MIXTURE-OF-POWERS: LEARNING TO FAST-FORWARD LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

A looped transformer repeatedly applies the same learned update to increase its computational depth. We introduce Mixture-of-Powers (MoP), which trains separate experts to advance by two, four, eight, or sixteen base updates per call. A supplied schedule combines these strides with unit steps, passing the hidden state from one expert to the next. We supervise predictions at each requested depth and after further base calls, keeping the base model, encoders, and readout fixed. On Sudoku with a prescribed sequence of forced placements, one stride-16 call correctly predicts the requested board on 98.24% of 512 test windows, compared with 99.22% for sixteen base calls. It executes 32 rather than 64 transformer blocks and reduces single-example end-to-end time from 54.49 to 27.25 ms. Across 25 call sequences excluded from training, mean endpoint exactness is 98.34%. For four successive stride-16 calls, 97.85% of windows have a correct endpoint and 81.05% match every scheduled intermediate board. The bank approximately halves inference time on this task, with additional parameter and training costs.

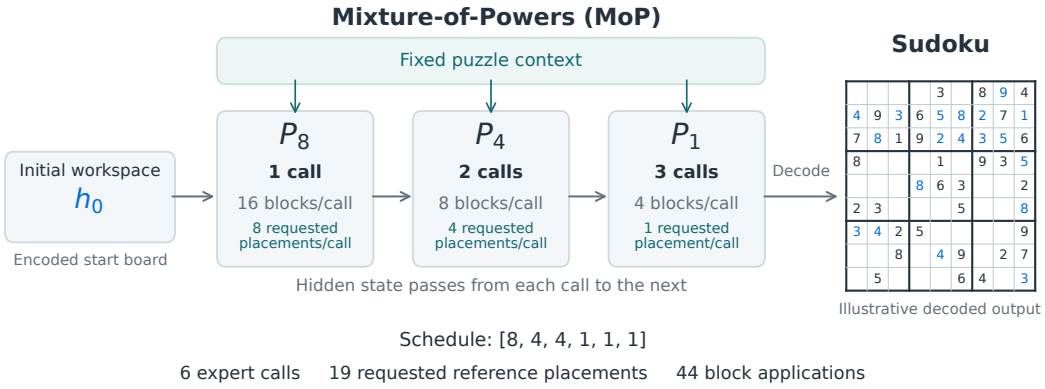

Figure 1: Combining larger strides and unit steps in one schedule. The starting board is encoded into a hidden workspace. Each expert receives the preceding call's hidden output and the same original puzzle. The schedule uses P_8 once, P_4 twice, and P_1 three times, targeting 19 reference placements in six calls. It executes 44 transformer blocks; 19 base calls execute 76. The final hidden state is decoded into a board. Blue digits mark added placements in this schematic board.

1 INTRODUCTION

A transformer performs computation through successive updates to an internal state (Vaswani et al., 2017). Looped transformers reuse the same transformation across depth, increasing computation without storing new weights for each update (Dehghani et al., 2019; Giannou et al., 2023; Geiping et al., 2025). In the conventional unit-step formulation, reaching depth D requires D sequential calls. We ask whether a model can advance several of these updates in one call and then continue with steps of different sizes. We call the number of base updates represented by a call its *stride*.

For example, a stride-16 expert should predict the output expected after sixteen base updates. A unit call applied to its hidden output should then predict the output after seventeen updates. We supervise each expert at its requested depth and after further recurrent calls. Each call receives the hidden state produced by the preceding call.

Mixture-of-Powers (MoP) trains experts for strides 1, 2, 4, 8, and 16. All experts read and write the same hidden workspace, while the base model, encoders, and readout remain fixed during power training. An externally supplied *schedule* selects the sequence of strides. Figure 1 shows $[8, 4, 4, 1, 1, 1]$: six calls advance through 19 reference updates, with unit steps used for the final three. Larger experts P_4 through P_{16} start from two copies of the trained half-stride expert. We then train them on intermediate targets and subsequent execution. The evaluated system uses supplied schedules throughout; it has no online stride selection or error detection.

LoopFormer studies variable strides in a shared, step-conditioned model (Jeddi et al., 2026), and trajectory distillation compresses steps in generative models (Salimans & Ho, 2022; Frans et al., 2025). MoP uses a bank of separately trained strides with a fixed recurrent interface. Our hypothesis is that these experts can reduce the computation needed to reach requested intermediate states while allowing later calls to continue accurately.

We test this hypothesis using Sudoku singles. A deterministic solver records the board after each forced digit placement, providing an exact target for every stride. Several placements may be available at once; the model must follow the recorded order. We compare predictions with the board expected after each call, including intermediate boards that a final-solution score would leave unchecked. The task tests compression of this prescribed computation.

The paper makes three contributions:

- We add separately parameterized stride experts to a frozen recurrent model, with all experts operating on the same hidden state.
- We train larger powers from copies of smaller ones, supervising intermediate boards and predictions after further calls through the frozen base.
- We evaluate accuracy and runtime for individual jumps, composed and unseen schedules, and compare the bank with shallower unit models and a shared stride core.

One stride-16 call reaches 98.24% endpoint exactness against 99.22% for sixteen base calls and approximately halves end-to-end time. P_2 learns the initial reduction from four to two blocks per reference placement, and larger experts retain that block budget. A stride-16 call executes 32 sequential blocks, giving sixteen times fewer calls and roughly twice the speed. Intermediate errors become more frequent on long paths, even when the final board is correct.

2 Related Work

Looped and recursive models. Universal Transformers share a transformation across depth and support adaptive computation (Dehghani et al., 2019). Giannou et al. (2023) construct looped transformers that execute programs, and Geiping et al. (2025) study additional latent recurrence for language-model reasoning. The Hierarchical Reasoning Model (HRM) uses interacting modules at different update frequencies (Wang et al., 2025), while the Tiny Recursive Model (TRM) repeatedly updates a hidden state and an answer representation (Jolicoeur-Martineau, 2025). We build on this use of recurrence by training experts to replace several base updates and predict the same requested intermediate outputs.

Computation depth and stride. Adaptive Computation Time learns how long to continue a recurrent computation (Graves, 2016); Mixture-of-Recursions assigns recurrence depths to individual tokens (Bae et al., 2025). Looped flows combine recurrent states with local denoising objectives and probability-flow inference, varying computation through time discretization (Suleymanzade et al., 2026). LoopFormer conditions a shared looped transformer on time and step size and trains shortcut-consistent trajectories (Jeddi et al., 2026). It is the closest architectural precedent for learned strides. MoP uses separate stride experts with a frozen base, encoder, and readout. We train on exact symbolic targets and evaluate predictions after subsequent calls. Our shared-core control tests one

implementation of parameter sharing; it does not reproduce LoopFormer, so it cannot establish a performance advantage over that method.

Trajectory shortcuts. Progressive diffusion distillation replaces successive teacher steps with fewer student steps (Salimans & Ho, 2022). Consistency Trajectory Models learn transitions between times on probability-flow trajectories (Kim et al., 2024), and shortcut models condition predictions on the desired step size (Frans et al., 2025). We apply a similar approach to a recurrent model trained on an explicit reference computation. The recorded states let us check the output at each requested depth. We also apply later calls to the expert's hidden output and check their predictions. All evaluations use supplied schedules; we do not train a model to select strides.

3 METHODS

We first train a base model to follow a reference trajectory. We then freeze it and train experts to advance several reference steps per call. At inference, a supplied schedule chooses which expert to apply next.

3.1 REFERENCE TRANSITIONS, POWERS, AND SCHEDULES

The reference solver produces a sequence of boards, placing one digit at each step. Let T denote this deterministic transition and $z_{j+1} = T(z_j)$ its trajectory. The learned base model $F = P_1$ instead updates a hidden workspace, initialized by the encoder as $h_0 = E(z_i, c)$. Here c is the fixed puzzle context. The readout R_q converts the hidden state into a board prediction using query q. Unlike the reference solver, the learned base can make errors.

A power is trained to behave like several base calls, as measured by decoded boards. For the evaluated inputs and a specified readout query, we seek agreement immediately after the jump and after further base calls:

$$\begin{aligned} R_q(P_s(h, c)) &\approx R_q(F^s(h, c)), \\ R_q(F^r(P_s(h, c), c)) &\approx R_q(F^{s+r}(h, c)), \quad r \in \{1, 2, 4\}. \end{aligned} \tag{1}$$

In the second line, we apply another one, two, or four base calls to the power's hidden output and compare the resulting board predictions. We call this *continuation*. Training also targets the exact reference board $T^s(z_i)$. When the base makes an error, its prediction differs from that reference target. We evaluate the decoded boards on sampled states, without requiring identical hidden tensors or assuming agreement on other inputs.

