LIBRARY OF LAYERS (LOL): ADAPTIVELY ORDERING TRANSFORMER LAYERS AT RUNTIME

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer programmers chain together the same reusable computational primitives, for example, conditional statements or loops, but in different orders to implement different algorithms. In contrast, existing language model architectures always execute their layers in the same order. We introduce Library of Layers (LoL), an architecture that dynamically orders its layers during inference to solve the problem at hand. LoL has a single hyperparameter that allows it to transition seamlessly between parameter reuse and sparsity, generalizing the parameter-efficiency and FLOPs-efficiency tradeoffs of dense, depth-recurrent, and mixture of experts architectures in one unified family. We show that LoL achieves competitive performance with all three above baselines when pretrained on text data. On synthetic sudoku and maze data, we observe that the model employs distinct layer orderings to implement different solution strategies, and we can intervene on the layer ordering to force a model to apply a specific solution strategy to a problem instance. In a multilingual natural language setting, we find that LoL chooses different layer order patterns based on language and script. LoL provides a path toward a future in which models dynamically compose their own computational graph, enabling more efficient and controllable reasoning.

1 INTRODUCTION

Many distinct algorithms are built from the same set of computational primitives, composed in different orders to implement different behaviors. For example, a database query engine can combine operations such as filtering, joining, sorting, and aggregation in different sequences depending on the query being executed. Modular programming lets the same reusable components be dynamically composed for many purposes within a single codebase. Transformer language models lack this flexibility: their layers are executed in a fixed, predefined order for every input. This rigidity may make these models less efficient, since useful computational primitives may need to be represented repeatedly at different depths.

Prior work has focused on increasing sparsity of models using mixture of experts (MoE) (Shazeer et al., 2017; Fedus et al., 2022) or reusing layers multiple times during the forward pass in looped models (Dehghani et al., 2018; Lan et al., 2019; Geiping et al., 2026). Other work has investigated skipping layers dynamically during the forward pass (Elhoushi et al., 2024; Raposo et al., 2024; Bae et al., 2025). However, all of these approaches still enforce the order in which the layers of the model must be executed.

In this paper, we introduce *Library of Layers* (LoL), an architecture in which a small router, trained end-to-end without explicit supervision, dynamically selects the next layer from a global library of transformer blocks. We note this architecture encompasses the two methods explored by prior work. A small library of layers forces layer reuse, whereas a large library of layers allows many layers to remain inactive on a given forward pass. By changing only the library size, LoL therefore spans a continuum from highly reused, dense computation to highly sparse computation. We visualize our novel architecture in Figure 1.

Dynamic layer selection introduces a challenge. We want to allow different token positions the flexibility to select different layer orderings, but this means that a vector at one token position extracted by a selected attention layer might attend to vectors at previous token positions extracted by different selected attention layers. We observe that this mismatch makes training unstable and harms

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.

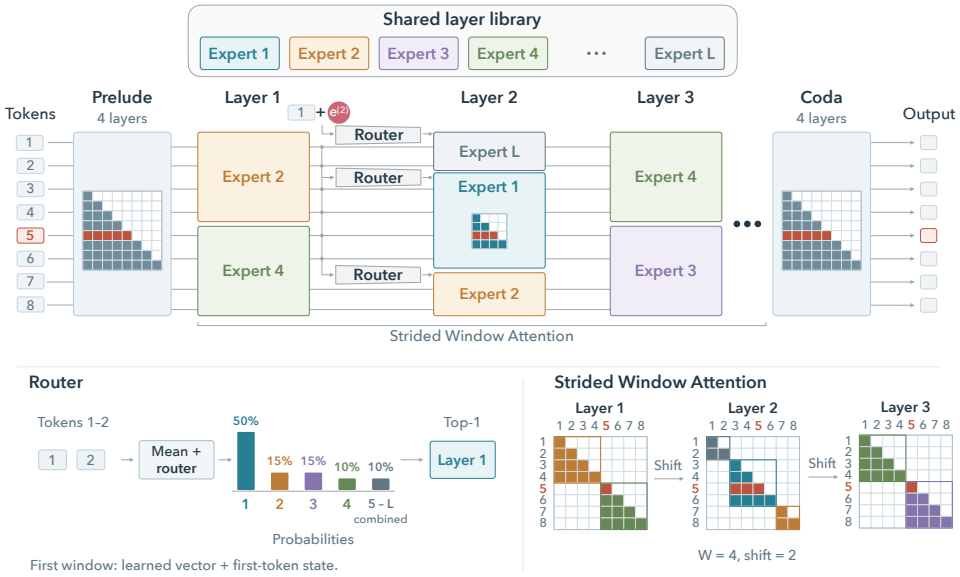

Figure 1: **We introduce Library of Layers (LoL), a novel architecture where the next executed layer is chosen dynamically at runtime.** Our architecture is made up of a prelude, routed core, and coda. Within the routed core, the next executed expert is chosen dynamically by a router from a shared library of layers. We introduce Strided Window Attention to maintain causality across the sequence whilst allowing expert selection to adapt to the current sequence.

performance. At the same time, if we chose a single layer ordering for the entire context at training time, we would risk making layer decisions with too little information or breaking causality.

We therefore introduce a windowed attention masking strategy that simultaneously allows different windows to adaptively select layer orderings while also preventing the above mismatch problem and preserving causality. We partition the context into windows, we route each window using only information available from earlier windows, and we shift the window boundaries across routed depths so that information can propagate between neighboring windows.

We use synthetic and natural-language experiments to study and interpret our models' learned layer orderings. On sudoku and maze tasks, LoL learns distinct layer-ordering patterns for different solution strategies; forcing a strategy's learned routing path at inference time causes the model to follow that strategy. This phenomenon is especially clear in mazes, where all algorithms we consider use the same output syntax and differ only in the order in which they explore cells. We find that reordering the experts controls the cell exploration order itself. In multilingual language modeling, different languages induce distinct execution paths, with languages that share scripts exhibiting more similar routing patterns. On general web text, we vary the library size to induce parameter reuse regimes with a FLOPs-to-parameters ratio comparable to that of looped transformers and also to induce sparse regimes with a FLOPs-to-parameters ratio comparable to MoE models, finding that LoL remains competitive across this range. Finally, at 1B parameters on math data, LoL is competitive with conventional dense transformers in terms of loss and benchmark accuracy in a FLOPs-matched setting.

These results suggest that layer ordering can become a learned part of the computation itself rather than a fixed architectural choice. LoL unifies behaviors associated with seemingly distinct dense, depth-recurrent, and MoE models by varying a single knob, the size of the layer library, while exposing the model's internal execution path directly. Because routing patterns can be extracted and intervened on, LoL also provides a simple mechanism for studying and controlling latent computation.

Our contributions are summarized as follows:

- We propose *Library of Layers* (LoL), a novel architecture in which a router selects blocks to apply from a global library of transformer layers, making layer selection, ordering, and the number of times each layer is used conditional on the input.
- We use maze, sudoku, and multilingual training data to empirically show the LoL router learns input-adaptive patterns from data. Moreover, these router patterns can then be used to control the model at test time by forcing only the layer execution ordering.
- We demonstrate that LoL models perform competitively across a spectrum of sparsity levels, from low sparsity looped models that repeatedly reuse the same layers to high sparsity mixture of experts models.

2 Related Work

Parameter Sharing. Looped transformers, which repeatedly reuse the same transformer blocks, are efficient and Turing complete (Dehghani et al., 2018; Lan et al., 2019; Giannou et al., 2023; Csordás et al., 2024; Saunshi et al., 2025; McLeish et al., 2025; Geiping et al., 2026; Prairie et al., 2026; Huang et al., 2026a). Looped transformers have also been studied in the context of synthetic algorithmic tasks, showing they can outperform fixed-depth counterparts in both learning efficiency and accuracy (Giannou et al., 2023; Yang et al., 2023; McLeish et al., 2024; Fan et al., 2024). Recurrence in other architectural regimes such as graph and convolutional neural networks has also shown promising results (Schwarzschild et al., 2021; Bansal et al., 2022; Ibarz et al., 2022; Schwarzschild, 2023; Bear et al., 2024). Parameter sharing has been explored for many components of the modern transformer architecture, for example, QKV weights and KV caches across layers (Sun et al., 2024; Kayyam et al., 2026).

Most similar to our work is that of Gong et al. (2024). However, Gong et al. (2024) pretrain their model before adding layer reuse and do not dynamically choose every layer in the final model. Gong et al. (2024) use top-k routing with k greater than one, unlike our top-1 routing. They avoid attention space mismatches by averaging the raw weights of the selected experts in the attention computation and combine the output of the FFN via a weighted mean based on the router output weighting.

Routing mechanisms. Mixture of experts (MoE) layers (Shazeer et al., 2017) have become the norm for large transformer-based models (OpenAI, 2025; Xu et al., 2026), often implemented with top-k token-based routing according to the current hidden state. Huang et al. (2026b) extend this to share a global pool of experts so that all layers can select any expert. Others add a skip option to the router, allowing tokens to dynamically skip layers (Raposo et al., 2024; Bae et al., 2025). Ternovtsii & Bilak (2026) route through multiple smaller transformer layers instead of only routing the MLP, as done in a conventional MoE.

3 How to Learn the Order of a Forward Pass

3.1 The Library

The architecture is built from decoder-only transformer blocks (Vaswani et al., 2017), organized into a prelude, a routed core, and a coda. Following prior work on looped transformers (Saunshi et al., 2025; Geiping et al., 2026), we use adaptivity only in the middle of our LoL models.

