SALSA: State Augmentation via Learned Selective Attention

Joel Manu[1] joelmanu@mit.edu
Frederick Hoffman[2 1] freddy@lyralabs.ai
Krithik Ramesh[2 1] krithik@lyralabs.ai

[1]Massachusetts Institute of Technology, Cambridge, MA, USA
[2]Lyra Labs, Boston, MA, USA

Abstract

Hybrid architectures combining attention and recurrent primitives have emerged as a popular paradigm in sequence modeling, achieving favorable tradeoffs between the in-context recall of transformers and the bounded state compression of recurrences. However, existing hybrids statically interleave primitives at fixed, hand-designed ratios, invoking attention operations regardless of whether the recurrent state is sufficient and raising concerns about adaptability and the effectiveness of primitive allocation. We introduce SALSA, a dynamic hybrid architecture that pairs the recurrent primitive with a learned router that selectively invokes attention on a per-token basis, when projected to improve the recurrent state quality. We evaluate this approach on short and long context benchmarks at scale. At 1.3B parameters, SALSA achieves lower pretraining perplexity than the static hybrid (9.72 vs. 9.99), outperforms it on 8 of 8 LM-Eval metrics by an average of 1.5 points, and improves PG19 perplexity over the static Hybrid at every sequence length evaluated from 2k to 32k. Additionally, we investigate the learned routing patterns of SALSA to reveal sparse, structured attention patterns that suggest SALSA acquires meaningful, content-dependent strategies for when global context is necessary. Together, these results position dynamic primitive allocation as a principled and practical direction for hybrid sequence modeling.

1. Introduction

Transformers (Vaswani et al., 2017) remain the dominant architecture in language modeling, yet the quadratic cost and diminishing effectiveness of full self-attention at long sequence lengths have motivated alternatives that compress context rather than attend over it in full. Among the most promising alternatives are State Space Models (SSMs) (Gu et al., 2022; Gu & Dao, 2024), a family of linear recurrent architectures that maintain a fixed-size hidden state updated incrementally across the sequence. However, a growing body of work finds that SSMs appear most effective as components of hybrid architectures (Glorioso et al., 2024; Lieber et al., 2024; Waleffe et al., 2024) that strategically combine recurrent and attention primitives. This raises a fundamental design question about how and when each primitive should be invoked.

When successful, hybrid architectures draw on the complementary strengths of each primitive, combining the precise recall of full attention with the compressed context modeling of recurrence to surpass the quality either achieves in isolation. The central challenge is therefore not merely to combine attention and recurrence, but to determine which computations require attention's exact recall and which are well captured by compressed recurrent state. There have been a number of exciting advances toward understanding and characterizing this optimal blend. Research on hybrid architectures has shown that as little as 8% of a network may need to consist of attention layers to achieve the best training loss, and that only a small percentage of tokens require the precise retrieval capabilities of attention (Bick et al., 2025).

These findings have helped drive strong results from hybrid architectures across scales and domains, but determining the optimal blend remains a challenging and actively researched problem. In practice, these hybrid architectures have

Accepted to FoGen 2026: Foundations of Deep Generative Models: Understanding Memorization, Generalization, and Reasoning, an ICML 2026 workshop (non-archival).

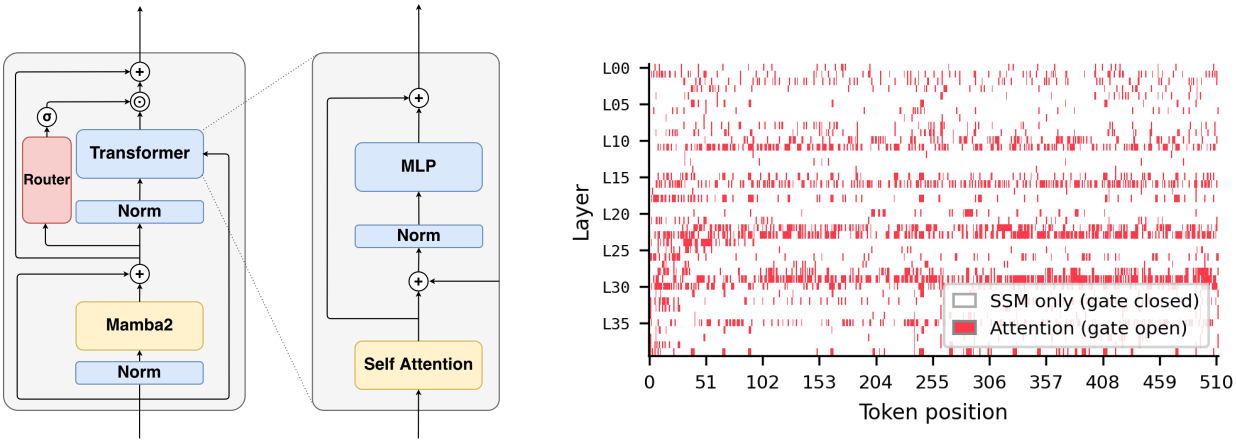


Figure 1. Overview of SALSA. **(Left)** Architecture overview: each SALSA layer routes tokens through a recurrent module unconditionally and selectively invokes attention via a learned gate. **(Right)** Per-token attention routing for the 780M SALSA model on a held-out sequence, revealing sparse, layer-banded attention usage resembling the attention usage of a traditional hybrid.

traditionally been implemented by interleaving layers of SSMs and transformers at a fixed, manually designed ratio. However, no single ratio is likely to be optimal across all applications, and even for a given application, allocating attention to the places where it most improves quality requires careful tuning. Further, recent work has observed that stronger individual components in hybrids can paradoxically weaken the overall model. For example, longer sliding-window attention contexts can degrade long-range performance by discouraging recurrent layers from learning to compress effectively (Cabannes et al., 2025).

We argue that this static interleaving is inherently limited. A fixed ratio cannot adapt to input content by design, forcing the model to expend the same attention budget in every context rather than learning when its components are most needed. Conditional computation provides a promising framework for addressing this. Mixture of Experts (Shazeer et al., 2017) models learn to route each token to a subset of specialized parameters, increasing effective capacity without proportional compute cost. On the other hand, H-Net (Hwang et al., 2026) learns to dynamically segment input into variable-length chunks rather than relying on fixed tokenization. While both introduce additional challenges in training and implementation, they demonstrate that learned, input-dependent architectural decisions can meaningfully improve upon static ones. We see hybrid primitive selection as a natural next application of this principle. Ideally, a hybrid model would learn to select which primitives to invoke on a per-token basis, adapting its computation to the demands of the input.

To this end, we make the following contributions:

1. We introduce SALSA, a dynamic hybrid architecture that replaces fixed primitive interleaving with a learned routing mechanism. At each token, a lightweight router uses the SSM's output to assess whether attention is necessary, invoking it selectively and combining outputs only when the learned gate predicts attention is useful. This design directly addresses the problem of where to optimally apply attention, and offers a natural pressure against against over-reliance on either primitive.

2. We devise a training recipe for SALSA, combining auxiliary losses with straight-through estimation and a continuous-to-binary gating schedule that stabilizes optimization and encourages sparse, decisive routing.

3. We evaluate SALSA at three scales (140M, 780M, 1.3B) on language modeling and long-context benchmarks, where it outperforms the static hybrid on nearly all metrics with an advantage that widens with scale. Analysis of learned routing patterns reveals structured, sparse attention usage, suggesting that SALSA acquires non-trivial strategies for primitive allocation.

2. Related Work

Transformers. Transformers (Vaswani et al., 2017) dominate sequence modeling but scale quadratically in sequence length. Efficiency improvements range from hardware-aware implementations (Dao et al., 2022) and architectural refinements (Shazeer, 2020; Ainslie et al., 2023; Touvron et al., 2023; Su et al., 2024) to sparse attention patterns (Child et al., 2019; Beltagy et al., 2020; Ding et al., 2023), yet even efficient variants remain super-linear.

Linear recurrent architectures. Linear recurrent models maintain a fixed-size hidden state with constant-time per-step inference. Beginning with linear attention (Katharopoulos et al., 2020), successive work has improved state capacity and utilization (Peng et al., 2022; Gu & Dao, 2024; Dao & Gu, 2024; Yang et al., 2024a;b; 2025), though recall-intensive tasks remain challenging due to finite state.

Hybrid architectures. Hybrid architectures bridge this gap by interleaving recurrent and attention layers at a fixed ratio, typically favoring recurrences (Lieber et al., 2024; Waleffe et al., 2024; Dong et al., 2025). Waleffe et al. (2024) find that as few as 7% of layers need attention for competitive perplexity, and other work prunes or distills attention further (Voita et al., 2019; Elbayad et al., 2020; Wang et al., 2024). However, these approaches fix the primitive allocation statically.

Conditional computation. Conditional computation methods activate model capacity selectively per token. Mixture-of-Experts models (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022) route tokens to specialist feed-forward modules, while H-Net (Hwang et al., 2026) applies learned routing to input segmentation, providing dynamic chunking of input sequences.

3. SALSA

We propose SALSA, a dynamic sequence modeling architecture that learns to mix state-space and Transformer sequence-mixing primitives. Through a set of architectural and training innovations, SALSA models can exploit the complementary strengths of both mechanisms while being trained seamlessly in a standard natural language modeling setting. We arrive at SALSA's specific design through a set of systematic ablations, reported in Appendix C.

