Beyond Parameter Scaling: Recursive Computation Enables Associative Recall in State Space Models

Adam Jacuch
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
adam.jacuch@stonybrook.edu

Bilal El Jamal
Department of Electrical and Computer Engineering
Stony Brook University
Stony Brook, NY 11794
bilal.eljamal@stonybrook.edu

Alex Doboli
Department of Electrical and Computer Engineering
Stony Brook University
Stony Brook, NY 11794
Alex.Doboli@stonybrook.edu

Abstract

Scaling sequence models usually means adding parameters: wider hidden states, larger recurrence states, or deeper stacks of untied layers. This paper asks whether some sequence-modeling failures instead reflect insufficient algorithmic time: the model is forced to perform storage, routing, and retrieval in a single pass. This work studies recursive depth, a parameter-efficient scaling axis that increases sequence-mixing computation while keeping the trainable parameter count fixed. The proposed method repeatedly applies a shared recurrent block over a dynamically updated value stream, so later passes can refine routing decisions using context produced by earlier passes. Recursive depth produces large gains on routing-intensive synthetic tasks. On MQAR-Hard, a 15M-parameter single-pass model achieves 55.2% accuracy, while the same parameter-matched model reaches 88.2% with two recursive steps and 99.5% with three. In a scaling-axis comparison, recursive depth reaches 55.4% MQAR-Hard accuracy at 1.10$\times$ train FLOPs, compared with 46.7% for width scaling and 39.8% for physical-depth scaling under the same compute budget. The effect also transfers to language modeling: at 180M parameters and 10B FineWeb-Edu tokens, recursive depth improves validation perplexity from 15.81 to 15.54 in our architecture and from 14.28 to 14.17 when applied to Mamba-2, without increasing parameter count or optimizer state. Memory and compute profiling show that checkpointing controls the activation cost, while strict FLOP-matched checkpoints expose the expected tradeoff between additional iterative computation and fewer observed training tokens. These results suggest that iterative computation with shared parameters is a useful and underexplored scaling axis for sequence models, especially when precise associative routing is the bottleneck.

1 Introduction

Modern sequence models are usually scaled by adding capacity: wider hidden states, larger recurrence states, or deeper stacks of untied layers [Vaswani et al., 2017, Gu and Dao, 2023, Dao and Gu, 2024]. This strategy has been highly effective, but it entangles two distinct resources: the number of parameters available to store computation, and the amount of computation the model is allowed to perform with those parameters. This paper studies the latter. We ask whether some failures of

Preprint.

efficient sequence models arise not because the model lacks capacity, but because it is forced to complete storage, routing, and retrieval in a single pass.

This question is especially natural for associative recall. Given many key–value bindings in a long context, a model must identify the key matching a query and recover the corresponding value [Arora et al., 2023]. Attention performs this kind of content-based routing directly through all-pairs token interactions. Linear recurrent models and state space models, by contrast, compress the sequence through a causal scan [Gu et al., 2022, Fu et al., 2022, Gu and Dao, 2023]. While this yields favorable scaling, it also requires the model to decide what to store, how to route, and what to retrieve during the same sweep over the sequence. A single pass offers no opportunity to first form a candidate association and then refine subsequent routing decisions based on that retrieved context.

We study *recursive depth* as a simple parameter-efficient way to provide this missing algorithmic time. Instead of increasing width, state size, or the number of untied layers, recursive depth repeatedly applies the same sequence-mixing block N times within each layer. Each pass operates over a value stream refined by earlier passes, allowing later routing decisions to condition on context produced by previous retrievals. Thus recursive depth increases sequence-mixing computation while keeping trainable parameter count and optimizer state fixed.

Our empirical results show that recursive depth is a strong scaling axis for routing-intensive sequence modeling. On the Multi-Query Associative Recall (MQAR) benchmark [Arora et al., 2023], a 15M-parameter single-pass model achieves 55.2% accuracy on the Hard tier, while the same parameter-matched model reaches 88.2% with two recursive steps and 99.5% with three. In a scaling-axis comparison at $1.10\times$ train FLOPs, recursive depth reaches 55.4% MQAR-Hard accuracy, compared with 46.7% for width scaling and 39.8% for physical-depth scaling.

The gains also transfer beyond synthetic recall. At 180M parameters and 10B FineWeb-Edu tokens [Lozhkov et al., 2024], recursive depth improves validation perplexity from 15.81 to 15.54 in our architecture, and from 14.28 to 14.17 when applied to Mamba-2 [Dao and Gu, 2024], without increasing parameter count. Finally, memory and compute profiling show that recursive depth exposes a practical tradeoff: it spends additional local sequence-mixer compute while preserving parameter memory and optimizer state, and gradient checkpointing keeps the activation footprint modest. Together, these results identify recursive depth as an underexplored scaling axis for efficient sequence models: not more parameters, but more iterative computation with the same parameters.

