformer baseline.** We report training loss, WikiText and Lambada perplexity, and average Commonsense Reasoning accuracy.

Params.	Model	Train Loss ↓	Wiki. ↓	LMB. ↓	CSR-Avg. ↑
200M	Transformer	2.799	43.590	92.093	38.89
	LoGo	**2.795**	**42.990**	**89.683**	**40.36**
470M	Transformer	2.461	28.015	30.787	44.27
	LoGo	**2.449**	**27.410**	**27.052**	**44.30**
900M	Transformer	2.294	22.037	17.717	47.63
	LoGo	**2.286**	**21.649**	**16.431**	**48.66**
1.5B	Transformer	2.162	18.241	12.025	**52.88**
	LoGo	**2.156**	**18.052**	**11.941**	52.68
3.3B	Transformer	2.000	14.911	8.216	58.11
	LoGo	**1.996**	**14.785**	**8.009**	**58.26**

4.2 COMPARISON WITH STATIC HYBRID PARADIGMS

We next evaluate the performance-compute trade-off by comparing LoGo with standard Transformers and static local-global hybrids under matched parameters and attention FLOPs.

Experimental Setup. We compare four attention paradigms at the 1.5B scale, using the same architecture and training hyperparameters: a standard Transformer, inter-layer and intra-layer local-global hybrids, and LoGo. The hybrid variants use the same local window size $w = 128$ and a matched global ratio of 0.5, corresponding to global span in half of the layers, heads, or tokens. We pretrain all models for 100B tokens at 8k context length and then perform two context-extension stages to 32k and 128k, each for 10B tokens. Detailed setup is provided in Appendix C.1.

Language Modeling. After 100B-token pretraining, we evaluate training loss, evaluation perplexity, and commonsense reasoning. As shown in Table 2, LoGo achieves the best overall results, outperforming both the full-attention Transformer and static hybrid baselines. The same relative ranking holds after 32k context extension (see Appendix C.3, Table 7).

Table 2: **Language modeling and commonsense reasoning comparison.** Attention budget denotes the global-attention ratio. Lower is better for loss and perplexity, and higher is better for accuracy.

Model	Attn	Loss / PPL ↓			Commonsense Reasoning ↑						
	Budget	Loss	Wiki.	LMB.	LMB.	PIQA	Hella.	Wino.	ARC-e	ARC-c	Avg.
Transformer	1.0	2.119	16.12	8.36	53.97	72.14	52.63	56.35	66.84	32.25	55.70
inter-layer-H	0.5	2.121	16.12	8.43	54.59	72.14	53.02	57.30	66.37	33.87	56.22
intra-layer-H	0.5	2.122	17.78	8.48	54.08	72.36	52.72	57.14	66.08	32.17	55.76
LoGo	0.5	**2.112**	**16.04**	**7.50**	56.57	71.76	53.67	57.06	66.20	32.51	**56.30**

Short-Context Recall. We further evaluate the 32k checkpoints on real-world recall tasks with relatively short inputs, typically a few thousand tokens. As shown in Table 3, LoGo remains competitive with the full-attention Transformer and the intra-layer hybrid, while the inter-layer hybrid shows the only noticeable gap.

Table 3: **Short-context recall comparison.** We evaluate the 32k checkpoints. Higher is better.

Model	Attn Budget	FDA	SWDE	SQD	TQA	NQ	Drop	Avg.
Transformer	1.0	81.67	84.70	52.38	62.68	**33.86**	29.80	57.52
inter-layer-H	0.5	78.49	84.34	51.41	62.32	32.34	**29.90**	56.47
intra-layer-H	0.5	83.39	**86.32**	**52.78**	**63.27**	32.34	28.89	**57.83**
LoGo	0.5	**83.85**	86.05	51.94	62.32	33.26	28.46	<u>57.65</u>

Long-Range Recall. To isolate long-range retrieval ability, we evaluate 32k and 128k context-extended checkpoints on needle-style RULER tasks. Table 4 reports average scores over the subtasks. LoGo achieves the best results in both stages, with larger gains at longer sequence lengths. On RULER-32k and RULER-64k, LoGo obtains the best score on five of the eight subtasks. This trend differs from short-context recall: the inter-layer hybrid becomes comparable to the full-attention Transformer, while the intra-layer hybrid exhibits a clear gap.

Table 4: **Long-range recall on needle-style RULER tasks.** We evaluate the 32k and 128k checkpoints on eight subtasks, covering three single-needle and five multi-needle variants. Each entry reports the average score for a train-test length setting; per-task results are in Tables 8 and 9.

Model	32K CT				128K CT		
	8k	16k	32k	Avg.	32k	64k	Avg.
Transformer	89.9	81.9	64.9	78.9	73.3	42.8	58.1
inter-layer-H	90.2	83.9	66.5	80.2	72.1	44.9	58.5
intra-layer-H	90.8	79.5	57.2	75.8	61.3	34.0	47.6
LoGo	**91.9**	**85.2**	**72.0**	**83.0**	**78.3**	**52.5**	**65.4**

Overall, LoGo improves the performance-compute trade-off under matched parameters and attention FLOPs. It maintains competitive language-modeling and short-context recall performance, while providing clear gains on long-range retrieval, supporting the hypothesis that token-level span allocation shifts global computation toward tokens that actually require long-range context.

4.3 Efficiency

We benchmark whether LoGo's query-sparse implementation converts reduced attention FLOPs into practical operator speedups, using a Triton kernel built on the FLA baseline (Yang & Zhang, 2024).

