DISCOVERING LEARNING-FRIENDLY GENERATION ORDERS FOR SEQUENTIAL COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

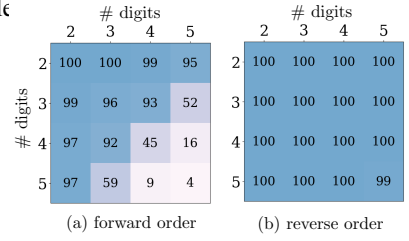
Sequential computation via autoregressive generation can make hard tasks learnable, but the generation order of intermediate steps strongly affects whether training succeeds. We address the problem of discovering a learning-friendly target order automatically, rather than designing it manually. Our key observation is that learning-friendly orders often exhibit a faster loss drop in the early stage of training. We exploit this by *loss profiling*, which ranks candidate orders by the early-stage loss of a single short run. To handle the factorial candidate space, we wrap loss profiling in a hierarchical global–local search over block- and within-block-level orderings. Experiments on benchmark tasks show that, in the best case, our method recovers the optimal forward order for sequences of length $L = 13$ from random initialization and $L = 40$ from structured initialization, lifting success rates from about 10% to near 100%. It also rediscovers the reverse-digit order for integer multiplication, reported to be efficient in prior studies. A case study in delay dynamical systems shows that learnability varies sharply even among valid topological sorts of the dependency graph. Namely, computational admissibility is not a sufficient condition for learning-friendliness.

1 INTRODUCTION

Discovering computational patterns in arithmetic and symbolic computation is challenging even for deep learning models. For example, learning the parity function—predicting the parity of an input bit string—has been theoretically shown to be hard for gradient-based learning (Shalev-Shwartz et al., 2017). This is because target functions (e.g., parity) are input-sensitive; a small input change generally alters the output. This contrasts with classical tasks such as image classification, where outputs are expected to be insensitive to random input perturbations. Hence, learning input-sensitive functions is of fundamental interest, and many techniques have been developed to improve empirical learnability on hard arithmetic and symbolic computation tasks (Wenger et al., 2022; Li et al., 2023a;b; Kera et al., 2024; 2026; Alfarano et al., 2024).

A powerful approach to address this is sequential computation via autoregressive generation. A computational task is broken down into subproblems, and the model autoregressively predicts a sequence of intermediate computation tokens towards the final answer. For example, Kim & Suzuki (2025) have proven that the step-by-step prediction of the parity of the first k bits for $k = 1, 2, \ldots$, makes parity function learning successful, as it turns the parity computation of $k + 1$ bits into 2-bit parity computation (i.e., the k-th sub-parity and the $(k+1)$-th bit). As such, fully exploiting sequential computation makes several almost unlearnable tasks learnable.

An important yet underexplored aspect is the order of sequential computation. Shen et al. (2023) have shown that Transformers learn multiplication of two integers with better generalization to larger integers when the product is predicted from least to most significant digits (cf. Fig. 1). Concretely, a GPT-2-small model trained on 300k instances of up-to-five-digit multiplication attains near-perfect accuracy with the least-significant-first order, but performs substantially worse with the most-significant-first order. In both cases, the model performs sequential prediction of digits,



(a) forward order (b) reverse order

Figure 1: Success rates for integer multiplication. The rows/columns are the numbers of digits in the two operands.

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.

but the order matters. Manual choices of an order rely on task-specific knowledge (e.g., that carries propagate from least to most significant digits in multiplication), and become infeasible when the underlying computational pattern is itself unknown. A systematic procedure for determining a learning-friendly order is therefore necessary.

This study addresses the task of reordering target tokens into a learning-friendly order for better learning of sequential computation tasks. Exploiting the observation that neural networks tend to learn from easy to hard instances during training (Arpit et al., 2017; Forouzesh & Thiran, 2024; Swayamdipta et al., 2020), we train a lightweight Transformer on a mixture of target sequences in different orders and identify those that lead to a faster loss drop in the early stages of training. To handle longer sequences, we propose a two-stage hierarchical approach, where the global stage finds block-level orders, while the local stage refines the order within each block.

