RANDOM RECURSIVE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive models create computational depth through parameter reuse, offering a parameter-efficient alternative to increasing model size. However, most recursive models repeatedly apply one learned transformation or a prescribed sequence of transformations, restricting computation to a fixed layer order. We introduce the Random Recursive Model (RRM), which maintains a pool of L learned layers and performs T recursive steps by sampling one layer independently with replacement for each example and step. This enables flexible layer reuse while retaining the parameter efficiency of recurrence. We evaluate RRM on challenging reasoning tasks, where it matches or exceeds the baselines, often with 50–75% fewer parameters. RRM can vary its depth at inference, including beyond that seen during training, without retraining or adding parameters, improving tasks that benefit from deeper iterative computation. RRM also supports Monte Carlo inference and probabilistic test-time scaling, both of which improve performance without retraining. These insights may open new directions in neural network architecture design.

1 INTRODUCTION

Recursive models have recently obtained strong results on challenging reasoning tasks (Geiping et al., 2025; Wang et al., 2025; Jolicoeur-Martineau, 2025). By reusing parameters across depth, they can allocate many computational steps to an example without requiring a distinct learned layer at every step. This makes recursion a direct way to separate model size from computation, including computation allocated through test-time scaling (Snell et al., 2024; Zhu et al., 2026).

Most recursive models nevertheless use a fixed-order recurrence. They either repeat one learned transformation or repeatedly apply a set of transformations in a prescribed order. Recent works add further structure through fixed update schedules, separate high- and low-level modules, distinct answer and reasoning states, or provisional-answer feedback (Wang et al., 2025; Jolicoeur-Martineau, 2025). We study a different and simpler form of recursion, which we call the *Random Recursive Model* (RRM), in which learned layers form a pool of reusable transformations and any layer can be reused at any point in the computation.

An RRM contains L learned layer transformations and randomly reuses them over T applications, or recursive steps. Unlike standard recurrence, which creates depth by applying the same transformation or the same fixed sequence, RRM can revisit the same layer or apply any of the other learned layers at each application. During both training and inference, at each of the T recursive steps, one layer is sampled uniformly and independently with replacement for each example. RRM therefore retains the parameter sharing of recurrence while allowing up to L^T possible layer sequences, without imposing a fixed recurrent order. The number of learned layers L determines model size, while T determines computational depth and can be increased or decreased at test time.

Across our experiments, we start from established fixed-order baselines and replace their backbones with RRM while retaining the task formulation and core model components. Across all five tasks, the strongest RRM configuration matches or exceeds the corresponding baseline. In controlled image classification and autoregressive logic puzzles, RRM does so with roughly 50–75% fewer parameters. The same RRM construction also matches or exceeds existing methods on more difficult reasoning tasks without any structured, hierarchical or prescribed structure.

We further study how RRM scales with the number of learned layers L and the number of applications T. RRM therefore naturally supports both training-time scaling and test-time scaling, since T

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

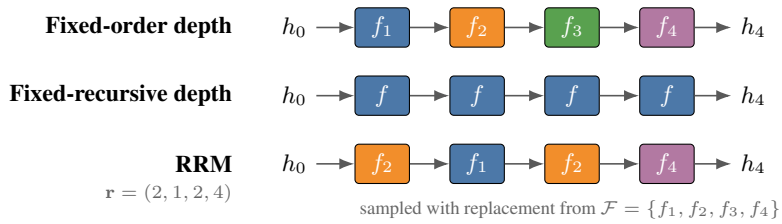


Figure 1: **Fixed-order depth, fixed-recursive depth, and RRM.** Four applications under the three depth constructions. Fixed-order depth applies each layer once in its prescribed position. Fixed-recursive depth repeatedly applies a prescribed sequence of learned functions. Here, the repeated sequence contains one function ($L = 1$); more generally, it may contain multiple layers and be surrounded by fixed-order layers. RRM samples from the complete set of learned layers at every step, allowing both reordering and reuse. Colors identify shared parameters.

can be increased without adding learned parameters. We find that additional applications improve performance when a task benefits from longer iterative computation, while gains plateau when fewer applications suffice and performance can degrade when test-time depth extends far beyond training depth. RRM is also compatible with Monte Carlo inference and probabilistic test-time scaling, both of which improve performance without retraining. With the latter, RRM achieves the best result among the three evaluated models.

2 Background

Most deep networks execute the learned transformations in a fixed order. This is true both for conventional feed-forward networks, which assign different parameters to different depths, and for modern recursive networks, which reuse parameters across depth. The two families differ in parameter sharing, but both prescribe the sequence of transformations before seeing an example.

2.1 Fixed-order depth

A feed-forward network of depth L maps an encoded input $h_0 = E(x)$ through an ordered sequence of functions,

$$h_{\ell+1} = f_{\ell+1}(h_\ell), \qquad \ell = 0, \dots, L-1. \tag{1}$$

The computation is therefore the fixed composition $f_L \circ \cdots \circ f_1$. Every example applies each function once and in the same order. Transformers are a prominent instance of this design. Each layer has its own parameters and occupies a predetermined position in the stack (Vaswani et al., 2023). Increasing depth ordinarily adds both computation and parameters.

2.2 Fixed-recursive depth

Recursive models instead create depth by applying a learned transformation repeatedly. In its simplest form,

$$h_{t+1} = f(h_t, E(x)), \qquad t = 0, \dots, T-1, \tag{2}$$

where the encoded input may optionally be injected at each step. Parameter sharing decouples computational depth T from model size. This principle underlies the Universal Transformer (Dehghani et al., 2019), looped Transformers (Yang et al., 2024), and recurrent-depth language models (Geiping et al., 2025). Recent reasoning systems similarly obtain large effective depth from small recurrent modules, including the Hierarchical Reasoning Model (HRM) (Wang et al., 2025) and the Tiny Recursive Model (TRM) (Jolicoeur-Martineau, 2025).

The recurrence becomes explicit when we unroll the computation. Define $f_x(h) = f(h, E(x))$ for a fixed input x. After T steps,

$$h_T = \underbrace{f_x \circ f_x \circ \cdots \circ f_x}_{T \text{ times}}(h_0) = f_x^{\circ T}(h_0). \tag{3}$$

For example, four recursive steps apply $f_x \circ f_x \circ f_x \circ f_x$, whereas a four-layer feed-forward network applies $f_4 \circ f_3 \circ f_2 \circ f_1$. The recursive model therefore follows a prescribed order, repeatedly applying the same learned function or fixed sequence of functions.