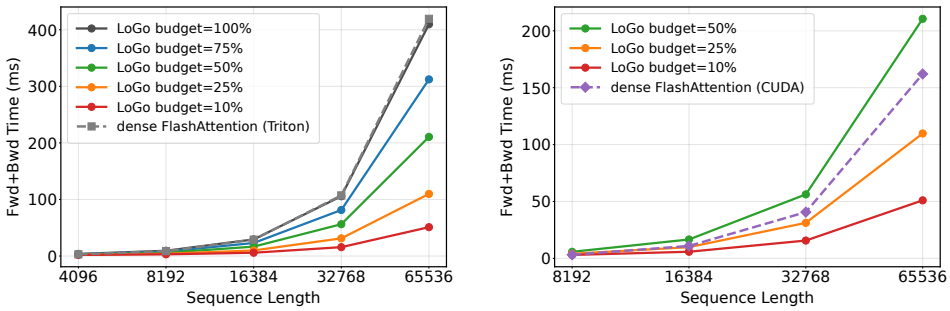

Figure 2: **Operator-level efficiency of query-sparse global attention.** Forward-plus-backward runtime is measured with batch size 1, 8 attention heads, and head dimension 128. *Left*: relative to dense FlashAttention in Triton, speedups scale with query sparsity. *Right*: compared against dense FlashAttention in CUDA, our Triton kernel is faster only at high sparsity, leaving room for a CUDA implementation.

We measure forward-plus-backward runtime from 4k to 64k sequence length, comparing against dense FlashAttention implementations in Triton (the FLA baseline) and CUDA.

As shown in Figure 2(a), the query-sparse kernel introduces little overhead relative to the dense Triton baseline, and its speedups scale with sparsity across all tested lengths. At a 0.5 attention budget, the measured speedup reaches 1.99× at 64k, close to the ideal reduction. Figure 2(b) compares against the dense CUDA baseline. Due to the implementation gap between Triton and CUDA baselines, the current query-sparse kernel is faster only at high sparsity. Overall, these results show that query-side span sparsity can translate proportional FLOPs savings into practical attention speedups.

4.4 Ablation Study

Architecture Ablations. We ablate LoGo's main architectural choices using the 1.5B pretraining setting. All variants follow the same training recipe and differ from LoGo by only one design choice. Table 5 reports the results. For branch parameterization, fully sharing branches hurts performance, while fully decoupling them adds substantial parameters and improves perplexity but not recall. LoGo's lightweight decoupling therefore gives the best trade-off. The remaining ablations show that ContextNorm is critical for stable fusion, weighted fusion is preferable to additive fusion, threshold-based budget control outperforms an auxiliary-loss alternative, and P-mask improves downstream performance. Additional interpretation is provided in Appendix C.4.1.

Table 5: **Architecture ablations.** Rows are grouped by the ablated component. "–" indicates a non-converged run.

Component	Variant	Train Loss ↓	Wiki. ↓	LMB. ↓	CSR-Avg. ↑	Recall-Avg. ↑
–	**LoGo**	2.112	16.04	**7.50**	56.30	**73.74**
Branch param.	shared params	2.116	16.12	7.83	55.99	69.92
	full params	**2.101**	**15.85**	7.71	**56.76**	73.42
	SO transform	2.114	16.07	7.75	56.21	71.33
Normalization	w/o ContextNorm	–	–	–	–	–
	SO w/o ContextNorm	2.121	16.36	8.39	55.43	70.90
Output fusion	additive fusion	2.113	16.14	7.90	56.43	72.52
Budget control	auxiliary loss	2.118	16.18	7.87	56.18	71.73
Training schedule	w/o P-mask	2.112	15.96	7.73	56.08	71.59

Different Local/Global Ratios. We also vary the global ratio and compare LoGo with inter-layer hybrids under matched budgets, using the same comparison setting as Section 4.2. As shown in Tables 10 and 11, LoGo consistently improves short-context recall and RULER-32k performance across ratios, showing robustness to the global budget.

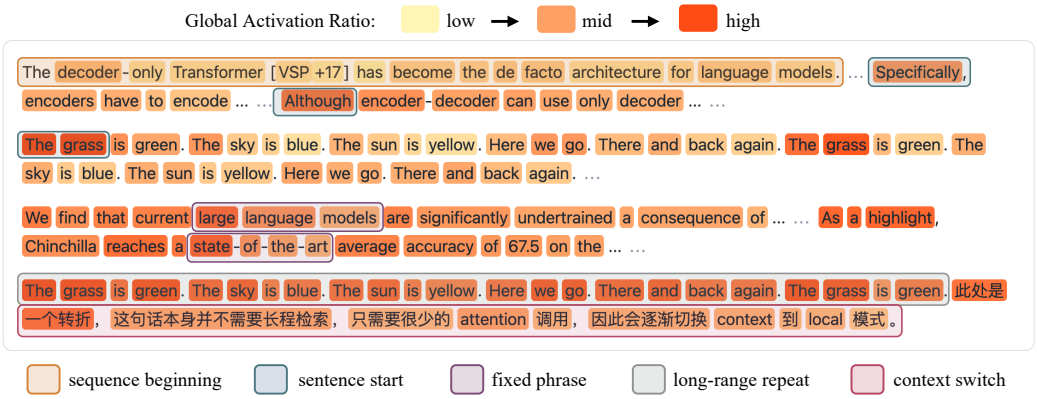

Figure 3: **Word-level span allocation patterns.** Warmer colors indicate more layers activating global attention. Boxed regions mark the five representative patterns in reading order.

Compatibility with Other Hybrids. Finally, we test whether LoGo can complement memory-saving hybrids by replacing the global-attention layers of an inter-layer hybrid with LoGo layers. Table 12 shows that this combination further reduces compute while improving recall, with details in Appendix C.4.1. Thus, LoGo can be combined with static hybrids to reduce both computation and memory cost.

4.5 Analysis

Span Allocation Statistics. We analyze learned span allocation on evaluation sets from four domains: Wiki (general text), Quest (long-context), GSM8K (math), and the LoGo codebase (code). As shown in Appendix C.4.2, the average global ratio stays close to the target 0.5, while allocation remains domain- and token-dependent. Different domains allocate global span at different depths, and tokens within the same domain vary substantially: some use global span in almost no layers or almost all layers, while most fall between these extremes. These patterns indicate that LoGo learns non-uniform span allocation under a controlled global budget.