2 Related Work

Efficient sequence models. Transformers provide powerful content-based routing through attention, but their quadratic sequence-length cost has motivated efficient alternatives based on structured state space models, long convolutions, gated recurrence, and selective SSMs [Vaswani et al., 2017, Gu et al., 2022, Gupta et al., 2022, Smith et al., 2023, Fu et al., 2022, Poli et al., 2023, Peng et al., 2023, Sun et al., 2023, De et al., 2024, Gu and Dao, 2023, Dao and Gu, 2024]. These models replace explicit all-pairs attention with linear or subquadratic sequence mixing, often improving long-context efficiency. Building on this line of attention-free sequence modeling, the evaluated base model is used to study a different scaling axis: increasing the number of tied sequence-mixing computations rather than increasing width, physical depth, or recurrence-state size.

Recursive and weight-tied computation. Recursive depth is most closely related to prior work on repeated computation over depth. Universal Transformers and ALBERT apply a shared recurrent transition or transformer block to iteratively refine token representations, while adaptive computation time allows recurrent models to allocate variable computation per input [Dehghani et al., 2019, Graves, 2016]. However, Universal Transformers retain self-attention at every recurrent step, giving each refinement step direct global token-token communication. The architectural setting here is distinctly different: recursive depth is applied inside a SSM-style sequence mixer, where the absence of direct attention-like routing is precisely the bottleneck targeted. Rather than repeatedly applying a full Transformer block, the base model repeatedly applies a shared linear-time recurrent mixer over a refined value stream. This establishes whether additional algorithmic time can improve attention-free recurrent sequence mixing while preserving the parameter footprint.

Adaptive, algorithmic, and implicit-depth models. This approach is also related to Neural GPUs, deep equilibrium models, and other approaches that increase computation through repeated or implicit transformations [Kaiser and Sutskever, 2016, Bai et al., 2019]. These methods study algorithmic recurrence, adaptive halting, or fixed-point computation as general mechanisms for increasing effective depth. In contrast, the base model utilizes a small fixed number of explicit recurrent refinements inside efficient sequence mixers, requiring no learned halting rule or fixed-point solver. Compared with conventional scaling axes, recursive depth increases sequence-mixer compute while keeping trainable parameter count and optimizer state fixed, making the compute–memory tradeoff explicit and directly measurable.

3 Methodology

Recursive depth is implemented as a weight-tied iterative refinement block inside each recurrent sequence-mixing layer. At $N = 1$, the block is a standard single-pass recurrent mixer. At $N > 1$, the same mixer is applied repeatedly over a dynamically updated value stream, increasing sequence-mixing computation without increasing parameter count.

Given layer input x_t, the block first computes a local causal context

$$c_t = \mathrm{SiLU}(\mathrm{DWConv}_{\mathrm{causal}}(x_{\le t})) \,. \tag{1}$$

The block maintains a value stream $v_t^{(n)}$, output accumulator $o_t^{(n)}$, and fetched recurrent state $h_t^{(n)}$, initialized as

$$v_t^{(0)} = c_t, \qquad o_t^{(0)} = c_t, \qquad h_t^{(0)} = 0. \tag{2}$$

At recursive step n, the routing input is

$$r_t^{(n)} = \begin{cases} \mathrm{RMSNorm}(c_t), & n = 1, \\ \mathrm{RMSNorm}(c_t) + G_h\Big(\mathrm{RMSNorm}(h_t^{(n-1)})\Big), & n > 1, \end{cases} \tag{3}$$

where G_h is a learned channelwise gate initialized at zero. The shared core computes

$$\alpha_t^{(n)} = \sigma(W_\alpha r_t^{(n)} + b_\alpha), \qquad \beta_t^{(n)} = \mathrm{SiLU}(W_\beta r_t^{(n)} + b_\beta), \tag{4}$$

with b_α initialized by a linearly spaced bias. The write stream is

$$u_t^{(n)} = v_t^{(n-1)} \odot \beta_t^{(n)} \odot \sqrt{\max\Big(1 - (\alpha_t^{(n)})^2, \epsilon\Big)}. \tag{5}$$

The fetched state is produced by the causal associative scan

$$h_t^{(n)} = \alpha_t^{(n)} h_{t-1}^{(n)} + u_t^{(n)}. \tag{6}$$

This recurrence is evaluated with an associative scan, enabling parallel computation over the sequence.