3 RANDOM RECURSIVE MODELS

We introduce the *Random Recursive Model* (RRM). RRM retains the input encoder E, all L learned layers of the backbone $\mathcal{F} = \{f_1, \ldots, f_L\}$, and the task-specific output head H. Unlike fixed-order feed-forward depth and fixed-recursive depth, RRM makes every backbone layer reusable rather than applying all layers once in their prescribed order or repeatedly following a fixed sequence. Given an input x, the encoder initializes the hidden state as $h_0 = E(x)$, although h_0 may instead be any task-specific starting state. For a fixed computational depth T, at each step $t = 0, \ldots, T-1$, RRM samples one of the L layers uniformly at random and applies it to the current state.

$$h_{t+1} = f_{r_t}(h_t, E(x)), \qquad r_t \sim \text{Uniform}\{1, \ldots, L\}, \tag{4}$$

where r_t is the layer index sampled at step t. Sampling is performed independently for every example and every recurrent step. The indices are sampled with replacement, so a function may be applied multiple times or not at all in a particular computation. The second argument $E(x)$ denotes optional input reinjection. When the backbone does not inject the encoded input at every step, Equation 4 reduces to $h_{t+1} = f_{r_t}(h_t)$. After T applications, the task-specific head returns $\hat{y} = H(h_T)$. RRM does not prescribe a particular layer type. Its layers may be Transformer blocks or other learned state transformations.

The complete computation for an example is determined by the random sequence $\mathbf{r} = (r_0, \ldots, r_{T-1})$ and can be written as

$$h_T = f_{r_{T-1}} \circ \cdots \circ f_{r_1} \circ f_{r_0}(h_0; E(x)). \tag{5}$$

Figure 1 compares RRM with fixed-order depth and fixed-recursive depth. Setting $L = 1$ recovers the simplest fixed recurrence, which repeats one learned transformation. More generally, fixed-recursive depth can repeat any prescribed sequence containing one layer, all L layers, or a subset of them. These cases are recovered by replacing the sampled indices with the corresponding fixed sequence. Fixed-order depth is recovered by setting $T = L$ and $r_t = t + 1$. The fixed-order cases can therefore be viewed as special cases of RRM. RRM occupies the simple intermediate regime, retaining the complete backbone while composing its layers recursively. This preserves the parameter-efficient depth and adjustable computation of recurrence while giving each step access to multiple learned state transformations. At recursive depth T, RRM admits L^T possible layer sequences instead of committing to one fixed composition, providing diverse computational trajectories without an additional learned mechanism.

3.1 TRAINING

Let $p(\mathbf{r})$ denote the uniform distribution over length-T index sequences. For a dataset $\mathcal{D}$ and task loss ℓ, RRM minimizes

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \mathbb{E}_{\mathbf{r}\sim p(\mathbf{r})} \left[\ell(H_\theta(h_T(x; \mathbf{r})), y) \right]. \tag{6}$$

We estimate this objective with one independently sampled sequence per example in each forward pass. Unless stated otherwise, standard backpropagation is applied through the sampled composition; at a given step, gradients update only the function that was applied. There is no auxiliary objective or additional learned mechanism.[1] Inference uniformly samples a sequence of layers for each sample, similar to training.

3.2 TRAINING- AND TEST-TIME SCALING

RRM separates the number of physical backbone layers, L, from the number of recursive applications, T. The former determines the parameterized depth of the backbone, while the latter determines

[1] We provide details on learned policies in Appendix A.3.

its effective computational depth. At training time, increasing T_{train} unrolls a longer random composition and allocates more computation to each example without introducing new layer parameters. Sampling with replacement makes reuse possible whenever $T > 1$ and guarantees that at least one layer is reused when $T > L$.

The recursive depth may also be changed after training. Given a model trained with depth T_{train}, we can evaluate the same parameters using a different depth T_{test}. Increasing T_{test} allocates more inference computation while the model size remains fixed. Allocating additional computation at inference is a widely studied approach to scaling reasoning (Snell et al., 2024; Geiping et al., 2025; Zhu et al., 2026). Thus, RRM supports compute scaling during both training and inference by varying T_{train} or T_{test} without changing model size. RRM also supports Monte Carlo inference and probabilistic test-time scaling, which are both described in Appendix A.4.

4 Experimental Setup

We evaluate RRM in three settings of increasing reasoning difficulty. For each task, we follow the task formulation, data splits, model components, training protocol, and evaluation procedure of established baselines. Our main change is to replace the fixed-order backbone with an RRM backbone. Any further task-specific changes are stated below. In every experiment, one layer index is sampled independently for each example and each application. Unless noted otherwise, the number of applications is fixed within a run. Where separate validation and test splits are available, we use validation performance for checkpoint selection and evaluate the selected model on the disjoint test set. Unless stated otherwise, RRM results are reported as mean $\pm$ sample standard deviation over three evaluation seeds for a fixed checkpoint.

CIFAR-10. We use CIFAR-10 as a controlled toy experiment (Krizhevsky & Hinton, 2009). The baseline is a custom eight-layer fixed-order Vision Transformer following Dosovitskiy et al. (2021), adapted to CIFAR-10 and trained under the same protocol as RRM. RRM replaces its eight-layer backbone with reusable Transformer layers. For training-time scaling, we train $L = 2$ models with $T_{\text{train}} \in \{4, 8, 12, 16\}$ and evaluate each at its training depth. For test-time scaling, we fix checkpoints trained with $T_{\text{train}} \in \{8, 16\}$ and evaluate $T_{\text{test}} \in \{4, 8, 12, 16, 20, 24\}$. All configurations are evaluated using both basic RRM inference and Monte Carlo inference with $M = 16$. We separately train an $L = 1$, $T_{\text{train}} = 16$ model to study single-layer reuse. Finally, we examine how the class-token representations of the $T_{\text{train}} = 16$ model evolve over $T_{\text{test}} \in \{8, 12, 16, 20, 24\}$ using one sampled sequence per image. We quantify class separation with a k-nearest-neighbor (k-NN) classifier using $k = 20$ and visualize the representations with t-SNE (van der Maaten & Hinton, 2008). Performance is measured by top-1 accuracy on the official test set. Architecture and training details are provided in Appendix A.2.

Sudoku and Zebra. We train on the processed Sudoku and Zebra datasets publicly released by Shah et al. (2024).[2] We use their strongest solver-ordered formulation. The causal Transformer has eight layers and provides the fixed-order baseline. RRM changes only the backbone composition. On Sudoku, we evaluate $L \in \{2, 4\}$ reusable layers, while Zebra uses $L = 2$. Every RRM is trained with $T_{\text{train}} = 16$ applications. All comparisons use the same 320,000-step training budget. We train on the full released training sets and evaluate on task-specific held-out splits using their greedy autoregressive protocol. For Zebra, we additionally evaluate inference-scaling depth using the trained $L = 2$, $T_{\text{train}} = 16$ RRM and $T_{\text{test}} \in \{4, 8, 12, 16, 20\}$.