The experiments demonstrate that the proposed method often enables successful training by reordering target sequences. We designed six benchmark tasks that are relatively easy for Transformers to learn in the forward order, but challenging in most other orders. Despite the large search space that grows factorially in the target sequence length L, successful recovery reaches $L = 13$ with random initial candidates and $L = 40$ with structured candidates in the best-performing task settings. We also applied our method to the multiplication task and successfully rediscovered the reverse-digit order reported by Shen et al. (2023). As a more realistic case study, we examine delay dynamical systems, a class of multi-variable recurrences common in physical and biochemical modeling. The results show that there is a large variance in learnability even among valid topological sorts of the dependency graph, which indicates that learning-friendly orders are not merely computationally admissible orders but require additional conditions.

Our contributions are summarized as follows:

- We address a novel task of discovering a learning-friendly order of target tokens. We define order-sensitive tasks and introduce six instances of recurrence-based sequential computation that serve as a benchmark for this new task. An empirical case study on delay dynamical systems shows that knowing dependency graphs may provide computationally admissible orders, but not all of them are necessarily learning-friendly.
- We propose a two-stage exploration method that efficiently determines learning-friendly orders. The exploration maintains a set of candidate permutations, and the candidates are ranked and replaced based on loss profiling, which exploits the nature of easy-to-hard learning of neural networks.
- Our experiments demonstrate the broad applicability of the method. On six recurrence tasks, it scales to sequence lengths up to $L = 13$ with random initial candidates and up to $L = 40$ with structured ones, and it rediscovers the reverse order reported by Shen et al. (2023) for integer multiplication.

2 Related work

Transformers for sequential computation. Transformers have been applied to a wide range of sequential computation tasks, e.g., arithmetic (Power et al., 2022; Jelassi et al., 2023; McLeish et al., 2024; Zhou et al., 2024), symbolic computation (Lample & Charton, 2020; Charton, 2022; Alfarano et al., 2024; Kera et al., 2024; 2026), and number-theoretic problems (Wenger et al., 2022; Li et al., 2023a;b; Charton, 2024; 2025). Some of these tasks are known to be difficult to learn end-to-end; for example, parity prediction is provably hard for gradient-based learners (Shalev-Shwartz et al., 2017) and for encoder-only Transformers (Hahn & Rofin, 2024). A remedy is to introduce intermediate computation steps, as Kim & Suzuki (2025) have shown for parity by predicting the parities of the first k bits step by step. Within this line of work, several axes have been identified as critical to learnability, including the choice of token representation and positional encoding (Jelassi et al., 2023; McLeish et al., 2024; Zhou et al., 2024), the training distribution and learned rules (Charton, 2024), and length generalization (Zhou et al., 2024; Jelassi et al., 2023). This study focuses on a less-explored axis: the order in which the intermediate states are emitted by the autoregressive decoder.

Generation order in autoregressive learning. The most relevant work is Shen et al. (2023), which showed better generalization to longer sequences in integer multiplication by reversing the digit order. This reverse-digit ordering was motivated by the observation that carries propagate from least to

most significant digits; other orderings may be suitable for other sequential computation tasks. A related but separate line of work studies generation order in natural language generation. Welleck et al. (2019) train text generators that decide their own generation order by recursively expanding a binary tree, and Asano et al. (2026) learn an unmasking order at inference time for masked diffusion language models. These studies focus on generation order for natural-language tasks. In contrast, our study focuses on learnable orders at training time; in the sequential computation tasks we consider, many orders make training itself fail or become substantially more difficult.

3 Problem Setup

Our problem is to discover a target sequence order of the Transformer decoder that improves learning. We first set up our task and then define order sensitivity. Lastly, we clarify the scope of this work.

3.1 Discovering learning-friendly orders.

Let Σ be the set of all tokens. We denote the set of all finite token sequences by Σ^* and its restriction to length-L sequences by Σ^L. Let $\mathcal{T}_\theta : \Sigma^* \times \Sigma^L \to \Sigma^L$ be a Transformer with parameters θ. We assume that the target sequence length is fixed. Let $(X, Y) \sim \mathcal{D}$ be an input–target sequence pair (X, Y) with $|Y| = L$ from a joint distribution $\mathcal{D}$. Let S_L be the set of all permutations over $\{1, \ldots, L\}$. The empirical risk minimizer $\theta^\pi_{\mathrm{ERM}}$ with a finite sample set $D = \{(X_i, Y_i)\}_{i=1}^m$ and permutation $\pi \in S_L$ is

$$\theta^\pi_{\mathrm{ERM}} = \arg\min_\theta \frac{1}{m} \sum_{i=1}^{m} \ell(\mathcal{T}_\theta, X_i, \pi(Y_i)), \tag{1}$$

where ℓ is a loss function. Our goal is to discover a permutation π that minimizes the expected risk:

$$\pi^* = \arg\min_{\pi \in S_L} \mathbb{E}_{(X,Y)\sim\mathcal{D}}\big[\ell\big(\mathcal{T}_{\theta^\pi_{\mathrm{ERM}}}, X, \pi(Y)\big)\big]. \tag{2}$$