3.1. SALSA Layer

Each SALSA layer composes two sequence modeling primitives: a **linear-time recurrent module** and a **full-context attention module**, with a learned router mediating their contributions. Given an input $x_l \in \mathbb{R}^{d_{\text{model}}}$ at layer l, the recurrent module produces

$$y_l = M_l(\mathrm{LN}(x_l))$$

where $M_l : \mathbb{R}^d \to \mathbb{R}^d$ denotes a recurrent state update with finite memory capacity, and LN denotes a layer norm. SALSA introduces a lightweight **routing mechanism** $R : \mathbb{R}^d \to \mathbb{R}$ that produces a scalar gate from the sum of the layer input and the recurrent output

$$g_l = R(x_l + y_l)$$

so that the allocation decision is conditioned on both the residual stream and the SSM's compressed state. The attention pathway computes full-sequence context from the same joint representation

$$h_l = \mathrm{MLP}(\mathrm{LN}(\mathrm{CA}(\mathrm{LN}(x_l + y_l)) + x_l))$$

where CA denotes causal self-attention and MLP is a feed-forward network. The layer output combines both primitives through a gated residual

$$x_{l+1} = x_l + y_l + f(g_l) \cdot h_l$$

where $f : \mathbb{R} \to [0, 1]$ is a gating function. If f is chosen such that $f(\cdot) \in \{0, 1\}$, SALSA gains access to conditional computation. For $f(g_l) = 0$ the layer reduces to a pure recurrent update; when $f(g_l) = 1$, full attention context is

incorporated. The design of f and the training procedure that encourages sparse, decisive gating are described in Section 3.3.

The pseudocode for the SALSA layer forward pass is provided in Appendix A.

3.1.1. MEMORY PRIMITIVE

While SALSA is agnostic to the specific choice of recurrent memory backbone, we instantiate the memory primitive M_l^d using a **state space model (SSM)** due to its linear-time updates and strong empirical performance on long-context sequence modeling. In particular, we employ **Mamba-2** (Dao & Gu, 2024), which models sequences through a parameterized linear dynamical system with input-dependent modulation.

$$h_t = A_t h_{t-1} + B_t x_t$$
$$y_t = C_t^\top h_t$$

where A_t, B_t, C_t are time-dependent structured matrices. More specifically, $A_t \in \mathbb{R}$ is a scalar decay, and $B_t \in \mathbb{R}^{d_{\text{state}}}$, $C_t \in \mathbb{R}^{d_{\text{state}}}$ are input-dependent projection vectors.

3.1.2. ROUTER

To avoid costly overhead while still allowing expressivity in the gate signal, we employ a lightweight two-layer MLP as the router. Given the joint representation $x_l + y_l$, the router produces a scalar logit via a single hidden layer with GELU (Hendrycks & Gimpel, 2016) activation:

$$R(x_l + y_l) = W_2\,\mathrm{GELU}(W_1(x_l + y_l) + b_1) + b_2$$

where $W_1 \in \mathbb{R}^{d_{\text{hidden}} \times d}$, $W_2 \in \mathbb{R}^{1 \times d_{\text{hidden}}}$. We use $d_{\text{hidden}} = 128$ across all configurations, keeping the router parameter count negligible relative to the full model.

3.2. Attention Pathway

The attention pathway in each SALSA layer follows the standard Transformer++ formulation (Touvron et al., 2023). Given the joint representation $x_l + y_l$, causal self-attention (Vaswani et al., 2017) computes queries, keys, and values via learned projections, with rotary position embeddings (RoPE; Su et al., 2024) applied to queries and keys to encode relative position information. The attention output is passed through a SwiGLU feed-forward network (Shazeer, 2020), yielding the full attention pathway output h_l as defined in Section 3.1. This pathway is only incorporated into the residual stream when the router gate $f(g_l)$ is non-zero, making it the conditionally computed component of the SALSA layer.

3.2.1. GATING AND HARD ROUTING

The router logit g_l is mapped to a gate value via a sigmoid with temperature τ:

$$p_l = \sigma(g_l/\tau)$$

During inference and late-stage training, we apply a hard threshold η to obtain a binary gate:

$$f(g_l) = \mathbb{I}[p_l > \eta]$$

with $\eta = 0.5$ throughout all experiments. To preserve gradient flow through this discrete operation, we apply straight-through estimation (Bengio et al., 2013):

$$g = \mathbb{I}[p > \eta] + (p - \mathrm{sg}(p))$$

We refer to binary thresholding with straight-through estimation as **hard routing**. In the soft regime, $f(g_l) = p_l$ directly. The transition between regimes is described in Section 3.3.

3.3. Training

The routing mechanism in SALSA introduces additional complexity to optimization, requiring careful training design to ensure stable and efficient learning.

3.3.1. WARMUP

In practice, we train without *hard routing*, that is, we use a continuous sigmoid gate, for an initial warmup fraction of training steps, after which we enable *hard routing*, thresholding gates at η with straight-through estimation, for the remainder of training. We find this warmup stabilizes early optimization before the model is asked to commit to binary routing decisions.

3.3.2. AUXILIARY LOSSES

Supplementing conditional architectures with auxiliary losses has become a standard technique (Shazeer et al., 2017; Fedus et al., 2022). We employ two auxiliary loss terms to encourage sparse and decisive routing behavior. This helps keep the FLOP utilization of SALSA comparable to baseline models.

L2 Auxiliary Loss. We train SALSA using an ℓ_2 penalty on the gate values g_l across each layer l:

$$\mathcal{L}_{\ell_2} = \alpha \cdot \frac{1}{N} \sum_{l=1}^{N} \|g_l\|^2$$

where N is the number of layers and α is a tunable coefficient. This encourages the model to invoke attention sparingly, reserving it for tokens where the recurrent state alone is insufficient.

Entropy Loss. The transition to hard routing replaces each soft gate p_l with its thresholded value $\mathbb{I}[p_l > \eta]$. When gates sit near 0.5, this substitution is large and abrupt, creating a sharp shift in behavior at the moment hard routing engages. To smooth the transition, we drive gate values toward the extremes ahead of time by penalizing their entropy. Treating each gate value $p_l \in (0, 1)$ as a Bernoulli parameter, we minimize:

$$\mathcal{L}_{\text{entropy}} = \beta \cdot \frac{1}{N} \sum_{l=1}^{N} \Big(-p_l \log p_l - (1 - p_l) \log(1 - p_l) \Big)$$

where β is a tunable coefficient and gate values are clamped to $[\varepsilon, 1 - \varepsilon]$ for numerical stability. The total auxiliary loss is $\mathcal{L}_{\text{aux}} = \mathcal{L}_{\text{L2}} + \mathcal{L}_{\text{entropy}}$, added to the standard language modeling objective.

3.4. Inference

We describe SALSA's inference behavior in two regimes: autoregressive decoding, where conditional computation provides direct compute savings, and prefill, where we rely on standard parallel kernels with masking.

3.4.1. AUTOREGRESSIVE DECODING

At decode time, SALSA maintains the same key-value cache structure used by standard transformers, alongside the recurrent state of the Mamba-2 module. For each generated token, the recurrent state is updated unconditionally in constant time, and the router produces a gate value that determines whether the attention pathway executes.

Unconditional KV computation. Even when the gate is closed and the attention path is skipped, we unconditionally compute and append the current token's keys and values to the KV cache. This is necessary because future tokens must be able to attend to all preceding positions, including those that themselves did not invoke attention. The KV projection cost is small relative to the full attention pathway, so this asymmetry preserves most of the savings from conditional computation: when the gate is closed, SALSA skips the attention computation, the output projection, and the MLP, retaining only the lightweight key and value projections required to keep the cache complete.

Conditional attention execution. When the gate is open, the attention pathway proceeds as in a standard transformer block: the current query attends to the full KV cache, the result is projected, and the MLP is applied. When the gate is closed, the attention pathway is bypassed entirely and the layer's output reduces to $x + s$.

Table 1. Downstream evaluation results. **Bold** indicates best, underline indicates second best within each model size.

Model	FW-Edu ppl ↓	LAMB. ppl ↓	LAMB. acc ↑	HellaS acc_n ↑	PIQA acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	WinoGr. acc ↑	OBQA acc ↑	Avg acc ↑
140M parameters										
Transformer	16.31	53.70	31.5	35.6	65.1	57.7	27.1	50.9	19.2	41.0
Mamba	16.24	53.05	29.1	36.1	65.8	57.1	26.1	51.0	**21.0**	40.9
Hybrid	15.91	46.58	31.9	36.6	65.5	57.8	26.9	50.0	20.8	41.4
SALSA	**15.10**	**45.42**	**32.5**	**37.8**	**66.1**	**60.7**	**28.6**	**51.4**	20.2	**42.5**
780M parameters										
Transformer	11.16	18.90	41.6	49.9	71.0	68.4	33.1	54.7	24.6	49.0
Mamba	11.01	21.94	38.5	49.8	70.7	68.7	32.2	54.1	**27.4**	48.8
Hybrid	10.85	18.91	41.7	50.7	**71.8**	67.6	34.6	55.3	27.0	49.8
SALSA	**10.51**	**16.57**	**44.6**	**52.1**	71.7	**68.9**	**36.6**	**56.9**	26.8	**51.1**
1.3B parameters										
Transformer	10.20	15.84	44.0	53.6	71.5	70.8	35.3	56.5	27.8	51.4
Mamba	10.27	17.47	41.4	53.2	**72.6**	70.7	36.2	54.4	27.8	50.9
Hybrid	9.99	15.00	45.7	54.6	72.1	71.0	37.5	54.9	28.4	52.0
SALSA	**9.72**	**13.93**	**46.3**	**55.7**	72.5	**72.6**	**40.6**	**57.5**	**29.2**	**53.5**

3.4.2. PREFILL

For prefill, we use the parallel forms of both primitives: the state-space dual scan for Mamba-2 (Dao & Gu, 2024) and FlashAttention (Dao et al., 2022) for the attention pathway. Conditional computation is applied post-hoc by masking the attention contribution at positions where the gate is closed. While this approach does not yet realize the full theoretical compute savings of conditional routing during prefill, as closed-gate positions still incur the attention computation before being masked out, it allows tractable prefill and decode times without requiring specialized kernels for sparse attention. We view further prefill optimization as a natural direction for future work.