Word-Level Visualization. We further inspect word-level allocation on a mixed passage with controlled repetitions and a bilingual context switch. Figure 3 highlights five representative patterns: (1) sequence beginnings mostly remain local; (2) sentence starts often receive more global span; (3) fixed phrases shift toward local attention as later words become predictable from nearby context; (4) long-range repeats trigger strong global activation; and (5) context switches quickly reduce global-span usage. These qualitative patterns indicate that LoGo learns to spend global attention on tokens with long-range demand while avoiding global computation for locally predictable tokens. Additional analysis and the full visualization are provided in Appendix C.4.2.

5 Conclusion

We introduced LoGo, a token-level dynamic local-global attention mechanism for improving the long-context performance-compute trade-off. LoGo treats attention span as a direct proxy for attention budget, giving every token efficient local attention while selectively activating global attention under a controlled budget. Its lightweight branch decoupling, threshold-based budget controller, progressive masking schedule, and query-sparse kernels make this routing stable, budget-controlled, and efficient. Experiments show that LoGo preserves Transformer scaling behavior and improves over full-attention and static hybrid baselines under matched parameters and computation, with clear gains on long-range retrieval. Ablations and analyses further show that the learned span allocation is robust, interpretable, and compatible with static hybrids. These results suggest that learned token-level span allocation is a scalable way to spend global attention where it is most useful.

AI USE STATEMENT

We used generative AI tools to assist with English writing, translation from internal drafts, language polishing, and LaTeX/table formatting. We did not use generative AI tools to formulate the research idea, design the methodology or experiments, implement the proposed method, generate experimental results, or interpret the results. All AI-assisted content was reviewed and edited by the authors, who take full responsibility for the final submission.

ETHICS STATEMENT

This work proposes an efficient attention mechanism for language model pretraining and does not involve human-subject studies, collection of personal data, or release of new datasets. Potential risks are those associated with improving language models in general, and we encourage responsible use consistent with standard safety and privacy practices.

REPRODUCIBILITY STATEMENT

We provide the model formulation and training procedure in Section 3, implementation details in Appendix B, and training, evaluation, and ablation details in Appendix C. We will provide anonymized supplementary code containing the full LoGo model implementation, configurations, tests, and query-sparse Triton kernels.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David C Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, et al. Colt5: Faster long-range transformers with conditional computation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 5085–5100, 2023.

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *Proceedings of the VLDB Endowment*, 17(2):92–105, 2023.

Simran Arora, Aman Timalsina, Aaryan Singhal, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. Just read twice: closing the recall gap for recurrent language models. In *Workshop on Efficient Systems for Foundation Models II@ ICML2024*, 2024.

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

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 2978–2988, 2019.

Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, ZIJIA CHEN, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Min-Hung Chen, Yoshi Suhara, Yingyan Celine Lin, et al. Hymba: A hybrid-head architecture for small language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, 2019.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020.

Tianyu Fu, Haofeng Huang, Xuefei Ning, Genghan Zhang, Boju Chen, Tianqi Wu, Hongyi Wang, Zixiao Huang, Shiyao Li, Shengen Yan, et al. Mixture of attention spans: Optimizing llm inference efficiency with heterogeneous sliding-window lengths. *arXiv preprint arXiv:2406.14909*, 2024.

Tianyu Fu, Yichen You, Zekai Chen, Guohao Dai, Huazhong Yang, and Yu Wang. Think-at-hard: Selective latent iterations to improve reasoning language models. *arXiv preprint arXiv:2511.08577*, 2025.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021.

Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.16712*, 2024.

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.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pp. 30016–30030, 2022.

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.

Quzhe Huang, Zhenwei An, Nan Zhuang, Mingxu Tao, Chen Zhang, Yang Jin, Kun Xu, Liwei Chen, Songfang Huang, and Yansong Feng. Harder task needs more experts: Dynamic routing in moe models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12883–12895, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL `https://arxiv.org/abs/2310.06825`.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng YAN. Moe++: Accelerating mixture-of-experts methods with zero-computation experts. In *The Thirteenth International Conference on Learning Representations*, 2025.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. Minimax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025.

Kevin Y Li, Asher Trockman, Ananda Theertha Suresh, and Ziteng Sun. Multi-mixer models: Flexible sequence modeling with shared representations. *arXiv preprint arXiv:2605.28769*, 2026a.

Yixing Li, Ruobing Xie, Zhen Yang, Xingwu Sun, Shuaipeng Li, Weidong Han, Zhanhui Kang, Di Wang, and Yu Cheng. Transmamba: A sequence-level hybrid transformer-mamba language model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31823–31833, 2026b.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning. *Advances in Neural Information Processing Systems*, 38:119406–119432, 2026.

Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. Openceres: When open information extraction meets the semi-structured web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3047–3056, 2019.

Luke McDermott, Robert W Heath Jr, and Rahul Parhi. Lola: Low-rank linear attention with sparse caching. *arXiv preprint arXiv:2505.23666*, 2025.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Lin Niu, Xin Luo, Linchuan Xie, Yifu Sun, Guanghua Yu, Jianchen Zhu, and S Kevin Zhou. Stem: Rethinking causal information flow in sparse attention. *arXiv preprint arXiv:2603.06274*, 2026.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, 2018.

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. *arXiv preprint arXiv:2404.02258*, 2024.

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.

Liliang Ren, Yang Liu, Yadong Lu, Chen Liang, Weizhu Chen, et al. Samba: Simple hybrid state space models for efficient unlimited context language modeling. In *International Conference on Learning Representations*, volume 2025, pp. 53551–53575, 2025.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 8732–8740, 2020.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

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. *arXiv preprint arXiv:1701.06538*, 2017.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *Advances in Neural Information Processing Systems*, 37:7339–7361, 2024.