The choice of π is crucial particularly for sequential computation tasks. With a generic order (e.g., a random order), the task is almost unlearnable, while an optimal order leads to nearly 100% success rate (cf. Tab. 1: forward vs. reverse orders). We next introduce a computational notion of *order sensitivity*.

3.2 Order-sensitive tasks

We call a task *order-sensitive* if the ratio of expected minimal operation counts between two output orders grows without bound with sequence length (see Sec. A for the formal definition). We use recurrence tasks to study how output order affects computation and learning.

Example 3.1 (ReLU sequence)**.** Let $X = [x_1, \ldots, x_L]$ be a sequence of integers. The ReLU sequence $Y = [y_1, \ldots, y_L]$ is: $y_1 = \mathrm{ReLU}(x_1) = \max(x_1, 0)$ and $y_i = \mathrm{ReLU}(x_i + y_{i-1})$ for $i = 2, \ldots, L$.

With the input X and the target in the natural causal order, i.e., $Y = [y_1, \ldots, y_L]$, autoregressive generation determines the next token y_i locally from the immediate context (e.g., y_{i-1} and x_i). In the reverse order $[y_L, \ldots, y_1]$, the preceding state y_{i-1} is not yet available as an output. When $x_i + y_{i-1} < 0$, ReLU maps multiple preceding states to the same value.

More generally, we consider recurrence tasks with a non-invertible transition function f: $y_1 = g(x_1)$ and $y_i = f(x_i, y_{i-1})$ for $i \geq 2$, with a task-specific initialization g. Table 1 presents six tasks with a pronounced forward–reverse gap in learnability, which we use as benchmarks. The MLP, Sine, and Cubic tasks have strong nonlinearity and are particularly challenging tasks.

Failure of alternative strategies. Two natural alternatives fail in this setting. (i) Jointly optimizing a soft permutation $\tilde{P} \in [0, 1]^{L\times L}$ together with model parameters (target preprocessed as $Y\tilde{P}$) leaks information across positions and undermines next-token prediction (Secs. B.1.1 and B.1.2). (ii) An evolutionary-strategy baseline for permutation search did not yield a practical alternative (Sec. B.2).

Table 1: Six recurrence tasks and success rates (%) at L=20 for Transformers trained with a fixed forward or reverse order; the forward order is substantially more learning-friendly across all tasks. L=50 results in Tab. 10; setup details in Sec. 5 and Sec. A.3.

Task	Recurrence ($i \geq 2$)	Forward	Reverse
RELU	$y_i = \mathrm{ReLU}(x_i + y_{i-1})$	**99.6**	5.6
SQUARE	$y_i = ((x_i^2 + y_{i-1}^2) \bmod N) - \lfloor N/2 \rfloor$	**100**	0.1
TRIANGLE	$y_i = \lvert((y_{i-1} + x_i) \bmod 2N) - N\rvert$	**100**	0.2
MLP	$y_i = \mathrm{round}\big(\mathrm{MLP}(x_i, y_{i-1})\big) \bmod N$	**100**	9.4
SINE	$y_i = \lfloor A \sin\big(\frac{2\pi}{P}(y_{i-1} + x_i)\big)\rfloor \bmod N$	**100**	6.2
CUBIC	$y_i = (y_{i-1}^3 + x_i) \bmod N$	**100**	1.6

Scope of this work. We target sequential computation tasks where the order of autoregressive generation critically affects learnability as the target computation is input-sensitive and a proper breakdown into subproblems is key. Natural language is beyond the scope of the current work because of its strong redundancy; words exhibit strong co-occurrence patterns (e.g., "live" and "in"), and even sentence- or paragraph-level reorderings leave function words such as "the" or "it" that hint at the original order. As such, suboptimal orders leave enough information to learn the target computation.

4 PROPOSED METHOD

We now present a method for discovering learning-friendly orders. It consists of two components, loss profiling and hierarchical exploration, and is based on the following assumptions.