4. Experiments

We evaluate SALSA across five axes. We examine pretraining loss, downstream performance on language modeling tasks, long-context generalization, computational efficiency, and interpretability of learned routing patterns.

4.1. Experimental Setup

We pretrain four architectures at three parameter scales (140M, 780M, 1.3B), yielding twelve runs. All architectures share the same residual-stream backbone (pre-norm RMSNorm (Zhang & Sennrich, 2019), SwiGLU FFN (Shazeer, 2020), tied embeddings (Press & Wolf, 2017)) so that observed differences are attributable to the token-mixing primitive. The baselines are: (1) a Transformer with causal self-attention (Vaswani et al., 2017) and RoPE (Su et al., 2024), (2) a pure Mamba-2 model (Dao & Gu, 2024), and (3) a static Hybrid interleaving Mamba-2 and attention blocks at a 1:7 ratio, matching recent hybrid recipes (Lieber et al., 2024; Waleffe et al., 2024). SALSA shares the Hybrid's SSM backbone but replaces fixed interleaving with a learned per-token router that selectively invokes attention. All architectures use RoPE in their attention blocks. Because attention is invoked conditionally, not every parameter is active for every token. We therefore report SALSA parameter counts as *average active parameters*, computed from the observed routing rate, to give a fair comparison against baselines that use all parameters unconditionally. For details about active parameter counts, we refer the reader to Appendix A.

All models are trained on FineWeb-Edu using the GPT-2 tokenizer (vocab size 50,257) for approximately 52B tokens (100K steps, global batch size 256×2048 tokens). Training uses mixed precision (FP16 activations, FP32 weights) on NVIDIA H100 GPUs, scaling from 16 to 64 GPUs across model sizes. Validation loss is estimated every 400 steps on a held-out split.

For SALSA, the router produces continuous gate values early in training, then anneals from continuous to discrete (binary) gating after the first 20% of training using a temperature schedule (Section 3.3), so that for the majority of training each token either fully invokes or fully skips attention. Full training and infrastructure details are provided in Appendix B.

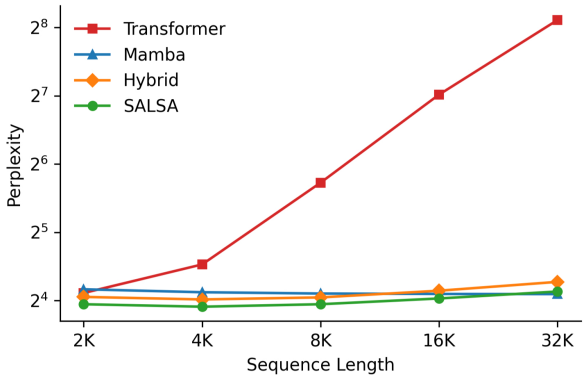


Figure 2. PG19 long-context perplexity for 1.3B models. Log-scale perplexity across sequence lengths.

4.2. Downstream Language Modeling Evaluations

We evaluate all models zero-shot using the LM-Eval harness (Gao et al., 2024) on 8 metrics from 7 tasks: LAMBADA perplexity and accuracy (Paperno et al., 2016), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2021), and OpenBookQA (Mihaylov et al., 2018). Results across all three scales are shown in Table 1. At 1.3B, SALSA outperforms the static Hybrid on all 8 metrics and achieves the best score overall on 7 of 8, with the highest average accuracy (53.5 vs. 52.0 for the Hybrid), representing a 1.5-point (2.8%) improvement. Notably, the gap between SALSA and the Hybrid widens with scale, from 1.1 points at 140M to 1.3 at 780M to 1.5 at 1.3B, suggesting that the dynamic routing mechanism becomes increasingly effective with greater model capacity. SALSA is competitive but slightly behind on PIQA and OBQA at some scales; these margins are within 1–2 points and may reflect evaluation variance rather than systematic weakness.

4.3. Long Context Evaluations

We evaluate long-context perplexity on PG19 (Rae et al., 2020) at five sequence lengths (2K, 4K, 8K, 16K, 32K), all beyond or at the 2K training length. As shown in Figure 2, SALSA achieves the lowest perplexity from 2K through 16K, outperforming the static Hybrid by 1.1–1.3 points across these lengths. At 32K, the pure Mamba model's length-invariant recurrence gives it a slight edge (17.1 vs. 17.6), though SALSA still outperforms the Hybrid (19.4) and the Transformer, which degrades sharply beyond the training length. These results suggest that SALSA's learned routing generalizes well to moderate length extrapolation, while the recurrence-only baseline retains an advantage at extreme lengths where positional encoding degradation dominates.

We further evaluate retrieval capability using the Needle-in-a-Haystack (NIAH) tasks from the RULER benchmark (Kamradt, 2023). While SALSA is competitive at short lengths (1K–2K), retrieval performance degrades faster than the Hybrid beyond the training length. We attribute this to RoPE degradation at unseen positions, which disproportionately affects selectively invoked attention layers. All reported results use position interpolation (Chen et al., 2023), which mitigates but does not eliminate the gap. Full results and discussion are provided in Appendix D.

4.4. Efficiency

We additionally evaluate the wall-clock efficiency of our pretrained 140M-parameter models. For each model, we measure both prefill and decode latency at a fixed context length T: the model first performs prefill over a sequence of length T, and then decodes an additional T tokens autoregressively. Each experiment is repeated eight times, and we report the average latency. All models use the appropriate cached or recurrent state during generation, so these measurements reflect the intended inference-time execution mode rather than a naive full-sequence recomputation baseline. More details about efficiency experiments are found in Appendix E.

Figure 4 shows that SALSA currently underperforms the other architectures in raw wall-clock latency. In particular, its prefill latency is higher across sequence lengths, and its decode throughput is lower than both the Transformer and Mamba-2 baselines. This result is somewhat expected as our implementation focuses on validating the model's sequence-modeling capability rather than on kernel-level or sparsity-aware optimization. These results suggest that

SALSA's current efficiency may be limited by implementation overhead. A natural direction for future work is to realize any theoretical sparsity gains with routing-aware kernels.

4.5. Understanding Attention Usage in SALSA

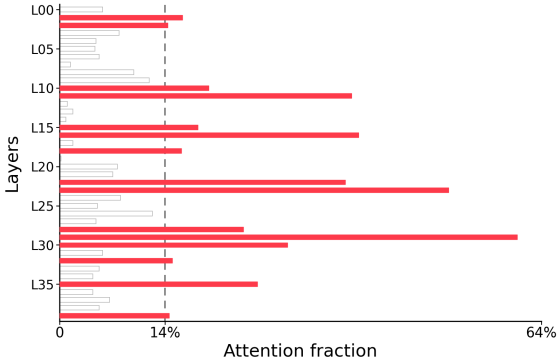


Figure 3. Per-layer attention usage for the 780M model scale. Bars indicate the fraction of tokens for which each layer's gate is open; the dashed line marks the model-wide average. This signifies that only some designated layers have significant attention usage

A central question for any dynamically routed architecture is whether the router learns meaningful structure or simply approximates a fixed allocation pattern. To investigate this, we visualize gate activations across all layers and token positions on a held-out sequence: President Kennedy's "We choose to go to the Moon" speech.[1] With hard routing enabled, we make observations consistent across model scales, training checkpoints, and auxiliary loss settings (Appendix F).

Attention is concentrated within a small subset of layers. Per-layer attention usage is highly non-uniform. A small set of layers, which we refer to as **attention-dense layers**, consistently invoke attention on a substantial fraction of tokens, while the majority of layers route sparsely. In Figure 3, for the 780M SALSA model, layers L11, L16, L22–23, and L28–30 each invoke attention on more than 25% of tokens, with L29 reaching 59%. The remaining layers fall well below this rate, and several invoke attention on less than 2% of tokens. This division of labor emerges purely from training: the architecture imposes no explicit specialization, yet the router learns to concentrate attention compute in specific layers.

Within attention-dense layers, routing is approximately position-invariant. The gate activations within attention-dense layers do not concentrate at sequence boundaries, recent tokens, or other positional landmarks. Instead, they appear roughly uniformly distributed across the sequence. This suggests that, conditional on a layer being attention-dense, the routing decision is driven by token content rather than position.

Robustness across training and scale. The qualitative patterns above are remarkably consistent across our experimental conditions. Layer banding emerges within the early stages of training and persists thereafter, and the same broad structure appears at all model scales we evaluate. Varying the auxiliary loss coefficient α shifts the overall sparsity level but does not disrupt the layer-banded structure: layers that are attention-dense at low sparsity targets remain the most attention-dense at high sparsity targets. We refer the reader to Appendix F for additional visualizations across these conditions.

Interpretation. While we make no strong causal claims, the consistency of these patterns suggests that SALSA discovers a functional specialization between its layers: a small number of layers act as global-context aggregators, while the remainder rely primarily on the recurrent state. We hypothesize that attention-dense layers serve to inject long-range information into the residual stream that downstream recurrent layers can then propagate locally. We leave a careful investigation of this hypothesis to future work.

[1] Speech transcript obtained from the John F. Kennedy Presidential Library and Museum.