Extreme reasoning tasks. For Sudoku-Extreme, Maze-Hard, and ARC-AGI-1 (Chollet, 2019), we follow the experimental setups used by HRM (Wang et al., 2025) and TRM (Jolicoeur-Martineau, 2025), including the released datasets, layer components, objective, and optimization settings. As in HRM and TRM, the fixed puzzle embedding $E(x)$ is supplied at every recurrent application. Maze-Hard and Sudoku-Extreme use exact-grid accuracy; ARC-AGI-1 uses Pass@2. We use these tasks to test whether useful recursive computation can emerge from simple random layer reuse, without the prescribed recurrent structures of HRM and TRM.

[2] https://github.com/kulinshah98/llm-reasoning-logic-puzzles

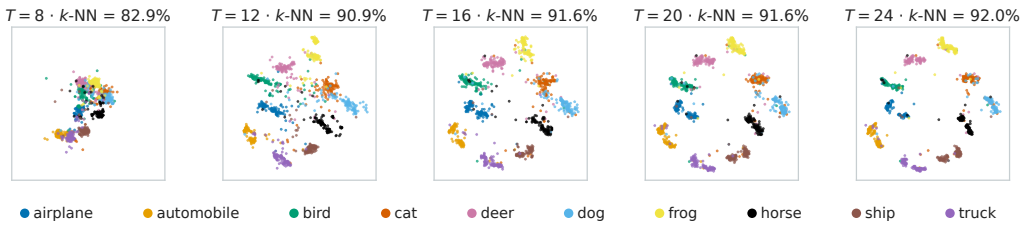


Figure 2: CIFAR-10 representations across recursive depth. We take the class-token representations from the RRM trained with $L = 2$ and $T_{\text{train}} = 16$ and plot them at $T_{\text{test}} \in \{8, 12, 16, 20, 24\}$, using one sampled layer sequence per image. Each panel is a t-SNE projection of the same test images, colored by class. Both the k-NN classification accuracy and the t-SNE projections show that the representations become more class-specific as the number of recursive applications increases.

Maze-Hard. On Maze-Hard, we use basic RRM with two non-causal Transformer layers ($L = 2$), randomly reused for $T_{\text{train}} = 16$ applications. We evaluate test-time scaling with $T_{\text{test}} \in \{4, 8, 12, 16, 20, 24\}$.

Sudoku-Extreme and ARC-AGI-1. For Sudoku-Extreme and ARC-AGI-1, we augment RRM with deep supervision, as detailed in Appendix A.1. For Sudoku-Extreme, we evaluate $L \in \{1, 2\}$ and use $N_{\text{sup}} = 32$ steps with $T_{\text{train}} = 21$ applications per step, of which the first seven are computed without gradients. For test-time scaling, we vary T_{test} over $\{11, 16, 21, 26, 32\}$. We also evaluate TRM+RRM, a variant that keeps the entire TRM setup but replaces only the fixed order of its two layers with RRM's random reuse. For ARC-AGI-1, we use $L = 2$, vary both the number of applications and supervision steps during training and inference. Further details are provided in Appendix A.2.

5 Results

5.1 Controlled Toy Image Classification

Table 1 compares RRM with the eight-layer fixed-order dense baseline. With basic inference, the two-layer RRM remains close to the dense baseline while using roughly one quarter of its parameters. Monte Carlo inference with $M = 16$ closes the gap, and the model trained at $T_{\text{train}} = 8$ slightly surpasses the baseline. Increasing T_{train} to 16 provides no consistent benefit, suggesting that CIFAR-10 saturates at modest computational depth.

Table 1: CIFAR-10 top-1 accuracy. M denotes Monte Carlo inference samples.

Model	L	T_{train}	Params.	M	Accuracy (%)
Dense	8	8	3.20M	1	92.520 ± 0.000
RRM	2	8	0.824M	1	91.403 ± 0.078
RRM	2	16	0.824M	1	91.670 ± 0.155
RRM	2	16	0.824M	16	92.337 ± 0.049
RRM	2	8	0.824M	16	$\mathbf{92.597 \pm 0.110}$

Figure 2 shows that the representations continue to become more class-specific as we increase recursive applications. At each application depth, we fit a k-NN classifier and measure its classification accuracy, which continues to improve beyond the training depth. The t-SNE projections provide a qualitative view of the same behavior, with progressively cleaner class clusters. Additional random recurrence therefore seems to refine the hidden representation even after the classifier already performs well.

5.2 Logic Puzzles as Language Modeling

Table 2 shows that RRM matches or improves upon the fixed-order dense baselines on Sudoku and Zebra. On Sudoku, the $L = 2$ RRM uses roughly one quarter as many parameters while remaining close to the dense baseline, whereas $L = 4$ exceeds it using about half as many parameters. On Ze-

Table 2: Sudoku and Zebra complete-puzzle accuracy.

Task	Model	L	T	Params.	Acc. (%)
Sudoku	Dense	8	8	42.66M	85.779 ± 0.000
	RRM	2	16	10.78M	84.523 ± 0.028
	RRM	4	16	21.40M	$\mathbf{85.954 \pm 0.131}$
Zebra	Dense	8	8	42.89M	94.980 ± 0.000
	RRM	2	16	11.01M	$\mathbf{95.547 \pm 0.103}$

Table 3: Maze-Hard results. † evaluates the same RRM with $T_{\text{test}} = 24$. Baseline results are from Jolicoeur-Martineau (2025).

Model	L	T	Params.	Acc. (%)
HRM	8	24	27M	74.50
TRM-Att	2	30	7M	85.30
RRM	2	16	7M	82.90 ± 0.17
RRM†	2	24	7M	$\mathbf{87.07 \pm 0.71}$

bra, the $L = 2$ RRM similarly uses roughly one quarter as many parameters and improves accuracy. The dense baseline results are our reproductions of the models from Shah et al. (2024).

5.3 Extreme Reasoning Tasks

Maze-Hard. Table 3 shows that basic RRM approaches TRM-Att at its training depth of $T_{\text{train}} = 16$. Increasing the same RRM to $T_{\text{test}} = 24$ moves it above TRM-Att, which uses $T = 30$ applications, without retraining or adding parameters. On Maze-Hard, RRM does not use deep supervision or a training-time halting objective. Unlike the prescribed recurrent structures of HRM and TRM-Att, it passes a single hidden state through $L = 2$ randomly reused layers.

Sudoku-Extreme. Table 4 shows that RRM closely matches TRM at the same 5M parameter budget and total depth of 672 layer applications. Rather than TRM's prescribed recurrent structure, RRM carries a single state through randomly reused layers. Specifically, RRM uses $T_{\text{train}} = 21$ applications over $N_{\text{sup}} = 32$ supervision steps, while TRM uses $T = 42$ over $N_{\text{sup}} = 16$ steps. TRM+RRM, which changes only TRM's fixed layer order to random reuse, improves upon both the published TRM result and our dense reproduction. Increasing basic RRM to $T_{\text{test}} = 32$ produces the strongest result without retraining.