The fetched state updates both the output accumulator and value stream:

$$o_t^{(n)} = o_t^{(n-1)} + G_o(h_t^{(n)}), \qquad v_t^{(n)} = v_t^{(n-1)} + G_v(h_t^{(n)}), \tag{7}$$

where G_o and G_v are learned channelwise gates, and G_v is initialized at zero. After N steps, $o_t^{(N)}$ is gated by $\mathrm{SiLU}(W_g x_t)$, normalized, projected with small initialization, passed through SiLU, and added residually to the layer input. A standard position-wise MLP is then applied once per layer.

All recurrent-step parameters are shared across recursive steps. Therefore recursive depth changes sequence-mixing compute but not trainable parameter count or optimizer state:

$$|\theta_N| = |\theta_1|, \qquad \mathrm{Cost}_{\mathrm{layer}}(N) \approx N\,\mathrm{Cost}_{\mathrm{mixer}} + \mathrm{Cost}_{\mathrm{MLP}}. \tag{8}$$

For Mamba-2 experiments, the same principle is applied by reusing the Mamba-2 mixer over a refined value stream while preserving physical depth, state expansion, and the 180M parameter budget.

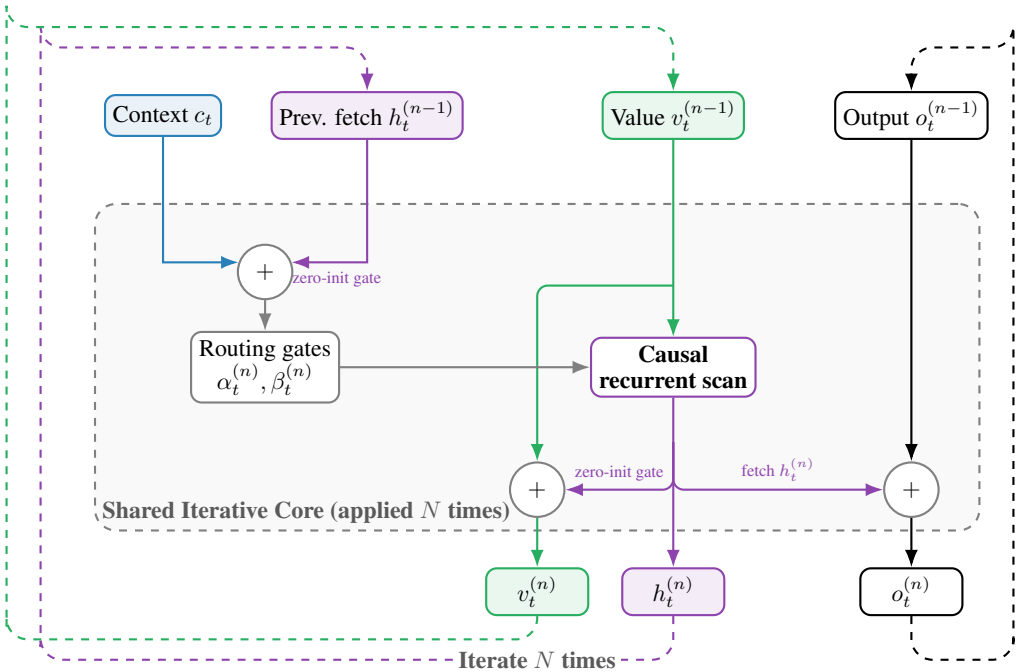

Figure 1: Weight-tied iterative block.

4 Algorithmic Analysis

This section analyzes recursive depth as a general computational principle. The goal is not to characterize a particular implementation detail, but to explain why repeatedly applying a shared sequence mixer can improve associative recall. A single recurrent scan performs one causal compression of the sequence. Recursive depth composes this scan with itself, allowing later passes to operate on representations produced by earlier passes. This gives the model additional *algorithmic time*: more sequential computation over the same context, without increasing the number of trainable parameters.

4.1 Single-Pass Recurrence

Consider a generic causal sequence mixer

$$h_t = \mathcal{R}_\theta(x_t, h_{t-1}), \tag{9}$$

where θ denotes the trainable parameters. A single-pass model applies this recurrence once over the sequence:

$$H^{(1)} = \mathcal{S}_\theta(X), \tag{10}$$

where $\mathcal{S}_\theta$ denotes the full causal scan induced by $\mathcal{R}_\theta$.

This computation is powerful, but it is constrained in an important way: each position must decide how to route information during the same pass in which information is being retrieved. Thus storage, matching, and retrieval are entangled in one causal sweep. For associative recall, this is a severe constraint. The model must use the query to identify a matching historical key and retrieve the associated value, but it has no opportunity to first form a candidate retrieval and then route again conditioned on that candidate.