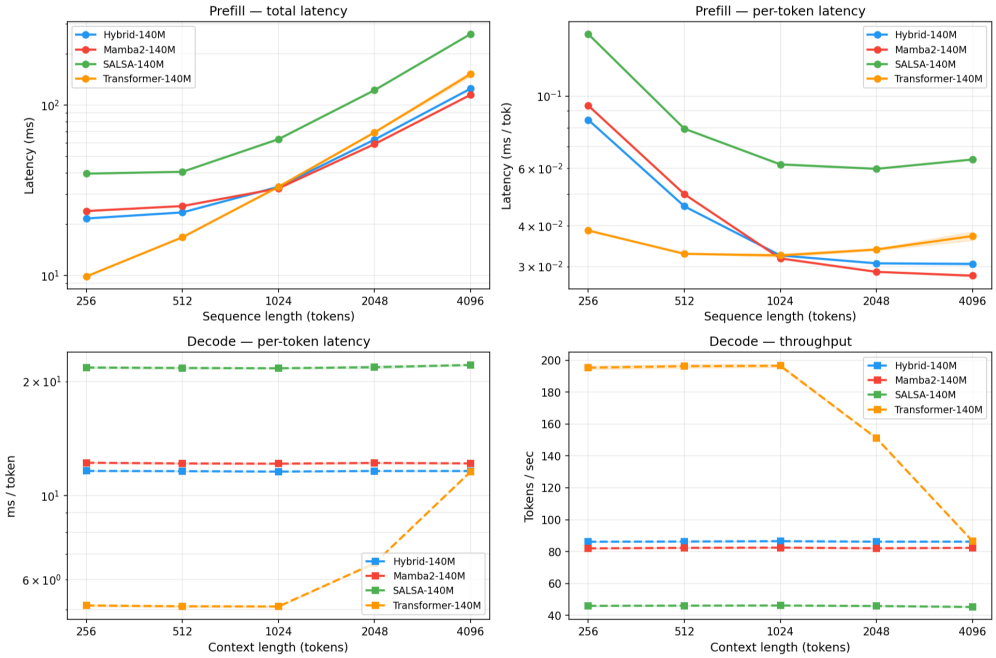


Figure 4. Wall-clock latency benchmark for pretrained 140M-parameter models. We measure prefill latency over a sequence of length T and autoregressive decode latency while generating T additional tokens. Each experiment is repeated eight times, and we report the average. All models use their corresponding cached or recurrent inference state. SALSA is slower than the baselines in the current implementation, suggesting that future work should explore sparsity-aware and routing-aware kernels to better exploit its conditional computation structure.

5. Discussion

5.1. Limitations

Despite these encouraging results, we identify several limitations of the current approach worth noting. While SALSA's conditional routing introduces sparsity that could in principle yield faster and cheaper inference, our implementation does not yet realize these gains (Section 4.4), and maintaining both SSM state and a full KV cache increases memory cost relative to either pure architecture. We view routing-aware kernels and more efficient state management as natural directions for future work. We further believe that research on the selection and enforcement of optimal attention budgets is warranted. The current routing rate is unconstrained at inference time, and although observed rates remained stable across our evaluations, a mechanism for bounding attention usage under a given compute budget would strengthen the architecture's practical applicability.

5.2. Conclusion

We have presented SALSA, a dynamic hybrid architecture that replaces fixed primitive interleaving with learned, per-token routing. SALSA's limitations are nontrivial, yet its consistent improvements over the static hybrid across scales, tasks, and sequence lengths provide evidence that dynamic primitive allocation offers advantages that fixed interleaving struggles to capture. We anticipate that continued progress on dynamic routing, alongside advances in the underlying recurrent and attention primitives themselves, will further strengthen the case for adaptive hybrid architectures.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Bick, A., Xing, E. P., and Gu, A. Understanding the skill gap in recurrent language models: The role of the gather-and-aggregate mechanism. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 4324–4344. PMLR, 2025.

Bisk, Y., Zellers, R., Le Bras, R., Gao, J., and Choi, Y. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020.

Cabannes, L., Beck, M., Szilvasy, G., Douze, M., Lomeli, M., Copet, J., Mazaré, P.-E., Synnaeve, G., and Jégou, H. Short window attention enables long-term memorization. *arXiv preprint arXiv:2509.24552*, 2025.

Chen, S., Wong, S., Chen, L., and Tian, Y. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Dao, T. and Gu, A. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. PMLR, 2024.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, 2022.

Ding, J., Ma, S., Dong, L., Zhang, X., Huang, S., Wang, W., Zheng, N., and Wei, F. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023.

Dong, X., Fu, Y., Diao, S., Byeon, W., Chen, Z., Mahabaleshwarkar, A. S., Liu, S.-Y., Van Keirsbilck, M., Chen, M.-H., Suhara, Y., Lin, Y. C., Kautz, J., and Molchanov, P. Hymba: A hybrid-head architecture for small language models. In *International Conference on Learning Representations*, 2025.

Elbayad, M., Gu, J., Grave, E., and Auli, M. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, September 2024. URL https://doi.org/10.5281/zenodo.13792558.

Glorioso, P., Anthony, Q., Tokpanov, Y., Whittington, J., Pilault, J., Ibrahim, A., and Millidge, B. Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.16712*, 2024.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling (COLM)*, 2024.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Hwang, S., Wang, B., and Gu, A. Dynamic chunking for end-to-end hierarchical sequence modeling. In *International Conference on Learning Representations*, 2026.

Kamradt, G. Needle in a haystack – pressure testing LLMs. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pp. 5156–5165, 2020.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Lieber, O., Lenz, B., Bata, H., Cohen, G., Osin, J., Dalmedigos, I., Safahi, E., Meirom, S., Belinkov, Y., Shalev-Shwartz, S., Abend, O., Alon, R., Asida, T., Bergman, A., Glozman, R., Gokhman, M., Manevich, A., Ratner, N., Rozen, N., Shwartz, E., Zusman, M., and Shoham, Y. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1525–1534, 2016.

Peng, H., Kasai, J., Pappas, N., Yogatama, D., Wu, Z., Kong, L., Schwartz, R., and Smith, N. A. ABC: Attention with bounded-memory control. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pp. 7469–7483, 2022.

Press, O. and Wolf, L. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 157–163, 2017.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

Sakaguchi, K., Le Bras, R., Bhagavatula, C., and Choi, Y. WinoGrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Shazeer, N. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008, 2017.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5797–5808, 2019.

Waleffe, R., Byeon, W., Riach, D., Norick, B., Korthikanti, V., Dao, T., Gu, A., Hatamizadeh, A., Singh, S., Narayanan, D., Kulshreshtha, G., Singh, V., Casper, J., Kautz, J., Shoeybi, M., and Catanzaro, B. An empirical study of Mamba-based language models. *arXiv preprint arXiv:2406.07887*, 2024.

Wang, J., Paliotta, D., May, A., Rush, A. M., and Dao, T. The mamba in the llama: Distilling and accelerating hybrid models. In *Advances in Neural Information Processing Systems*, pp. 62432–62457, 2024.

Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning*, pp. 56501–56523, 2024a.

Yang, S., Wang, B., Zhang, Y., Shen, Y., and Kim, Y. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, pp. 115491–115522, 2024b.

Yang, S., Kautz, J., and Hatamizadeh, A. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. 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.

Zhang, B. and Sennrich, R. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, pp. 12381–12392, 2019.

A. Architecture

Algorithm 1 SALSA Layer Forward Pass. All normalization layers are RMSNorm. The hard routing branch uses straight-through estimation to preserve gradient flow through the discrete threshold $\eta = 0.5$. During training, the temperature τ is annealed toward zero and hard routing is activated after an initial warmup fraction of training steps, encouraging increasingly decisive gating.

Require: Input $x \in \mathbb{R}^{B\times T\times D}$, temperature τ, hard_routing flag
$x_1 \leftarrow \mathrm{Norm}_1(x)$
$s \leftarrow \mathrm{Mamba2}(x_1)$
$\ell \leftarrow \mathrm{Router}(x + s)$ ▷ gate logits
$p \leftarrow \sigma(\ell/\tau)$ ▷ continuous gate
if hard_routing **then**
  $g_{\text{hard}} \leftarrow \mathbb{I}[p > \eta]$
  $g \leftarrow g_{\text{hard}} + (p - \mathrm{sg}(p))$ ▷ straight-through
else
  $g \leftarrow p$
end if
$a \leftarrow \mathrm{Attention}(\mathrm{Norm}_2(x + s))$
$m \leftarrow \mathrm{MLP}(\mathrm{Norm}_3(x + a))$
$a' \leftarrow a + m$
$y \leftarrow x + s + g \odot a'$
return y

A.1. Parameter Counts and Compute

The compute efficiency of SALSA models is controlled by the average activation rate of the attention gates across layers and tokens. We refer to this quantity as routing rate p, where p is the fraction of layer-token positions at which the attention branch is active. In our experiments, we generally train SALSA models to operate near $p \approx 0.2$ (see Figure 5).

In Table 2, we characterize how decode-time FLOPs and active parameters per token vary as a function of p. This isolates the effect of the routing rate on inference cost. Our SALSA models always maintain comparable compute costs to Hybrid models at the same active parameter count.

Table 2. Decode-time compute cost, measured in GFLOPs, for 1.3B-parameter models across sequence lengths. We compare dense Hybrid, Mamba-2, and Transformer baselines against SALSA variants with different average attention activation rates p. Lower values of p reduce compute by activating attention less frequently, while $p = 1$ corresponds to the fully active attention setting. We bold the SALSA configuration with $p = 0.2$, which most closely matches the activation rate used in our main experiments. We also include the expected active parameters of each model.