A *window* is a segment of a reference trajectory, identified by its starting board and number of placements. A *schedule* $\pi = (s_1, \ldots, s_C)$ lists the strides used to traverse that segment. Executing the schedule produces a *path*. Each call ends at a *landing*, whose hidden state is decoded and compared with the corresponding reference board:

$$h_k = P_{s_k}(h_{k-1}, c), \qquad a_k = \sum_{j=1}^{k} s_j, \qquad \text{target } z_{i+a_k}. \tag{2}$$

For example, a schedule $[8, 4, 1]$ has three landings, targeting the boards after 8, 12, and 13 reference placements from its start. The reference advance, number of expert calls, and number of executed transformer blocks are

$$D = \sum_{k=1}^{C} s_k, \qquad C = |\pi|, \qquad B(\pi) = \sum_{k=1}^{C} b_{s_k}, \tag{3}$$

where b_s counts blocks inside one stride-s call. These are distinct measures of computation.

Schedules are supplied externally. For a whole puzzle, the number of blank cells gives the required reference length without revealing the solution. A *pure* schedule repeats one stride and requires the length to be divisible by that stride. A *mixed* schedule combines strides and can use unit calls to cover a remainder. No learned router or stopping rule is evaluated.

3.2 ARCHITECTURE AND STACKED POWERS

The workspace has 97 slots, each a 256-dimensional vector: 96 state positions and one scratch position. A 26-token vocabulary represents digits and control markers; row, column, and box embeddings represent Sudoku structure. Each transformer block uses workspace self-attention, cross-attention to the fixed puzzle context, and a feedforward network. The one-layer context encoder supplies keys and values that are cached for each expert and reused during the path.

A shared one-layer readout predicts board cells in parallel. Training uses a state query for the current board and an answer query for the completed board. These queries use different embeddings and the same readout parameters. Predicted boards are never re-encoded between calls: the next expert receives the preceding expert's hidden output. There is no clock embedding. The base learns the one-placement target, but its predictions can change more than one cell or none.

Both P_1 and P_2 contain four blocks. P_2 starts from the trained P_1 weights. Each larger power starts from two copies of its trained half power, placed in sequence with separate parameters and their original output normalizations:

$$P_{2s}^{\text{init}}(h, c) = P_s(P_s(h, c), c), \qquad s \in \{2, 4, 8\}. \tag{4}$$

Thus $(b_1, b_2, b_4, b_8, b_{16}) = (4, 4, 8, 16, 32)$. For example, P_4 begins as two trained four-block P_2 copies, then trains those eight blocks as one expert. All appended powers use two blocks per reference placement. A schedule containing only these powers executes $2D$ blocks, compared with $4D$ for the base; with n_1 unit calls, $B(\pi) = 2D + 2n_1$. Larger strides reduce call count without further halving this block budget. Appendix B gives the representation and parameter details.

3.3 PROGRESSIVE TRAINING

First, we jointly train the base, encoders, and readout to predict the reference board after each unit call. Training begins with short paths and gradually permits longer ones. Starts that require a hidden-single rule receive extra sampling weight. Every call receives the model's preceding hidden output; reference boards supply loss targets only. We then freeze the selected base, encoders, and readout.

Next, we train powers in increasing stride order, updating only the new expert. Half of the batches start from a directly encoded board. The other half first run the frozen base to obtain a starting hidden state representative of ongoing computation. If these preliminary p calls start at z_i, later reference targets are indexed from z_{i+p}. The first half of each stage repeats the new power. The second half divides batches equally between repeated calls and mixed schedules containing the new power and earlier experts. These paths train the new expert to work with the hidden states produced by preceding calls.

We supervise the board predicted at each landing, selected boards within the expert, and predictions after further base calls:

$$\mathcal{L} = \mathcal{L}_{\text{state}} + 0.1\mathcal{L}_{\text{KL}} + 0.25\mathcal{L}_{\text{deep}} + 0.25\mathcal{L}_{\text{cont}} + 0.25\mathcal{L}_{\text{answer}}. \tag{5}$$

The state term targets the reference board, using focal loss to emphasize poorly predicted tokens. The KL term encourages agreement with the frozen base's output probabilities after the same number of reference steps. Deep supervision predicts intermediate boards from internal blocks of the new expert. Continuation supervision checks boards after further frozen-base calls. The answer term targets the completed board when the path reaches the solution.

Continuation is attempted on 25% of batches using one, two, or four further base calls. That sampling probability is separate from its loss weight of 0.25. We backpropagate the continuation loss through these calls to update the new expert, while keeping the base weights fixed. Hidden tensors are not directly matched by a distance loss. Appendix C gives the training details and lists those missing from the available records.

4 EXPERIMENTAL SETUP

We test whether the powers predict the requested boards, remain accurate when composed, and reduce runtime. We also compare the bank with shallower unit models and a single shared stride core:

- **Q1: Intermediate-state accuracy.** Does one power call predict the board after the requested number of reference placements?
- **Q2: Reuse across calls.** Can later experts use that output, including in call sequences excluded from training?
- **Q3: Runtime.** How much do executed blocks and elapsed time decrease, and what accuracy is lost?
- **Q4: Shallower base models.** Can a unit model with fewer blocks per call match the bank at the same executed-block budget?
- **Q5: Shared stride core.** How do accuracy, runtime, and parameter count change when all four strides share one core?

Task and data. We select puzzles from three sources within Sudoku-Extreme.[1] A *naked single* is a blank cell with only one legal digit. A *hidden single* is a digit that has only one legal cell within a row, column, or box. The solver checks for naked singles from left to right and top to bottom, then for hidden singles in a fixed order of rows, columns, boxes, and digits. It rejects puzzles that require guessing and records every intermediate board. On average, training states have 8.94 forced placements available at once. The task is to follow the recorded order, even when another move would also be valid.

There are 100,000 training, 2,000 validation, and 5,000 test puzzles. A further 1,000 form the *depth holdout*, a test of longer puzzles: 999 require 64 placements and one requires 60, compared with at most 59 in training. These puzzles also differ in clue count and source composition, so this test does not isolate the effect of length. Every 64-placement window starts at the initial board and ends at the solution. Duplicate filtering compares exact puzzle strings; separation under Sudoku symmetries or shared solutions was not audited (Appendix A).

Models and training budgets. All runs use seed 0, bf16 training, AdamW, and an RTX PRO 6000 Blackwell GPU. The base and the two shallower unit models each train for 180,000 updates, with 256 windows per batch, on the same 20,000 puzzles. MoP uses four further disjoint 20,000-puzzle pools: 180,000 updates for P_2 and 60,000 each for P_4, P_8, P_{16}.

The shared stride core reuses the frozen base, encoders, and readout and trains on the same four power pools for 180,000 updates. It receives the requested stride as an input and repeats its four-block core $s/2$ times for stride s, matching MoP's executed blocks. The powers start from progressively trained smaller experts; the shared core starts from the base.

Adding the individually listed components gives 5.94 M parameters for the base system, 69.16 M for MoP, and 10.76 M for the base plus shared core. These sums do not account for every parameter in the recorded totals (Appendix B). The reported base-plus-bank training takes 48.39 hours and 540,000 updates; the shared core adds 24.40 hours to the common base. These totals exclude preliminary attempts, and one optimizer update can involve different amounts of computation. The models in Q4 and Q5 differ in capacity, data exposure, and training history. We report their performance as trained, without attributing the differences to architecture alone.

What counts as correct. *Endpoint exactness* is the percentage of examples whose final predicted board matches all 81 reference cells. Here "final" means the end of the requested window, which may be partway through a puzzle. *All scheduled landings exact* requires a correct board after every expert call. One call has one check; a longer call sequence has more opportunities to fail. Different valid partial boards count as mismatches if they differ from the recorded order.

Table 1 separates the evaluations and their sample sizes. We report the trajectory scores separately from the completed-answer scores. The logs do not identify the readout query at every landing.

[1] https://huggingface.co/datasets/sapientinc/sudoku-extreme

When a path reaches the correct endpoint after earlier mismatches, we therefore cannot verify that it recovered under the same query (Appendix D).

Table 1: Which examples each Sudoku evaluation uses. A window is a reference-trajectory segment; a path adds a sequence of expert calls. A start can be the initial board or an intermediate board unless specified. The fixed-window and unseen-schedule tests draw separate examples, so the same schedule can receive different scores. Whole-puzzle results are subsets of the predefined paths.

Evaluation	Data and starting state	Examples evaluated	Results
Stage validation	Validation; encoded board or state after four base calls	256 windows of each start type per jump	Tables 9, 10
Fixed-window tests	Test for 16/20 placements; depth holdout for 64, from the initial board	512 windows per schedule; 489/487/405 distinct puzzles for 16/20/64 placements	Tables 2, 11
Unseen schedules	Test for 16/20/35 placements; depth holdout for 64	25 complete call sequences, each tested on 512 windows	Table 13
Predefined paths	Test and depth holdout; whole puzzles and shorter windows	Test: 15,157 paths, 3,388 puzzles. Holdout: 7,684 paths, 513 puzzles	Tables 14, 15

Comparisons and uncertainty. Each power path is paired with a base rollout on the same example. Reported 95% confidence intervals use 1,000 bootstrap resamples, keeping all windows from one puzzle together. These intervals estimate uncertainty from the sampled evaluation puzzles for one trained model. They do not measure variation across training runs, and each applies only to its stated comparison. Comparisons between MoP and the other trained systems have no direct confidence intervals. Their evaluation settings match, but the logs lack example identifiers needed to verify identical windows across systems. Appendix D documents the sampling limits.