The *prelude* P and *coda* C are fixed-order stacks of decoder layers with full causal attention. They surround the *routed core*, which selects layers from a library $\{E_1, \ldots, E_L\}$ over D routed depths. Hence, given a core depth of D, a library of L decoder layers, and an input sequence $\boldsymbol{x} \in V^n$, these three groups produce the output probabilities $\mathbf{p} \in \mathbb{R}^{n \times |V|}$ as shown in Algorithm 1. We visualize our forward pass in Figure 1. For our $360M$ parameter models, we use 4 layers in the prelude, 4 layers in the coda, and a depth of 16 in the routed core.

3.2 Maintaining Causality

The main problem when dynamically choosing experts to execute is maintaining causality when choosing the next expert, as at inference time we will not have the whole context to base our routing decisions on. One could route for the whole sequence based upon the first token, but this will fix

Algorithm 1 Library of Layers Forward Pass

Require: Input $\mathbf{x} \in V^n$, core depth D, window size W, stride S; model parameters $\Theta = \{P, \{E_m\}_{m=1}^{L}, \text{Router}, \{\mathbf{e}^{(i)}\}_{i=1}^{D}, C\}$

$\mathbf{s}^{(0)} \leftarrow P(\mathbf{x})$ ▷ Full causal attention
for $i = 1, \ldots, D$ **do**
  Partition $p \in \{0, \ldots, n-1\}$ into $I_0, \ldots, I_{K-1}$ s.t. $w_p^{(i)} = \lfloor (p + ((i-1)S \bmod W))/W \rfloor$
  $\boldsymbol{\mu}_k \leftarrow \text{Mean}(\mathbf{s}_{I_k}^{(i-1)}), \quad k = 0, \ldots, K-1$
  for $k = 0, \ldots, K-1$ **do**
    if $k = 0$ **then**
      $\mathbf{r}_k \leftarrow \frac{1}{2}(\mathbf{e}^{(i)} + \mathbf{s}_0^{(i-1)})$ ▷ $\mathbf{e}^{(i)}$: learned vector per depth
    else
      $\mathbf{r}_k \leftarrow \frac{1}{k}\sum_{j=0}^{k-1} \boldsymbol{\mu}_j$ ▷ Mean of prior windows only
    end if
    $(\rho_k, \alpha_k) \leftarrow \text{Top1}(\text{Router}(\mathbf{r}_k))$ ▷ $\boldsymbol{\rho_k}$: selected layer; $\boldsymbol{\alpha_k}$: weight
    $\mathbf{y}_k \leftarrow E_{\rho_k}(\mathbf{s}_{I_k}^{(i-1)})$ ▷ Causal attention within window
    $\mathbf{s}_{I_k}^{(i)} \leftarrow \mathbf{s}_{I_k}^{(i-1)} + \alpha_k\big(\mathbf{y}_k - \mathbf{s}_{I_k}^{(i-1)}\big)$
  end for
end for
$\mathbf{p} \leftarrow \text{softmax}(C(\mathbf{s}^{(D)}))$ ▷ Full causal attention
return $\mathbf{p}$

expert choice too early. One could route per token, but then the attention spaces between neighboring tokens may be different at each layer. To overcome this problem we route per chunk of the sequence and stride the chunking between transformer layers to allow information to flow between adjoining chunks. Specifically, at routed depth i, let W be the window size and S the stride between depths. A position $p \in \{0, \ldots, n-1\}$ belongs to window I_k when $w_p^{(i)} = k$, where:

$$w_p^{(i)} = \left\lfloor \frac{p + ((i-1)S \bmod W)}{W} \right\rfloor$$

All tokens in a window share one selected expert and router weight. Within that expert, token p may attend only to $\mathcal{A}_p^{(i)} = \{q \mid 0 \le q \le p,\ w_q^{(i)} = w_p^{(i)} = k\}$. The striding of boundaries between depths causes successive windows to overlap, allowing information to flow between adjoining windows while preserving causality. In addition to this, in the prelude and coda we use full-sequence causal attention.

To maintain causality across router decisions, the router for each window k is only given information from prior windows in the sequence (i.e., windows j for $0 \le j < k$). Precisely, we input the mean of the mean hidden states of the prior windows to the router. For the first window I_0 of the sequence, as there are no prior windows to condition on, during training we learn a depth-specific vector $\mathbf{e}^{(i)}$ and mean this with the hidden state of the first token of the sequence, and input this to the router. We apply loss-free load balancing (Wang et al., 2024; Liu et al., 2024) and apply a learned depth-dependent scale and bias after each normalization in the transformer block, using FiLM (Perez et al., 2018). We describe our implementation of FiLM in Appendix B for completeness.

3.3 Growing

As D and L do not have to be related, we can increase D over the course of training, saving FLOPs during early training when both the router and layers are undertrained. This allows the routers to explore layers quickly and cheaply during early training, as the depth of the model is smaller. This can be viewed as similar to increasing the number of loops of a recurrent model over training (McLeish et al., 2025) or progressive stacking (Gong et al., 2019) when training fixed-depth transformers.

Additional architecture and training details are provided in Appendix B. We provide ablations for our design decisions in Appendix A.

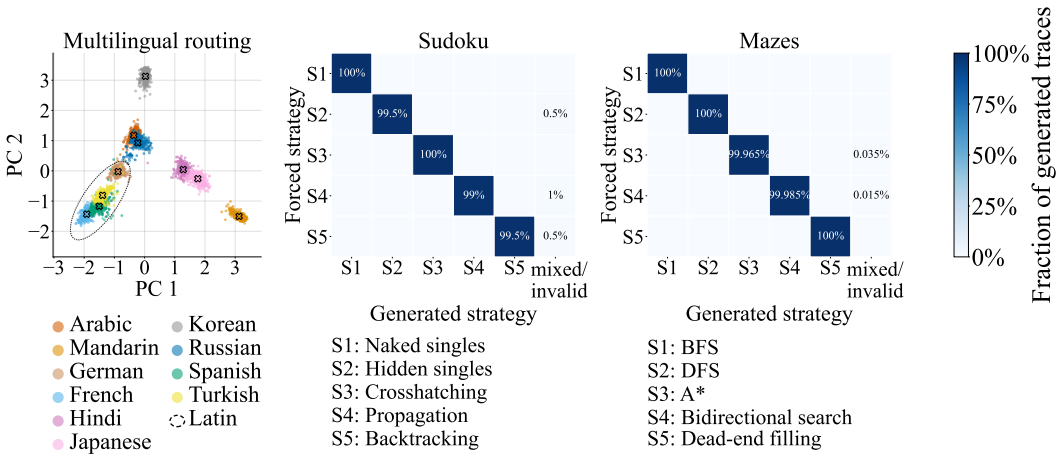

Figure 2: **Library of Layers learns distinct paths for different languages and distinct strategies when solving sudoku puzzles and mazes.** Left: PCA projection of the layer execution paths of a $360M$ parameter LoL model trained on an equal mixture of 10 languages. We see the paths taken are distinct per language and Latin-script languages use similar paths. Middle and Right: We train $360M$ parameter LoL models to solve sudoku (middle) and mazes (right), using 5 different strategies. We then force the mode path per strategy at inference time and see the models use the correct strategy (algorithm) with close to 100% accuracy.

4 Experiments

We conduct an analysis of LoL's capabilities using a range of synthetic and real-world data. We begin by using multilingual and synthetic tasks to trace what LoL learns during training and find the path the model uses is often interpretable in these regimes. Next, we train LoL on a mixture of math data, showing LoL can learn to perform language modeling and mathematical reasoning. Finally, we vary the sparsity of LoL models while training on web text and find LoL is competitive with widely used architectures at a wide range of sparsities.

4.1 Teaching the Router to Be a Librarian

We begin by studying the router behavior in three training tasks: multilingual text, sudoku, and mazes. We observe that the models learn distinct routing paths for each of these tasks.

4.1.1 Multilingual

First, we train $360M$ parameter models on an equal mix of 10 different languages from FineWeb-2 (Penedo et al., 2025) for 10 billion tokens. On a held-out validation set, for each sampled window, we encode the router's selected layer at every depth as a one-hot vector. We then concatenate these one-hot vectors and project the resulting vector into two dimensions using PCA. In Figure 2 (left), we see that the layers executed by the model are distinct per language and the Latin-script languages all share a similar layer execution order. In Appendix Figure 10, we see LoL gradually learns to distinguish between languages and route them through different computational paths by dynamically reordering the layers in its library over the course of training.

4.1.2 Sudoku

Next, we analyze the toy task of sudoku. To construct our dataset, we generate sudoku grids by randomized depth-first filling under the rules of sudoku. We use five solving strategies, each with its own step syntax, fully described in Table 2 in the appendix. We then retain between 30 and 36 digits in the grid, ensuring it is still solvable by all strategies. Duplicate grids are not included in the data. Each puzzle is solved using all five strategies, and every trace becomes one sequence

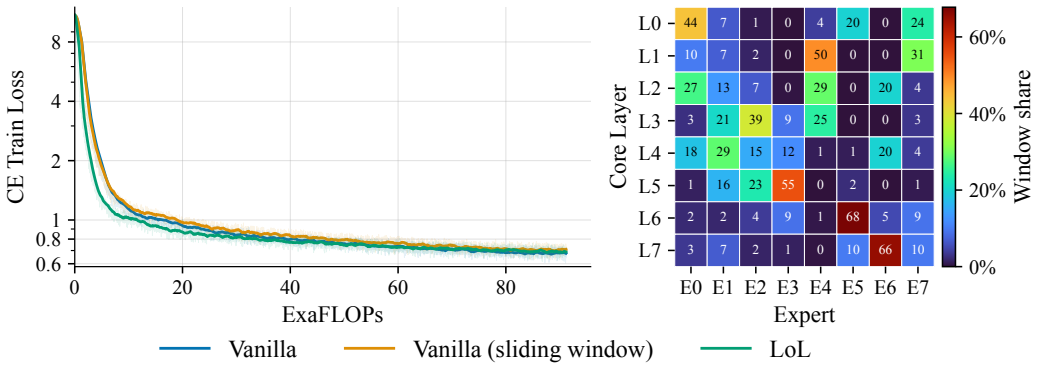

Figure 3: **Library of Layers is competitive with vanilla dense models.** Left: Cross-entropy training loss over training FLOPs. We see growing gives LoL an early loss advantage and all models finish with comparable training loss. Right: Heatmap of expert utilization of one LoL model. We see experts mainly tend to be used in groups of neighboring layers, but some are used throughout the model.

of the dataset. In total, we have 20M sequences, which equates to approximately $10B$ training tokens. Our data is formatted as `Q : <Input Grid> [Sx] R : <Solving Steps> A : <Final Grid> <eos>`, where `[Sx]`, with x$\in \{1, 2, 3, 4, 5\}$, identifies the solving strategy used to generate the reasoning trace.