4.2 Recursive Depth

Recursive depth replaces a single scan with repeated applications of the same scan:

$$H^{(n)} = \mathcal{S}_\theta\left(Z^{(n-1)}\right), \qquad n = 1, \ldots, N, \tag{11}$$

where $Z^{(n-1)}$ is the sequence representation available before recursive step n. A generic refinement update can be written as

$$Z^{(n)} = \Psi\Big(Z^{(n-1)}, H^{(n)}\Big), \tag{12}$$

for some pointwise or local update rule Ψ. The important structural property is that the same sequence mixer $\mathcal{S}_\theta$ is reused at every recursive step. Therefore, increasing N increases computation but not the number of parameters:

$$\#\theta_N = \#\theta_1, \qquad \mathrm{Compute}(N) \propto N. \tag{13}$$

This is distinct from ordinary depth scaling. Physical depth adds new parameterized transformations:

$$H = \mathcal{S}^{(L)}_{\theta_L} \circ \cdots \circ \mathcal{S}^{(1)}_{\theta_1}(X), \tag{14}$$

where each layer has its own parameters. Recursive depth instead performs

$$H^{(N)} = \underbrace{\mathcal{S}_\theta \circ \mathcal{S}_\theta \circ \cdots \circ \mathcal{S}_\theta}_{N \text{ applications}}(X), \tag{15}$$

possibly with intermediate refinement updates. Thus recursive depth is a scaling axis for computation, not parameter count.

4.3 From Myopic to Iterative Routing

The key benefit of recursive depth is that later routing decisions can depend on information retrieved by earlier passes. In a single-pass model, the routing signal is computed only from the input representation available before retrieval:

$$r^{(1)}_t = r_\theta(z^{(0)}_t). \tag{16}$$

With recursive depth, later routing signals may depend on previous retrieved states:

$$r^{(n)}_t = r_\theta\Big(z^{(n-1)}_t, h^{(n-1)}_t\Big), \qquad n > 1. \tag{17}$$

Thus a recursive model can decompose associative recall into an initial scan that forms candidate bindings and later scans that refine retrieval conditioned on those candidates. In this sense, N acts as an algorithmic-depth parameter: it grants the model more routing iterations per token while preserving the parameter footprint.

5 Experiments

The empirical evaluation studies recursive depth under fixed-parameter and fixed-token budgets, and analyzes its resource profile relative to standard width and physical-depth scaling. Unless otherwise stated, baseline comparisons are parameter-matched and token-matched: models have the same number of trainable parameters and are trained on the same token budget using identical tokenizers, dataloaders, optimizers, and learning-rate schedules.

5.1 Multi-Query Associative Recall (MQAR)

The Multi-Query Associative Recall benchmark [Arora et al., 2023] evaluates a model's ability to recover values from many key–value bindings over long contexts. Three MQAR difficulty tiers are defined by jointly scaling sequence length (L), number of key–value pairs (K), number of queries (Q), and vocabulary size (V): Easy ($L = 512$, $K = 64$, $Q = 32$, $V = 256$), Medium ($L = 768$, $K = 96$, $Q = 48$, $V = 384$), and Hard ($L = 1024$, $K = 128$, $Q = 64$, $V = 512$). Recursive depth $N \in \{1, 2, 3\}$ is evaluated at parameter counts of 5M, 10M, and 15M. The $N = 1$ model represents the standard single-pass architecture. For $N > 1$, the same recurrent sequence-mixing parameters are iteratively applied N times per layer.

Figure 2 reports mean final evaluation accuracies over two independent training runs. On MQAR-Easy, the single-pass model reaches near-saturation at 10M parameters, while recursive models solve the task effectively at 5M parameters. On MQAR-Hard at 15M parameters, the single-pass model achieves 55.2% accuracy, whereas recursive depth yields 88.2% ($N = 2$) and 99.5% ($N = 3$).

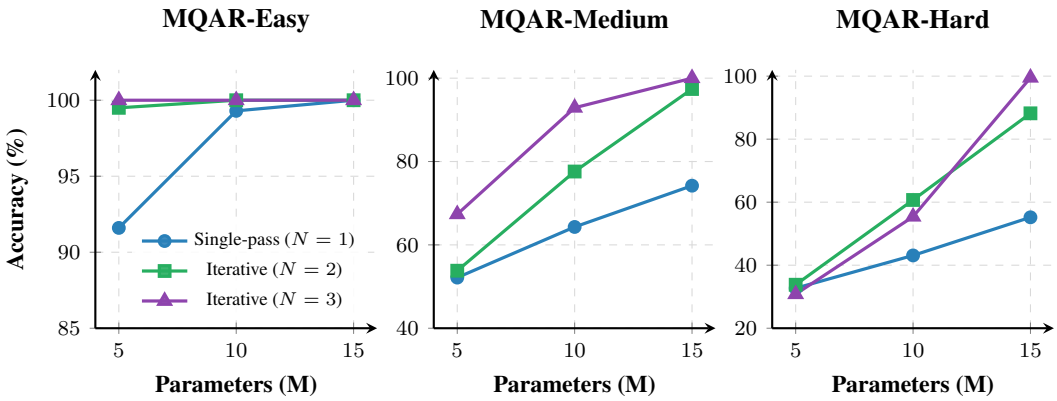

Figure 2: MQAR scaling across difficulty levels.