Runtime measurement. We report expert calls, executed transformer blocks, and elapsed time separately. Timing uses the median of five synchronized GPU repetitions at batch sizes one and 256. Recurrent time covers expert calls. End-to-end time also includes encoding the board and context, computing the experts' context projections, and decoding the final board. MoP and the shared core were timed on an otherwise idle GPU. Shallower-model timings were collected during concurrent training and are excluded from speedup claims. The reported gains apply to this implementation, hardware, and workload.

5 Results

5.1 Q1: Predicting the requested intermediate board

We encode a starting board and predict the board 16 reference placements later. One P_{16} call is correct on 98.24% of 512 fixed test windows, compared with 99.22% for sixteen base calls (Table 2). The difference is -0.98 percentage points (95% CI $[-2.33, 0.38]$), corresponding to nine failed windows for the power and four for the base. Of the targets, 506 are intermediate boards and six are completed puzzles. Across the four powers, stage-validation jump accuracy ranges from 98.83% to 99.61% (Table 9). The confidence interval for P_{16} is too wide to establish a loss of at most one percentage point.

5.2 Q2: Continuing computation after a power call

We next apply successive experts to one another's hidden outputs. At 16 placements, two P_8 calls reach 98.83% endpoint exactness, four P_4 calls reach 98.24%, and eight P_2 calls reach 97.66%. Each sequence executes 32 blocks. Two P_8 calls have the highest point estimate at this block budget. Applying P_4 after P_{16} predicts the board 20 placements from the start on 98.63% of windows, compared with 98.83% for twenty base calls (difference -0.20 points, CI $[-1.18, 0.78]$). In validation, one, two, or four further frozen-base calls after a power produce mean endpoint scores of 98.96% to 99.54% across powers (Table 10).

Table 2: Board accuracy for several schedules reaching the same depth. Each Power bank (MoP) schedule uses half as many transformer blocks as the four-block base sequence and is evaluated on 512 Sudoku windows. P_s requests s reference placements; arrows give call order. Endpoint exact checks all 81 cells at the final landing; all scheduled landings exact requires a match after every call. Schedules check different intermediate positions. Base endpoint uses the paired four-block P_1 sequence. Block counts exclude encoding and readout. The 16/20-placement rows use the test split; the 64-placement rows solve depth-holdout puzzles from the initial board. For P_{16}, the endpoint difference is -0.98 percentage points (95% CI $[-2.33, 0.38]$); all intervals appear in Table 11.

Call schedule	Reference placements	Expert calls	Block applications	Endpoint exact (%)	All scheduled landings exact (%)	Base endpoint exact (%)
1 call of P_{16}	16	1	32	98.24	98.24	99.22
2 calls of P_8	16	2	32	98.83	98.44	99.22
4 calls of P_4	16	4	32	98.24	97.46	99.22
8 calls of P_2	16	8	32	97.66	93.95	99.22
16 calls of P_1	16	16	64	99.22	98.83	99.22
$P_{16} \to P_4$	20	2	40	98.63	97.85	98.83
20 calls of P_1	20	20	80	98.83	98.83	98.83
4 calls of P_{16}	64	4	128	97.85	81.05	99.41
64 calls of P_1	64	64	256	99.41	84.77	99.41

We also evaluate 25 complete call sequences excluded from training. Their mean endpoint exactness is 98.34%, with individual scores from 95.70% to 99.61%. Each sequence is evaluated on 512 windows and contributes equally to the mean. Twenty-two sequences cover 16, 20, or 35 placements; three cover 64 and use the longer puzzles from the depth holdout. The individual strides and some shorter subsequences occur in training; the complete call sequences do not (Table 13).

Longer sequences often predict the correct endpoint after mismatching earlier targets. Four P_{16} calls reach the depth-64 board on 97.85% of windows, compared with 99.41% for the base (difference -1.56 points, CI $[-3.33, 0.00]$). Only 81.05% match all four landings; the base matches all 64 of its landings on 84.77%. These scores check different sets of intermediate boards. The unseen sequence $[4, 4, 8, 16, 16, 16]$ reaches 98.44% endpoint exactness, with 70.70% matching every landing. Scoring only the endpoint would miss these intermediate errors. Some mismatches may be valid placements made in a different order, and the missing readout-query records prevent us from determining whether the model recovered under the same query (Section 4).

5.3 Q3: Runtime and executed blocks

For one example, P_{16} reduces recurrent time from 52.28 to 23.59 ms, a $2.22\times$ speedup, and end-to-end time from 54.49 to 27.25 ms, approximately $2.00\times$ (Table 3). For a batch of 256, total batch time falls from 60.38 to 31.10 ms, a $1.94\times$ speedup. At 64 placements, four P_{16} calls reduce single-example end-to-end time from 210.34 to 97.84 ms, or $2.15\times$.

The other 16-placement power sequences each execute 32 blocks and take 26.20 to 26.41 ms of recurrent time. Their runtimes are similar despite their different call counts. Compared with the base, these schedules use half the blocks and take about half the recurrent time. A single P_{16} call is somewhat faster than the other power sequences; these timings do not identify the source of that smaller difference.

5.4 Q4: Comparison with shallower base models

At 16 placements, the two-block unit model and P_{16} each execute 32 blocks, reaching 90.82% and 98.24% endpoint exactness (Figure 2). At 64 placements and 128 blocks, their scores are 60.55% and 97.85%. The three-block model reaches 97.66% and 91.80% while executing more blocks. Neither tested shallower model matches the bank's accuracy at these block budgets. MoP also stores more parameters and receives additional training on four further puzzle pools. We have not tested whether the shallower models could match its accuracy with more training or a different training procedure.

Table 3: Runtime for Power bank (MoP) and its four-block base on an idle RTX PRO 6000. The power schedules execute half as many core blocks and take approximately half the end-to-end time. P_s requests s reference placements. Recurrent time covers expert calls; end-to-end time adds input encoding and final board decoding. Block counts include only expert-core blocks. Entries are median milliseconds over five synchronized repetitions for the entire batch. Speedup divides the four-block base's end-to-end time by the listed time at batch size one.

Call schedule	Expert calls	Block applications	Batch size 1 Recurrent (ms)	Batch size 1 End-to-end (ms)	Batch size 256 End-to-end (ms)	End-to-end speedup (batch size 1)
16 calls of P_1	16	64	52.28	54.49	60.38	1.00×
8 calls of P_2	8	32	26.31	28.44	31.50	1.92×
4 calls of P_4	4	32	26.20	28.68	31.85	1.90×
2 calls of P_8	2	32	26.41	29.52	32.78	1.85×
1 call of P_{16}	1	32	23.59	27.25	31.10	2.00×
64 calls of P_1	64	256	208.02	210.34	234.16	1.00×
4 calls of P_{16}	4	128	93.92	97.84	112.18	2.15×

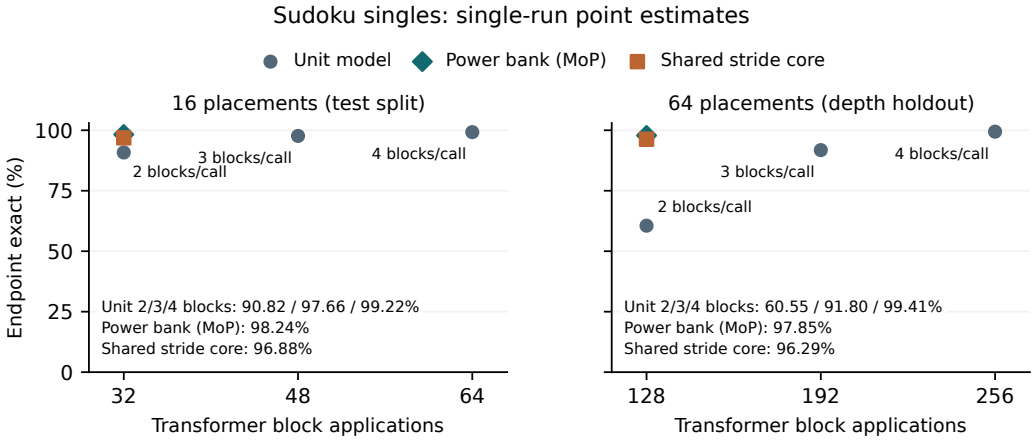

Figure 2: Board accuracy at different executed-block budgets. Power bank (MoP) is more accurate than the two-block unit model at the same block count. Each point uses 512 Sudoku windows; endpoint exactness requires all 81 cells to match the reference board. Power bank (MoP) and Shared stride core use one stride-16 call for 16 placements and four for 64. Unit models use one call per placement, with their blocks per call labeled. All points come from single training runs; no confidence intervals between systems are available. Both panels use the same accuracy scale. Evaluation settings match, but the logs do not identify every window, so we cannot verify identical examples across systems. Capacity, initialization, and training exposure differ.