During training, we compute the loss over only the solving steps and final grid, masking the question and strategy description. We mask the strategy description tokens from all parts of the network but the router; that is, no token outside the strategy description can attend to it in the prelude, coda, or routed core.

For each strategy, we find the *mode path* by running the trained model with teacher forcing on approximately 450 held-out validation puzzles containing the strategy description. At each window position in the document and at each routed depth, we independently select the expert chosen most frequently across these puzzles. We report both the mode paths and how consistently the router agrees with the mode path at each depth in Appendix Figures 12 and 11, respectively.

We then test LoL on 200 new sudoku grids, replacing the strategy description with a neutral filler, `[...]`, and forcing each window to follow the corresponding *mode path* for the requested strategy. This tests whether choosing the path alone is enough to make the model follow that strategy. When measuring accuracy, we identify strategy from step syntax; a single step in another syntax, or a malformed step, makes the trace mixed or invalid. We replay placements to check each step's legality and strategy adherence, consistency with the generated grid, and agreement of that grid with the true solution.

In Figure 2 (middle), we plot the confusion matrix when we force the execution order of the layers to the mode path for each strategy. We see that the model uses the expected strategy with close to 100% accuracy when the layer execution order is forced to the mode path for that strategy. Appendix Figure 13 also shows that LoL succeeds in solving the sudoku grid with close to 100% accuracy.

4.1.3 MAZES

Finally, we study mazes, where all five strategies share the same trace syntax, a list of visited cells, and only differ in the order in which they visit cells. We generate loop-free mazes on an 8×8 grid using randomized recursive backtracking and randomly choose distinct start and goal cells. We solve each maze using BFS (Cormen et al., 1990), DFS (Cormen et al., 1990), A* (Hart et al., 1968), bidirectional search (Pohl, 1969), and dead-end filling. See Appendix Table 3 for examples of each strategy.

Model	ARC-E	ARC-C	HellaSwag	PIQA	GSM8K
Random	25.0	25.0	25.0	50.0	0.0
Vanilla	40.8 ± 1.0	28.2 ± 1.3	31.4 ± 0.5	56.6 ± 1.2	29.7 ± 1.3
Vanilla (sliding window)	49.5 ± 1.0	28.5 ± 1.3	31.3 ± 0.5	58.8 ± 1.1	27.5 ± 1.2
LoL	50.3 ± 1.0	29.1 ± 1.3	31.2 ± 0.5	59.7 ± 1.1	27.9 ± 1.2

Table 1: **Library of Layers is competitive across benchmarks.** We benchmark the same models shown in Figure 3 on ARC, HellaSwag, PIQA, and GSM8K, seeing all models perform comparably.

We ensure all mazes have a unique solving trace per strategy and a solution path of at least 12 cells. We also require DFS to backtrack at least once and BFS to expand at least three cells outside the solution path. As in our sudoku experiments, we organize each example into a sequence of problem-solving steps, remove duplicate mazes, only allow the router to view the strategy description, and compute the loss only over the solving steps and final answer. We train on approximately $10B$ tokens of data made up of 7.2M mazes and 36M sequences, one per strategy for each maze.

We then repeat the sudoku procedure for mazes, extracting each algorithm's mode path from approximately 900 held-out validation sequences using the same teacher-forced procedure. We evaluate LoL on 20,000 unseen mazes per strategy. Unlike the strategies used for sudoku, the maze solving strategies all share the same output syntax and differ in the order in which they visit cells. We therefore ask whether LoL can learn these different exploration orders by changing the order of its own experts.

Figure 2 (right) shows that forcing the corresponding mode path produces the requested algorithm's trace with nearly 100% accuracy. Appendix Figure 16 highlights that LoL solves all the unseen mazes with the requested algorithms with a score close to 100%. These results show that forcing LoL to follow a particular computational path can control the algorithm used, even when all strategies share the same output syntax.

4.2 Mathematical Reasoning at the 1B Scale

Next, we scale training to $1B$ parameters with models that are 16 layers deep. For LoL, we divide the 16 layers into 4 prelude layers, 4 coda layers, and a depth of 8 in the routed core. We train on a mixture of Nemotron (Mahabadi et al., 2025) and OpenMath (Toshniwal et al., 2024) data. We also grow the LoL models using a $1 - sqrt$ schedule for 75% of training, following McLeish et al. (2025). We train for approximately 90 ExaFLOPs; this equates to approximately $20B$ tokens for the most efficient model.

In Figure 3 (left), we see our models are competitive in terms of training loss with the parameter equivalent dense baselines. We use both full attention and sliding window attention as baselines, where we set the size of the sliding window to be equal to the window size in the strided attention in LoL. In Appendix Figure 19 and Table 4, we include a comparison to the vanilla model with strided attention. This model performs slightly worse than the vanilla and vanilla sliding window attention models. We note that the math training data is synthetic; therefore, loss values in Figure 3 are lower than the FineWeb ones in other plots. Moreover, in Figure 3 (right), we see our models are learning to use different experts at different routed depths. In this setting, where the number of experts is equal to the depth, we see clusters form where experts are often used across neighboring layers. However, some experts are used across a wider range of depths.

In Table 1, we benchmark the same models on ARC-Easy (Clark et al., 2018), ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), and GSM8K (Cobbe et al., 2021). We use `lm-evaluation-harness` (Gao et al., 2024) and report the standard errors for each benchmark in the table. We use these benchmarks as they are known to provide a stronger, more reliable signal at smaller model sizes and token counts (Magnusson et al., 2025; McLeish et al., 2026). For all evaluations except GSM8K, we use 5-shot; for GSM8K, we use 3-shot to ensure responses stay within the 1024 token context. We see almost all evaluations are competitive and remain within 2 standard errors of each other on all benchmarks, showing LoL is competitive across a range of modern architectures. The only exception is the vanilla model, which scores poorly on ARC-Easy.

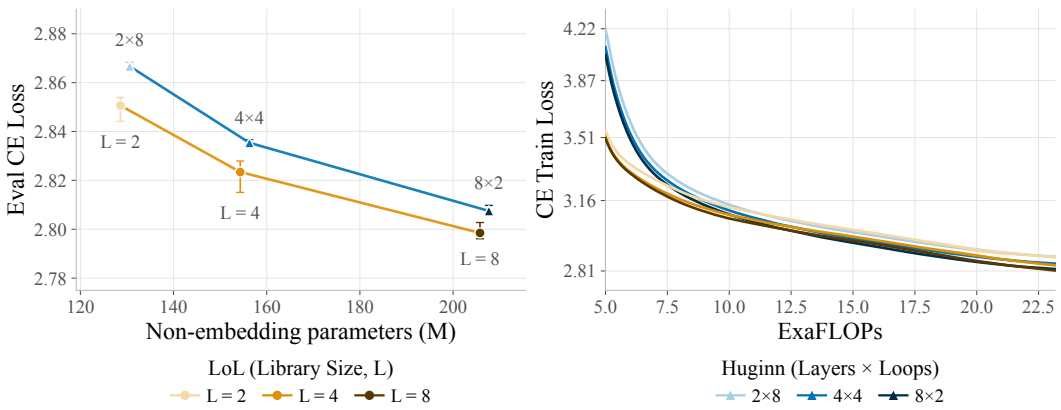

Figure 4: **Library of Layers is competitive with looped transformer models.** Left: Cross-entropy evaluation loss, matching the number of unique layers in the core block of the LoL and looped transformer. We see LoL performs competitively with looped transformer models and offers more flexibility in layer ordering. Error bars are min/max loss over 3 random seeds. We note larger variance for the LoL models as changing the random seed changes the paths explored during training. Right: Cross-entropy training loss over training FLOPs, averaged over 3 random seeds per model.

4.3 How Big Is Your Library?

The shared library of layers can be either small or large. To demonstrate this, we compare to models across a range of sparsity levels. For all comparisons, we sweep at least 6 learning rates, experiment with different learning rate schedules, and test different learning rates for the Muon and AdamW parameters.

4.3.1 Comparison to Looped Transformers

First, we compare to looped transformers. Following the analysis of Huang et al. (2026a), we use the Huginn (Geiping et al., 2026) architecture for our looped model baseline. We compare models at several distinct parameter counts, reusing layers to reach an effective parameter count of $360M$. To ensure a fair comparison, we fix the library size in our LoL models to match the size of the recurrent block in the Huginn models. We train for $10B$ tokens of FineWeb data (Penedo et al., 2024).

In Figure 4, we see that our LoL models are competitive in terms of loss with the parameter matched looped models. We note that the LoL models have slightly higher variance in final loss over three seeds. We hypothesize this is due to sensitivity to the initialization of the router causing different paths to be explored during training. In Appendix Figure 17, we show the routing diversity for each LoL model shown in Figure 4. Unlike Huginn, LoL can offer sparse behavior by simply scaling the size of its library.