Table 4: Sudoku-Extreme exact-grid accuracy. HRM, TRM, and TRM+RRM use $N_{\text{sup}} = 16$, while RRM uses $N_{\text{sup}} = 32$. † evaluates the same RRM with $T_{\text{test}} = 32$ without retraining. The TRM row is our reproduction; the published result is 87.40% (Jolicoeur-Martineau, 2025).

Model	L	T	Params.	Accuracy (%)
HRM	8	24	27M	55.00
TRM	2	42	5M	87.63
RRM	2	21	5M	87.22 ± 0.03
TRM+RRM	2	42	5M	89.17 ± 0.02
RRM†	2	32	5M	$\mathbf{89.21 \pm 0.03}$

ARC-AGI-1. Similarly, Table 5 shows that RRM trained for fewer steps matches TRM baseline. We also observe reducing N_{sup} from 32 to two preserves nearly all accuracy with $16\times$ fewer applications. Additional details and results are provided in Appendix A.6.

Table 5: RC-AGI-1 Pass@2 accuracy. Baseline results are from Jolicoeur-Martineau (2025).

Model	L	T	Params.	Pass@2 (%)
HRM	8	24	27M	40.30
TRM-Att	2	30	7M	44.60
RRM	2	15	7M	$\mathbf{44.79 \pm 0.19}$

On both Sudoku-Extreme and ARC-AGI-1, deep supervision was important during training.[3] Without it, RRM memorized the Sudoku-Extreme training data and generalized poorly, while truncating gradients through the early applications improved optimization.

Parameter efficiency and one reusable layer. Figure 3 visualizes the parameter efficiency across four tasks and shows that RRM matches or outperforms the baselines with substantially fewer parameters. The $L = 1$ points in panels (a,d) further test whether RRM can be reduced to repeatedly applying a single learned transformation. This retains recursive computation but removes random layer selection, showing that repeatedly applying one function can already learn useful computation.

[3] An ARC Prize analysis likewise identifies HRM's outer loop as an essential performance driver.

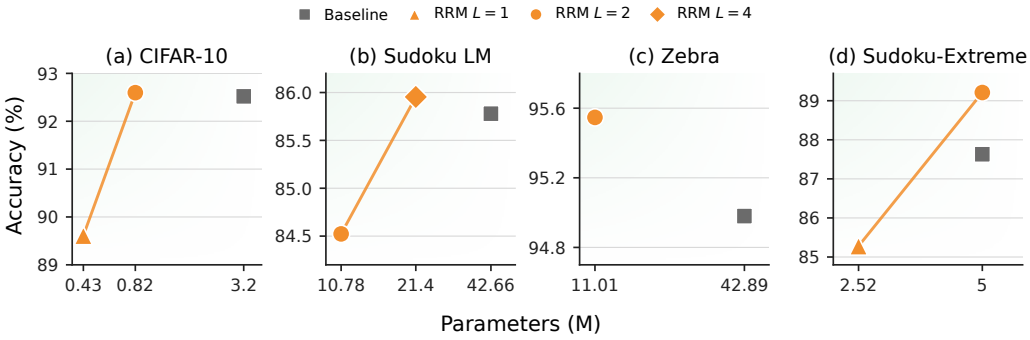


Figure 3: Parameter efficiency across four tasks. RRM matches or outperforms the corresponding baselines with substantially fewer parameters in (a–c) and at a comparable parameter count in (d).

Since $L = 2$ adds parameters and improves both tasks, this comparison does not isolate random ordering from the additional model capacity.

5.4 Test-Time Scaling

Figure 4 compares how the tasks respond when the same RRM parameters are run for longer. The CIFAR-10 RRM trained with $T_{\text{train}} = 16$ remains stable beyond its training depth, showing robustness to additional computation rather than a meaningful accuracy gain. Additional sweeps in Appendix A.5 show that the RRM trained with $T_{\text{train}} = 8$ degrades when extended substantially beyond its training depth, even with Monte Carlo inference. Zebra follows a similar pattern. Its accuracy rises sharply toward the training depth and remains high at the next tested depth, but does not improve monotonically as computation is extended further. We observe a similar pattern on ARC-AGI-1. When trained with $T_{\text{train}} = 8$, increasing T_{test} from 8 to 15 yields only a marginal gain, while with $T_{\text{train}} = 15$, increasing T_{test} from 15 to 21 provides no gain.

By contrast, Maze-Hard and Sudoku-Extreme benefit from additional test-time computation. Maze-Hard improves beyond the training depth, and Sudoku-Extreme improves monotonically over all tested depths. Additional random recurrence at test time may therefore be most useful for tasks that benefit from longer iterative computation, while gains plateau or degrade when fewer applications suffice.

Probabilistic test-time scaling. We further evaluate RRM using PTRM's probabilistic test-time scaling approach (Sghaier et al., 2026). PTRM injects Gaussian noise into the recurrent state to produce multiple trajectories, then uses the existing Q head from the deep-supervision setup, trained to predict whether a complete solution is correct, to select one final answer. Unlike Monte Carlo inference for RRM, it selects the highest-scoring trajectory instead of averaging predictions.

We use the paper's main Sudoku-Extreme configuration of 100 trajectories, $N_{\text{sup}} = 64$ supervision steps, and $\sigma = 0.3$. As shown in Table 6, this approach transfers to all three architectures without retraining and substantially improves accuracy, with RRM achieving the highest result. Because evaluating 100 complete trajectories is expensive, we use a fixed 1,000-puzzle subset. Further evaluation details are provided in Appendix A.4.2.

Table 6: Sudoku-Extreme accuracy with standard inference and PTRM test-time scaling.

Model	Standard (%)	PTRM (%)
TRM	87.20	98.43 ± 0.12
TRM+RRM	88.57 ± 0.25	98.83 ± 0.06
RRM	87.20 ± 0.26	$\mathbf{99.50 \pm 0.10}$

6 Related Work

RRM is a simple form of parameter-sharing recursion, rather than a replacement for existing recursive or capacity-scaling methods. By removing a fixed layer order, it provides a direct way to study how flexible layer reuse interacts with model capacity, computational depth, and reasoning. Our experiments provide initial evidence at the tasks and scales considered, while broader architectures,

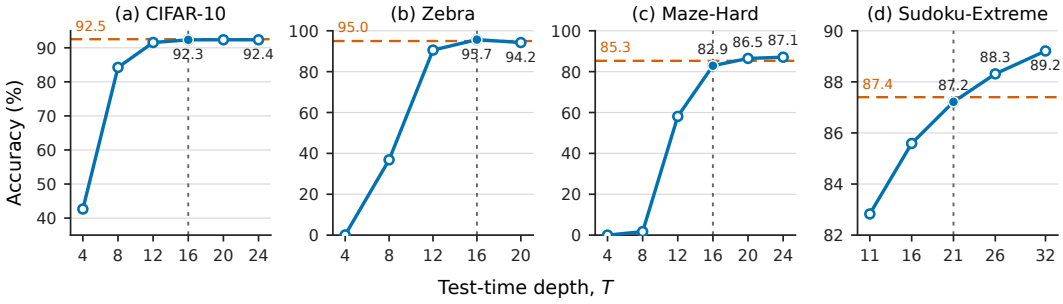


Figure 4: Test-time scaling of RRM without retraining or adding parameters. Vertical dotted lines mark T_{train}. Horizontal dashed lines show the corresponding baselines. Additional test-time applications leave performance in (a,b) largely unchanged but improve it in (c,d).

scales, and reuse strategies remain open for further study. Below, we first distinguish methods that increase capacity at roughly fixed computation from methods that increase computation through parameter sharing, where RRM fits most directly. We then review work on recursive reasoning, test-time scaling, layer order, and randomized networks.