5.5 Q5: Comparison with a shared stride core

At the same executed-block budget, the shared core reaches 96.88% on a stride-16 call and 96.29% on four such calls, compared with MoP's 98.24% and 97.85% (Figure 2). Across the 25 unseen schedules, its mean is 96.63%, against 98.34% for MoP. Single-example end-to-end time is 31.89 ms for the stride-16 jump, a 1.74× speedup against its separately timed base. It adds 4.81 M parameters, compared with 63.21 M for the bank.

On 15,157 predefined test paths, the shared core reaches 99.59% endpoint exactness and MoP reaches 99.51%. MoP matches every scheduled landing on 96.45% of paths, compared with 92.38% for the shared core. This evaluation pools whole puzzles, shorter windows, and base-only schedules, with each path weighted equally (Appendix D). A puzzle can contribute several paths, so these percentages do not give the fraction of distinct puzzles solved.

On depth-holdout paths, endpoint scores fall to 93.68% for MoP and 92.39% for the shared core, against 95.71% for the paired base (Table 14). MoP's deficit is 2.03 points (95% CI $[-3.27, -0.76]$). Repeated P_2 over whole holdout puzzles reaches 91.03% endpoint exactness, with only 37.62% matching every landing (Table 15).

The shared core uses far fewer parameters and has a slightly higher pooled test endpoint score. MoP has higher point estimates on the fixed-window and unseen-schedule tests. It also receives twice as many power-training updates and starts from trained smaller experts. We cannot separate the effect of these training differences from the choice to use distinct experts.

6 Limitations

We evaluate one training seed on Sudoku puzzles with a fixed placement order and exact intermediate labels. The depth holdout changes length, clue count, and source together, so it cannot isolate the effect of longer execution. The continuation tests cover only selected states and call sequences. Errors on longer paths also leave open how reliably experts can replace base calls over longer computations.

The bank requires additional parameters and training to reduce inference cost. Every appended expert uses two blocks per reference placement, so larger strides do not further reduce this block budget. The controls differ in capacity, data exposure, and initialization, and we do not ablate individual loss terms. We have not evaluated learned stride selection, stopping, or downstream uses of intermediate states.

Some readout-query, sampling, and implementation details are missing from the records (Appendix D). These omissions prevent full reproduction and leave it unclear whether a correct endpoint after earlier mismatches reflects recovery under the same readout query. Experiments on other tasks and with matched training budgets are needed to assess how broadly the measured savings apply.

7 Conclusion

Mixture-of-Powers trains experts to advance a looped transformer's computation by different numbers of base updates. A supplied schedule combines the experts through a shared hidden state. On prescribed Sudoku-singles trajectories, one stride-16 call uses half as many transformer blocks and approximately half the end-to-end time of sixteen base calls, with a 0.98-percentage-point decrease in endpoint exactness. Later calls remain accurate on the tested schedules, although intermediate errors become more frequent on long paths. The bank requires more stored parameters and training than the base. Further experiments should test whether these savings hold with matched training budgets, longer trajectories, and tasks without exact intermediate supervision.

AI use statement

Generative AI tools assisted with formulation, experimental design, literature review, interpreting results, drafting and editing, checking result exports, table and plotting code, and LaTeX formatting. The authors are responsible for the final content.

Reproducibility statement

Appendices A through D document the data, model, training, and evaluation; Appendix E gives expanded results. The manuscript package includes LaTeX source, references, figures, derived tables, and display-generation utilities, and rebuilds the accompanying PDF. Experiment code, training and evaluation routines, corpus files, and checkpoints are not included. Appendix D identifies implementation details unresolved by the available reports and configurations. The package rebuilds the numerical displays; reproducing the experiment requires the original training and evaluation implementation.

REFERENCES

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. arXiv:2507.10524, 2025. URL https://arxiv.org/abs/2507.10524.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OlzB6LnXcS.

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. arXiv:2502.05171, 2025. URL https://arxiv.org/abs/2502.05171.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Alex Graves. Adaptive computation time for recurrent neural networks. arXiv:1603.08983, 2016. URL https://arxiv.org/abs/1603.08983.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2602.11451.

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

Dongjun Kim, Chieh-Hsin Lai, Wei-Hsiang Liao, Naoki Murata, Yuhta Takida, Toshimitsu Uesaka, Yutong He, Yuki Mitsufuji, and Stefano Ermon. Consistency trajectory models: Learning probability flow ODE trajectory of diffusion. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.02279.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2202.00512.

Ayhan Suleymanzade, Chanhyuk Lee, Floor Eijkelboom, Nicholas M. Boffi, İsmail İlkan Ceylan, and Jinwoo Kim. Thinking with looped flows. arXiv:2609.11801, 2026. URL https://arxiv.org/abs/2609.11801.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. 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. arXiv:2506.21734, 2025. URL https://arxiv.org/abs/2506.21734.

A CORPUS CONSTRUCTION AND SYMBOLIC TRANSITIONS

Sources and filtering. The dataset uses the Kaggle, unbiased, and 17-clue sources within Sudoku-Extreme. We generate each reference sequence with the prescribed singles solver, without using the supplied solution. We retain puzzles that the solver completes without guessing. Before constructing a trajectory, we assign the puzzle to a split according to its source partition, blank count, and the remaining quota. The source test partition supplies our test split; its training partition supplies training, validation, and depth holdout.

The documented duplicate filter compares exact puzzle strings, including removal of source-training puzzles found in the source test partition. The generation report records zero dropped duplicates. The report does not include separate checks for every pair of final splits, Sudoku symmetries, or shared solutions.

The generator attempts 263,307 puzzles and accepts 108,000. Across all splits, Kaggle contributes 23,805 accepted puzzles from 23,805 attempts; the unbiased source contributes 83,196 from 237,233; and the 17-clue source contributes 999 from 2,269. Attempt counts are 243,998 for training, 4,902 for validation, 12,137 for test, and 2,270 for depth holdout. Counts for every source within every split and an exact upstream dataset revision are unavailable. The accompanying records preserve hashes of the input files. We apply no data augmentation. Holdout puzzles differ from training puzzles in both length and source mixture.

The one-placement reference rule. For each blank cell, list the digits that do not already occur in its row, column, or box. If any cell has only one candidate, fill the first such cell in left-to-right, top-to-bottom order. Otherwise, check each row, then each column, then each box. Within each group, use unit-index order and then digit order. Fill the first digit that has exactly one legal position in that unit. A unit is a row, column, or 3×3 box. After each placement, recompute candidates and restart the rule checks. Reject a puzzle if no rule applies before completion.

Each recorded transition stores the changed cell, its digit, and the rule used. The current board determines the next placement. The trajectory ends when the board is complete; solved boards are never repeated to create longer examples.

Table 4: Size of the Sudoku dataset. A reference board is the initial puzzle or the board after one more forced digit placement. The number of placements equals the puzzle's initial blank count. Training includes three 44-placement puzzles. The depth holdout tests longer puzzles: 999 have 64 placements and one has 60.

Split	Puzzles	Reference boards	Placements per puzzle: range	Placements per puzzle: mean
Training	100,000	5,468,896	44 to 59	53.689
Validation	2,000	109,460	45 to 59	53.730
Test	5,000	273,254	45 to 59	53.651
Depth holdout	1,000	64,996	60 to 64	63.996

The training split contains 5,368,896 placements: 4,421,460 naked singles, 750,814 hidden singles in rows, 178,998 in columns, and 17,624 in boxes. The mean number of simultaneously available forced moves is 8.94 in training, 9.01 in test, and 5.16 in the holdout. We score intermediate boards against the prescribed placement order. A solver that made several forced placements at once could finish in fewer steps.

Training pools and stored evaluation paths. We sort original puzzle identifiers by their SHA-256 hashes and divide them into five groups of 20,000. The base and each successive power use a different group, so new powers do not train on the base's own puzzles. We use this allocation for all runs and do not test whether disjoint pools improve learning. The shallower unit models use the base's group. Each stride of the shared core uses the group assigned to the corresponding power.

The dataset also stores evaluation paths: whole-puzzle executions and eligible shorter windows of 16, 32, or 64 placements, with pure or mixed schedules. We report the paths evaluated by each model.

MoP and the shared core use the same path definitions. Unit-only models support fewer schedules, so we compare them on matching schedule types rather than on pooled scores from different path sets.

B ARCHITECTURE AND PARAMETERIZATION

Board encoding and workspace. The vocabulary contains 26 tokens: ten digits, eight control markers (padding, beginning, end, separator, question, context, state query, and answer query), and eight reserved controls. The embedding table allocates 32 identifiers, leaving six outside this vocabulary. A board is represented by 81 digits in row order, with zero for a blank. The context contains 85 tokens including markers and represents the original puzzle. It remains fixed when execution starts at an intermediate board or after preliminary base calls. Future reference boards are used only as training and evaluation targets.