Assumption 1: Learning prefers computationally efficient orders. For order-sensitive sequential computation tasks, we hypothesize that Transformer training is biased toward orders that require fewer operations. This is consistent with the empirical tendency of deep models to fit simpler patterns first.

Assumption 2: Weak locality property. We assume that small local reorderings do not drastically change the learning quality of an order. In the RELU recurrence, for example, swapping neighboring targets around the forward order usually introduces only a one-step skip in sequential computation, so the required function class expands only mildly. In contrast, moving tokens in the late-stage outputs toward the beginning severely breaks the causal structure.

Assumption 1 is natural to expect. Interestingly, Sec. 6 will show that it is not sufficient for being learning-friendly. Assumption 2 is a reasonable restriction on our task; without this, tiny local swaps can abruptly flip learnability, and exhaustive search over full permutations becomes necessary.

4.1 LOSS PROFILING

We first present *loss profiling*. Given a set of candidate permutations, loss profiling narrows down the set to a few learning-friendly orders through short training runs of a small Transformer model on a mixed dataset of variously ordered target sequences.

The key observation is that the validation loss drops faster for learning-friendly orders (Fig. 2). This derives from a characteristic of the training dynamics of deep neural networks: they tend to learn easy samples in the early stages of training, and gradually adapt to harder samples later. Several studies have already leveraged this characteristic, such as Arpit et al. (2017) for learning with noisy labels and Baldock et al. (2021) for identifying difficult examples.

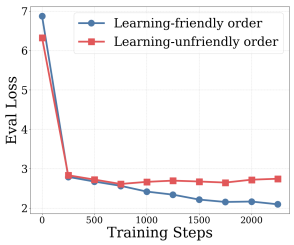


Figure 2: Early-stage validation loss curves.

In loss profiling, a Transformer is trained only for short periods on a mixture of target orders; it then identifies learning-friendly orders as "easy samples," for which the loss drops faster.

Formally, let $D = \{(X_i, Y_i)\}_{i=1}^{m}$ and $D' = \{(X_i', Y_i')\}_{i=1}^{m'}$ be training and validation sets, respectively. Let $\mathcal{P} = \{P_1, \ldots, P_T\}$ be the set of T candidate permutation matrices. Let D^{P_t} be the set D

with target sequences reordered by P_t, i.e., $D^{P_t} = \{(X_i, Y_i P_t)\}_{i=1}^m$. The learning-friendly orders are determined as follows.

P1. Train a Transformer for n_e epochs on a mixed dataset $\bar{D} := \bigcup_{t=1}^T D^{P_t}$.

P2. Compute the validation loss for each $P_t \in \mathcal{P}$ as $\mathcal{L}(D', P_t) = \frac{1}{m'} \sum_{i=1}^{m'} \ell(\mathcal{T}_{\theta'}, X_i', Y_i' P_t)$, where $\mathcal{T}_{\theta'}$ is the trained model at **P1**. Then, output $P^* = \arg\min_{P_t \in \mathcal{P}} \mathcal{L}(D', P_t)$.

We empirically observed that loss profiling can handle a few thousand permutations, but the number of permutations grows factorially in the sequence length. Next, we introduce two-stage hierarchical optimization, which calls loss profiling as a subroutine.

4.2 Hierarchical Exploration

We present the global–local pipeline (cf. Fig. 8) with the initial set of permutation candidates $\mathcal{P}_0 = \{P_1, \ldots, P_T\}$. The global stage splits each token sequence into blocks and improves block-level permutations. The local stage refines this by permuting the tokens within each block.

Global stage. Let the search depth be K and $T = 2K!$. Let $\mathcal{P}_1 := \mathcal{P}_0$. For $k = 1, \ldots, K$, we split each target sequence into k blocks, and apply loss profiling to the new permutation set:

$$\bigcup\nolimits_{P \in \mathcal{P}_k} \{PQ_1, \ldots, PQ_{k!}\} \cup \{PQ_1 J, \ldots, PQ_{k!} J\}, \tag{3}$$

where $Q_i \in [0,1]^{L \times L}$ are the block-level permutations and $J \in [0,1]^{L \times L}$ is the exchange matrix (reversal permutation). The best $\lfloor T/(2k!) \rfloor$ permutations define the new candidate set $\mathcal{P}_{k+1}$, where $\lfloor \cdot \rfloor$ denotes the floor function. We denote the best permutation obtained in the final iteration ($k = K$) as P_g. This permutation is then refined with the local stage.