4.3.2 Comparison to Sparse Mixture of Experts

Finally, we compare to sparse mixture of experts models. We compare at a $3:2$ sparsity ratio: the MoE model uses three experts per layer with two active, while LoL uses 24 library layers over 16 routed depths. We increase the MoE experts' intermediate expansion factor relative to our standard 360M model to match active parameters per token as closely as possible. To align computation, LoL uses MoE prelude and coda layers, and the MoE baseline uses strided attention at the same depths as LoL. We again use growing for the LoL models and train all models for approximately 17 ExaFLOPs; this equates to approximately $10B$ tokens for the most efficient model.

In Figure 5 (left), the MoE baseline achieves a slightly lower training loss than LoL, with a gap of approximately 0.05 (2.75 vs. 2.8). We note it is common for higher cross-entropy loss to coexist with strong downstream performance in latent reasoning models (Saunshi et al., 2025; Huang et al., 2026a). In Figure 5 (right), we visualize the expert utilization across the core layers on an evaluation

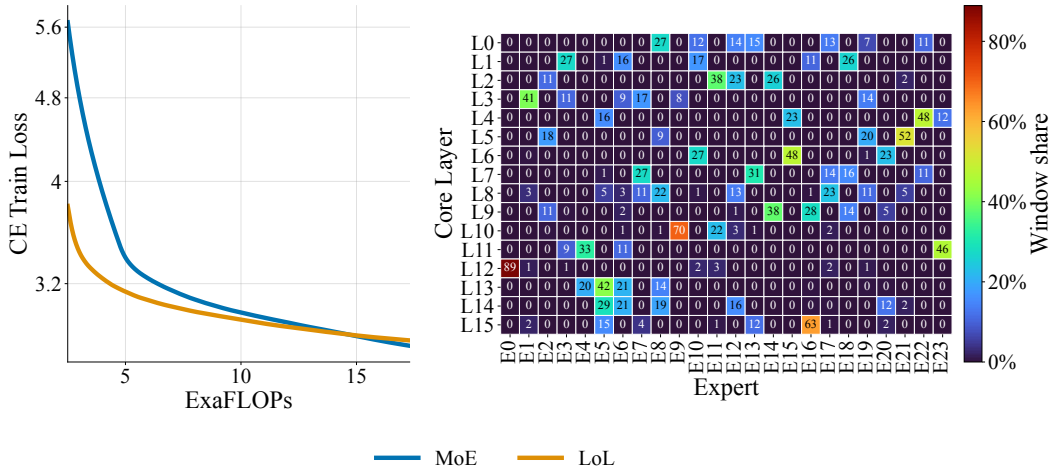

Figure 5: **Library of Layers is competitive with mixture of experts models.** Left: Cross-entropy training loss over training FLOPs, averaged over 3 random seeds. We see growing gives LoL an early loss advantage and both models finish training with similar training losses. Right: Heatmap of expert utilization of one LoL model. We see some experts are heavily used by specific core layers or neighboring groups of core layers, while others are used diversely over depths, showing LoL learns to use the sparsity effectively.

set. Experts vary in depth specialization: some favor earlier or later layers, while others are used across depths, highlighting the flexibility of LoL models on real-world language modeling data.

5 Discussion

LoL generalizes the parameter reuse and sparsity of looped models, standard feedforward transformers, and MoEs into a single model family, allowing us to switch between these regimes simply by varying the library size. While our design addresses several challenges we discuss above, this paper is our initial exploration into dynamic layer ordering, and we believe that future work will vastly improve on our design choices. Moreover, the goal of our experiments is to show not merely that an architecture with dynamic layer ordering can achieve competitive performance, but also that it provides a new tool for model interpretability and control over a model's reasoning strategies.

We now enumerate several directions for future work that we believe are especially promising. First, LoL is built with controllability at the core. We see in Figure 2 that we can find distinct routing paths underlying languages and algorithms, and we can intervene on the layer order at inference to control the model. This control may be extended to chat settings by finding patterns related to lying, scheming, and safety (Park et al., 2024), providing a native layer of safety in latent reasoning. Next, LoL can in principle employ a larger computational depth at inference time than at training time, offering a mechanism for test-time scaling and increasing its capacity for latent reasoning. Finally, another promising direction is to combine test-time learning (Chen et al., 2026; Hotsko et al., 2026; Zhao & Jones, 2026) with our top-1 routing. For example, instead of routing over full transformer blocks, one may route over a growing library of test-time learnable adapters for each layer. This approach would allow the parametric memory to be available to all layers of the model.

Limitations. The biggest limitation of our work is model size. Constrained by GPU access, we train models with up to $1B$ parameters with token budgets at or above compute-optimal (Hoffmann et al., 2022). Future work may implement these models at larger sizes to show that benefits scale. Moreover, top-1 routers are prone to collapse during training. Although most of our runs avoid dead experts, we do see this in some cases and encourage future work on efficiently optimizing top-1 routers.

AI USE STATEMENT

In this work, we used generative AI tools for:

1. Design or provide feedback on research methodology or experiments. The core ideas of all experiments are human generated. We use AI to improve the rigor of experiments. For example, we use AI to help select the exact algorithms used to solve mazes to make sure the algorithms are sufficiently distinct from each other.
2. Implement methods. We use coding agents for implementation. We check all code and take full responsibility for its correctness.
3. Support qualitative and thematic data analysis. We use agents to help with making plots and visualizations.

We have not used generative AI tools for:

1. Clean and reformat datasets.
2. Interpret results.
3. Propose or refine hypotheses.

These required disclosure tasks are not applicable to this work:

1. Help develop theoretical models or conceptual frameworks.
2. Formulate mathematical claims.
3. Provide critical ingredients for proving mathematical claims.
4. Assist in the writing of proofs.
5. Assist with translation.

Additionally, we used generative AI tools for:

1. Editing the research paper to improve readability. We use agents to find and correct grammar mistakes and point out sections which require more human editing.
2. Identifying relevant literature. We use agents to perform an additional literature search on top of our human search.

We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work describes a novel architecture with promising performance. The approach described in this work is complementary to other scaling approaches, for example, via parameter count or verbalized chain of thought. Similar concerns around model capabilities apply, none of which we feel are specific to this new architecture. Finally, we observe that LoL exhibits a degree of controllability when forcing layer execution order; this may have broader ethical implications for the governance and deployment of controllable and interpretable AI systems.

REPRODUCIBILITY STATEMENT

We give architecture and training details in Section 3, Section 4, and Appendix B. Upon acceptance, we will release all code.

REFERENCES

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. *Advances in Neural Information Processing Systems*, 35:20232–20242, 2022.

Jay Bear, Adam Prugel-Bennett, and Jonathon Hare. Rethinking deep thinking: Stable learning of algorithms using lipschitz constraints. *Advances in Neural Information Processing Systems*, 37: 97027–97052, 2024.

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

Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, pp. 95–136, 2022.

Zeming Chen, Angelika Romanou, Gail Weiss, and Antoine Bosselut. Perk: Long-context reasoning as parameter-efficient test-time learning. In *International Conference on Learning Representations*, volume 2026, pp. 10350–10377, 2026.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of machine learning research*, 24(240):1–113, 2023.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Thomas H Cormen, Charles Eric Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 1990.

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D. Manning. Moeut: Mixture-of-experts universal transformers. *arXiv preprint arXiv:2305.13245*, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. *arXiv preprint arXiv:2409.15647*, 2024.

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2026.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of bert by progressively stacking. In *International conference on machine learning*, pp. 2337–2346. PMLR, 2019.

Zhuocheng Gong, Ang Lv, Jian Guan, Wei Wu, Huishuai Zhang, Minlie Huang, Dongyan Zhao, and Rui Yan. Mixture-of-modules: Reinventing transformers as dynamic assemblies of modules. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 20924–20938, 2024.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Liliana Hotsko, Yinxi Li, Yuntian Deng, and Pengyu Nie. Code2lora: Hypernetwork-generated adapters for code language models under software evolution. *arXiv preprint arXiv:2606.06492*, 2026.

Benhao Huang, Chufan Shi, Junlin Chen, Shicheng Wen, Zhengzhong Liu, Eric Xing, and Xuezhe Ma. Towards looped models done right — part i: Topology, input injection, recurrent-state design, July 2026a. URL https://ifm-research.notion.site/Towards-Looped-Models-Done-Right-3ade511912ec8128987dfeb7a5580043.

Minbin Huang, Han Shi, Chuanyang Zheng, Yimeng Wu, Guoxuan Chen, Xintong Yu, Yichun Yin, and Hong Cheng. Unipool: A globally shared expert pool for mixture-of-experts. *arXiv preprint arXiv:2605.06665*, 2026b.

Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, et al. A generalist neural algorithmic learner. In *Learning on graphs conference*, pp. 2–1. PMLR, 2022.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Ali Kayyam, Anusha Madan Gopal, and M Anthony Lewis. Do transformers need three projections? systematic study of qkv variants. *arXiv preprint arXiv:2606.04032*, 2026.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ian Magnusson, Nguyen Tai, Ben Bogin, David Heineman, Jena D Hwang, Luca Soldaini, Akshita Bhagia, Jiacheng Liu, Dirk Groeneveld, Oyvind Tafjord, et al. Datadecide: How to predict best pretraining data with small experiments. *arXiv preprint arXiv:2504.11393*, 2025.

Rabeeh Karimi Mahabadi, Sanjeev Satheesh, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc-math: A 133 billion-token-scale high quality math pretraining dataset. *arXiv preprint arXiv:2508.15096*, 2025.

Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, et al. Transformers can do arithmetic with the right embeddings. *Advances in Neural Information Processing Systems*, 37: 108012–108041, 2024.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. *arXiv preprint arXiv:2511.07384*, 2025.