The hidden workspace contains 96 state positions and one extra scratch position, each with 256 values. The scratch position is available for internal computation and is not associated with a cell. State tokens occupy positions after it and receive token, position, and Sudoku-structure embeddings. Eighty-one positions correspond to cells; the rest accommodate markers and unused positions. The records do not specify the exact marker order, initialization of unused positions, or attention masks.

Transformer blocks and context. The context encoder has one transformer layer. A four-block expert uses four attention heads and a feedforward width four times the workspace width. Each block applies normalization before self-attention, cross-attention, and the feedforward network. Self-attention combines workspace positions; cross-attention reads the fixed puzzle context. Learned biases in each attention head represent whether two cells share a row, column, or box. These biases start at zero. Each four-block core has an output layer normalization and no dropout.

Context keys and values are computed once for each expert used in a path and reused on later calls to that expert. The model receives no explicit clock embedding. The accumulated requested stride is tracked for scheduling and target selection, separately from the hidden workspace.

Predicting a board. The readout combines a state-query or answer-query embedding with cell-structure embeddings at its output positions. One transformer layer predicts all board positions together, and the highest-probability allowed token is selected at each position. Allowed outputs are digits and the end marker. The predicted tokens are used to score the board, never as the next expert's input. The next expert receives the hidden workspace. Because all cells are predicted in parallel, a unit call can appear to change several cells or none; one placement is its training target.

The result records distinguish trajectory scores from separate completed-answer checks. The trajectory tables use the former. Whole-puzzle validation completion is reported separately. Appendix D states which output-query details are known and which remain unverified.

The listed components sum to 5,942,352 parameters for the base system, 69,157,152 for the bank, and 10,755,968 for the base plus shared core. The three- and two-block unit systems sum to 4,888,900 and 3,835,448. The reported system totals are larger: 72,509,633 for the bank, 14,108,449 for the shared system, 8,241,381 for the three-block system, and 7,187,929 for the two-block system. Each total contains an additional 3,352,481 parameters not assigned to a component in the available report. We report the listed sums as component subtotals. We cannot verify the total number of parameters used during execution from this report.

The full bank stores every expert, while each schedule executes the experts it names. A deployment restricted to one stride could omit the unused experts. We do not measure such a deployment and report parameter storage, executed blocks, and elapsed time separately.

Building larger powers. Each larger expert starts as two copies of the trained half-stride expert. The copies have separate parameters and preserve their block order and output normalizations. Before further training, the new expert performs the half-stride computation twice. We update only the new copies. Stacking adds stored parameters and keeps the cost at two blocks per reference placement, so each larger-stride call executes more blocks.

Table 5: Parameters in the individually reported model components. P_s is trained to advance s reference placements per call. Expert block counts are per call; the shared stride core repeats four blocks $s/2$ times. The appended-power total counts P_2 through P_{16} and excludes the base, encoders, and readout. The recorded system totals include further parameters that are not assigned to a component; see the text below. n/a indicates that no block count applies.

Component	Transformer blocks	Parameters
State encoder	n/a	65,536
Context encoder	1	823,040
Readout	1	839,456
Unit model (4 blocks/call), P_1	4	4,214,320
Power expert P_2 (stride 2)	4	4,214,320
Power expert P_4 (stride 4)	8	8,428,640
Power expert P_8 (stride 8)	16	16,857,280
Power expert P_{16} (stride 16)	32	33,714,560
Appended power experts, total	n/a	63,214,800
Shared stride core	$4(s/2)$	4,813,616
Unit model (3 blocks/call)	3	3,160,868
Unit model (2 blocks/call)	2	2,107,416

Shared stride core. The comparison model has one four-block core for all four strides. A 64-dimensional embedding of the requested stride controls scale, shift, and residual gates at three normalization sites per block. For stride s, the same core is applied $s/2$ times. It starts from the base weights, while the original base remains a separate frozen model. A general conditioning site can be expressed as

$$u_s = \alpha_s \odot \mathrm{LN}(h) + \beta_s, \qquad h' = h + \gamma_s \odot H(u_s), \tag{6}$$

where H is an attention or feedforward branch. Preserving an unconditioned copied branch requires $\alpha_s = \gamma_s = 1$ and $\beta_s = 0$. The records describe a zero-initialized conditioning projection but do not specify whether offsets of one are added to its scale and gate outputs. We cannot confirm that the initialized shared core exactly matches the base without these offsets. The original implementation is also needed to confirm which normalizations and context projections are shared.

C Training details and checks

C.1 Checks before training

The four-block recurrent model is first checked on 32 puzzles. Changing future target labels leaves the hidden computation unchanged, while altering the workspace changes the output. Execution counters confirm the requested calls and accumulated strides. Gradients pass through frozen-base continuation to the new power; the base, encoders, and readout remain unchanged. Converting 192 checked reference boards into their encoded representation and back recovers every board without a parsing failure.

C.2 Base training and curriculum

AdamW uses $(\beta_1, \beta_2) = (0.9, 0.95)$, weight decay 0.01, and gradient clipping at 1.0. Training uses bf16 autocast and batches of 256 windows. Gradients pass through the entire call sequence. Activation checkpointing reduces memory use by recomputing intermediate expert and readout activations during backpropagation.

The learning rate rises over 500 updates to 3×10^{-4}, remains at that peak, then decays with a cosine schedule over the last 45,000 of 180,000 updates. The maximum permitted path length doubles every 22,500 updates and is limited to 59 by the available training trajectories. Within this limit, three length ranges have sampling probabilities 0.25 for one placement, 0.25 for 2 through 8, and 0.50 for 9 through 64; probabilities are renormalized when a range is unavailable. Half the batch starts where the next reference move is a hidden single, divided equally among row, column, and box rules. The other half is drawn equally from initial and intermediate boards.

At landing j, the state loss uses focal-weighted negative log-likelihood of the target tokens:

$$\ell_{\text{state}}(h_j, z_{i+j}) = -\frac{1}{N}\sum_{u=1}^{N}(1-p_{j,u})^2 \log p_{j,u}, \tag{7}$$

where $p_{j,u}$ is the probability of the correct token at scored position u. The focal factor gives more weight to poorly predicted tokens. Losses are averaged over scored positions and landings. Before any expert call, the encoded state is also trained to reproduce the starting board. A completed-board answer loss, weighted by 0.25, is added when the path reaches the solution.

Validation occurs every 2,000 updates, using 256 windows at each available depth and 256 complete puzzles. The validation split has no 64-placement trajectory. The selected checkpoints are at updates 180,000 for the four-block base, 168,000 for the three-block model, and 162,000 for the two-block model; all three runs train for 180,000 updates. These checkpoints have the best recorded validation score, although the exact formula combining validation metrics is unavailable. The selected main base, encoders, and readout are then frozen.

C.3 Starting states, call sequences, and losses

Half the power-training batches begin directly from an encoded board. The other half first applies 1 to 16 frozen-base calls. The powers then train on hidden states produced by preceding base calls. From a reference start z_i, p preliminary calls produce $h_0 = \text{stopgrad}(F^p(E(z_i, c), c))$; $p = 0$ denotes direct encoding. The choice is made per batch. Let $j_0 = i + p$ and $a_k = \sum_{j=1}^{k} s_j$. The power path and its base teacher start from the same hidden state:

$$h_k = P_{s_k}(h_{k-1}, c), \qquad \bar{h}_{a_k} = F^{a_k}(h_0, c), \qquad z_{j_0+a_k} = T^{a_k}(z_{j_0}). \tag{8}$$

The preliminary base calls may already have made an error. We still train the power against both the base prediction and the recorded solver board, scoring agreement and correctness separately.

Pure schedules repeat the new power at least twice, with a power-of-two call count. The first half of each stage uses only pure schedules. The second half divides batches equally between pure and mixed schedules. Mixed schedules must contain the new power. They are formed from large-stride decompositions of the requested length, subdivisions into smaller strides, and different stride orders. A subdivision replaces one stride by two half-sized strides and has recorded probability 0.5. Unit calls account for at most one quarter of the requested placements. The 25 complete evaluation sequences are excluded from training. The records do not fix the relative probabilities of the three mixed-schedule constructions, whether subdivision is repeated recursively, or how rejected samples are replaced.

The state loss first averages across scored tokens, then across landings within a path, then across windows in the batch:

$$\ell_k = -\frac{1}{|U_k|}\sum_{u\in U_k}(1-p_{k,u})^2 \log p_{k,u}, \qquad \mathcal{L}_{\text{state}} = \frac{1}{N_{\text{win}}}\sum_{w=1}^{N_{\text{win}}}\frac{1}{C_w}\sum_{k=1}^{C_w}\ell_{w,k}. \tag{9}$$

Here U_k is the set of scored positions. The loss also checks reconstruction of the starting board. The available description does not specify the inclusion of the end marker in each loss, the reconstruction term's weight, the KL temperature, or every auxiliary averaging rule. KL is computed from the teacher distribution to the student distribution. The separate completed-answer loss has weight 0.25 and applies only at a terminal endpoint.