ByteDance Seed Team. Seed-oss open-source models. `https://github.com/ByteDance-Seed/seed-oss`, 2025.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

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.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.

Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL `https://github.com/fla-org/flash-linear-attention`.

Yuanhang Yang, Shiyi Qi, Wenchao Gu, Chaozheng Wang, Cuiyun Gao, and Zenglin Xu. Xmoe: Sparse models with fine-grained and adaptive expert selection. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11664–11674, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Zihao Zeng, Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. Adamoe: Token-adaptive routing with null experts for mixture-of-experts language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6223–6235, 2024.

Jintao Zhang, Haoxu Wang, Kai Jiang, Shuo Yang, Kaiwen Zheng, Haocheng Xi, Ziteng Wang, Hongzhou Zhu, Min Zhao, Ion Stoica, et al. Sla: Beyond sparsity in diffusion transformers via fine-tunable sparse–linear attention. In *International Conference on Learning Representations*, volume 2026, pp. 138288–138305, 2026.

Michael Zhang, Simran Arora, Rahul Chalamala, Benjamin Spector, Alan Wu, Krithik Ramesh, Aaryan Singhal, and Christopher Ré. Lolcats: On low-rank linearizing of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 46200–46253, 2025.

Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. Mixture of attention heads: Selecting attention heads per token. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 4150–4162, 2022.

Jingwei Zuo, Maksim Velikanov, Ilyas Chahed, Younes Belkada, Dhia Eddine Rhayem, Guillaume Kunsch, Hakim Hacid, Hamza Yous, Brahim Farhat, Ibrahim Khadraoui, et al. Falcon-h1: A family of hybrid-head language models redefining efficiency and performance. *arXiv preprint arXiv:2507.22448*, 2025.

A Detailed Related Work

A.1 Conditional Computation in Transformers

Token-Adaptive Expert and Depth Budgets. Conditional computation improves the performance-compute trade-off by assigning different compute budgets to different tokens (Bengio et al., 2013; 2015). Existing approaches can be organized by the axis along which computation is made adaptive. Expert-wise methods build on MoE architectures (Shazeer et al., 2017) and dynamically adjust expert computation, for example through top-p expert allocation (Huang et al., 2024; Yang et al., 2024) or null-expert designs (Zeng et al., 2024; Jin et al., 2025). These methods mainly target feed-forward or expert computation, whereas LoGo targets attention span and the associated long-context cost. Depth-wise methods adapt the number of layers processed by each token, with representative lines including early-exit mechanisms (Elbayad et al., 2020; Schuster et al., 2022) and mixture-of-depths routing (Raposo et al., 2024; Fu et al., 2025). They allocate computation along model depth, while LoGo allocates attention span within every layer.

Token-Adaptive Attention Budgets. Attention-wise conditional computation is closest to LoGo. Some methods adapt attention by selecting heads (Zhang et al., 2022) or assigning token-wise sparse attention budgets (Lin et al., 2026; Niu et al., 2026). These approaches show that attention computation need not be uniform across tokens. LoGo focuses on attention span because it directly determines both accessible context and attention cost, making it a natural proxy for token-level attention-budget allocation. The most related attention-wise methods dynamically activate specific attention modules. CoLT5 (Ainslie et al., 2023) routes tokens through light and heavy branches for both feed-forward and attention computation, improving the quality-efficiency trade-off. However, it is designed for an encoder-decoder architecture and applies dynamic computation on the encoder side, whereas LoGo targets decoder-only pretrained backbones and autoregressive long-context modeling. SMA (Ren et al., 2023) activates a GAU-style attention module based on SSM states, achieving efficient sequence modeling, but activated tokens still share the same restricted sparse attention pattern. Its router therefore decides whether to enter an attention module rather than which context span a token should receive.

Difference from LoGo. LoGo differs from these conditional-computation methods in three ways. First, it uses attention span as a direct proxy for attention budget, routing each token between local-only and local-plus-global computation. Second, it combines lightweight branch decoupling with an auxiliary-loss-free threshold controller, enabling comparisons under matched parameters and attention budgets. Third, we evaluate this design at pretrained scale with matched parameters and FLOPs, emphasizing long-context retrieval rather than only short-context quality or inference speed.

A.2 Local-Global Hybrid Attention

Inter-Layer Hybrids. Local-global hybrid attention reduces long-context cost by combining restricted-span computation (Beltagy et al., 2020; Dai et al., 2019; Jiang et al., 2023) with full-context access (Vaswani et al., 2017). A common design is the inter-layer hybrid, which assigns different attention mechanisms to different layers. Recent systems alternate global attention with local attention (Team et al., 2024; Agarwal et al., 2025; Xiao et al., 2026), linear attention (Li et al., 2025; Team et al., 2025), or SSM-style modules (Lieber et al., 2024; Glorioso et al., 2024; Ren et al., 2025). This layer-wise mixture is efficient and simple to scale, but all tokens in the same layer share the same span decision. As a result, tokens that only require local evidence may still pass through global layers, while tokens requiring long-range retrieval cannot receive additional global computation in local layers.

Intra-Layer Hybrids. Another common design is the intra-layer hybrid, which mixes mechanisms within a layer, typically by assigning different heads to different attention types (Dong et al., 2025; Zuo et al., 2025; Fu et al., 2024). Head-wise mixtures provide both local and global capacity in each layer and are often considered more capable than layer-wise alternation. However, the span allocation remains static: the same subset of heads is global for every token, regardless of token-level contextual demand. LoGo instead shares each token's routing decision across heads for

implementation efficiency, while allowing different tokens to activate global attention in different numbers of layers.