Sean McLeish, John Kirchenbauer, David Miller, Siddharth Singh, Abhinav Bhatele, Micah Goldblum, Ashwinee Panda, and Tom Goldstein. Gemstones: A model suite for multi-faceted scaling laws. *Advances in Neural Information Processing Systems*, 38:123419–123454, 2026.

Meta. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, September 2024. URL https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Peter S. Park, Simon Goldstein, Aidan O'Gara, Michael Chen, and Dan Hendrycks. Ai deception: A survey of examples, risks, and potential solutions. *Patterns*, 5(5):100988, 2024. ISSN 2666-3899. doi: https://doi.org/10.1016/j.patter.2024.100988. URL https://www.sciencedirect.com/science/article/pii/S266638992400103X.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. Fineweb2: One pipeline to scale them all–adapting pre-training data processing to every language. *arXiv preprint arXiv:2506.20920*, 2025.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Ira Pohl. *Bi-directional and heuristic search in path problems*. PhD thesis, Department of Computer Science, Stanford University, 1969.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025.

Avi Schwarzschild. *Deep Thinking Systems: Logical Extrapolation With Recurrent Neural Networks*. PhD thesis, University of Maryland, College Park, 2023.

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. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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

Ivan Ternovtsii and Yurii Bilak. Mixture of layers with hybrid attention. *arXiv preprint arXiv:2605.09516*, 2026.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Tianyu Zhao and Llion Jones. Fast-weight product key memory. *arXiv preprint arXiv:2601.00671*, 2026.

TABLE OF CONTENTS

A Additional Architecture Ablations **16**
A.1 Library of Layers 16
A.2 Baselines 17

B Additional Architecture and Training Details **17**

C Additional Experimental Information **18**
C.1 Multilingual 18
C.2 Sudoku 19
C.3 Maze 21
C.4 Looped Transformers 22
C.5 MoE 23
C.6 Math 23

A Additional Architecture Ablations

A.1 Library of Layers

In Figure 6, we ablate the impact of growing LoL during training. In Figure 7, we ablate the impact of auxiliary loss versus loss-free load balancing. In Figure 8, we see sharing router weights across layers has minimal impact on final training loss.

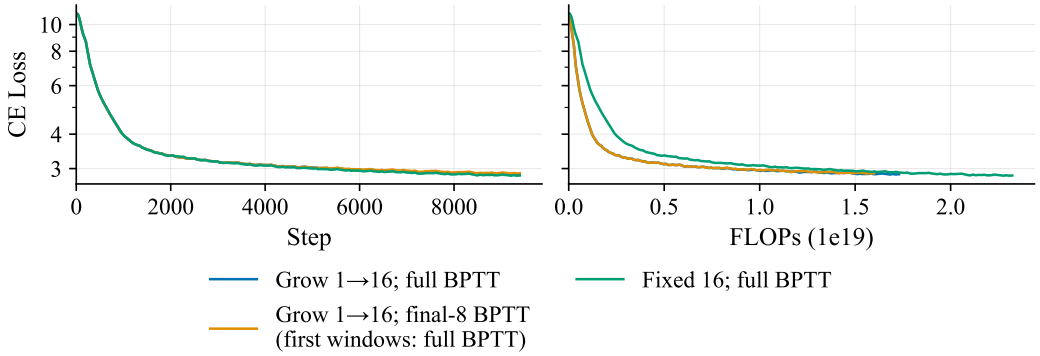

Figure 6: Increasing depth over training saves FLOPs with only a minor increase in final loss.

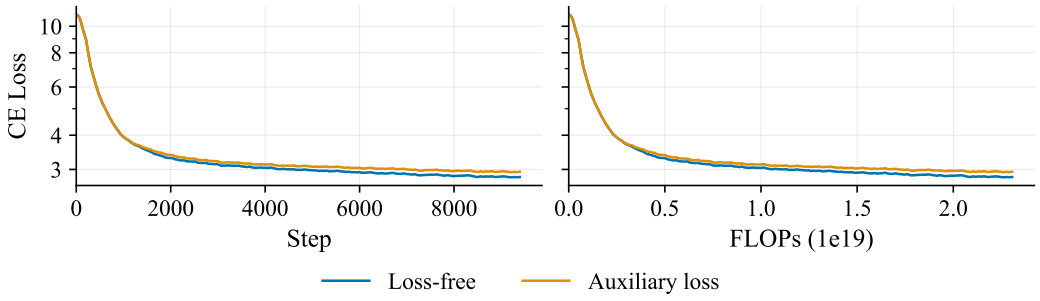

Figure 7: Loss-free load balancing (Wang et al., 2024) achieves a lower final cross-entropy loss when compared to auxiliary loss load balancing methods.

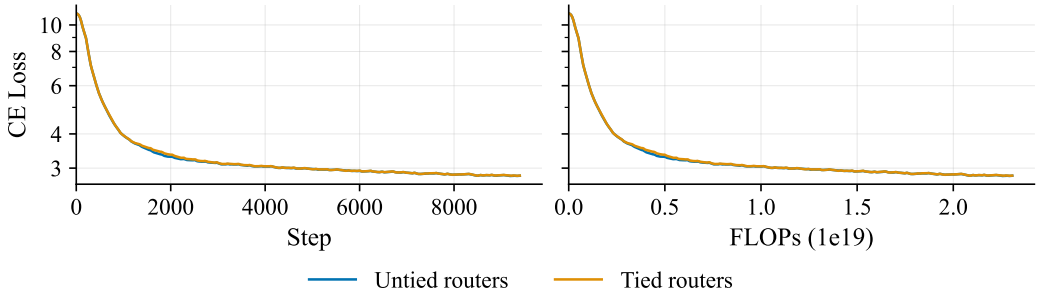

Figure 8: Sharing router weights across layers causes minimal change in cross-entropy loss.

A.2 BASELINES

In Figure 9, we see varying attention patterns in vanilla models has a small impact on training loss.

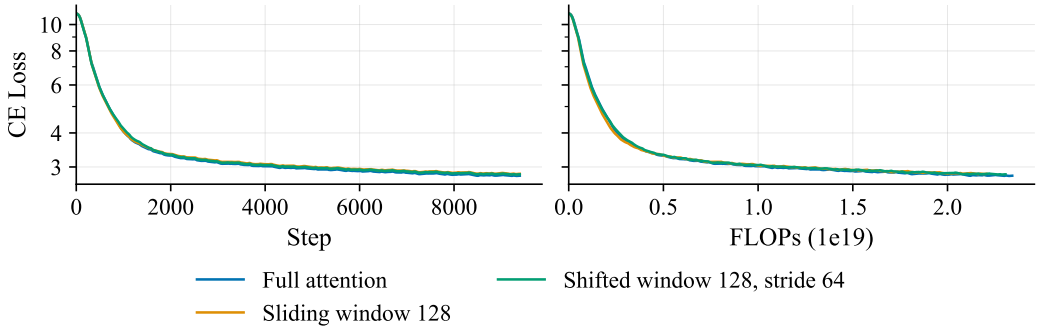

Figure 9: Varying the attention pattern for dense fixed-depth models, we see full attention achieves the lowest loss.

B ADDITIONAL ARCHITECTURE AND TRAINING DETAILS

Our main experiments are conducted at the 360M parameter scale based on the GPT-2 Medium architecture (Radford et al., 2019). Unlike GPT-2, our models use RoPE (Su et al., 2024), RMSNorm (Zhang & Sennrich, 2019), SwiGLU MLPs (Shazeer, 2020), and bias free attention (Chowdhery et al., 2023). Our 1B model shape is based upon the Llama-3.2-1b shape (Meta, 2024), using the same architecture as the 360M model. We use the GPT-NeoX tokenizer (Black et al., 2022) for our predominantly English FineWeb training experiments and the larger OLMo3 (Olmo et al., 2025) tokenizer for our multilingual experiments. For our toy sudoku and maze experiments, we use small tokenizers specifically for the vocabulary of each task. For natural language experiments, we use $W = 128$ and $S = 64$, with a total context length of 1024 tokens. For sudoku and mazes, we set the first window to span the question and strategy description. Subsequent windows use $W = 128$ and $S = 64$ for sudoku, and $W = 64$ and $S = 32$ for mazes.

To train the router, we use loss-free load balancing (Wang et al., 2024), adding a per-expert bias only for expert selection, $\rho_k = \arg\max_m (a_{k,m} + b_m)$, while weighting the selected expert using the unmodified router score, $\alpha_k = a_{k,\rho_k}$. As in MoE routing (Shazeer et al., 2017; Fedus et al., 2022), this retains a differentiable path from the loss to the router through α_k, without differentiating through the discrete selection.

For our LoL models, we apply a learned, depth-dependent scale and bias after each normalization in the transformer block. Specifically, in place of the regular normalization in the transformer block, we use: $\boldsymbol{\gamma}_s \odot \mathrm{Norm}_s(\mathbf{x}) + \boldsymbol{\beta}_s, \qquad s \in \{\mathrm{attn}, \mathrm{ffn}\}$, where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are depth-dependent.

We use the Muon optimizer (Jordan et al., 2024) with AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017). We note we use PyTorch Muon with `adjust_lr_fn=match_rms_adamw`, and all learning rates should be taken in this context. We use loss-free load balancing (Wang et al., 2024) to train routers. We use document blocked attention, meaning documents never cross attend to each other in any experiments.