Deep supervision asks internal blocks of the newly trained expert to predict intermediate reference boards. A site representing offset d within call k targets $z_{j_0+a_{k-1}+d}$, accounting for all earlier calls and the preliminary base steps. The methods description places a target after every two blocks in stacked powers; some configuration comments describe an earlier mapping. The exact selected blocks and output normalization require implementation confirmation.

On one quarter of batches, one, two, or four additional frozen-base calls provide continuation supervision with loss weight 0.25. Backpropagation through those calls updates the new expert. All sampled steps must fit within a recorded trajectory; completed boards are not repeated to extend a path. The precise sampling probabilities within length ranges and pure call counts, and the handling of a continuation that does not fit, are unavailable.

One training update. The documented procedure is:

1. Choose a puzzle, a starting board, optional preliminary base calls, and a valid call sequence. Determine the reference board needed after each call.
2. Encode the starting board once and apply the preliminary base calls, if selected. Start the power path and frozen-base teacher from that same hidden state.
3. Execute each expert on the preceding hidden output. Compare its decoded board with the reference and its output probabilities with the teacher. Add intermediate-board losses inside the new expert.
4. If selected and valid, continue through the frozen base and score the resulting board. Add the answer loss when the path reaches the completed puzzle.
5. Backpropagate through the call sequence and update only the new power. Earlier experts, encoders, and readout retain their weights.

Reproducing these updates requires the sampling and loss details listed above.

Table 6: Learning-rate schedules and maximum training-path lengths. Warm-up gradually raises the learning rate; the cosine tail reduces it during the final listed updates. The span cap is the maximum number of reference placements in a sampled path. It doubles at the listed interval, subject to available trajectory length. Durations count optimizer updates; the shared core uses separate stride-selection counts for its span caps. The base schedule also applies to the shallower unit models.

Training stage	Optimizer updates	Peak learning rate	Warm-up (updates)	Cosine tail (updates)	Span-cap doubling (updates)
P_1	180,000	$3 \cdot 10^{-4}$	500	45,000	22,500
P_2	180,000	$3 \cdot 10^{-4}$	300	60,000	30,000
P_4	60,000	10^{-4}	2,000	20,000	13,333
P_8	60,000	10^{-4}	2,000	20,000	20,000
P_{16}	60,000	$2 \cdot 10^{-5}$	2,000	20,000	40,000
Shared stride core	180,000	10^{-4}	2,000	60,000	Stride-specific

For stride s, the maximum training-path length begins at $2s$ and doubles toward 64 before learning-rate decay. The shared core selects each of its four strides once in a randomly ordered group of four updates. Each stride is selected in exactly 45,000 updates. This *anchor stride* determines the update's data pool and path-length curriculum. Mixed paths can also include other strides.

Each stride's maximum path length follows its own 45,000-selection schedule, with a 15,000-selection tail. Doubling intervals for strides 2, 4, 8, and 16 are 7,500, 10,000, 15,000, and 30,000 selections. The shared core's global learning rate follows Table 6. Several stride conditions may contribute gradients in one mixed-path update, so 45,000 anchor selections does not mean only 45,000 uses of that stride. All shared-core calls remain differentiable, including through later frozen-base continuations. Exact gradients for the conditioning embeddings and reductions of internal losses are not separately logged. The components designated as frozen remain unchanged in every completed run.

C.4 Training cost and validation thresholds

Table 7 reports the measured training costs. The base and bank stages total 48.39 hours and 540,000 updates, excluding preliminary checks and unsuccessful development attempts. The shared model reuses the base checkpoint and adds 24.40 hours. The two shallower controls train concurrently, so their wall times do not measure standalone throughput. The logs record sampled windows, supervised landings, target tokens, teacher calls, calls with and without gradients, and elapsed time.

The reported power results correspond to 180,000 updates for stride 2 and 60,000 for each larger stride. The shared-core results use 180,000 total updates. Development changes included replacing a four-block stride-4 expert with stacked powers and reducing the stride-16 learning rate after a divergent run. Those attempts are excluded from the reported training totals. The records do not establish when every design choice was made relative to test access. We use the term fixed-window testing because we cannot verify that the test remained sealed throughout development.

Table 7: Training cost for each reported stage. P_s targets s reference placements per call. Expert parameters count the recurrent component only; peak GPU memory is the allocation recorded for that process. The shared stride core reuses the frozen base, encoders, and readout. The two shallower models trained concurrently, so their wall times are not measurements of training speed on an otherwise idle GPU.

Training stage	Expert parameters (millions)	Optimizer updates	Wall time (hours)	Peak GPU memory (GB)
Power bank (MoP)				
P_1	4.21	180,000	10.38	3.44
P_2	4.21	180,000	14.12	8.80
P_4	8.43	60,000	5.37	8.82
P_8	16.86	60,000	7.58	10.87
P_{16}	33.71	60,000	10.94	16.56
Shared stride core				
All four strides	4.81	180,000	24.40	18.30
Unit model (3 blocks/call)				
P_1	3.16	180,000	12.22	3.04
Unit model (2 blocks/call)				
P_1	2.11	180,000	10.29	2.64

Stage validation uses 256 directly encoded windows and 256 starting after four base calls. Repeated-power tests use two and four calls when enough reference steps remain; continuation tests use one, two, and four additional base calls. Four stride-16 calls do not fit any validation trajectory. We compare endpoint deficits against a one-percentage-point reference threshold. Stride 2 exceeds it by 0.17 points and remains in the bank; no experts were removed by this threshold. A point estimate within one percentage point does not demonstrate statistical equivalence.

D EVALUATION PROTOCOL

Fixed-window tests. Each listed schedule uses 512 windows, starting from a directly encoded board, and a base rollout on the same window. The 16- and 20-placement tests cover 489 and 487 distinct puzzles, with six and seven targets that are completed boards. The 64-placement test uses 405 distinct puzzles among the 999 eligible length-64 puzzles. Its 512 windows all start at the initial board and end at the solution, so some puzzles appear more than once.

The summaries do not preserve every selected puzzle identifier and starting offset or fully specify sampling with replacement. The systems have matching evaluation settings, seeds, counts, and base scores. Without all window identifiers, we cannot verify that they were tested on identical examples.

Unseen call sequences. This test contains 22 complete sequences covering 16, 20, or 35 placements and three covering 64. None occurs among the 18,196 recorded bank-training sequences or 11,127 shared-core training sequences. Individual strides and shorter subsequences may have been seen. The 64-placement sequences also exceed every training trajectory and use a different mixture of puzzle sources.

The procedure and date for selecting these 25 sequences were not preserved. We give each sequence equal weight in the reported mean. We have not estimated how often these schedules would occur in use. Its windows are drawn separately from the fixed-window tests, so the same call sequence can have different scores in the two tables.

Predefined paths and evaluated subsets. The dataset stores 115,754 test paths and 24,999 depth-holdout paths. Each path specifies a puzzle, a starting board, a reference length, and a call schedule. The evaluation configuration limits each path group to 512 examples, subject to schedule and length eligibility. For example, the recorded test set for repeated stride 8 contains 288 length-48 paths and 512 length-56 paths, totaling 800. These counts are consistent with separate group caps. The exact ordering or random selection within each capped group is not retained.

MoP and the shared core evaluate 15,157 test paths from 3,388 puzzles and 7,684 holdout paths from 513 puzzles. These include pure and mixed whole-puzzle paths and shorter pure-stride windows. Unit-only models evaluate 4,412 and 1,537 paths because they support fewer schedules. Every path contributes once to the pooled mean; a puzzle evaluated under more schedules contributes more times. Because the aggregate includes base-only paths, it cannot measure puzzle-solving accuracy with powers alone.

Pure schedules require the puzzle length to be divisible by their stride. Repeated stride 16, for example, can use only length-48 puzzles among test lengths 45 through 59. Rows for different strides therefore contain different puzzle subsets. The mixed-schedule holdout group contains only one path.

Output queries and reported scores. Training distinguishes a state query for the current board and an answer query for the completed board. The reports store trajectory accuracy and separate completed-answer checks. We use the trajectory scores in the path tables. The reports do not record which query was used at every evaluated landing (Table 8). Without these query records, we cannot determine whether a correct endpoint after earlier mismatches reflects recovery under the same query. A path could also reach the reference board after making valid placements in a different order, without ever correcting an invalid digit.

Table 8: What is known about the output query behind each reported score. State and answer queries use different embeddings with the same readout network. The paper uses the recorded trajectory scores separately from completed-answer scores. Without the missing query records, we cannot confirm that intermediate and final predictions used the same query.