Sequence-Adaptive Hybrids. A smaller set of sequence-adaptive hybrids varies the attention pathway across positions. Some methods use fixed predefined patterns, giving different positions different pathways without learning token-specific budgets (Zhang et al., 2025; McDermott et al., 2025; Zhang et al., 2026). More closely related methods such as TransMamba (Li et al., 2026b) and Oryx (Li et al., 2026a) assign different per-token budgets across a sequence, but the allocation is largely handcrafted rather than learned jointly with the language-modeling objective. These designs suggest that sequence-wise variation in attention is useful for efficient context modeling. LoGo instead learns the local/global span decision from token representations, rather than prescribing per-position budgets, allowing global attention to be allocated dynamically under a controlled budget.

Difference from LoGo. LoGo is complementary to static local-global hybrids rather than a replacement for all hybrid designs. It keeps the attention form fixed and learns span allocation end-to-end: every token receives local attention, and a learned gate decides whether to additionally activate global attention under a controlled budget. This makes LoGo directly comparable with inter-layer and intra-layer hybrids under matched global-attention ratios. It also makes LoGo compatible with memory-saving hybrids: global-attention components in a static hybrid can be replaced by LoGo layers to preserve the memory benefits of the underlying layout while reducing global computation.

B Efficient Implementation Details

Query-Sparse Prefill and Training. During prefill and training, the global branch uses a selected-query full-causal attention kernel. Selected queries are packed in ascending sequence order into a variable-length compacted layout, while keys and values remain dense over the original sequence. For each compacted query row, the kernel stores its original sequence position, together with compacted query offsets and dense key-value offsets. Causal visibility is therefore determined by real positions rather than compacted indices. In the forward pass, each compacted query block has a known range of original positions. KV blocks before the minimum selected position in the block are fully visible and are processed without a causal predicate, blocks after the maximum selected position are skipped, and only the diagonal region applies a real-position causal mask. This gives the same result as dense full-causal attention on the selected rows while avoiding the significant overhead of applying dense causal masks to every KV block.

The backward pass preserves the same row-sparse structure. The query-gradient kernel mirrors the forward traversal over visible KV blocks and writes gradients only for selected query rows. Gradients for unselected query rows remain zero and are scattered back to the original layout. For key and value gradients, kernels are scheduled over dense KV blocks. A lightweight precomputed search identifies, for each KV block, the first compacted-query block that can causally attend to it. The kernel skips earlier query blocks and processes only the valid region: at most two partially masked diagonal compacted-query blocks, followed by fully causal query blocks that need no causal predicate. As a result, the query-sparse kernels preserve the outputs and gradients of dense global attention restricted to the selected query rows while reducing both forward and backward computation.

Autoregressive Decoding. During autoregressive decoding, each layer maintains the key-value caches required by the local and global branches. The local branch always attends to at most the most recent w cached tokens. Since the gate is computed before global attention is launched, a token routed to local span can skip the global attention call entirely and use the local output. If the token activates global span, it attends over the full global cache and fuses local and global outputs using the same rule as in Section 3.2.

Our current batched decoding implementation adopts a conservative policy to avoid modifying the existing decode kernel. If no query in a microbatch activates the global branch, the global call is skipped. Otherwise, the standard dense decode kernel is launched for the microbatch, and inactive rows are masked in the output. This policy captures all-local decoding steps without introducing per-

step query compaction overhead. A more specialized serving kernel could compact active decode rows in the same spirit as the prefill kernel, further reducing global decoding computation.

Complexity and Compatibility. Ignoring projections and MLPs, dense causal attention in one layer costs $\mathcal{O}(T^2Hd_h)$, and sliding-window attention costs $\mathcal{O}(TwHd_h)$. LoGo adds the local branch for all tokens but evaluates the global branch only on selected query rows, giving total attention cost $\mathcal{O}(TwHd_h + \rho^{(\ell)}T^2Hd_h)$ under target global ratio $\rho^{(\ell)}$. Since w is small in our setting, LoGo has the same dominant attention-compute order as a static local-global hybrid with global fraction $\rho^{(\ell)}$, but it allocates the global computation dynamically at the token level.

The key-value cache remains dense and linear in context length, as in standard global attention. Therefore, LoGo does not by itself provide proportional memory savings. For memory-constrained settings, LoGo can instead be combined with memory-saving local-global designs by replacing their global-attention components with query-sparse LoGo layers, reducing compute while preserving the memory benefits of the underlying hybrid.

LoGo introduces row sparsity rather than head sparsity or key-value sparsity. All heads in a layer share the same token-level routing decision, so each head processes the same number of global query rows and avoids head-wise load imbalance. Meanwhile, the dense key-value layout preserves sequence order and causal structure. Together, these properties make LoGo compatible with tensor parallelism and sequence-parallel implementations. Although token selection is dynamic, the irregularity is converted before the kernel launch into metadata and a compacted query tensor, so the GPU kernel still operates on regular blocks rather than fine-grained branches.

C Experimental Details and Additional Results

C.1 Training Setup

Scaling Experiments. The LoGo models and Transformer baselines use the same backbone configuration and training hyperparameters at each scale, trained on an in-house corpus similar to Team (2025). Table 6 summarizes the model hyperparameters, which follow the scaling setup of Sun et al. (2024). We set the FFN intermediate dimension to $8d/3$. For LoGo, we use target global ratio $\rho^{(\ell)} = 0.5$, sliding-window size $w = 128$, threshold update rate $\gamma = 0.0005$, and initialize $b_{\text{th}}^{(\ell)}$ according to a 20% P-mask duration. For training, we fix the tokens-per-parameter (TPP) ratio to 20 and set the maximum sequence length to 8192. The batch size is 0.5M tokens for the first four scales and 1M tokens for the 3.3B scale. We use AdamW with weight decay 0.1. The learning rate peaks at the value shown in Table 6 after a 1% warmup and then cosine-decays to 10% of the peak.