Model	Active Params	GFLOPs by Sequence Length				
		512	1024	2048	4096	8192
Hybrid	1.34B	471.7	969.2	2041.4	4495.2	10639.6
Mamba-2	1.34B	163.8	327.7	655.4	1310.7	2621.4
Transformer	1.31B	1395.3	2893.7	6199.6	14048.5	34694.1
SALSA ($p = 0$)	1.15B	164.9	329.8	659.6	1319.2	2638.5
SALSA ($p = 0.1$)	1.25B	276.8	570.7	1210.1	2695.1	6489.7
SALSA ($p = 0.2$)	**1.35B**	**388.6**	**811.6**	**1760.6**	**4071.0**	**10341.0**
SALSA ($p = 0.5$)	1.65B	724.2	1534.3	3412.1	8198.6	21894.8
SALSA ($p = 1$)	2.15B	1283.5	2738.7	6164.6	15078.0	41151.1

B. Training details

Data. All models are trained on FineWeb-Edu (`HuggingFaceFW/fineweb-edu`) using the GPT-2 byte-level BPE tokenizer (vocabulary size 50,257). Documents are encoded with the EOS token appended, concatenated, and

Table 3. Decode compute across sequence lengths for baseline and SALSA models at two model scales. GFLOPs are reported as a function of context length. SALSA compute scales with the attention activation rate p, interpolating between Mamba-2-like compute at $p = 0$ and dense-attention compute at $p = 1$. Active parameter counts reflect the parameters used at the corresponding activation rate.

Model	**Active Params**	**GFLOPs by Sequence Length**				
		512	1024	2048	4096	8192
Hybrid	138.9M	73.32	151.5	322.3	721.8	1752.9
Mamba-2	136.5M	51.4	102.8	205.6	411.2	822.3
Transformer	137.7M	152.5	327.5	745.2	1851.3	5145.6
SALSA ($p = 0$)	123.6M	51.81	103.6	207.2	414.5	828.9
SALSA ($p = 0.1$)	131.4M	61.58	126.7	267.6	591.8	1410.4
SALSA ($p = 0.2$)	139.2M	71.34	149.8	327.9	769.2	1991.9
SALSA ($p = 0.5$)	162.6M	100.7	219.0	508.9	1301.3	3736.4
SALSA ($p = 1$)	201.5M	149.5	334.4	810.6	2188.1	6643.9

(a) 140M-scale models.

Model	**Active Params**	**GFLOPs by Sequence Length**				
		512	1024	2048	4096	8192
Hybrid	788.9M	296.6	612.6	1302.5	2914.2	7065.4
Mamba-2	794.8M	123.8	247.6	495.2	990.3	1980.7
Transformer	785.1M	844.9	1770.3	3862.7	9013.8	23181.6
SALSA ($p = 0$)	671.0M	123.7	247.4	494.7	989.5	1978.9
SALSA ($p = 0.1$)	727.6M	188.2	389.3	830.1	1866.4	4557.3
SALSA ($p = 0.2$)	784.2M	252.7	531.2	1165.5	2743.2	7135.8
SALSA ($p = 0.5$)	954.1M	446.3	956.9	2171.6	5373.9	14871.0
SALSA ($p = 1$)	1.24B	768.8	1666.5	3848.4	9758.3	27763.0

(b) 780M-scale models.

packed into non-overlapping windows of length $T = 2048$ with a rolling token buffer; document boundaries are not respected within a window. The dataset is consumed in streaming mode and sharded across data-parallel ranks via `datasets.distributed.split_dataset_by_node`, so each rank sees a disjoint document stream. We use a `StatefulDataLoader` so that streaming dataloader state can be checkpointed and resumed bit-exactly after preemption. The same train and validation splits (held out as a streaming subset of FineWeb-Edu) are used for all runs.

Token Budget. Every run uses a global batch size of 256 sequences of length 2048, i.e. $524{,}288$ tokens per optimizer step, and is trained for $T_{\max} = 100{,}000$ steps, corresponding to a fixed budget of approximately 5.24×10^{10} (≈ 52B) tokens per model. Identical token budgets across architectures and scales let us interpret all comparisons as compute- and data-matched.

Optimization. We optimize with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$), weight decay 0.1, and global gradient clipping at norm 1.0. The learning rate follows a linear warmup over $2{,}000$ steps to a peak of 3×10^{-4}, then a cosine decay to a floor of 3×10^{-5} (10% of the peak) over the remaining steps:

$$\eta(t) = \begin{cases} \eta_{\max} \cdot \dfrac{t+1}{T_{\text{warm}}}, & t < T_{\text{warm}} \\ \eta_{\min} + \dfrac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\pi \dfrac{t - T_{\text{warm}}}{T_{\max} - T_{\text{warm}}}\right)\right), & t \geq T_{\text{warm}} \end{cases} \tag{1}$$

Mixed Precision. Training is performed with `torch.amp.autocast("cuda")` and a `GradScaler`; weights and optimizer state remain in FP32, while activations and matmuls execute in FP16. All runs use a single global seed of 1337 for model initialization, dataloader sharding, and shuffling.

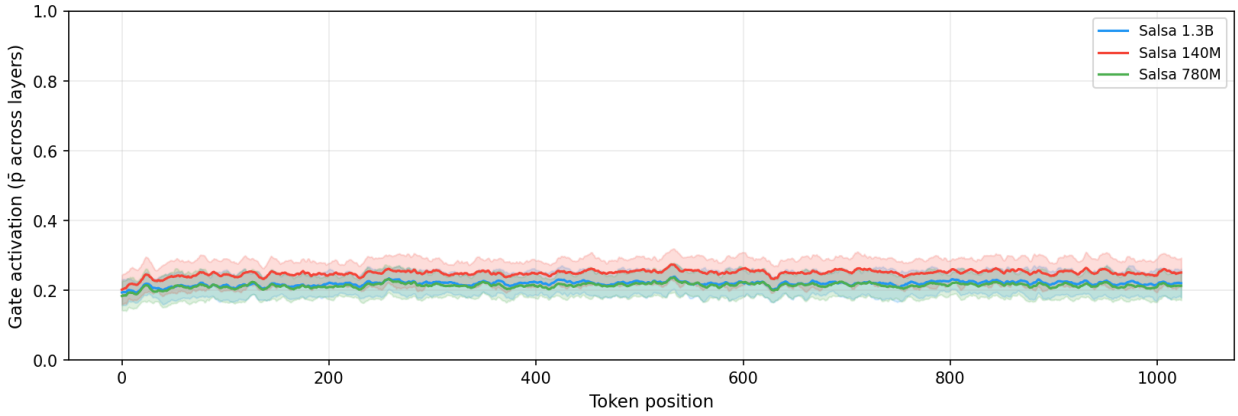


Figure 5. Mean gate activation over token position for pretrained SALSA models. For each token, we average the binary attention-gate activations across layers, yielding a per-token estimate of the routing rate p. The models are evaluated on a held-out length-1024 sequence from the FineWeb-Edu training distribution. Across scales, SALSA maintains activation rates close to the target value of $p \approx 0.2$, with relatively low variance across token positions. Larger models exhibit slightly lower mean gate activation, suggesting that the target compute budget is preserved or improved as scale increases.

Evaluation During Training. Every 400 optimizer steps we estimate validation loss and perplexity over 50 mini-batches of size 8 on the held-out FineWeb-Edu split. All reported runs train for the full $T_{\max}$.

Distributed System. Training is run with `torchrun` under SLURM using PyTorch DistributedDataParallel (DDP) with NCCL. The number of nodes scales with model size: 140M runs use 2 nodes, 780M runs use 4 nodes, and 1.3B runs use 8 nodes, with 8 NVIDIA H100 (80 GB) GPUs per node (16, 32, and 64 H100s respectively). The global batch size of 256 is held constant by adjusting the per-rank batch size accordingly (16, 8, and 4 sequences per rank). Checkpoints (model, optimizer, `GradScaler`, and dataloader state) are written every $2{,}000$ steps to enable bit-exact resumption.

B.1. Model Configurations

All four architectures share the same residual-stream backbone: pre-norm with RMSNorm ($\varepsilon = 10^{-5}$), SwiGLU feed-forward blocks with MLP ratio 4, tied input/output embeddings, and $\mathcal{N}(0, 0.02)$ initialization. Architecture-specific details are as follows:

- **Transformer.** Standard causal self-attention with rotary position embeddings (RoPE, base $10{,}000$).
- **Mamba-2.** Model dimension $d_{\text{model}} \in \{768, 1536, 2048\}$ across the three scales, state dimension d_{state} between 32 and 128 depending on scale, with conv width $d_{\text{conv}} = 4$ and an expansion factor of 2.
- **Hybrid.** Interleaves Mamba-2 and attention blocks at a $1{:}7$ ratio, matching recent hybrid recipes. Attention blocks use the same RoPE configuration as the Transformer baseline.
- **SALSA.** Shares the Hybrid's SSM backbone but replaces fixed interleaving with a learned per-token router. Each block contains both an SSM and an attention pathway; the router determines per-token whether the attention output is incorporated.

B.2. SALSA training recipe

Routing Mechanism. The router produces continuous gate values early in training, then anneals from continuous to discrete (binary) gating using a temperature schedule. At all three scales, the transition occurs after the first 20% of training steps (`hard_routing_after_frac = 0.2`). The per-step router temperature is communicated to all blocks via the model's `set_train_step` hook. Validation uses the same routing mode (continuous or binary) that the model is currently in at training time, so reported validation perplexity is consistent with the optimization objective.

Auxiliary Losses. Two auxiliary losses are added to the language-modeling loss before backpropagation: an entropy regularizer encouraging decisive routing, and a layerwise ℓ_2 penalty encouraging the routing rate to stay near the 20% target. The coefficients are 0.25 (ℓ_2) and 0.01 (entropy) at 140M and 780M, and 0.30 (ℓ_2) and 0.01 (entropy) at 1.3B. No additional reweighting is applied.