Reported measurement	Available information
Jump, continuation, fixed-window, and unseen-schedule accuracy	Scores compare predicted and reference boards. The query used at each landing is not recorded.
Predefined-path endpoints and all landings	Trajectory scores are separate from completed-answer scores. Whether the query changes at completion is unverified.
Whole-puzzle validation	A separate completion score is recorded; its query is unspecified.
Completed-answer checks	These explicitly use the answer readout and are excluded from the reported trajectory scores.
Final board in timing tests	Decoding is included in elapsed time, but the query used is unspecified.

Confidence intervals. A window or path is scored correct only if its predicted board matches all reference cells. Paired differences compare model and base on the same examples. The reported 95% confidence intervals use 1,000 bootstrap resamples with seed 0, keeping every window from a given puzzle together. This accounts for dependence between examples from the same puzzle. Each interval estimates uncertainty from the evaluation sample for its stated comparison. The models use one training seed, so the intervals cannot measure variation across training runs.

The summaries do not specify the exact bootstrap interval formula or all weighting rules within a resample. They also provide no direct intervals for bank-versus-shared or bank-versus-shallower comparisons. We report point estimates for those comparisons.

Checking board decoding. The main readout predicts board positions in parallel. A separate check selects each position's highest-probability token, converts the result to board text, and parses that text back into a board. It checks for format or length failures and agreement with the direct comparison against target labels. The labels are used to score predictions, not to generate them.

There are no score mismatches, parsing failures, or maximum-length failures in 1,280 checked windows each for MoP and the shared core, and 1,152 each for the shallower controls, using 128 windows per checked schedule. We sampled these windows separately from the main evaluations. The decoding checks do not identify which query was used at each landing in the main evaluations.

Timing procedure. The recorded timings are median milliseconds over five GPU-synchronized repetitions, for batches of one and 256. Encoding includes both input encoders and each expert's

context projections. Recurrent time covers the scheduled expert calls; decoding covers the final board prediction. Intermediate board checks are excluded from this timing breakdown. End-to-end time is measured separately, so it need not equal the sum of the median phase times. MoP and the shared core use an idle GPU and separately measured base timings. The shallower models were timed during concurrent training and are excluded from speedup comparisons.

The reports do not give the number of warm-up runs, inference precision and mode, model-runtime framework and CUDA versions, attention backend, compilation settings, or treatment of data transfer and model loading. Context projections are charged to encoding and reused within a path, but reuse across timing repetitions is unspecified. Only medians are retained, so variation across repetitions cannot be reported. We report speedups for the measured setup. The timings do not isolate the sources of small differences between systems.

Details still needed for full reproduction. The records do not specify all workspace masks, stride-conditioning offsets, loss averaging rules, sampling fallbacks, or internal-supervision mappings. Full reproduction requires these details from the original implementation or additional metadata. The supplied package rebuilds the paper and its numerical displays; it does not include the complete experiment implementation.

E EXPANDED RESULTS

E.1 INDIVIDUAL JUMPS, REPEATED CALLS, AND CONTINUATION

We validate each expert with a single call, repeated calls to the same expert, and further base calls applied to its hidden output (Table 9). Table 10 separates the one-, two-, and four-call continuation results. Stride 2 and all shared-core strides exceed the descriptive one-percentage-point deficit threshold. They remain included in every applicable evaluation.

Table 9: Validation of one-call jumps and subsequent execution on Sudoku. Endpoint exactness requires all 81 reference cells to match. Jump scores combine 256 encoded starts and 256 starts after four base calls. Base endpoint uses the paired four-block unit model. Repeated-call scores require every landing to match and average two-call and four-call tests, with 256 windows each; stride 16 has only a two-call test. Continuation averages endpoint scores after one, two, or four further base calls. Differences are model minus base in percentage points (pp); confidence intervals (CI) resample whole puzzles. These tests use different total depths across strides.

Stride call	Endpoint exact (%)	Base endpoint exact (%)	Difference vs. base (pp)	95% CI (pp)	Repeated calls All scheduled landings exact (%)	Continuation Endpoint exact (%)
Power bank (MoP)						
P_2	98.83	100.00	-1.17	$[-2.16, -0.38]$	96.68	98.96
P_4	99.41	99.80	-0.39	$[-0.98, 0.00]$	96.29	99.54
P_8	99.61	99.80	-0.20	$[-0.59, 0.00]$	97.27	99.28
P_{16}	99.22	100.00	-0.78	$[-1.58, -0.19]$	98.44	99.35
Shared stride core						
P_2	96.68	100.00	-3.32	$[-5.04, -1.79]$	87.30	97.27
P_4	96.09	99.80	-3.71	$[-5.36, -2.02]$	88.09	97.07
P_8	96.68	99.80	-3.12	$[-4.56, -1.60]$	93.55	96.68
P_{16}	96.88	100.00	-3.12	$[-4.61, -1.57]$	96.88	96.74

Table 10: Continuing execution with the frozen base after a power call. Each row begins with a stride-s call, then applies one, two, or four base calls to its hidden output. The comparison uses s base calls followed by the same continuation, targeting the same board. Entries are percentages with all 81 reference cells correct, averaged over 256 directly encoded starts and 256 starts produced by four preliminary base calls. All starts come from validation.

Stride call	Stride call + continuation			Base rollout + continuation		
	1 base call	2 base calls	4 base calls	1 base call	2 base calls	4 base calls
Power bank (MoP)						
P_2	98.83	99.02	99.02	99.80	99.80	99.80
P_4	99.61	99.41	99.61	99.80	99.80	99.80
P_8	99.22	99.22	99.41	99.80	99.80	99.80
P_{16}	99.22	99.41	99.41	100.00	100.00	100.00
Shared stride core						
P_2	96.88	97.07	97.85	99.80	99.80	99.80
P_4	96.68	97.07	97.46	99.80	99.80	99.80
P_8	96.09	96.88	97.07	99.80	99.80	99.80
P_{16}	96.48	96.68	97.07	100.00	100.00	100.00

E.2 ALL FIXED-WINDOW TEST RESULTS

Table 11 reports all fixed-window results. We score the final call and all scheduled calls separately and give paired intervals against each condition's own unit model.

Table 11: Complete fixed-window Sudoku results, with 512 windows per row. P_s requests s placements and arrows specify call order. Endpoint exact checks the board after the last call; all scheduled landings exact checks the board after every call. All 81 cells must match the corresponding reference. Different schedules check different intermediate positions. Base endpoint is the condition's own unit model; n/a denotes comparison with itself. Differences are model minus paired base, in percentage points (pp), with 95% confidence intervals (CI) obtained by resampling puzzles. Depths 16/20 use test windows; depth 64 uses complete puzzles from the depth holdout.

Call schedule	Reference placements	Endpoint exact (%)	All scheduled landings exact (%)	Base endpoint exact (%)	Difference vs. base (pp)	95% CI (pp)
Power bank (MoP)						
1 call of P_{16}	16	98.24	98.24	99.22	-0.98	$[-2.33, 0.38]$
2 calls of P_8	16	98.83	98.44	99.22	-0.39	$[-1.39, 0.78]$
4 calls of P_4	16	98.24	97.46	99.22	-0.98	$[-2.34, 0.20]$
8 calls of P_2	16	97.66	93.95	99.22	-1.56	$[-3.11, -0.19]$
16 calls of P_1	16	99.22	98.83	99.22	n/a	n/a
$P_{16} \to P_4$	20	98.63	97.85	98.83	-0.20	$[-1.18, 0.78]$
20 calls of P_1	20	98.83	98.83	98.83	n/a	n/a
4 calls of P_{16}	64	97.85	81.05	99.41	-1.56	$[-3.33, 0.00]$
64 calls of P_1	64	99.41	84.77	99.41	n/a	n/a
Shared stride core						
1 call of P_{16}	16	96.88	96.88	99.22	-2.34	$[-3.92, -0.78]$
2 calls of P_8	16	96.68	94.14	99.22	-2.54	$[-4.29, -0.79]$
4 calls of P_4	16	96.29	88.28	99.22	-2.93	$[-4.80, -1.18]$
8 calls of P_2	16	96.48	84.38	99.22	-2.73	$[-4.37, -1.18]$
16 calls of P_1	16	99.22	98.83	99.22	n/a	n/a
$P_{16} \to P_4$	20	96.88	94.53	98.83	-1.95	$[-3.72, -0.39]$
20 calls of P_1	20	98.83	98.83	98.83	n/a	n/a
4 calls of P_{16}	64	96.29	59.18	99.41	-3.12	$[-5.23, -1.17]$
64 calls of P_1	64	99.41	84.77	99.41	n/a	n/a
Unit model (3 blocks/call)						
16 calls of P_1	16	97.66	96.88	97.66	n/a	n/a
20 calls of P_1	20	97.85	95.70	97.85	n/a	n/a
64 calls of P_1	64	91.80	57.81	91.80	n/a	n/a
Unit model (2 blocks/call)						
16 calls of P_1	16	90.82	84.57	90.82	n/a	n/a
20 calls of P_1	20	90.23	79.49	90.23	n/a	n/a
64 calls of P_1	64	60.55	3.52	60.55	n/a	n/a

E.3 UNIT-MODEL CONTROLS

Table 12: Board accuracy (%) for unit models with different blocks per call. The first three columns evaluate 1, 16, and 32 reference placements on 256 validation examples each. Whole-puzzle validation is the separately recorded completion score, also on 256 examples. Whole-puzzle test and depth-holdout scores use trajectory endpoints after the full reference length, on 3,388 and 513 puzzles. Correctness requires all 81 cells to match. Table 8 explains the output-query uncertainty.