Local stage. Let $P := P_g$ be the current permutation. For each block length $l \in \{2, 3, \ldots, \lfloor L/2 \rfloor\}$, we split the current target sequence into $\lceil L/l \rceil$ blocks, with any remainder forming the last block. Let s_i be the length of block i (typically $s_i = l$), and let $R_1^i, \ldots, R_{s_i!}^i \in \{0,1\}^{L \times L}$ enumerate its within-block permutations, leaving all other positions unchanged. We process the blocks sequentially. For each block i, loss profiling ranks the candidate set $\{PR_1^i, \ldots, PR_{s_i!}^i\}$. We update P to the lowest-loss candidate before proceeding to the next block. After processing all blocks, we keep their internal orders fixed and apply loss profiling once to all $\lceil L/l \rceil!$ block-reordering candidates. The best candidate initializes P for block length $l + 1$.

Computational overheads. Three aspects keep the framework efficient. (i) Each profiling run trains for only 800–1,600 steps (1–2 epochs with 10^5 samples and batch size 128), since loss-drop differences emerge early. (ii) A single run handles up to $2 \times 7! = 10{,}080$ permutations. (iii) The global stage uses K training runs, while the local implementation, which profiles and updates blocks sequentially, uses $O(L \log L)$ training runs (see Sec. F.6 for the exact count). This counts training runs, not total runtime, which also includes candidate evaluation. We use small Transformers for exploration. The sensitivity to model configuration is tested in Sec. 5.1.

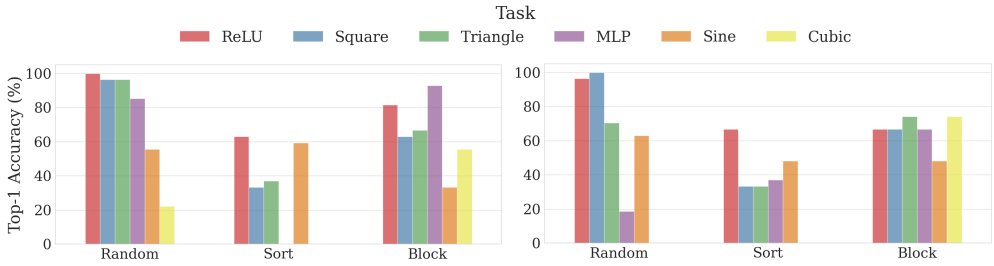
5 Experiments

This section validates loss profiling and global–local exploration. We first summarize the basic experimental setup. Further details will be specified as needed or deferred to the appendix.

Datasets. We generated datasets for the six tasks given in Tab. 1. We considered both fixed-length and variable-length cases. We used $L \in \{5, 6, \ldots, 50\}$ for the former, and L from two ranges, $L \in \{20, 21, \ldots, 30\}$ and $L \in \{40, 41, \ldots, 50\}$, for the latter. We prepared 10^5 training and 10^3 validation samples for exploration, and 10^3 evaluation samples for final training.

Exploration. As described in Sec. 4, exploration repeatedly trains small, lightweight GPT-2 models (Radford et al., 2019) for a small number of steps, and loss profiling ranks permutation candidates using the validation set. Training settings are detailed in Sec. D.1. The configuration of the small models and the initial candidate permutations of global–local search will be specified later.

Evaluation (Final Training). Unless otherwise specified, we train a GPT-2 model with six layers, eight attention heads, and $(d_{\text{emb}}, d_{\text{ffn}}) = (512, 2048)$ using the discovered order. Success rate measures sequence-level exact-match accuracy over evaluation samples.



(a) $L = 50$ (b) $L \in \{40, 41, \ldots, 50\}$

Figure 3: Top-1 identification accuracy of loss profiling: (a) fixed-length targets; (b) variable-length targets. Each run is initialized with one of the permutation sets Random $\mathcal{P}_{\mathrm{r}}$, Sort $\mathcal{P}_{\mathrm{s}}$, or Block $\mathcal{P}_{\mathrm{b}}$.

Importantly, note that this section focuses on the benchmark tasks for which the optimal orders are known since this allows more rigorous evaluation; otherwise, we cannot evaluate whether the exploration attains optimality and how close the discovered order is to optimality. The case with unknown optimal orders will be addressed in Sec. 6.

5.1 Loss profiling for discovering the forward order

We first validate that loss profiling is effective for discovering learning-friendly orders from candidate sets. We consider three types of candidate sets. Figure 9 provides a visualization.