5.2 Compute-Matched Scaling-Axis Comparison

The MQAR sweep above varies recursive depth while holding the parameter budget fixed. The subsequent evaluation compares recursive depth against conventional width and physical-depth scaling under approximately matched train-step compute. Starting from the same single-pass base model, the experiment scales one architectural axis at a time: hidden width, number of untied layers, or recursive refinement depth. Width- and depth-scaled baselines are selected using an architecture-level train-FLOP proxy to match the recursive $N = 2$ model, and all models are trained on MQAR-Hard with the same tokenizer, optimizer, data distribution, and token budget.

Table 1: Compute-matched scaling-axis comparison on MQAR-Hard.

Scaling axis	Dim	Depth	N	Params	Train FLOPs	MQAR-Hard Acc.
Base single-pass	702	2	1	1.00×	1.00×	43.1%
Recursive depth	702	2	2	1.00×	1.10×	**55.4%**
Width scaling	752	2	1	1.10×	1.10×	46.7%
Physical-depth scaling	660	3	1	1.10×	1.10×	39.8%

Table 1 reports MQAR-Hard accuracy for each scaling axis. The recursive-depth model matches the width- and physical-depth baselines in estimated train-step FLOPs while preserving the base model's parameter count.

5.3 Language Modeling

To evaluate general sequence modeling at scale, 180M-parameter models are trained from scratch on 10B tokens from FineWeb-Edu [Lozhkov et al., 2024]. For all 180M language-modeling runs, an 8-layer backbone is used; since recursive depth applies the sequence mixer $L \times N$ times per forward pass, $N = 2$ and $N = 3$ correspond to 16 and 24 tied mixer applications while preserving the same parameter count. The evaluated architecture is compared against standard Transformer [Vaswani et al., 2017] and Google Hawk [De et al., 2024] baselines. To test whether recursive depth transfers beyond the custom architecture, we also apply the same recursive formulation to a faithful JAX implementation of Mamba-2 [Dao and Gu, 2024], a state-of-the-art SSM architecture. These Mamba-2 models are physical-depth-matched to the primary architecture, use the standard state expansion ($E = 64$), and are width-scaled to match the 180M parameter budget.

Table 2 reports final FineWeb-Edu validation perplexity and zero-shot LAMBADA performance. Values represent the mean and standard deviation across 3 independent training runs. In the Base architecture, recursive depth improves FineWeb-Edu perplexity from 15.81 ($N = 1$) to 15.54 ($N = 3$), while reducing LAMBADA perplexity from 167.52 to 145.14. The $N = 2$ Base model obtains the highest LAMBADA accuracy within the Base family, improving from 15.36% to 16.73%. The same trend appears in Mamba-2: recursive depth improves FineWeb-Edu perplexity from 14.28 ($N = 1$) to 14.17 ($N = 3$), reduces LAMBADA perplexity from 71.28 to 65.38, and improves

Table 2: Parameter- and token-matched language modeling results.

Architecture	Params	FW-Edu PPL ↓	LAMB. PPL ↓	LAMB. Acc ↑
Transformer	180M	13.74 ± 0.03	64.10 ± 3.47	$28.41 \pm 0.36\%$
Hawk	180M	14.94 ± 0.02	120.06 ± 1.82	$17.49 \pm 0.07\%$
Mamba-2				
Single-Pass ($N = 1$)	180M	14.28 ± 0.03	71.28 ± 2.31	$23.53 \pm 0.04\%$
Iterative ($N = 2$)	180M	14.18 ± 0.04	69.57 ± 2.91	$24.82 \pm 0.43\%$
Iterative ($N = 3$)	180M	$\mathbf{14.17 \pm 0.03}$	$\mathbf{65.38 \pm 4.04}$	$\mathbf{25.30 \pm 0.56\%}$
Base				
Single-Pass ($N = 1$)	180M	15.81 ± 0.02	167.52 ± 6.15	$15.36 \pm 0.17\%$
Iterative ($N = 2$)	180M	15.58 ± 0.02	147.84 ± 2.83	$\mathbf{16.73 \pm 0.40\%}$
Iterative ($N = 3$)	180M	$\mathbf{15.54 \pm 0.04}$	$\mathbf{145.14 \pm 6.22}$	$16.63 \pm 0.45\%$

LAMBADA accuracy from 23.53% to 25.30%. The Transformer baseline remains the strongest overall model in this language-modeling setting.