6.1 Scaling Capacity without Proportional Increase in Computation

Capacity-scaling methods increase the number of available parameters while activating only a subset for each input. Mixtures of local experts introduced learned specialization through a gating network (Jacobs et al., 1991), while later conditional-computation methods generalized this idea to avoid evaluating the entire model (Bengio et al., 2013; 2016). Sparsely gated mixture-of-experts models made this approach practical at scale (Shazeer et al., 2017), and subsequent systems expanded model capacity while limiting the number of experts applied to each token (Lepikhin et al., 2020; Fedus et al., 2022). Later work improved how tokens and experts are assigned (Zhou et al., 2022; Puigcerver et al., 2024). These methods provide access to a large parameter pool without a proportional increase in computation. In contrast, RRM changes the reuse order of a fixed set of learned layers at a given computational depth and remains compatible with MoEs, since each reusable layer could itself be an MoE.

6.2 Scaling Computation with Shared Parameters

Parameter-sharing methods instead increase computational depth by applying the same parameters multiple times. Neural GPUs use a recurrent convolutional transition to perform multiple computational steps with shared parameters (Łukasz Kaiser & Sutskever, 2016). Universal Transformers apply the same principle to self-attention by repeatedly applying a shared transition across depth (Dehghani et al., 2019), while ALBERT uses cross-layer parameter sharing to prevent the parameter count from growing with network depth (Lan et al., 2020). More recently, looped and recursive Transformers have been studied for iterative algorithms, length generalization, parameter-efficient language modeling, latent reasoning, and adaptive computational depth (Yang et al., 2024; Fan et al., 2025; Bae et al., 2025a; Geiping et al., 2025; Chen et al., 2025; Bae et al., 2025b; Zhu et al., 2026; Sapunov, 2026; Jeddi et al., 2026). Although these approaches differ in how depth is trained or allocated, they repeatedly apply one transformation or a fixed sequence of transformations. RRM retains parameter sharing across depth but samples each application from a pool of learned layers.

6.3 Recursive Reasoning and Test-Time Scaling

Recursive reasoning methods refine an internal state over multiple steps using shared parameters. Recurrent Relational Networks showed that repeated relational updates can solve tasks including Sudoku (Palm et al., 2018). Later work found that recurrent networks trained on easier problems can extrapolate to harder instances by using additional recurrences at inference, but also identified *overthinking*, where performance deteriorates when recurrence extends too far beyond training (Schwarzschild et al., 2021; Bansal et al., 2022). More recently, the Hierarchical Reasoning Model and Tiny Recursive Model have shown that compact recurrent networks can solve difficult Sudoku,

maze, and abstraction tasks (Wang et al., 2025; Jolicoeur-Martineau, 2025). RRM instead studies a simpler form of recursive reasoning, where layers are trained under random reuse and may learn computations that other models encode through a prescribed recurrent structure.

Varying the number of recurrent steps also connects recursive reasoning to test-time scaling. Test-time scaling can allocate additional inference computation through search and output refinement (Snell et al., 2024). Recursive models can instead scale latent computation by increasing recurrent depth (Geiping et al., 2025; Zhu et al., 2026). RRM provides this form of scaling through randomly reused layers. In the basic RRM, T is set directly and can be increased during training or at test time without adding learned parameters. RRM also supports probabilistic test-time scaling without retraining (Sghaier et al., 2026).

6.4 Layer Order and Randomized Networks

Many methods alter the computational path through a network. Early-exit and layer-skipping methods shorten a fixed sequence of layers while preserving its relative order (Teerapittayanon et al., 2017; Wang et al., 2018; Wu et al., 2019). PathNet and Routing Networks select task-specific paths through reusable modules (Fernando et al., 2017; Rosenbaum et al., 2017), while Modular Networks and Distributed Neural Architectures learn input- or token-dependent compositions that may change module order (Kirsch et al., 2018; Cowsik et al., 2025). Other work searches at test time for ways to skip or repeat contiguous blocks of a pretrained language model (Li et al., 2025). These approaches either retain a constrained order, specialize routes at the task level, or learn discrete routing decisions, whose non-differentiability complicates end-to-end training.

Randomized-network methods instead introduce stochasticity without learning an explicit routing policy. LayerShuffle randomly permutes all layers once per batch to make a full-size Vision Transformer robust to reordering (Freiberger et al., 2024), whereas RRM samples with replacement per example and step, enabling $T \neq L$, parameter-efficient recursion, and inference-depth scaling. Dropout randomly masks individual activations (Srivastava et al., 2014). At the level of depth, stochastic depth and LayerDrop randomly omit entire layers, while Swapout and FractalNet sample paths through larger fixed architectures (Huang et al., 2016; Fan et al., 2019; Singh et al., 2016; Larsson et al., 2017). These methods vary which fixed components are active while preserving their relative order. At the level of network topology, randomly wired neural networks sample graph connectivity before training, after which the sampled graph defines the architecture (Xie et al., 2019). RRM instead samples a layer at each step, allowing arbitrary order and reuse without learning or searching for a route.

7 Conclusion and Future Work

We introduced the Random Recursive Model (RRM), which replaces fixed-order recurrence with random reuse of a small set of learned layers. RRM matches or exceeds the corresponding baselines, often with substantial reductions in parameter count. Separating the number of learned layers from the number of applications also allows RRM to scale computation without increasing model size. Additional inference time applications improve performance on Maze-Hard and Sudoku-Extreme, while gains plateau or eventually decline on tasks that require less computation. Monte Carlo inference and probabilistic test-time scaling provide further improvements without retraining. Together, these results suggest that random layer reuse acts as an effective form of regularization, improving generalization while mitigating overparameterization.

Future work should study how the sampling distribution and the diversity of sampled sequences affect learning. We also studied learned sampling policies, but did not obtain useful gains (Appendix A.3). Making learned ordering work remains an important direction, since an adaptive policy could reduce computation through earlier exiting and improve parameter efficiency by using a smaller subset of reusable layers. Future work should also study how scaling the number of layers and applications behaves in larger models and tasks. More fundamentally, it remains unclear why random layer reuse is effective, and studying its theoretical foundations is left for future work. More broadly, these findings may guide the design of networks that retain flexible layer reuse while being even more computationally efficient.