Table 6: **Model hyperparameters for scaling experiments.**

Size	Hidden Dim.	#Layers	#Heads	LR
200M	768	12	12	6×10^{-4}
470M	1024	24	16	6×10^{-4}
900M	1536	24	12	6×10^{-4}
1.5B	2048	24	16	3×10^{-4}
3.3B	2560	32	20	3×10^{-4}

Controlled 1.5B Comparison. For the controlled 1.5B comparison in Section 4.2, all models use the same MHA-SwiGLU backbone as in the scaling experiments and differ only in their attention span pathway. We pretrain each model for 100B tokens with a maximum sequence length of 8k and a global batch size of 4M tokens. We use AdamW with weight decay 0.1 and gradient clipping 1.0. The learning rate linearly warms up to 6×10^{-4} over the first 1% of training and then cosine-decays to 6×10^{-5}. After pretraining, we perform two progressive context-extension stages to context lengths of 32k and 128k, each for 10B tokens. The two stages use cosine learning-rate decay to 1×10^{-5} and global batch sizes of 4M and 8M tokens, respectively. For LoGo, we use a threshold update rate of $\gamma = 0.0005$ and initialize $b_{\text{th}}^{(\ell)}$ according to a 20% P-mask duration. Concretely, for a P-mask duration of S steps, we set $b_{\text{th},0}^{(\ell)} = 0.5 - \gamma S$. For 100B-token pretraining, 20% duration corresponds to $S = 5000$ steps, giving an initial threshold of -2.0.

C.2 Evaluation Setup

We evaluate language modeling with training loss and perplexity on WikiText (Wiki.; (Merity et al., 2017)) and Lambada (LMB.; (Paperno et al., 2016)). Zero-shot commonsense reasoning (CSR) reports average accuracy over Lambada, PIQA (Bisk et al., 2020), HellaSwag (Hella.; (Zellers et al., 2019)), WinoGrande (Wino.; (Sakaguchi et al., 2020)), and ARC-Easy / ARC-Challenge (ARC-e / ARC-c; (Clark et al., 2018)). Short-context recall includes FDA (Arora et al., 2023), SWDE (Lockard et al., 2019), SQuAD (SQD; (Rajpurkar et al., 2018)), TriviaQA (TQA; (Joshi et al., 2017)), Natural Questions (NQ; (Kwiatkowski et al., 2019)), and DROP (Dua et al., 2019), following the evaluation setting of Arora et al. (2024). Long-range recall uses all needle-style RULER (Hsieh et al., 2024) tasks: three single-needle variants (S-NIAH) and five multi-needle variants (MK-NIAH, MQ-NIAH, and MV-NIAH). All evaluations are run with `lm-evaluation-harness` (Gao et al., 2021).

C.3 Additional Experimental Results

Table 7 reports language-modeling and commonsense reasoning results after 32k context extension.

Tables 8 and 9 provide per-task RULER scores for the 32k and 128k checkpoints.

Table 7: **Language modeling and commonsense reasoning after 32k context extension.** Attention budget denotes the global-attention ratio. Lower is better for loss and perplexity, and higher is better for accuracy.

Model	Attn	Loss / PPL ↓			Commonsense Reasoning ↑						
	Budget	Loss	Wiki.	LMB.	LMB.	PIQA	Hella.	Wino.	ARC-e	ARC-c	Avg.
Transformer	1.0	1.428	16.05	9.01	53.08	70.51	52.36	57.54	66.58	31.57	55.27
inter-layer-H	0.5	1.424	15.93	8.75	54.28	71.71	52.43	57.85	65.66	33.02	55.82
intra-layer-H	0.5	1.425	16.05	8.82	53.50	72.09	52.69	56.67	65.70	32.42	55.51
LoGo	0.5	**1.417**	**15.89**	**7.92**	55.89	72.14	53.24	57.38	66.12	32.34	**56.18**

Table 8: **Per-task RULER results for 32k checkpoints.** We report needle-style RULER scores at 8k, 16k, and 32k input lengths.

Model	S-NIAH-1			S-NIAH-2			S-NIAH-3		
	8k	16k	32k	8k	16k	32k	8k	16k	32k
Transformer	**100.0**	**100.0**	94.6	**100.0**	**100.0**	95.0	98.8	98.4	83.0
inter-layer-H	**100.0**	**100.0**	97.6	**100.0**	**100.0**	90.6	94.6	79.2	64.0
intra-layer-H	**100.0**	**100.0**	96.6	**100.0**	**100.0**	83.8	99.0	81.6	59.2
LoGo	**100.0**	**100.0**	**99.0**	**100.0**	**100.0**	**99.4**	**100.0**	**99.6**	**92.2**

Model	MK-NIAH-1			MK-NIAH-2			MK-NIAH-3		
	8k	16k	32k	8k	16k	32k	8k	16k	32k
Transformer	86.4	78.8	62.0	98.0	97.6	73.6	60.8	55.6	27.2
inter-layer-H	92.2	93.2	65.2	96.6	96.8	77.4	**63.0**	55.6	**53.0**
intra-layer-H	**93.8**	82.8	55.6	94.8	75.4	46.0	58.6	54.8	49.2
LoGo	93.4	**93.8**	**70.8**	**99.4**	**99.0**	**91.0**	61.4	**59.2**	47.4

Model	MQ-NIAH			MV-NIAH		
	8k	16k	32k	8k	16k	32k
Transformer	86.4	72.8	**52.3**	89.2	51.7	31.5
inter-layer-H	84.0	71.9	40.2	91.3	**74.6**	**44.2**
intra-layer-H	**88.1**	**73.5**	38.6	91.6	67.6	28.6
LoGo	86.1	68.6	49.7	**94.6**	61.7	26.3

Table 9: **Per-task RULER results for 128k checkpoints.** We report needle-style RULER scores at 32k and 64k input lengths.