5.4 Memory Footprint and Scaling Profiling

To quantify the resource footprint of training with recursive depth, we profile peak activation VRAM and training-step compute for a 12-layer model at sequence length $S = 4096$. All memory profiles were benchmarked in standard IEEE 32-bit float (`float32`) precision on a single consumer-grade NVIDIA RTX 5070 Ti. The model is evaluated under a standard full-sequence forward/backward training step, both with and without layer-wise gradient checkpointing (rematerialization). For the Base architecture, TFLOPs are computed directly from the explicit operations in the implementation. For Recursive Mamba-2, TFLOPs are reported as architecture-level estimates, since compiler cost analysis can undercount custom recurrent kernels used by the implementation.

Table 3: Training memory and compute profile at $L = 12$, $S = 4096$, and `float32` precision.

Architecture	Steps	Naive VRAM	Checkpt. VRAM	Compute
Mamba-2	$N = 1$	3842.3 MB	1070.1 MB	3.67 TFLOPs†
	$N = 2$	4446.0 MB	1070.2 MB	4.09 TFLOPs†
	$N = 3$	5054.8 MB	1070.3 MB	4.52 TFLOPs†
Base	$N = 1$	3651.8 MB	628.1 MB	3.71 TFLOPs
	$N = 2$	4953.4 MB	708.2 MB	4.55 TFLOPs
	$N = 3$	6329.0 MB	788.9 MB	5.39 TFLOPs

† Architecture-level estimate.

Table 3 reports the measured memory footprint and training-step compute profile. VRAM is measured using compiler memory analysis. Without checkpointing, peak VRAM grows approximately linearly with recursive depth, since intermediate states from each recursive mixer application must be retained for the backward pass. With layer-wise checkpointing, the marginal memory cost is much smaller: in the Base model, checkpointed VRAM increases by approximately 80 MB per recursive step, while Recursive Mamba-2 remains nearly flat under this profiling setup. Compiler optimizations can further reduce temporary memory in favorable short-sequence regimes, but we report the more conservative $S = 4096$ profile.

The tradeoff is additional recomputation during the backward pass. For the Base model, train-step compute increases by 0.84 TFLOPs per recursive step, corresponding to a $1.23\times$ increase at $N = 2$ and a $1.45\times$ increase at $N = 3$ relative to the single-pass model. For Recursive Mamba-2, TFLOPs are marked as architecture-level estimates because compiler cost analysis can undercount custom recurrent kernels. Notably, because these benchmarks were conducted in full `float32` precision, the absolute memory footprint is heavily penalized compared to standard half-precision training; nonetheless, the checkpointed recursive models fit comfortably within the constraints of consumer-grade hardware.

5.5 Strict Training-FLOP Matching

To isolate the effect of cumulative training compute from the architectural benefits of recursive depth, a strict training-FLOP-matched comparison is performed. For the $N > 1$ FineWeb-Edu runs, the checkpoint nearest to the point where cumulative training FLOPs match the total compute used by the full $N = 1$ baseline run is evaluated. Values are mean and standard deviation across 3 independent training runs.

Table 4: Strict training-FLOP-matched checkpoint comparison on FineWeb-Edu.

Architecture	Checkpoint criterion	Cumulative training FLOPs	FW-Edu PPL ↓
Base Architecture			
Single-Pass ($N = 1$)	Final checkpoint	$1.000\times$	15.81 ± 0.02
Iterative ($N = 2$)	FLOP-matched checkpoint	$1.000\times$	$\mathbf{15.78 \pm 0.01}$
Iterative ($N = 3$)	FLOP-matched checkpoint	$1.000\times$	16.29 ± 0.06
Recursive Mamba-2			
Single-Pass ($N = 1$)	Final checkpoint	$1.000\times$	$\mathbf{14.28 \pm 0.03}$
Iterative ($N = 2$)	FLOP-matched checkpoint	$1.000\times$	$\mathbf{14.28 \pm 0.02}$
Iterative ($N = 3$)	FLOP-matched checkpoint	$1.000\times$	14.58 ± 0.04

Table 4 shows that under exact cumulative FLOP matching, the Base architecture at $N = 2$ effectively matches the single-pass baseline, achieving 15.78 perplexity compared with 15.81 for $N = 1$. In contrast, the $N = 3$ configuration degrades to 16.29, indicating that deeper recursive refinement can become inefficient when the added per-token computation reduces the number of unique training tokens observed within the same total FLOP budget. The Recursive Mamba-2 results follow the same pattern: $N = 2$ matches the single-pass baseline at 14.28 perplexity, while $N = 3$ degrades to 14.58. These results support treating $N = 2$ as the practical operating point for strict compute-constrained training.