C. Ablations

All ablations are conducted at the 140M scale on 30B tokens of FineWeb-Edu (30K steps) and compared on validation perplexity. Each experiment uses the best configuration from all previous ablation stages as its baseline, varying only the dimension under study.

C.1. Router Design and Input

We ablate the router input representation and router capacity jointly. The router input determines what information the routing decision is based on: `ssm_out` (the SSM output), `x` (the residual stream), `sum` (their sum), or `concat` (their concatenation). We test two router sizes: a simple single-layer linear projection and a larger MLP with one hidden layer. Results are shown in Table 4.

Table 4. Router design ablation. Validation perplexity (↓) for combinations of router input and router capacity. **Bold** indicates the selected configuration.

Router	`ssm_out`	`x`	`sum`	`concat`
Simple (linear)	16.90	16.94	16.91	16.85
Large (MLP)	16.69	16.57	**16.57**	16.61

The larger router consistently outperforms the simple projection across all inputs, and the residual stream (`x`) and sum inputs perform best with the larger router. The residual stream and sum inputs are statistically indistinguishable with the large router (16.57 each); we select sum so that the routing decision has access to both the residual stream and the SSM's compressed state.

C.2. Attention Input

Given the selected router configuration, we ablate what input is provided to the attention pathway. We vary the source for queries (q) and key-value pairs (kv) independently between the residual stream (`x`), the SSM output (`ssm_out`), and their sum (`sum`). Results are shown in Table 5.

Table 5. Attention input ablation. Validation perplexity (↓) for different attention input configurations. **Bold** indicates the selected configuration.

Configuration	Val PPL
q: `sum`, kv: `x`	16.58
q: `ssm_out`, kv: `x`	16.74
all `x`	16.57
all `sum`	**16.44**
all `ssm_out`	16.76

Providing the sum of the residual stream and SSM output to all attention inputs performs best, suggesting that the attention pathway benefits from access to both the original representation and the SSM's compressed context.

C.3. Auxiliary Losses

We ablate the auxiliary loss coefficients used to regularize routing behavior. The ℓ_2 loss penalizes deviation from the target routing rate, and the entropy loss encourages decisive (low-entropy) routing. For this ablation, hard routing is enabled after 20K of 30K steps. We report both validation perplexity and the observed routing rate p (fraction of tokens invoking attention). Results are shown in Table 6.

Without entropy regularization, lower ℓ_2 coefficients yield better perplexity but allow the routing rate to drift well above

Table 6. Auxiliary loss ablation. Validation perplexity (↓) and observed routing rate p under different loss configurations. **Bold** indicates the selected configuration. Higher entropy loss coefficients resulted in unstable training and are omitted.

ℓ_2 coeff	Entropy coeff	Val PPL	Routing rate p
0.10	—	17.20	0.37
0.50	—	17.74	0.20
1.00	—	17.80	0.14
0.10	0.01	17.28	0.29
0.25	**0.01**	**17.50**	**0.21**
0.40	0.005	17.67	0.19

the 20% target. The combination of $\ell_2 = 0.25$ and entropy $= 0.01$ provides a good balance, achieving a routing rate close to the target while maintaining competitive perplexity. We note that higher entropy coefficients caused training instability and are not reported.

C.4. Routing Annealing Schedule

Finally, we ablate when the router transitions from continuous to discrete (binary) gating during training. We report validation perplexity at the end of a full 100K-step run for each schedule. Results are shown in Table 7.

Table 7. Routing annealing schedule ablation. Validation perplexity (↓) as a function of when binary gating is enabled, reported as the fraction of total training steps. **Bold** indicates the selected configuration.

Hard routing after	Val PPL
20% of training	**15.13**
40% of training	15.20
60% of training	15.30
80% of training	15.61

Earlier transitions to binary gating consistently yield lower perplexity, suggesting that the model benefits from learning under the discrete routing objective for as long as possible. However, some initial period of continuous gating is critical for stability: enabling hard routing from step 0 in a separate 780M experiment resulted in a perplexity increase of approximately 50% over baseline, indicating that the router requires a warm-up phase with gradient flow through both pathways before discrete decisions can be made reliably.

D. Full downstream results

D.1. Expanded Language Modeling Evaluations

Tables 8 and 9 report both standard accuracy and norm-adjusted accuracy (acc_n) for tasks where the LM-Eval harness provides both metrics. Norm-adjusted accuracy corrects for sequence length bias in multiple-choice evaluation by normalizing log-likelihoods by completion length.

The main body reports results using the primary metric for each task. Here we note that the norm-adjusted results reinforce the same conclusions: at 1.3B, SALSA achieves the best norm-adjusted accuracy on HellaSwag (55.7 vs. 54.6), PIQA (72.7 vs. 71.8), ARC-Easy (67.2 vs. 65.7), and ARC-Challenge (40.6 vs. 37.5), with the ARC-Challenge gap of 3.1 points being the largest single-task improvement over the Hybrid at any scale. The one exception is OBQA acc_n at 1.3B, where the Hybrid leads (40.4 vs. 39.0); this is consistent with the small OBQA margins noted in the main text and does not alter the overall picture.

At smaller scales the pattern holds as well. At 780M, SALSA leads on norm-adjusted accuracy for 4 of 5 applicable tasks, with the Transformer narrowly matching on ARC-Easy acc_n (62.0). At 140M, SALSA sweeps all norm-adjusted metrics. Across all three scales, the average accuracy advantage of SALSA over the Hybrid is consistent whether computed from standard or norm-adjusted metrics, providing additional confidence that the improvements are not artifacts of evaluation methodology.

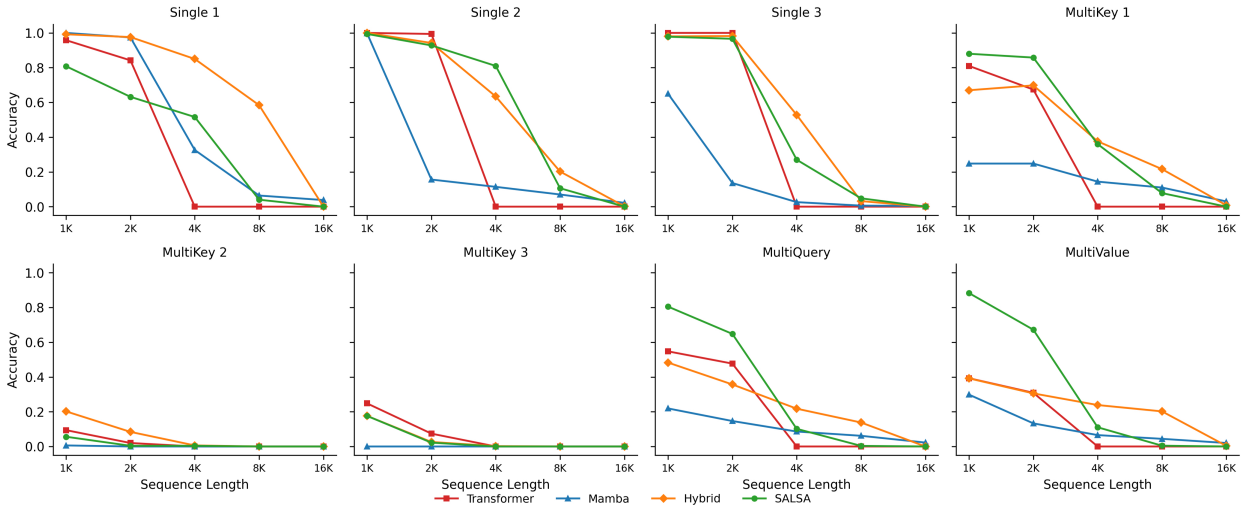


Figure 6. RULER NIAH task accuracy for 1.3B models across sequence lengths. All models degrade beyond the 2K training length, with Hybrid retaining the strongest retrieval at mid-lengths due to its fixed attention layers.

D.2. Expanded Long Context Evaluations

We further evaluate retrieval capability using the NIAH (Needle-in-a-Haystack) tasks from the RULER benchmark (Figure 6). On simpler single-needle tasks at short lengths (1K–2K), SALSA is competitive with or outperforms the Hybrid and Transformer. However, all models that rely on positional encodings degrade rapidly beyond the 2K training length, and SALSA degrades faster than the Hybrid on several tasks at mid-lengths (4K–8K). We attribute this to RoPE degradation disproportionately affecting SALSA: because the router learns to invoke attention selectively, the attention layers that do fire may encounter positions further from those seen during training than the Hybrid's fixed attention layers, which see every token and thus build more robust positional representations. The Hybrid's consistent attention at every eighth layer provides a more stable retrieval mechanism at extrapolated lengths. We note that all models, including the Transformer, effectively collapse on the harder multi-key and multi-query tasks beyond 4K, indicating that this is largely a positional encoding limitation rather than an architecture-specific weakness.

E. Efficiency

We report wall-clock inference latency for two phases: **prefill** (processing a prompt of T tokens in a single forward pass) and **decode** (autoregressive generation of T additional tokens one step at a time). All measurements are taken under `torch.no_grad()` with models in `eval()` mode.

Input data. To avoid measuring on synthetic or unrepresentative inputs, all token sequences are drawn from real documents in **FineWeb-Edu** (`HuggingFaceFW/fineweb-edu`, training split, streamed). Sequences are tokenized with the GPT-2 tokenizer, packed to the maximum sequence length under test, and transferred to the GPU before timing begins. A total of $N_{\text{warmup}} + N_{\text{trials}}$ batches are collected once and reused across every model and sequence-length combination in a given run.

Timing Each timed region is wrapped with `torch.cuda.synchronize()` calls immediately before and after to ensure all GPU kernels have completed. Wall-clock time is recorded with `time.perf_counter()`. We report the mean over N_{trials} timed trials per (model, sequence length) pair.