C Additional Experimental Information

C.1 Multilingual

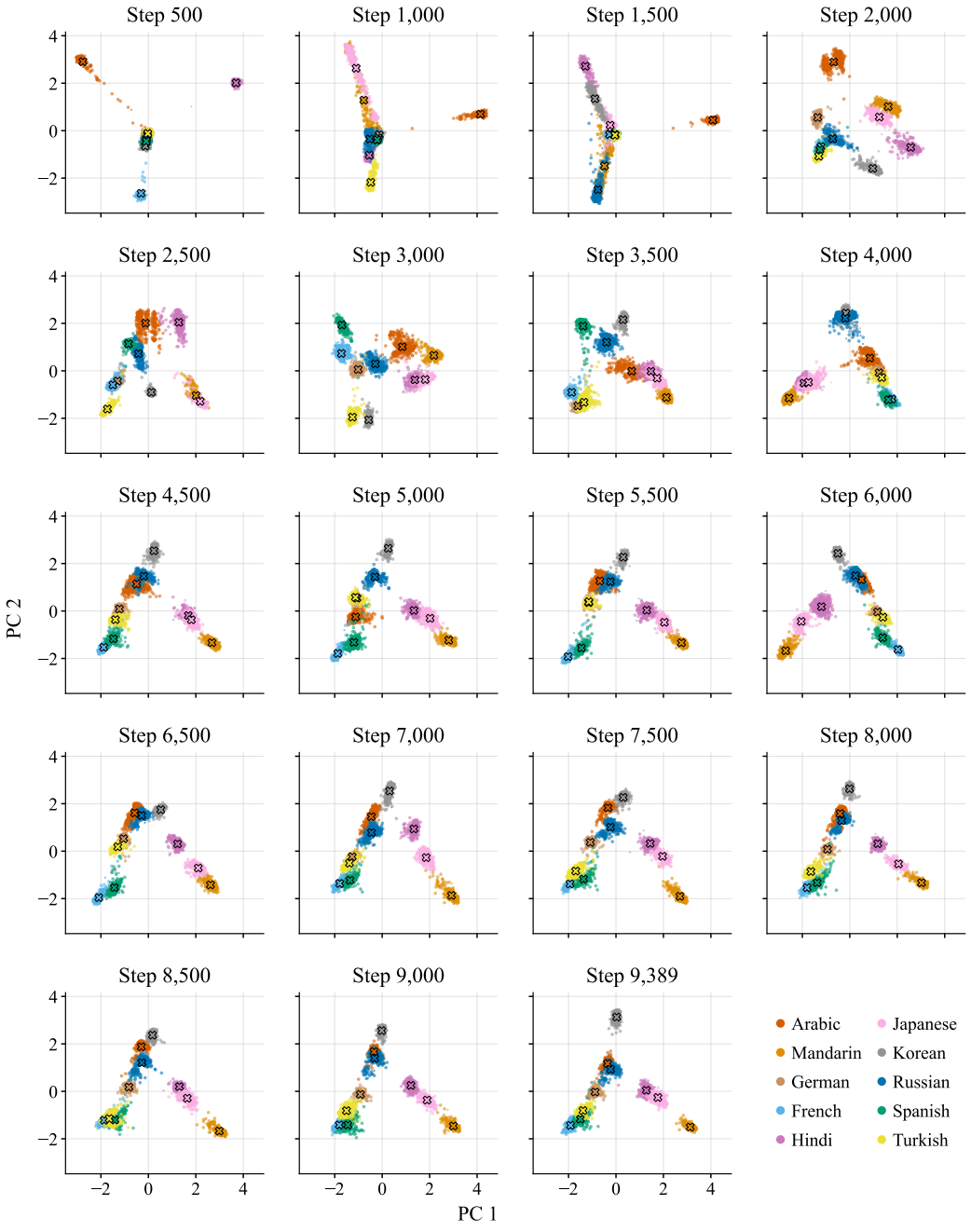

Figure 10: Evolution of multilingual routing during training.

C.2 SUDOKU

Strategy	Solving Step Format
Repeated row-major naked-single sweeps.	`r5c5 = { 5 }`
Hidden singles, scanning rows, columns, then boxes.	`row3 5 -> r3c7`
Digit-major crosshatching over boxes.	`3 box4 -> r6c3`
First-in-first-out propagation of forced cells, recording newly forced peers.	`set r5c5 = 5 : r5c2 -> { 2 }`
Deterministic DFS backtracking: first empty cell, digits ascending.	`r1c3 = 1 x`

Table 2: Sudoku data strategies and data formatting.

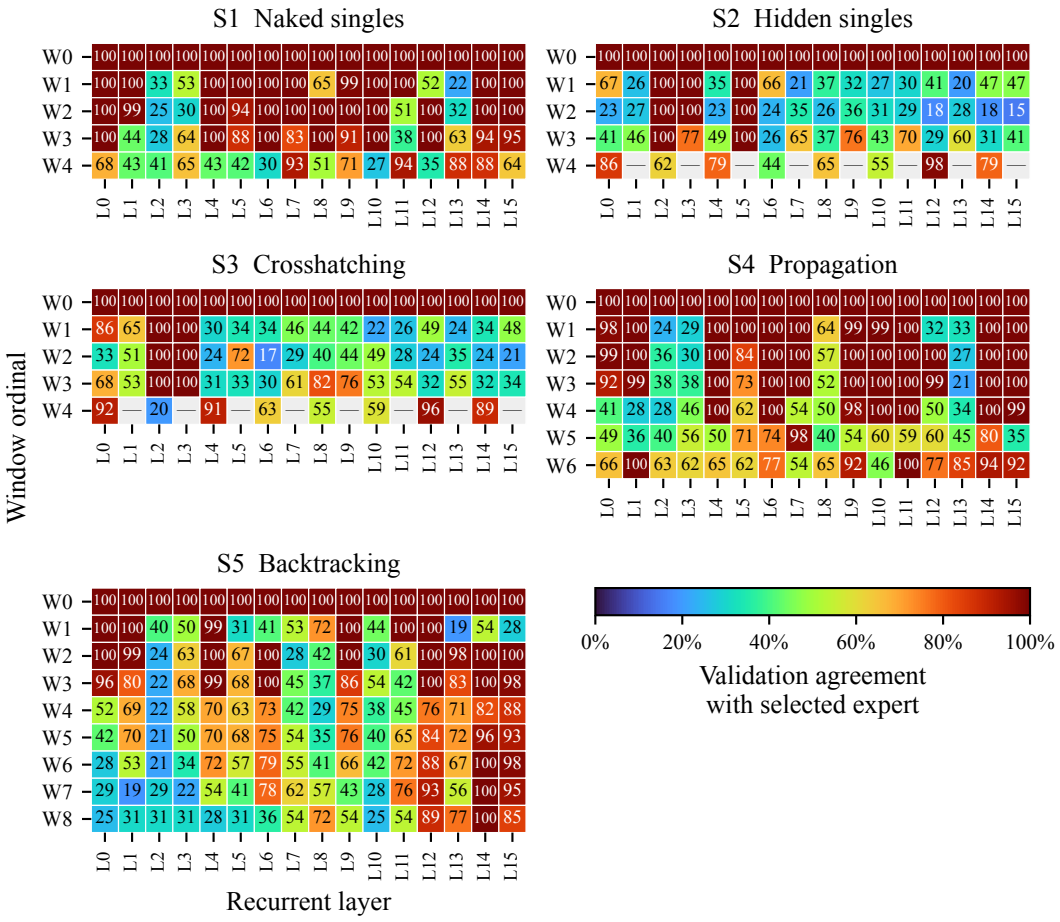

Figure 11: Routing agreement on sudoku. Each panel shows one strategy, with window positions as rows and recurrent depths k as columns. Each cell gives the percentage of observed routing decisions that agree with the mode-path expert at that position. Gray cells indicate positions with no observations.

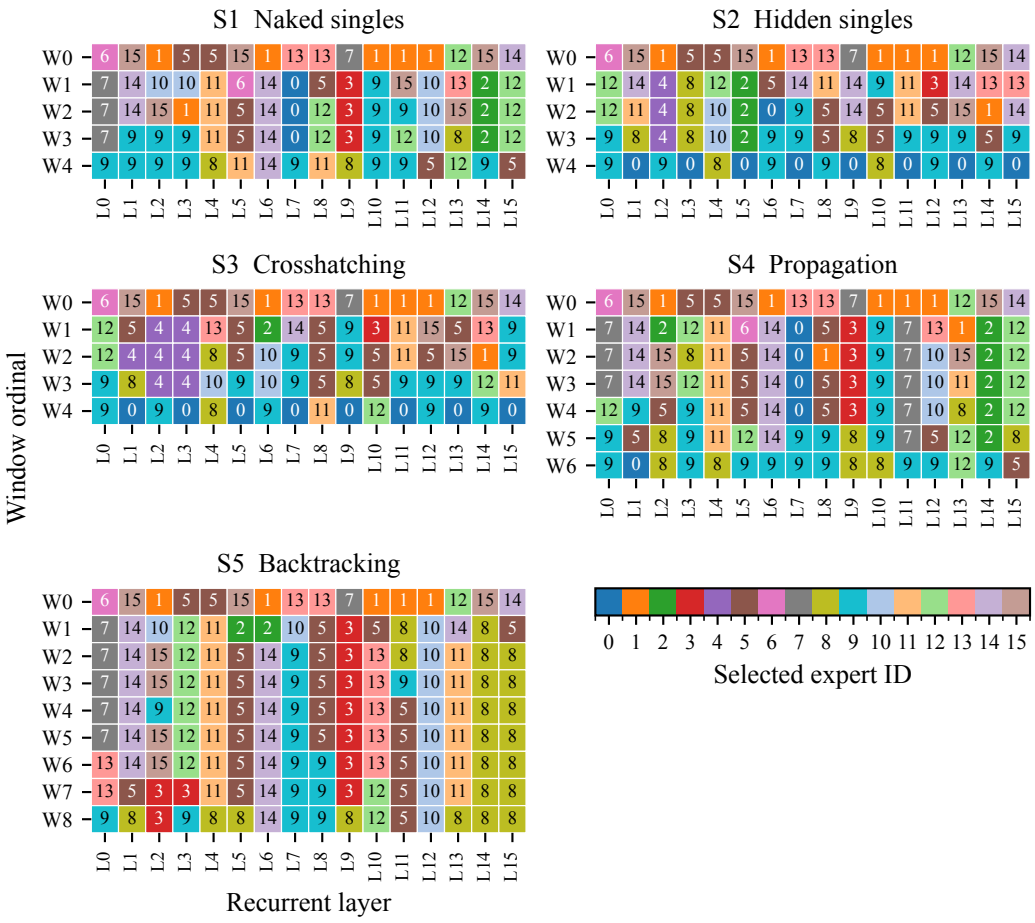

Figure 12: Mode routing paths on sudoku. Each panel shows one strategy, with window positions as rows and recurrent depths k as columns. Cell labels and colors identify the expert assigned to each (strategy, window position, recurrent depth) tuple.