6 Discussion

The experiments identify recursive depth as a distinct scaling axis for recurrent and state-space sequence models. Unlike width or physical-depth scaling, recursive depth keeps trainable parameter count and optimizer state fixed, while increasing the amount of sequence-mixing computation applied by the same parameters. It is therefore not a compute-free improvement, but a compute–memory tradeoff: additional local mixer computation is exchanged for fixed model size.

6.1 Recursive Depth as Iterative Routing

The MQAR results provide the clearest evidence that recursive depth addresses a bottleneck not solved by parameter count alone. On MQAR-Hard, the 15M-parameter single-pass model reaches 55.2% accuracy, while the parameter-matched recursive models reach 88.2% at $N = 2$ and 99.5% at $N = 3$. The compute-matched scaling-axis comparison further supports this interpretation: at approximately $1.10\times$ train-step FLOPs, recursive depth reaches 55.4% accuracy, compared with 46.7% for width scaling and 39.8% for physical-depth scaling.

These results suggest that associative recall is limited not only by representational capacity, but also by routing time. A single recurrent scan must store, match, and retrieve in one causal sweep. Recursive depth allows later scans to condition on representations produced by earlier scans, giving the model additional opportunities to refine candidate associations while reusing the same parameters.

6.2 Transfer Beyond the Base Architecture

The language-modeling results show that the effect is not limited to synthetic recall. The gains are smaller than on MQAR, but consistent within both recurrent/SSM-style model families. In the Base architecture, recursive depth improves FineWeb-Edu perplexity from 15.81 to 15.54 and reduces LAMBADA perplexity from 167.52 to 145.14. In Mamba-2, recursive depth improves FineWeb-Edu perplexity from 14.28 to 14.17, reduces LAMBADA perplexity from 71.28 to 65.38, and improves LAMBADA accuracy from 23.53% to 25.30%.

These should be interpreted as within-architecture scaling gains, not as a claim that recursive SSMs dominate all baselines. The Transformer remains the strongest overall language-modeling model in this setting. The key result is instead that recursive depth improves both the custom architecture and a state-of-the-art SSM architecture without increasing parameter count or optimizer state.

The Mamba-2 result is also likely conservative. Mamba-2 is highly optimized around fused single-pass kernels, making it difficult to insert recursive refinement in an architecture-native way. A Mamba-style model designed around recursive depth from the start could expose cleaner recurrent-step boundaries and use more tailored gating, rather than inserting recursion into an implementation optimized for a single fused pass.

6.3 Compute–Memory Tradeoff

Recursive depth is most useful when memory, optimizer state, or parameter communication are more limiting than local arithmetic. Width scaling increases dense compute, parameters, optimizer state, and activation size. Physical-depth scaling adds untied layers and therefore increases full-layer compute and parameter memory. Recursive depth instead reuses the same parameters and applies extra computation only inside the sequence mixer; the MLP is still executed once per layer.

The profiling results support this view. Without checkpointing, recursive unrolling increases activation memory because intermediate recursive states must be retained for the backward pass. With layer-wise checkpointing, the marginal memory cost becomes much smaller: in the Base model, checkpointed VRAM increases by only about 80 MB per recursive step at $S = 4096$ in `float32`. Thus, recursive depth is not intended as a universally superior scaling axis, but as a practical option in memory-bound regimes where additional computation is cheaper than additional parameters or optimizer state.

The strict FLOP-matched experiment should be read as a conservative compute-bound stress test. It asks what happens when recursive models are not allowed to spend any additional total training compute, even though their main advantage is fixed memory rather than fixed FLOPs. Under this accounting, $N = 2$ remains competitive with the single-pass baseline, while $N = 3$ degrades because it sees fewer unique training tokens before exhausting the same FLOP budget. This clarifies the operating regime: recursive depth is less attractive when training is strictly compute-bound, but useful when training is memory- or parameter-bound.

6.4 Practical Operating Point

Across experiments, $N = 2$ is the most practical setting in the regimes tested here. It gives large MQAR gains, improves language modeling in both Base and Mamba-2, and remains competitive under strict FLOP matching. Larger recursive depths can be useful for strongly routing-dominated tasks, as shown by MQAR-Hard at $N = 3$, but are less reliable under fixed cumulative compute. We do not claim that $N = 2$ is universally optimal: deeper recursion may become more effective with architectures designed for multi-pass refinement, improved normalization or gating, larger model scales, or training regimes where memory rather than compute is the primary bottleneck.

The main limitations are that the largest gains occur on a synthetic associative-recall benchmark, the language-modeling gains are modest, and end-to-end efficiency depends on implementation details such as sharding, rematerialization, and kernel fusion. Overall, recursive depth should be viewed as a complementary scaling axis: it does not replace width, physical depth, or attention, but provides a parameter-efficient way to add routing iterations when model memory is the bottleneck.