Prefill A single forward pass over a batch of B sequences of length T is timed. The first N_{warmup} batches are used as warmup (GPU warm-up and cache priming) and are excluded from reported times. Reported latency is the total wall-clock time of the forward pass in milliseconds, from which we derive ms/token as t_{ms}/T and throughput as $B \cdot T \,/\, (t_{\text{ms}}/1000)$ tokens per second.

Decode Decode is measured at batch size $B = 1$. All models implement `prefill()` and `decode_step()`, enabling $O(1)$-per-step autoregressive generation via cached state:

- **Transformer** uses a standard growing KV cache (keys and values appended each step; attention is $O(T)$ per step in sequence length).
- **Mamba-2** uses the Mamba recurrent state (`conv_state`, `ssm_state`) seeded during prefill via `InferenceParams`; each decode step calls `Mamba2.step()` for true $O(1)$ recurrent execution.
- **Hybrid** (interleaved Mamba-2 and Transformer blocks) maintains a per-layer cache: Mamba blocks use their recurrent state and Transformer blocks use a growing KV cache.
- **SALSA** combines the Mamba recurrent state with a growing KV cache for the conditional attention path. At each decode step the router evaluates a gate: tokens for which the gate is closed (logit ≤ 0) skip the attention and MLP paths entirely; only the Mamba recurrent step executes. Keys and values are still appended to the cache on every step so that future attending tokens see the full context.

Prefill (seeding the cache) is performed *outside* the timed region. Timing begins immediately after the first prefill token is selected (greedy decoding, `argmax` over the final prefill logit) and ends after N_{dec} additional tokens have been generated. Reported latency is the total time for N_{dec} decode steps in milliseconds; per-token latency is $t_{\text{ms}}/N_{\text{dec}}$ and throughput is $N_{\text{dec}} \,/\, (t_{\text{ms}}/1000)$ tokens per second.

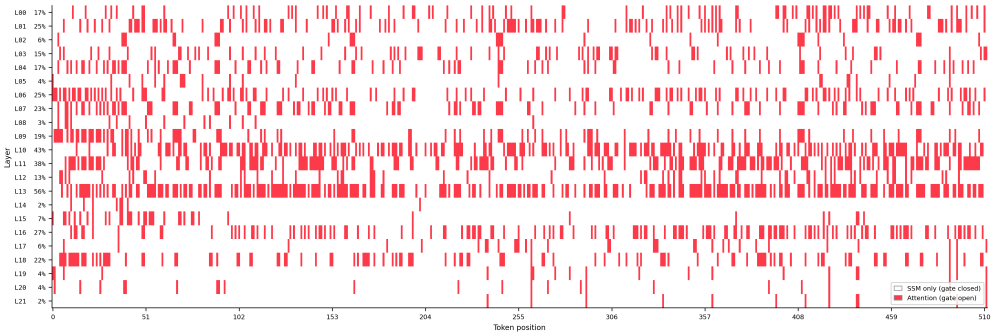
Hyperparameters Unless stated otherwise, experiments use the following defaults:

Parameter	Value
Warmup passes (N_{warmup})	3
Timed trials (N_{trials})	8
Prefill batch size (B)	4
Decode batch size	1
Decode tokens (N_{dec})	equal to context length T
Sequence lengths (T)	256, 512, 1024, 2048, 4096
Tokenizer	GPT-2
Input data	FineWeb-Edu (streaming, train split)

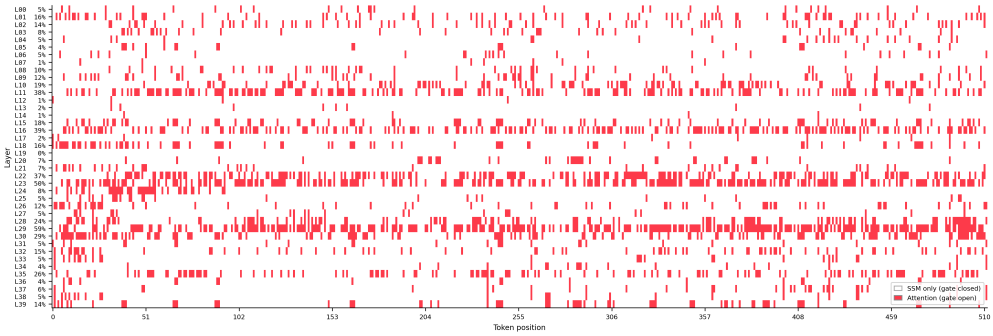
GPU memory is explicitly freed between checkpoints (`torch.cuda.empty_cache()`) to prevent cross-model interference. All experiments are run on a single NVIDIA H100 80GB GPU.

F. Additional Routing Visualizations

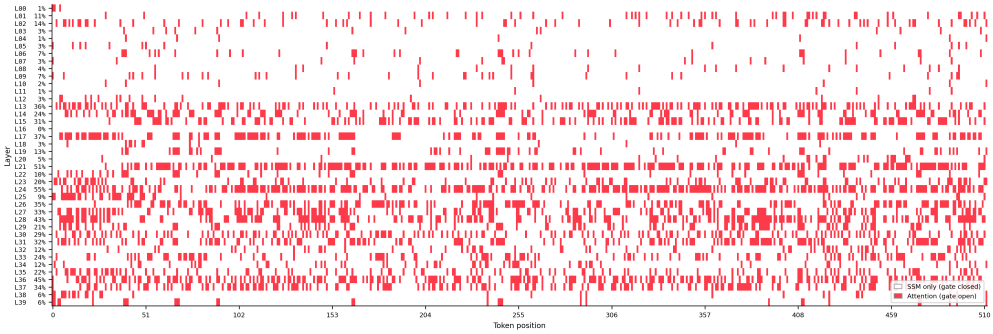
F.1. Routing Patterns Across Model Scales



(a) 140M parameters.



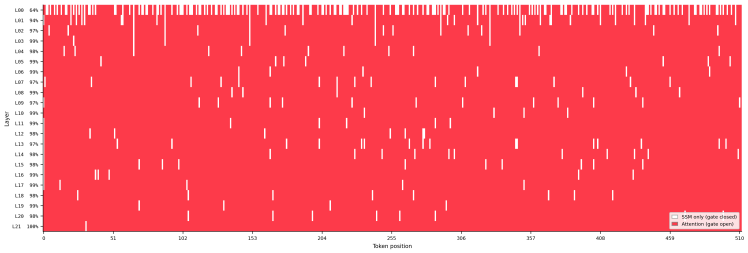
(b) 780M parameters.



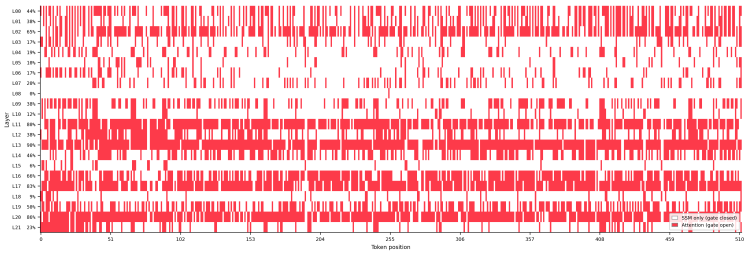
(c) 1.3B parameters.

Figure 7. Routing patterns across model scales. Layer-banded sparsity emerges at all scales, with attention-dense layers consistently appearing in similar regions of the network depth.

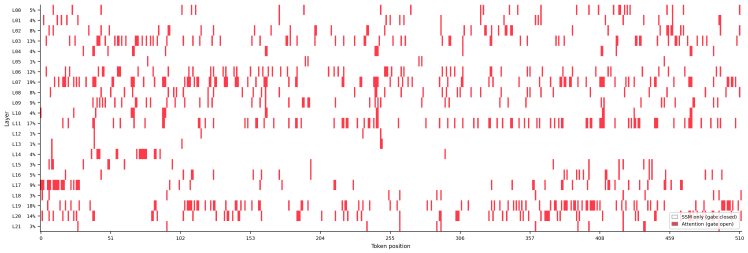
F.2. Routing Patterns Across Auxiliary Loss Coefficients



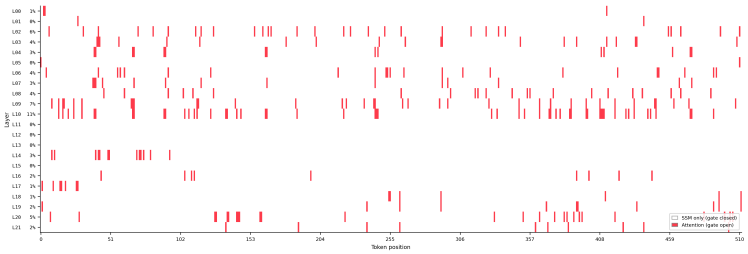
(a) $\alpha = 0.00$.



(b) $\alpha = 0.03$.



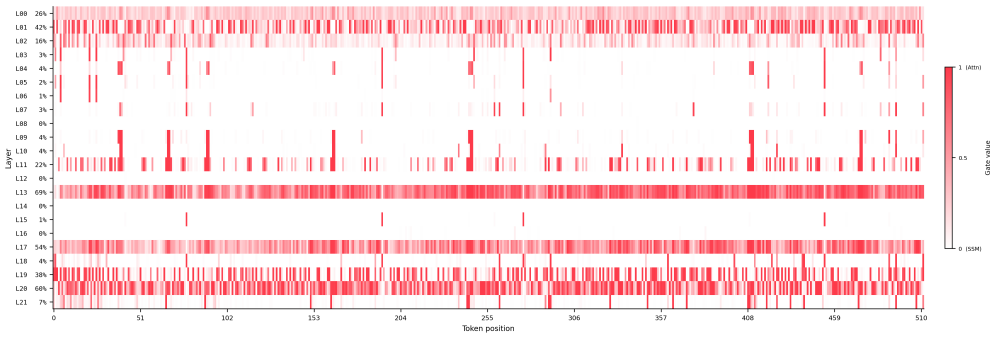
(c) $\alpha = 0.10$.