Model	S-NIAH-1		S-NIAH-2		S-NIAH-3	
	32k	64k	32k	64k	32k	64k
Transformer	99.8	80.0	**99.8**	59.4	93.0	43.4
inter-layer-H	**100.0**	**95.6**	98.8	53.4	77.0	35.8
intra-layer-H	99.8	70.4	95.0	41.8	70.2	27.6
LoGo	**100.0**	83.4	99.6	**65.0**	**96.4**	**57.6**

Model	MK-NIAH-1		MK-NIAH-2		MK-NIAH-3	
	32k	64k	32k	64k	32k	64k
Transformer	61.6	43.2	89.8	41.0	56.8	13.0
inter-layer-H	72.4	47.0	88.8	46.8	53.2	28.8
intra-layer-H	58.6	34.4	53.6	25.4	51.0	20.4
LoGo	**80.8**	**48.6**	**97.0**	**68.2**	**70.2**	**34.6**

Model	MQ-NIAH		MV-NIAH	
	32k	64k	32k	64k
Transformer	53.8	**35.6**	31.6	**27.2**
inter-layer-H	45.3	25.0	**41.1**	27.0
intra-layer-H	37.2	25.3	25.2	26.4
LoGo	**53.9**	35.5	28.6	26.8

C.4 Additional Ablations and Analysis

C.4.1 Additional Ablations

Architecture Ablations. We provide additional interpretation for Table 5. **(1) Branch parameterization.** Fully sharing all branch parameters limits specialization and hurts both modeling and recall performance. Fully decoupling branches increases parameters by about 20% and improves perplexity, but does not improve recall, likely because the sparse global branch becomes harder to train. Replacing the linear-projection (LP) branch transformation with a scale-and-offset transformation (SO) is also slightly worse, supporting LoGo's lightweight LP decoupling design. **(2) ContextNorm.** Removing ContextNorm prevents proper convergence, as the two branch outputs have substantially different variances. The SO transformation reduces branch discrepancy, but removing ContextNorm under SO still hurts convergence, showing that explicit output normalization remains important for stable fusion. **(3) Output fusion.** Weighted fusion outperforms additive fusion, supporting explicit competition between local and global spans. **(4) Budget control.** Replacing threshold-based control with an auxiliary loss degrades modeling performance and yields a higher training loss, justifying the auxiliary-loss-free controller. **(5) Progressive masking.** Masking the global branch from the beginning hurts downstream performance, confirming the benefit of early dense branch learning before sparse routing takes effect.

Different Local/Global Ratios. Tables 10 and 11 provide the full ratio-sweep results. Parentheses indicate the difference from the matched inter-layer baseline.

Compatibility with Other Hybrids. LoGo is complementary to memory-saving hybrid designs. Starting from a 1:1 inter-layer hybrid (global ratio 0.5), we replace its global-attention layers with LoGo layers and report recall performance and three representative multi-needle tasks in Table 12. Adding LoGo further reduces compute while improving average recall (65.6/66.9 vs. 65.1). The 0.5-threshold variant also outperforms an equal-FLOPs inter-layer hybrid with a 1:3 global/local ratio (65.6 vs. 64.7). These results show that LoGo can be combined with memory-saving static hybrids to improve the compute-memory-performance trade-off.

Table 10: **Short-context recall under different local/global ratios.** We compare LoGo with inter-layer hybrids under matched global-attention budgets using pretrained checkpoints. Because FDA varies noticeably with the global ratio, we additionally report the average excluding FDA.

Model	Attn Budget	FDA	SWDE	SQD	TQA	NQ	Drop	Avg.	Avg. (w/o FDA)
inter-layer-H	0.25	85.48	84.07	50.17	61.55	33.01	27.55	56.97	51.27
inter-layer-H	0.50	79.49	84.34	51.51	63.15	33.70	28.65	56.81	52.27
inter-layer-H	0.75	84.12	84.79	50.97	63.68	33.86	28.27	57.62	52.32
LoGo	0.25	85.21	83.53	50.80	61.67	35.76	28.03	**57.50 (+0.53)**	**51.96 (+0.69)**
LoGo	0.50	84.03	85.06	52.18	64.51	34.50	28.80	**58.18 (+1.37)**	**53.01 (+0.74)**
LoGo	0.75	82.58	85.60	51.51	65.23	35.51	30.81	**58.54 (+0.92)**	**53.73 (+1.41)**

Table 11: **RULER performance under different local/global ratios.** We compare LoGo with inter-layer hybrids under matched global-attention budgets using 32k checkpoints.

Model	Attn Budget	RULER 16k ↑	RULER 32k ↑
inter-layer-H	0.25	83.3	67.1
inter-layer-H	0.50	83.9	66.5
inter-layer-H	0.75	75.1	59.9
LoGo	0.25	80.9 (-2.4)	**73.2 (+6.1)**
LoGo	0.50	**85.2 (+1.3)**	**72.0 (+5.5)**
LoGo	0.75	**80.1 (+5.0)**	**67.0 (+7.1)**

Table 12: **Compatibility with memory-saving hybrids.** We evaluate pretrained checkpoints and report recall performance and three representative multi-needle tasks; tasks with smaller differences, such as S-NIAH, are omitted.