7 Conclusion

This work introduced recursive depth as a parameter-efficient scaling axis for recurrent and state-space sequence models. Instead of adding width, state size, or untied layers, recursive depth repeatedly applies a shared sequence mixer over a refined value stream, giving the model additional algorithmic time while preserving trainable parameter count and optimizer state.

The results show that this extra iterative computation is especially valuable for associative recall. On the hardest MQAR tier, a 15M-parameter single-pass model reaches 55.2% accuracy, while recursive depth reaches 88.2% with two steps and 99.5% with three. The same mechanism also

improves 180M-parameter language modeling in both the base architecture and a modified Mamba-2 implementation without increasing parameter count.

Recursive depth is not a free improvement: it trades additional sequence-mixer compute for fixed parameter memory and optimizer state. With checkpointing, this makes it a practical scaling axis in memory-bound regimes where adding parameters, optimizer states, or untied layers is costly. Strict FLOP-matched checkpoints show that $N = 2$ is the most practical operating point, while deeper recursion can underperform when it observes fewer training tokens. Overall, these findings suggest that iterative computation with shared weights is an underexplored and practical scaling axis for efficient sequence models, particularly when precise associative routing is the bottleneck.

References

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando de Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Llion Jones. Universal transformers. In *International Conference on Learning Representations*, 2019.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. In *Advances in Neural Information Processing Systems*, volume 35, pages 22982–22994, 2022.

Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. In *International Conference on Learning Representations*, 2016.

Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Fineweb-edu: A large-scale high-quality dataset for educational pretraining. *arXiv preprint arXiv:2406.17557*, 2024.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Kiran Gv, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan S. Wind, Stanisław Woźniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*, 2023.

Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified state space layers for sequence modeling. In *International Conference on Learning Representations*, 2023.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

A Experimental Details

A.1 Language Modeling

All 180M-parameter language-modeling runs are trained from scratch on tokenized FineWeb-Edu sequences of length 4096 using a Mistral tokenizer with a 32k-token vocabulary. Models use an 8-layer backbone and are trained for 76,500 optimizer steps with global batch size 32, corresponding to approximately 10B training tokens. We use AdamW with learning rate 3×10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1, and a cosine decay schedule with final learning-rate multiplier 0.1. Dropout is set to zero. Training uses full-sequence next-token prediction with standard cross-entropy loss.

Validation is performed every 100 steps on held-out FineWeb-Edu validation batches using the same sequence length and tokenizer. LAMBADA evaluation is performed zero-shot using the final checkpoints. All reported language-modeling values are means and standard deviations over three independent training runs.

A.2 MQAR

MQAR batches are generated synthetically during training. Each sequence contains a set of key–value bindings followed by query keys; the target is the value associated with each query key. Loss and accuracy are computed only at query positions, with all non-query positions masked out. The Easy, Medium, and Hard settings are:

Task	Seq. length	KV pairs	Queries	Vocab.
Easy	512	64	32	256
Medium	768	96	48	384
Hard	1024	128	64	512

MQAR models are trained for 250,000 steps with batch size 32. We use AdamW with a warmup-cosine learning-rate schedule, peak learning rate 2×10^{-3}, 1000 warmup steps, final learning rate 10^{-5}, global gradient clipping at 1.0, and weight decay 0.0. Reported MQAR accuracies are mean final evaluation accuracies over two independent runs.

Reported $\pm$ values denote standard deviation across independent training runs.

B Compute and Resource Accounting

For language modeling, cumulative training FLOPs are estimated from architecture-level operation counts and used consistently across recursive depths and baselines for relative comparison. For the Base architecture, TFLOPs in the resource profile are computed from the explicit operation count in the implementation. For Recursive Mamba-2, TFLOPs are reported as architecture-level estimates because compiler cost analysis can undercount custom recurrent kernels.

Peak VRAM is measured using compiler memory analysis during a full-sequence forward/backward training step at sequence length $S = 4096$ and batch size 1, both with and without layer-wise gradient checkpointing. Resource profiling is conducted in `float32` precision on a single NVIDIA RTX 5070 Ti. The reported memory numbers are therefore conservative relative to standard mixed-precision training.

Compute resources. Language-modeling runs were trained on 8 TPU v6e cores using data-parallel sharding. MQAR experiments were trained on a single NVIDIA RTX 5070 Ti with batch size 32. Resource profiling was also performed on a single NVIDIA RTX 5070 Ti in `float32` precision.

C Assets and Licenses

The experiments use publicly available datasets, benchmarks, and model references. FineWeb-Edu is used for language-model training, LAMBADA is used for zero-shot evaluation, and MQAR is generated synthetically following the benchmark definition. Transformer, Hawk, and Mamba-2 baselines are cited in the main paper. We use these assets for research evaluation and respect their published licenses and terms where available.