AI USE STATEMENT

We used generative AI tools to polish the writing for grammar and readability, generate and edit implementation, experimental, and plotting scripts, and search for potentially relevant related work. The authors reviewed all AI-assisted text, inspected and ran the generated code, checked its outputs against the intended behavior, and independently read and verified each suggested source before citing it. We take responsibility for the final content of this work, including all text, claims, code, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Our experiments use publicly available datasets and follow established baselines. The experimental setup and appendix document the model configurations, training procedures, and inference protocols used in our experiments. We plan to release the code, configurations, and scripts needed to reproduce the reported results.

REFERENCES

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora, 2025a. URL https://arxiv.org/abs/2410.20672.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025b. URL https://arxiv.org/abs/2507.10524.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder, 2021. URL https://arxiv.org/abs/2107.05407.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Logical extrapolation without overthinking, 2022. URL https://arxiv.org/abs/2202.05826.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models, 2016. URL https://arxiv.org/abs/1511.06297.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking, 2025. URL https://arxiv.org/abs/2502.13842.

François Chollet. On the measure of intelligence, 2019. URL https://arxiv.org/abs/1911.01547.

Aditya Cowsik, Tianyu He, and Andrey Gromov. Towards distributed neural architectures, 2025. URL https://arxiv.org/abs/2506.22389.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data, 2019. URL https://arxiv.org/abs/1805.09501.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers, 2019. URL https://arxiv.org/abs/1807.03819.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout, 2019. URL https://arxiv.org/abs/1909.11556.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization, 2025. URL https://arxiv.org/abs/2409.15647.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022. URL https://arxiv.org/abs/2101.03961.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks, 2017. URL https://arxiv.org/abs/1701.08734.

Matthias Freiberger, Peter Kun, Anders Sundnes Løvlie, and Sebastian Risi. Layershuffle: Enhancing robustness in vision transformers by randomizing layer execution order, 2024. URL https://arxiv.org/abs/2407.04513.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach, 2025. URL https://arxiv.org/abs/2502.05171.

Alex Graves. Adaptive computation time for recurrent neural networks, 2017. URL https://arxiv.org/abs/1603.08983.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth, 2016. URL https://arxiv.org/abs/1603.09382.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation, 2026. URL https://arxiv.org/abs/2602.11451.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025. URL https://arxiv.org/abs/2510.04871.

Louis Kirsch, Julius Kunze, and David Barber. Modular networks: Learning to decompose neural computation, 2018. URL https://arxiv.org/abs/1811.05249.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020. URL https://arxiv.org/abs/1909.11942.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals, 2017. URL https://arxiv.org/abs/1605.07648.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020. URL https://arxiv.org/abs/2006.16668.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? test-time depth adaptation of pretrained llms, 2025. URL https://arxiv.org/abs/2507.07996.

Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks, 2018. URL https://arxiv.org/abs/1711.08028.

Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts, 2024. URL https://arxiv.org/abs/2308.00951.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning, 2017. URL https://arxiv.org/abs/1711.01239.

Grigory Sapunov. Universal transformers need memory: Depth-state trade-offs in adaptive recursive reasoning, 2026. URL https://arxiv.org/abs/2604.21999.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks, 2021. URL https://arxiv.org/abs/2106.04537.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model, 2026. URL https://arxiv.org/abs/2605.19943.

Kulin Shah, Nishanth Dikkala, Xin Wang, and Rina Panigrahy. Causal language modeling can elicit search and reasoning capabilities on logic puzzles, 2024. URL https://arxiv.org/abs/2409.10502.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures, 2016. URL https://arxiv.org/abs/1605.06465.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks, 2017. URL https://arxiv.org/abs/1709.01686.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model, 2025. URL https://arxiv.org/abs/2506.21734.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks, 2018. URL https://arxiv.org/abs/1711.09485.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks, 2019. URL https://arxiv.org/abs/1711.08393.

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition, 2019. URL https://arxiv.org/abs/1904.01569.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms, 2024. URL https://arxiv.org/abs/2311.12424.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew Dai, Zhifeng Chen, Quoc Le, and James Laudon. Mixture-of-experts with expert choice routing, 2022. URL https://arxiv.org/abs/2202.09368.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models, 2026. URL https://arxiv.org/abs/2510.25741.

Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms, 2016. URL https://arxiv.org/abs/1511.08228.

A Additional Details

A.1 Deep Supervision

We observe that deep supervision (Wang et al., 2025) is helpful for Sudoku-Extreme and ARC-AGI tasks. For the latter, ARC-AGI, it helps with training optimization, but at inference, using a small number of deep supervision steps, such as $N_{\text{sup}} = 2$, gives similar results. It also seems to be a main driver of performance on these tasks.[4] Like other recursive networks, RRM appears to benefit from deep supervision while retaining a simpler, more flexible layer-reuse options.

In deep supervision, a single hidden state is carried across supervision steps. Let $h_t^{(j)}$ denote the state after application t of supervision step j, for $j = 1, \ldots, N_{\text{sup}}$ and $t = 0, \ldots, T-1$. Then

$$h_0^{(1)} = E(x), \qquad h_0^{(j)} = \text{sg}\left(h_T^{(j-1)}\right), \qquad h_{t+1}^{(j)} = f_{r_t^{(j)}}\left(h_t^{(j)}, E(x)\right),$$

where the second expression applies for $j > 1$ and sg stops the gradient. A task loss is applied after each supervision step. The next step therefore continues from the preceding state, but gradients do not cross between supervision steps.

Deep supervision also uses early exiting, a popular method in earlier works on conditional computation, including Graves (2017); Banino et al. (2021). We observe that this is very helpful for probabilistic test-time scaling (Sghaier et al., 2026), since it uses the Q head to select the best candidate.

During deep supervision, we can also truncate gradients within a sequence of random recursive applications. We observe that this improves optimization on Sudoku-Extreme while reducing training computation. Simply, if we have T applications, we truncate gradients through the first M applications.

A.2 Experimental Details

CIFAR-10. The custom Vision Transformer divides each 32×32 image into 4×4 patches and uses post-normalized Transformer blocks with hidden width 256, eight attention heads, an MLP ratio of one, and no dropout. The dense baseline contains eight distinct blocks, while RRM uses the same block design with one or two reusable layers. All models are trained from scratch for 800 epochs with batch size 512 using Adam, an initial learning rate of 10^{-3}, weight decay 5×10^{-5}, label smoothing 0.1, and a learning-rate schedule with 10% warmup, a 40% constant phase, and 50% linear decay. Training uses random crops, horizontal flips, and the CIFAR-10 AutoAugment policy (Cubuk et al., 2019). We reserve a stratified 20% of the official training set for validation, select checkpoints by validation accuracy, and report top-1 accuracy on the official test set.