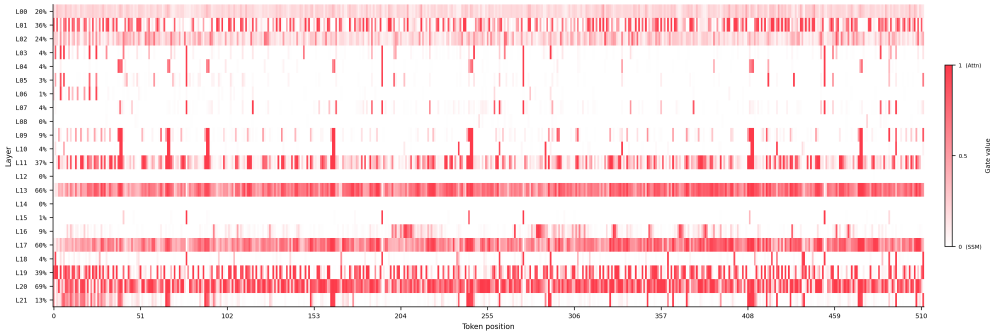
(d) $\alpha = 0.50$.

Figure 8. Routing patterns across auxiliary loss coefficients α. As α increases, overall attention usage decreases, but the layer-banded structure is preserved: attention-dense layers remain attention-dense, while sparsely-routed layers become further suppressed.

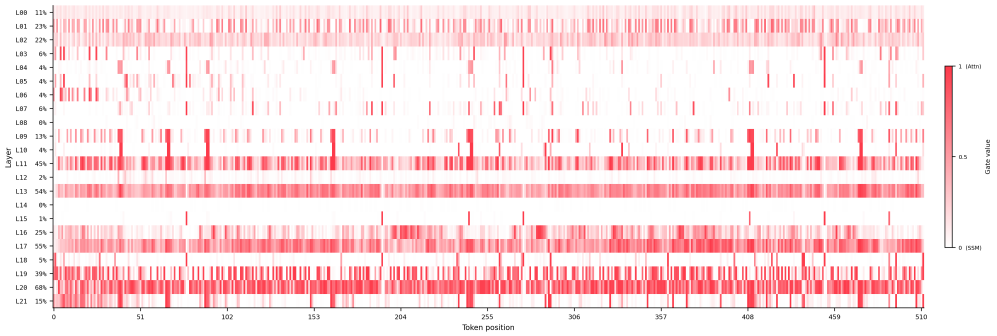
F.3. Emergence of Routing Patterns During Training



(a) Step 2,000 (soft routing).

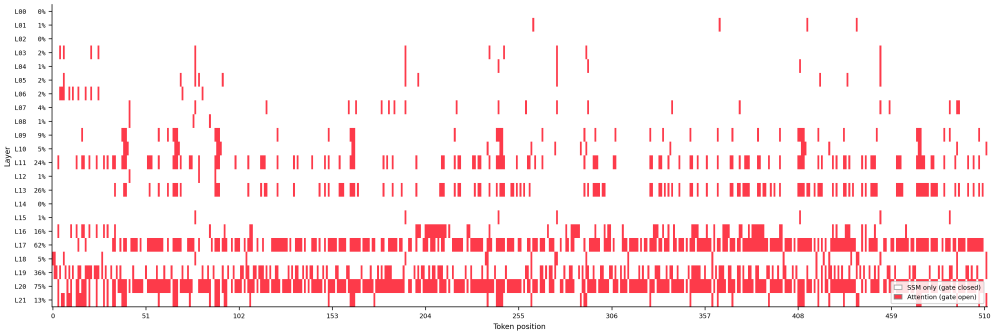


(b) Step 4,000 (soft routing).

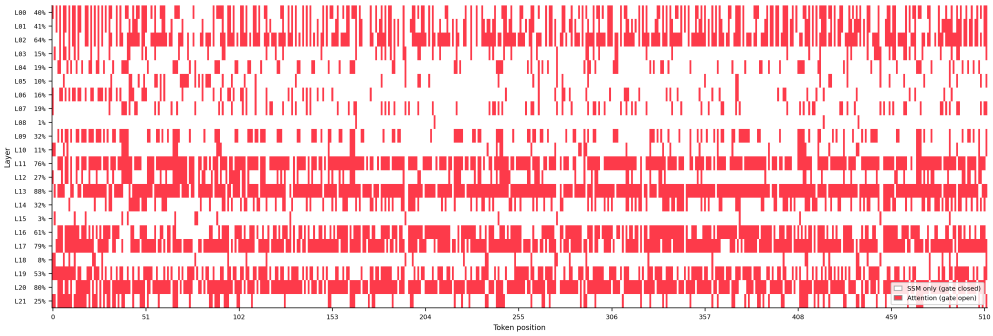


(c) Step 10,000 (soft routing).

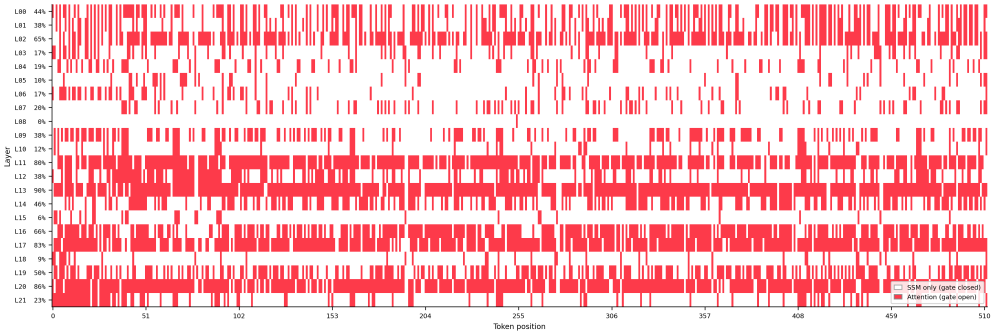
Figure 9. Routing patterns during the soft-routing warmup phase (140M model). Layer banding is already visible early in training, before the hard-routing transition.



(a) Step 20,000 (hard routing enabled).



(b) Step 50,000 (hard routing).



(c) Step 100,000 (hard routing).

Figure 10. Routing patterns after the hard-routing transition (140M model). Layer banding sharpens as the model commits to discrete routing decisions; the same set of attention-dense layers identified during the soft phase (Figure 9) remains attention-dense.

Table 8. Pretraining and downstream evaluation results (part 1). **Bold** indicates best, underline indicates second best within each model size.

Model	FW-Edu ppl ↓	LAMB ppl ↓	LAMB acc ↑	HellaS acc ↑	HellaS acc_n ↑	PIQA acc ↑	PIQA acc_n ↑	Avg acc ↑
140M parameters								
Transformer	16.31	53.70	31.5	31.2	35.6	65.1	63.5	41.0
Mamba	16.24	53.05	29.1	31.6	36.1	65.8	65.4	40.9
Hybrid	15.91	46.58	31.9	31.3	36.6	65.5	64.8	41.4
SALSA	**15.10**	**45.42**	**32.5**	**32.8**	**37.8**	**66.1**	**65.8**	**42.5**
780M parameters								
Transformer	11.16	18.90	41.6	39.4	49.9	71.0	70.8	49.0
Mamba	11.01	21.94	38.5	39.2	49.8	70.7	71.1	48.8
Hybrid	10.85	18.91	41.7	39.5	50.7	**71.8**	71.1	49.8
SALSA	**10.51**	**16.57**	**44.6**	**40.7**	**52.1**	71.7	**71.3**	**51.1**
1.3B parameters								
Transformer	10.20	15.84	44.0	41.5	53.6	71.5	71.7	51.4
Mamba	10.27	17.47	41.4	41.7	53.2	**72.6**	71.9	50.9
Hybrid	9.99	15.00	45.7	42.4	54.6	72.1	71.8	52.0
SALSA	**9.72**	**13.93**	**46.3**	**42.8**	**55.7**	72.5	**72.7**	**53.5**

Table 9. Downstream evaluation results (part 2). **Bold** indicates best, underline indicates second best within each model size.

Model	ARC-e acc ↑	ARC-e acc_n ↑	ARC-c acc ↑	ARC-c acc_n ↑	WinoG acc ↑	OBQA acc ↑	OBQA acc_n ↑
140M parameters							
Transformer	57.7	51.6	**23.8**	27.1	50.9	19.2	31.8
Mamba	57.1	49.4	22.4	26.1	51.0	**21.0**	32.2
Hybrid	57.8	50.5	23.5	26.9	50.0	20.8	32.0
SALSA	**60.7**	**51.8**	23.4	**28.6**	**51.4**	20.2	**32.8**
780M parameters							
Transformer	68.4	**62.0**	29.9	33.1	54.7	24.6	38.0
Mamba	68.7	**62.0**	31.0	32.2	54.1	**27.4**	38.0
Hybrid	67.6	61.2	**32.2**	34.6	55.3	27.0	37.8
SALSA	**68.9**	61.9	**32.2**	**36.6**	**56.9**	26.8	**39.4**
1.3B parameters							
Transformer	70.8	63.7	33.4	35.3	56.5	27.8	38.8
Mamba	70.7	64.7	34.4	36.2	54.4	27.8	39.2
Hybrid	71.0	65.7	34.1	37.5	54.9	28.4	**40.4**
SALSA	**72.6**	**67.2**	**37.6**	**40.6**	**57.5**	**29.2**	39.0