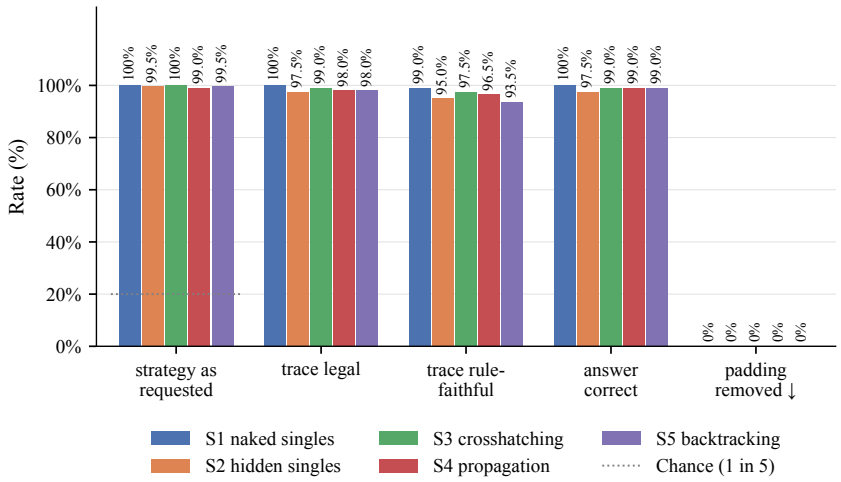

Figure 13: Strategy adherence and sudoku solution accuracy under forced routing.

In Figure 13, we report different evaluation metrics:

- **Strategy as requested:** the percentage of outputs whose solving steps all use the requested strategy's syntax.
- **Trace legal:** the percentage of outputs whose placements are all legal and lead to the true solution. Traces that do not match any strategy count as failures.
- **Trace rule-faithful:** the percentage of outputs that pass the legality check and correctly apply every claimed deduction.
- **Answer correct:** the percentage of outputs whose final 81 digit grid, after the answer marker, matches the true solution.
- **Padding removed ↓:** the fraction of tokens before the end-of-sequence token that are padding, averaged across outputs.

We see that LoL, with close to 100% accuracy, uses the expected strategy with legal traces and succeeds in solving the sudoku grid without adding any unneeded padding.

C.3 MAZE

Strategy	Example Solving Steps
Breadth-first search in discovery order.	`r1c1 r1c2 r2c1 r1c3 r2c2 r3c1`
Depth-first traversal with backtracking.	`r1c1 r1c2 r1c3 r1c4 r1c3 r1c2`
A* search using Manhattan distance.	`r1c1 r1c2 r2c2 r2c3 r3c3 r3c4`
Alternating BFS from the start and goal.	`r1c1 r8c8 r1c2 r8c7 r1c3 r8c6`
Repeated row-major dead-end filling.	`r8c1 r8c2 r8c3 r1c8 r2c8 r3c8`

Table 3: Maze data strategies and data formatting.

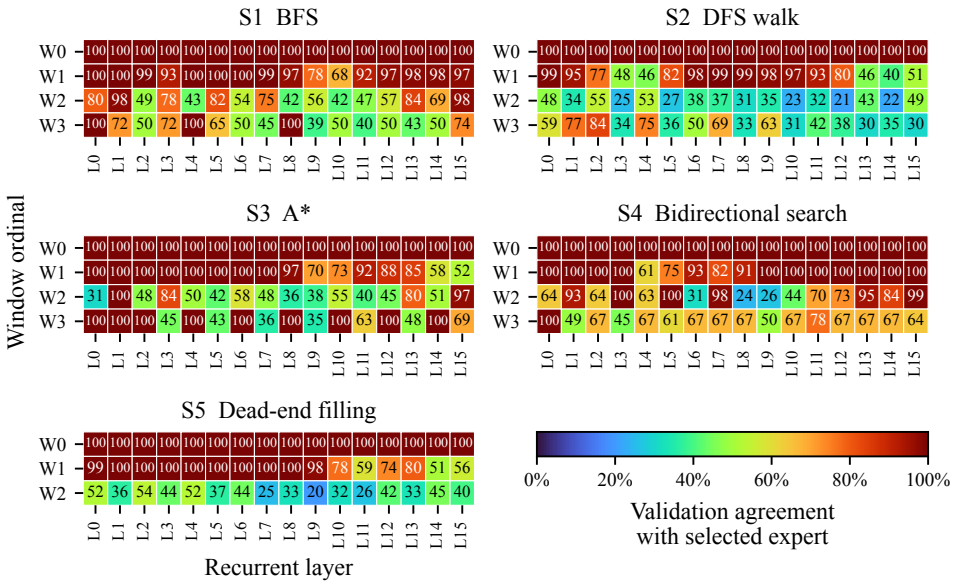

Figure 14: Routing agreement on mazes. Each panel shows one strategy, with window positions as rows and recurrent depths k as columns. Each cell gives the percentage of observed routing decisions that agree with the mode-path expert at that position. The gray row marks window W3 for dead-end filling; no validation observations are available for that window-strategy combination.

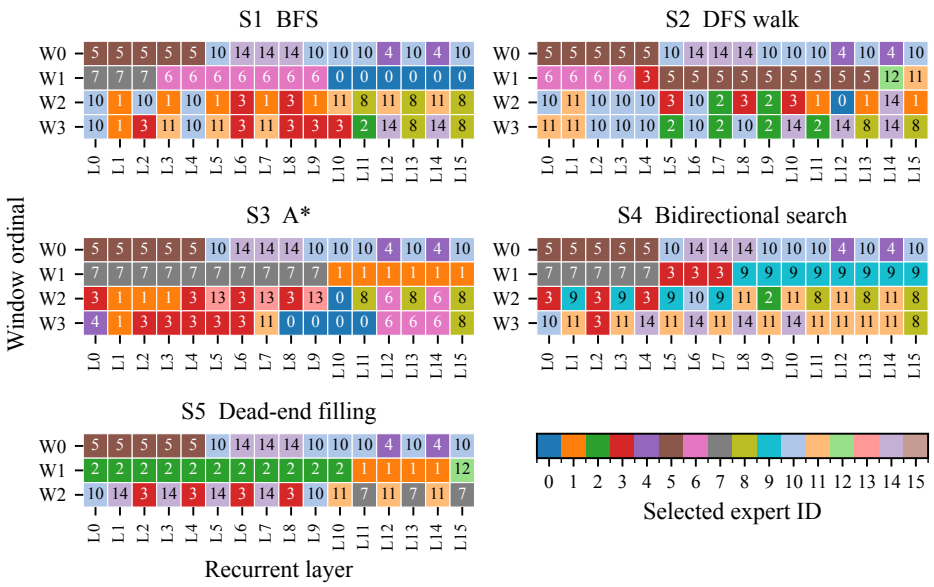

Figure 15: Mode routing paths on mazes. Each panel shows one strategy, with window positions as rows and recurrent depths k as columns. Cell labels and colors identify the most frequently selected expert for each (strategy, window position, recurrent depth) tuple during validation. For dead-end filling, W3 has no validation observations and reuses the W2 expert sequence if reached.

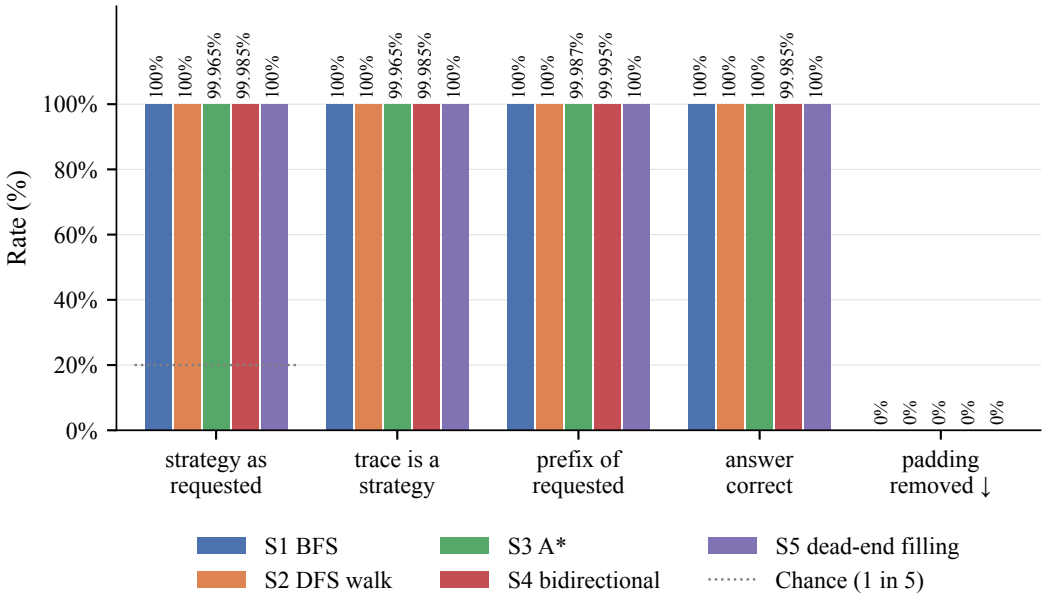

Figure 16: Strategy and maze solution generation under forced routing.