Sudoku and Zebra. Following Shah et al. (2024), we use the released solver-ordered datasets and eight-layer causal Transformer configuration, with hidden width 576, eight attention heads, MLP width 3456, and dropout 0.2. All models are trained for 320,000 optimizer steps. For Sudoku, we use the complete released training and test sets. For Zebra, we use the complete released training set and a fixed subset of 15,000 released test puzzles, matching the evaluation-set size used by Shah et al. (2024).

Maze-Hard and Sudoku-Extreme. We use the Maze-Hard and Sudoku-Extreme datasets introduced by HRM (Wang et al., 2025) and follow the corresponding setups used by TRM (Jolicoeur-Martineau, 2025). Maze-Hard contains fixed sets of 1,000 training and 1,000 test mazes. For Sudoku-Extreme, we use 1,000 training puzzles with 1,000 augmentations per puzzle and the provided test split. Both backbones have hidden width 512. Maze-Hard uses non-causal Transformer blocks with eight attention heads and an MLP expansion ratio of four, while Sudoku-Extreme uses TRM's position- and channel-mixing blocks with the same expansion ratio. Neither dataset provides a separate validation split, so the provided test splits are also used for periodic checkpoint evaluation.

[4] The ARC Prize analysis also found that HRM's outer refinement loop drove substantial gains on ARC-AGI.https://arcprize.org/blog/hrm-analysis

How we compute T for the baselines. For HRM, T counts every layer in both recurrent modules, giving $T = 2(2 \times 4 + 4) = 24$ from two high-level cycles, two low-level cycles, and four layers in each module. For TRM, each of its three high-level cycles performs the low-level updates and one answer update through its two-layer network, giving $T = 3(4 + 1) \times 2 = 30$ on Maze-Hard and ARC-AGI-1 and $T = 3(6 + 1) \times 2 = 42$ on Sudoku-Extreme.

ARC-AGI-1. We use the ARC-AGI-1 dataset (Chollet, 2019) and follow the HRM and TRM experimental setup (Wang et al., 2025; Jolicoeur-Martineau, 2025). For RRM, we use $L = 2$ and train with $T_{\text{train}} \in \{8, 15\}$ applications and $N_{\text{sup}} \in \{16, 32\}$ maximum supervision steps. The eight-application variants differentiate all applications; for the main 15-application configuration, the first five applications are detached and the remaining ten are differentiated, and we also train a fully differentiated variant. At inference, we vary T_{test} from 8 to 21 and independently study reducing the supervision steps budget from its trained depth down to two.

A.3 Learned Policies for Adaptive Ordering

We also tested replacing uniform layer sampling with a learned routing policy. At application t, the policy samples the next layer from $\pi_\phi(r_t \mid h_t)$, and the layer parameters are trained with the task loss. Because the sampled layer indices are discrete, we train the policy with REINFORCE (Williams, 1992). For a sampled sequence $\mathbf{r}_i$ and reward $R_i = -\ell_i$, the policy loss is

$$\mathcal{L}_{\text{policy}}(\phi) = -\frac{1}{B} \sum_{i=1}^{B} \text{sg}(R_i - \bar{R}) \sum_{t=0}^{T-1} \log \pi_\phi(r_{i,t} \mid h_{i,t}),$$

where $\bar{R}$ is the minibatch mean reward.

The policy also has an exit logit, allowing it to stop before the maximum number of applications. It can therefore learn to reduce computation by using a smaller T. A policy could also concentrate its routes on a subset of the L layers, making unused layers candidates for removal and potentially reducing the parameter count.

However, learning the policy is difficult because it must choose among L^T possible layer orders while the layers themselves are still changing. The reward is observed only after executing the sampled sequence, which makes credit assignment across its routing decisions noisy. Table 7 shows this behavior in an eight-layer CIFAR-10 ablation. The learned policy uses an average of $T = 7.99$ applications and retains all eight layers, closely matching the dense model's computation and parameter count while also adding policy parameters. We also added a penalty on T to encourage shorter computations, but it substantially degraded accuracy. The learned policy therefore provides neither a parameter nor a computation benefit over the dense model and obtains lower accuracy. Uniform random reuse avoids this additional policy optimization. Learning policies that realize both computation and parameter savings is left for future work.

Table 7: CIFAR-10 learned-ordering ablation. The learned policy uses the same eight layers as the dense model, and its reported T is the mean number of applications.

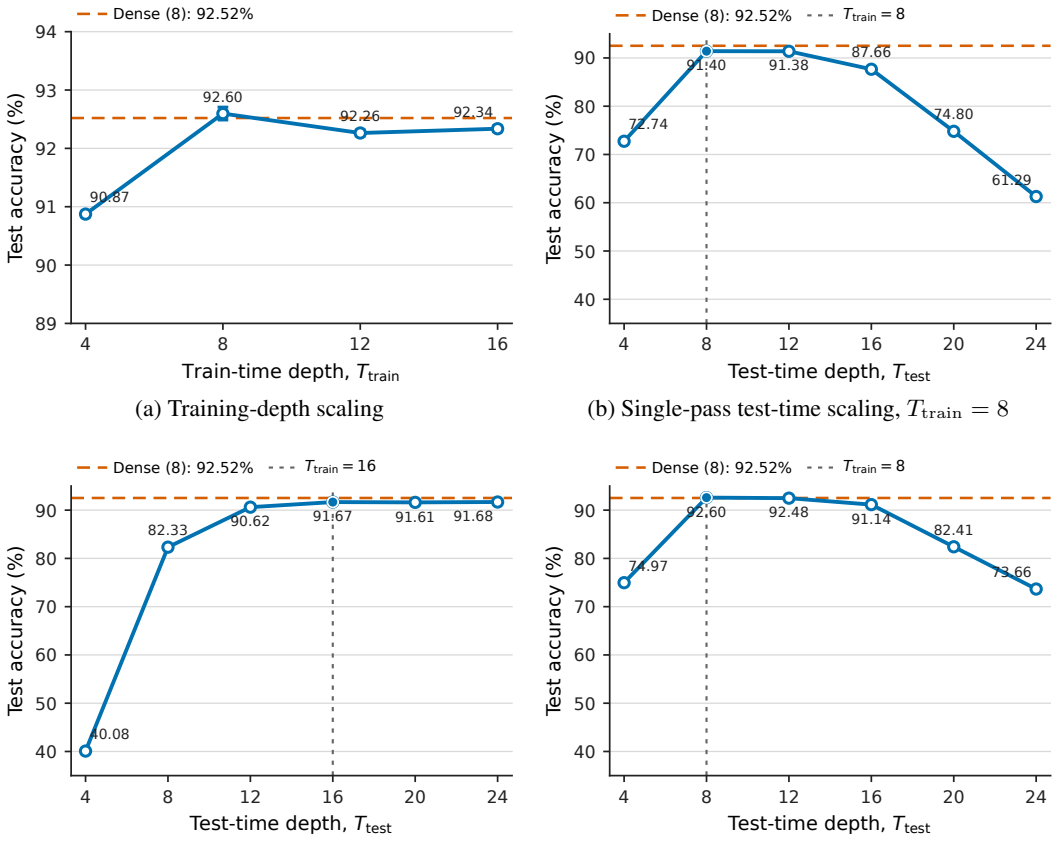
Model	L	T	Accuracy (%)
Dense	8	8	92.71
RRM	8	32	**93.67**
Learned policy	8	7.99	85.07