Model	Validation: reference placements			Whole-puzzle scores		
	1	16	32	Validation	Test	Depth holdout
Unit model (4 blocks/call)	100.00	99.61	98.05	99.61	99.85	98.83
Unit model (3 blocks/call)	100.00	99.22	98.83	96.09	98.67	92.40
Unit model (2 blocks/call)	99.61	96.88	84.77	83.20	88.99	58.09

E.4 CALL SEQUENCES EXCLUDED FROM TRAINING

Table 13: Results for all 25 complete call sequences excluded from training. Brackets give strides in execution order: $[16, 4]$ applies P_{16}, then P_4. Each uses 512 Sudoku windows drawn separately from the fixed-window tests. Endpoint exact requires all 81 reference cells to match after the last call; all scheduled landings exact requires a match after every call. The number of checks varies with the schedule. Depths 16/20/35 use the test split; depth 64 uses the longer puzzles in the depth holdout.

Stride schedule	Reference placements	Expert calls	Power bank (MoP)		Shared stride core	
			Endpoint exact (%)	All scheduled landings exact (%)	Endpoint exact (%)	All scheduled landings exact (%)
$[2, 8, 4, 2]$	16	4	98.83	97.27	97.27	89.84
$[8, 2, 2, 4]$	16	4	98.05	96.88	96.88	92.58
$[8, 4, 2, 2]$	16	4	98.05	96.88	96.88	94.14
$[4, 2, 4, 2, 4]$	16	5	98.05	95.70	96.88	89.84
$[1, 4, 4, 4, 2, 1]$	16	6	99.22	98.44	96.09	94.14
$[2, 4, 4, 2, 2, 2]$	16	6	98.05	96.09	97.27	87.50
$[1, 2, 2, 2, 2, 2, 2, 2, 1]$	16	9	98.05	96.48	96.48	92.19
$[2, 2, 1, 2, 1, 2, 2, 2, 2]$	16	9	95.70	92.19	91.80	80.08
$[16, 4]$	20	2	99.61	99.61	96.48	94.53
$[4, 16]$	20	2	99.61	98.05	96.09	91.41
$[2, 8, 8, 2]$	20	4	99.22	96.88	97.66	89.45
$[8, 2, 8, 2]$	20	4	98.83	98.44	97.66	92.58
$[2, 2, 8, 4, 4]$	20	5	99.22	95.70	95.70	87.50
$[4, 4, 2, 4, 2, 4]$	20	6	99.61	96.88	94.92	87.50
$[2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2]$	20	11	99.22	92.97	94.14	79.69
$[16, 16, 2, 1]$	35	4	97.66	96.09	98.44	94.92
$[16, 2, 16, 1]$	35	4	98.05	97.27	98.83	93.75
$[2, 1, 16, 16]$	35	4	96.48	91.80	94.92	82.81
$[1, 2, 16, 8, 8]$	35	5	97.66	96.48	98.83	93.75
$[4, 8, 16, 4, 2, 1]$	35	6	98.05	95.70	98.05	83.98
$[8, 4, 4, 16, 2, 1]$	35	6	97.66	92.97	98.44	89.45
$[8, 8, 8, 8, 2, 1]$	35	6	98.44	94.92	98.05	89.45
$[16, 16, 16, 16]$	64	4	98.05	80.08	95.70	60.55
$[8, 16, 16, 8, 16]$	64	5	98.83	78.91	97.66	35.94
$[4, 4, 8, 16, 16, 16]$	64	6	98.44	70.70	94.53	20.31

E.5 PREDEFINED PATHS AND WHOLE-PUZZLE RESULTS

Table 14 pools whole puzzles and shorter windows. Table 15 restricts the comparison to paths that start at the initial board and cover the full puzzle. Their intermediate scores show how often a correct solution is accompanied by agreement with every scheduled reference board.

Table 14: Accuracy across predefined Sudoku paths, combining whole puzzles and shorter trajectory segments. Both models evaluate 15,157 test paths from 3,388 puzzles and 7,684 holdout paths from 513 puzzles. Each path counts equally, including paths using only the base; a puzzle with several paths contributes several times. Endpoint exact checks the last board; all scheduled landings exact checks every call's board against all 81 reference cells. Base endpoint is the paired four-block model. Differences are model minus base in percentage points (pp), with 95% confidence intervals (CI) obtained by resampling puzzles.

Model	Endpoint exact (%)	All scheduled landings exact (%)	Base endpoint exact (%)	Difference vs. base (pp)	95% CI (pp)
Test split					
Power bank (MoP)	99.51	96.45	99.58	−0.08	[−0.27, 0.15]
Shared stride core	99.59	92.38	99.58	0.01	[−0.25, 0.27]
Depth holdout					
Power bank (MoP)	93.68	78.64	95.71	−2.03	[−3.27, −0.76]
Shared stride core	92.39	59.85	95.71	−3.32	[−4.76, −1.79]

Table 15: Whole-puzzle results from the initial board. Repeat P_s uses only stride s, so the number of blanks must be divisible by s; mixed strides combines sizes. Endpoint exact requires all 81 cells to match the solution, while all scheduled landings exact also requires every intermediate call to match its reference board. Base endpoint uses the paired four-block model. Different rows select different eligible puzzles and check different numbers of boards, so cross-row percentages are not matched comparisons. The mixed-stride holdout row contains only one path.

Call schedule	Paths	Power bank (MoP)		Shared stride core		Base endpoint exact (%)
		Endpoint exact (%)	All scheduled landings exact (%)	Endpoint exact (%)	All scheduled landings exact (%)	
Test split						
Repeat P_{16}	288	100.00	99.65	100.00	100.00	100.00
Repeat P_8	800	99.88	95.88	100.00	92.00	99.88
Repeat P_4	810	99.88	95.43	99.88	86.91	99.88
Repeat P_2	1,651	99.27	89.58	99.88	79.53	99.94
Repeat P_1	3,388	99.85	98.08	99.85	98.08	99.85
Mixed strides	3,100	99.90	97.68	99.81	93.71	99.84
Depth holdout						
Repeat P_{16}	512	98.83	83.59	96.48	61.13	98.83
Repeat P_8	512	99.22	76.17	95.90	35.16	98.83
Repeat P_4	513	97.08	63.35	94.54	16.76	98.83
Repeat P_2	513	91.03	37.62	94.54	8.19	98.83
Repeat P_1	513	98.83	82.26	98.83	82.26	98.83
Mixed strides	1	100.00	100.00	100.00	100.00	100.00

E.6 COMPLETE TIMING RESULTS

Table 16 includes all idle-GPU timings for MoP and the shared core. Each model's speedup uses its own separately measured base time.

Table 16: Complete Sudoku timing results for Power bank (MoP), Shared stride core, and their four-block base models on an idle RTX PRO 6000. P_s requests s placements; arrows give call order. Recurrent time covers expert execution. End-to-end time includes encoding the inputs and decoding the final board. Values are median milliseconds for the entire batch over five synchronized repetitions. Each system's base was timed separately, so speedups use the base measurement from the same run.

Call schedule	Batch size 1		Batch size 256	
	Recurrent (ms)	End-to-end (ms)	Recurrent (ms)	End-to-end (ms)
Power bank (MoP)				
1 call of P_{16}	23.59	27.25	27.16	31.10
2 calls of P_8	26.41	29.52	29.44	32.78
4 calls of P_4	26.20	28.68	29.16	31.85
8 calls of P_2	26.31	28.44	29.13	31.50
16 calls of P_1	52.28	54.49	57.96	60.38
$P_{16} \rightarrow P_4$	30.27	34.60	33.98	38.60
20 calls of P_1	65.36	67.65	72.53	75.02
4 calls of P_{16}	93.92	97.84	107.92	112.18
64 calls of P_1	208.02	210.34	231.64	234.16
Shared stride core				
1 call of P_{16}	29.75	31.89	32.57	34.88
2 calls of P_8	30.44	32.75	32.67	34.99
4 calls of P_4	30.08	32.17	32.75	35.04
8 calls of P_2	30.52	32.64	33.25	35.53
16 calls of P_1	53.07	55.37	57.77	60.26
$P_{16} \rightarrow P_4$	37.45	39.59	40.68	43.06
20 calls of P_1	65.60	68.04	72.39	74.92
4 calls of P_{16}	118.65	121.07	129.84	132.36
64 calls of P_1	207.67	210.24	238.72	241.44