The evaluations reported in the maze experiment are the same as those in the sudoku experiment.

C.4 Looped Transformers

In Figure 17, we visualize the expert utilization of the models shown in Figure 4.

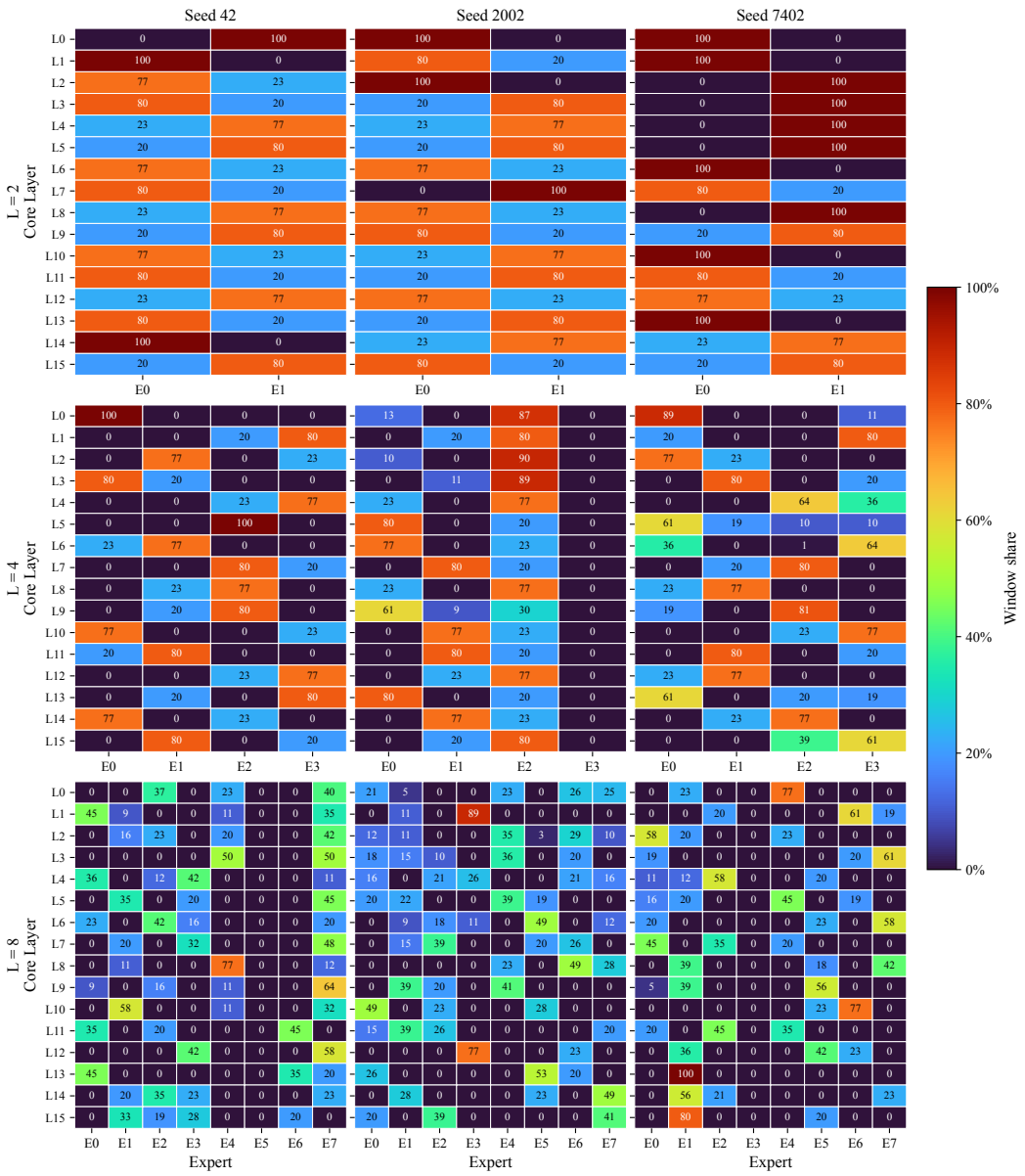

Figure 17: Heatmap of expert utilization of LoL models shown in Figure 4.

C.5 MoE

In Figure 18 we visualize the expert utilization of the models shown in Figure 5.

C.6 Math

In Figure 19 and Table 4, we extend Figure 3 and Table 1 with "Vanilla (strided window)" and "LoL (untied router)."

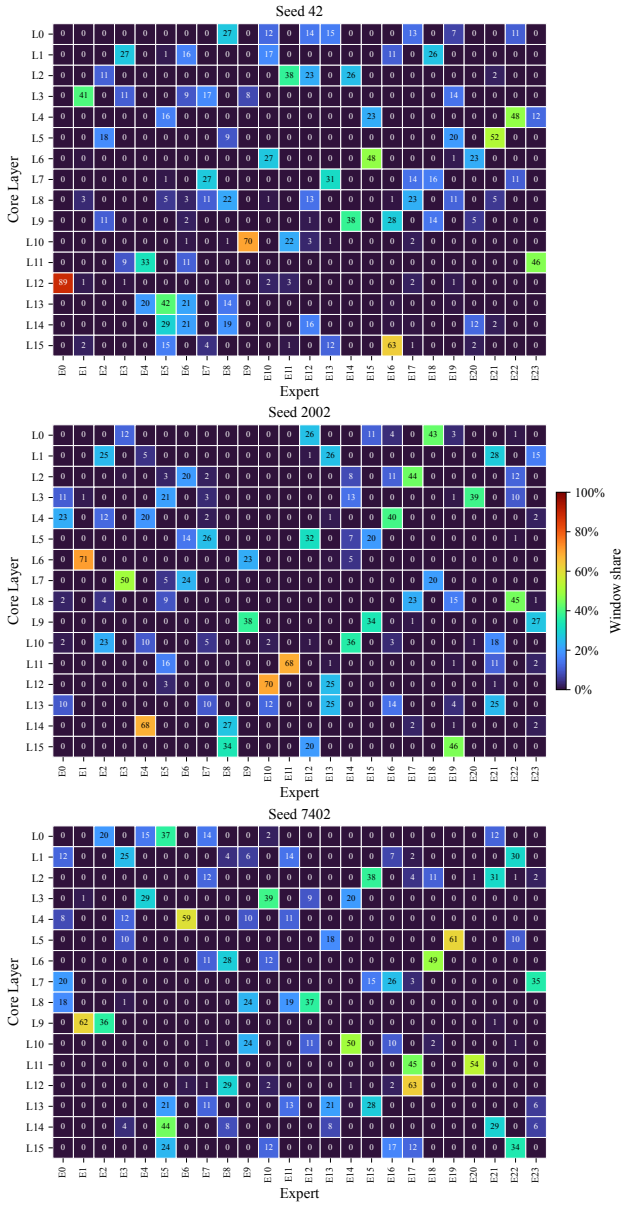

Figure 18: Heatmap of expert utilization of LoL models shown in Figure 5.

Model	ARC-E	ARC-C	HellaSwag	PIQA	GSM8K
Random	25.0	25.0	25.0	50.0	0.0
Vanilla	40.8 ± 1.0	28.2 ± 1.3	31.4 ± 0.5	56.6 ± 1.2	29.7 ± 1.3
Vanilla (sliding window)	49.5 ± 1.0	28.5 ± 1.3	31.3 ± 0.5	58.8 ± 1.1	27.5 ± 1.2
LoL	50.3 ± 1.0	29.1 ± 1.3	31.2 ± 0.5	59.7 ± 1.1	27.9 ± 1.2
Vanilla (strided window)	48.5 ± 1.0	28.0 ± 1.3	30.8 ± 0.5	59.0 ± 1.1	25.9 ± 1.2
LoL (untied router)	50.1 ± 1.0	30.0 ± 1.3	31.4 ± 0.5	57.1 ± 1.2	27.7 ± 1.2

Table 4: Extended version of Table 1 with "Vanilla (strided window)" and "LoL (untied router)."

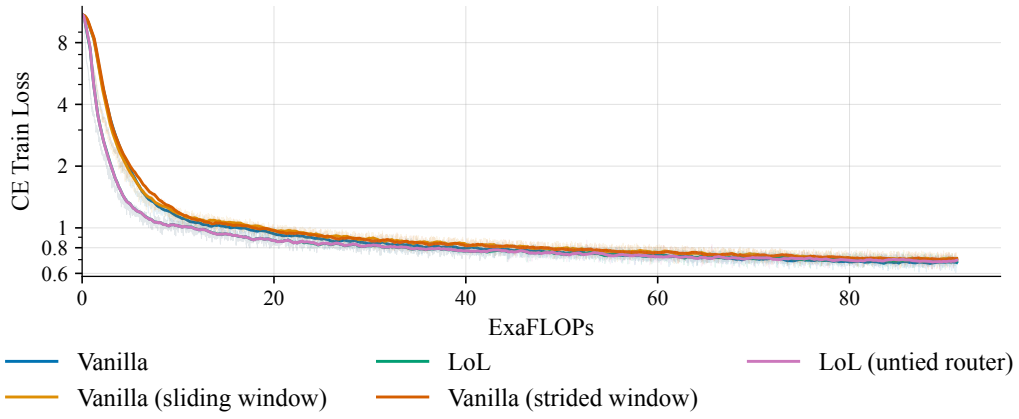

Figure 19: Extended version of Figure 3 with "Vanilla (strided window)" and "LoL (untied router)."