A.4 Inference Procedures

A.4.1 Monte Carlo Inference

For a fixed depth T, the predictive distribution marginalizes over random layer sequences. We estimate this expectation using M independent samples,

$$\begin{aligned} p_\theta(y \mid x) &= \mathbb{E}_{\mathbf{r} \sim p(\mathbf{r})} \left[p_\theta(y \mid x, \mathbf{r}) \right], \\ \widehat{p}_{\theta,M}(y \mid x) &= \frac{1}{M} \sum_{m=1}^{M} p_\theta\Big(y \mid x, \mathbf{r}^{(m)}\Big), \qquad \mathbf{r}^{(m)} \sim p(\mathbf{r}). \end{aligned} \tag{7}$$



(a) Training-depth scaling

(b) Single-pass test-time scaling, $T_{\text{train}} = 8$

(c) Single-pass test-time scaling, $T_{\text{train}} = 16$

(d) Monte Carlo test-time scaling, $T_{\text{train}} = 8$

Figure 5: CIFAR-10 training-depth and test-time-scaling experiments. All RRMs use $L = 2$. Vertical dotted lines mark T_{train}, and horizontal dashed lines show the dense eight-layer baseline.

Here, $\widehat{p}_{\theta,M}$ is the M-sample Monte Carlo estimator of $p_\theta(y \mid x)$. Setting $M = 1$ uses a single random computation, while larger M reduces variation across sampled layer sequences at the cost of additional forward passes, without changing the model parameters.

A.4.2 PTRM Test-Time Scaling

PTRM (Sghaier et al., 2026) produces K stochastic trajectories for the same input. Before each supervision step j of trajectory m, it perturbs the recurrent latent state with independent Gaussian noise,

$$\widetilde{h}^{(j,m)} = h^{(j,m)} + \epsilon^{(j,m)}, \qquad \epsilon^{(j,m)} \sim \mathcal{N}(0, \sigma^2 I). \tag{8}$$

For RRM, each trajectory also uses independently sampled layer sequences. A trajectory produces a complete candidate solution $\hat{y}^{(m)}$ and a scalar score $q^{(m)}$ from the existing Q head, which is trained to predict whether the candidate solution is correct. PTRM returns

$$m^\star = \arg\max_{m \in \{1,\dots,K\}} q^{(m)}, \qquad \hat{y} = \hat{y}^{(m^\star)}. \tag{9}$$

Unlike Monte Carlo inference, which averages predictive distributions, PTRM preserves the individual candidates and selects one complete solution. The approach requires no retraining. We use the main Sudoku-Extreme configuration of $K = 100$, $N_{\text{sup}} = 64$ supervision steps, and $\sigma = 0.3$. Since this method is expensive, we use a fixed subset of 1,000 test puzzles for our evaluations.

A.5 Additional CIFAR-10 Results

Training-depth scaling. We train separate $L = 2$ RRMs with $T_{\text{train}} \in \{4, 8, 12, 16\}$ and evaluate each RRM at $T_{\text{test}} = T_{\text{train}}$ using Monte Carlo inference with $M = 16$. Figure 5(a) shows a clear

Table 8: ARC-AGI-1 ablations. M is the number of initial applications, out of T_{train}, through which gradients are truncated. Total gives $T_{\text{test}} \times N_{\text{sup}}$. All models use $L = 2$. The TRM-Att result is from Jolicoeur-Martineau (2025).

Model	M	T_{train}	T_{test}	N_{sup}	Total	Pass@2 (%)
TRM-Att	–	30	30	16	480	44.60
RRM	0	8	8	32	256	42.67 ± 0.14
RRM	0	8	15	32	480	42.79 ± 0.26
RRM	5	15	15	32	480	43.79 ± 0.29
RRM	5	15	15	2	30	43.71 ± 0.14
RRM	5	15	21	32	672	43.79 ± 0.14
RRM	0	15	15	32	480	$\mathbf{44.79 \pm 0.19}$

improvement from four to eight applications, followed by a plateau. Increasing the training depth alone therefore does not continue to improve CIFAR-10 accuracy once the model has sufficient depth for the task.

Test-time scaling. We next evaluate the RRMs trained with $T_{\text{train}} \in \{8, 16\}$ using one sampled layer sequence per image and vary T_{test}. Panels (b) and (c) of Figure 5 show that the RRM trained with $T_{\text{train}} = 8$ remains close to its trained-depth performance at $T_{\text{test}} = 12$, but deteriorates when extended further. The RRM trained with $T_{\text{train}} = 16$ is stable for $T_{\text{test}} \in \{16, 20, 24\}$. The stable test-time range therefore depends on the depth encountered during training. Deeper training supports a wider range of test-time applications, but neither RRM improves consistently simply by running longer.

Monte Carlo inference. Finally, we repeat the test-time sweep for the RRM trained with $T_{\text{train}} = 8$ using $M = 16$ sampled sequences per image. As shown in Figure 5(d), Monte Carlo inference raises the accuracy at each tested depth, but it does not prevent the degradation far beyond the training depth. Averaging multiple layer sequences therefore improves accuracy, but does not make additional test-time applications consistently beneficial.

These observations should be treated as specific to CIFAR-10. Maze-Hard and Sudoku-Extreme instead improve beyond their training depths, while Zebra and ARC-AGI-1 largely plateaus. The useful test-time range therefore depends on the task and training setup, with additional recurrence eventually becoming unhelpful or detrimental in some settings. This may reflect the *overthinking* behavior reported in recurrent models (Bansal et al., 2022).

A.6 Additional ARC-AGI-1 Results

Table 8 adds further ARC-AGI-1 ablations. RRM matches the reported TRM baseline at the same total number of applications. We further study gradient truncation during training, which helps optimization on Sudoku-Extreme. In our ARC-AGI-1 ablations, however, the RRM that differentiates all applications performs best at the same inference cost. The currently reported result comes from step 264,282 of 518,227 training steps, and completing training may yield a stronger result. For the RRM that truncates gradients through the first five applications, reducing N_{sup} from 32 to two retains nearly all performance while using $16\times$ fewer applications. We have not yet evaluated the better fully differentiated RRM with fewer supervision steps. Increasing T_{test} from 15 to 21 does not improve the truncated-gradient RRM. Similarly, the model trained with $T_{\text{train}} = 8$ gains only marginally when evaluated at $T_{\text{test}} = 15$.