Model	Attn Budget	FDA	SWDE	SQD	TQA	NQ	Drop	MK1-8k	MQ-8k	MV-8k	Avg.
inter-layer-H (1:3)	0.25	**85.48**	84.07	50.17	61.55	33.01	27.55	84.4	73.8	82.4	64.7
inter-layer-H (1:1)	0.50	79.49	84.34	51.51	63.15	33.70	28.65	**85.6**	77.0	82.0	65.1
inter-layer-H (1:1) + LoGo (0.5)	0.25	79.77	**84.79**	52.28	**64.93**	34.62	27.12	84.2	80.2	82.4	65.6
inter-layer-H (1:1) + LoGo (0.7)	0.35	82.85	84.07	**53.25**	64.75	**35.86**	**28.89**	83.8	**84.2**	**84.2**	**66.9**

C.4.2 Additional Analysis

Span Allocation Statistics. Figure 4 provides the full layer-wise and token-wise routing statistics across four domains. The average global ratio stays close to the target ratio of 0.5, with mild variation across domains, confirming that the budget controller maintains the intended operating point. However, the realized allocation is not uniform across layers. Wiki and Quest concentrate more global span near the final layer, GSM8K peaks in the first layer, and code uses the least global span in the final layer, suggesting that different domains use long-range context at different stages of representation processing. Token-level span usage also varies substantially within each domain: some tokens use global span in almost no layers or almost all layers, while most fall between these extremes. This confirms that LoGo learns token-level dynamic allocation rather than merely matching the average budget.

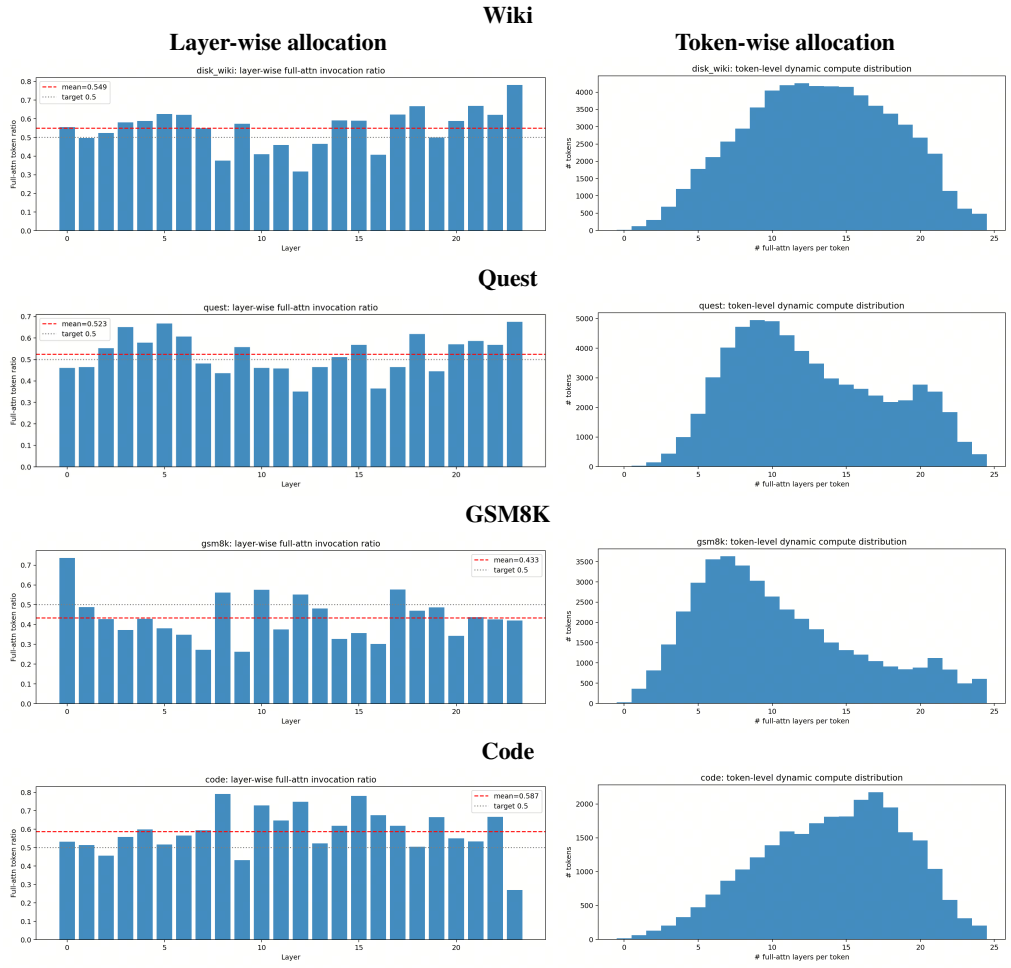

Figure 4: Span allocation statistics across domains. For each domain, the left panel reports the realized global activation ratio by layer, and the right panel reports the distribution of token-level global-span usage, measured by the number of layers with global activation (0–24).

Word-Level Visualization. For Figure 3, we construct an input passage from YOCO (Sun et al., 2024) and Chinchilla (Hoffmann et al., 2022), two randomly selected papers cited in this work, and insert two controlled cases: repeated short sentences and a bilingual context switch. Because routing is attached to the query state used to predict the next token, we shift the recorded activation by one position, so each word is colored by the span used to predict it. We then compute word-level global-span activation by averaging over constituent tokens. The full visualization is shown in Figure 5. The visualization shows five interpretable patterns. **(1) Sequence beginnings.** Tokens near the beginning of the input sequence mostly use local attention, avoiding unnecessary global-span



Figure 5: **Full word-level span allocation visualization.** Warmer colors indicate more layers activating global attention.

allocation before long-range context becomes useful. **(2) Sentence starts.** Sentence-initial tokens after punctuation, especially periods, often receive more global span, consistent with the limited local context available at a new sentence boundary. **(3) Fixed expressions.** Fixed expressions such as "large language models" and "state-of-the-art" become increasingly local as later words are locally predictable. **(4) Long-range repetition.** Repeated content triggers global activation when the match lies outside the local window, whereas nearby repetition uses less global span. We hypothesize that once a few anchor tokens, such as "The grass is green", match distant context, subsequent tokens rely on global retrieval to continue the repeated span. **(5) Context switches.** After switching to a new context, global-span usage drops quickly and rises again only when the passage resumes long-range matching. These patterns provide an interpretable account of how LoGo allocates global computation according to token-level contextual demand.