- $\mathcal{P}_{\mathrm{r}}$ (**Random**) consists of the forward order and random orders (permutations). The forward order typically outperforms others by a large margin.
- $\mathcal{P}_{\mathrm{s}}$ (**Sort**) consists of partially sorted orders. The bubble sort algorithm reorders a random permutation toward the forward order, and we evenly sample permutations from the sorting trajectory. This set ranges from learning-friendly (nearly sorted) to unfriendly (random) orders.
- $\mathcal{P}_{\mathrm{b}}$ (**Block**) consists of orders that are randomly shuffled at the block level. We fix the block size to $b = 5$. The last block can be smaller than b if b does not divide the sequence length. This candidate set is reasonable when one can assume prior knowledge of the local structure.

The structured initializations $\mathcal{P}_{\mathrm{s}}$ and $\mathcal{P}_{\mathrm{b}}$ are motivated by prior knowledge of partial/local token ordering. For example, monomial-level permutation on polynomial tasks and time-step-level permutation on multivariate time-series tasks naturally lead to a block-level permutation, i.e., $\mathcal{P}_{\mathrm{b}}$.

Setup. To show the robustness of loss profiling to the choice of model configurations, we tested 27 GPT-2 configurations by varying the number of layers $n_{\mathrm{layer}} \in \{1, 2, 4\}$, attention heads $n_{\mathrm{head}} \in \{1, 2, 4\}$, and model dimensions $(d_{\mathrm{emb}}, d_{\mathrm{ffn}}) \in \{(128, 512), (256, 1024), (512, 2048)\}$. Given a set of permutations $\mathcal{P}$, loss profiling assigns a score to each permutation. Top-k accuracy measures whether the forward order appears among the top k.

Stability across model configurations. Figure 3 shows that loss profiling discovers the forward order with high accuracy across 27 model configurations in both fixed-length and variable-length settings. The top-1 accuracy moderately degrades for the structured initializations $\mathcal{P}_{\mathrm{s}}$ and $\mathcal{P}_{\mathrm{b}}$ as their candidates are more similar. Section E reports further results for other tasks, lengths, and top-k accuracies. Model-size analysis can be found in Sec. E.2.

Correlation with final test accuracy. Table 2 shows high correlation between the profiling ranks and the final test accuracies (Sec. E.3). In the hardest task, Cubic, the correlation is limited, but the top-5 overlap is still moderate, indicating that the profiling ranks are still informative.

Table 2: Rank agreement with final test accuracy at $L = 30$.

Task	Spearman ρ	Top-5 overlap
ReLU	0.924	68.9%
MLP	0.898	58.5%
Cubic	0.200	43.7%

5.2 Global–local exploration of learning-friendly orders

We now demonstrate that the proposed method can discover learning-friendly orders.

Table 3: Success rates (%) of final training using the permutations discovered by the global–local pipeline. **Bold** indicates that the optimal (forward) order is successfully discovered.

Initialization	Target Length	Task					
		RELU	SQUARE	TRIANGLE	MLP	SINE	CUBIC
$\mathcal{P}_{\mathrm{r}}^{-}$	$L = 10$	**99.6**	**100.0**	**100.0**	38.2	54.2	0.3
	$L = 13$	**99.2**	5.6	7.4	10.8	26.3	0.2
	$L = 20$	24.2	10.4	0.8	5.9	8.2	0.0
$\mathcal{P}_{\mathrm{s}}^{-}$	$L = 20$	**99.6**	**100.0**	**100.0**	89.4	**100.0**	1.9
	$L = 30$	**97.6**	**100.0**	**100.0**	**100.0**	**100.0**	3.8
	$L = 40$	3.4	0.0	0.0	0.4	1.2	0.0
$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 20$	**99.6**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	$L = 30$	**97.6**	**100.0**	**100.0**	**100.0**	1.7	0.7
	$L = 40$	**96.0**	2.1	1.2	**100.0**	0.0	0.0

Setup. The search depth is $K = 6$ for short sequences ($10 \leq L \leq 13$) and $K = 7$ for longer ones ($20 \leq L \leq 40$) for better convergence. The exploration starts with either of the three initial candidate sets: $\mathcal{P}_{\mathrm{r}}^{-}, \mathcal{P}_{\mathrm{s}}^{-}, \mathcal{P}_{\mathrm{b}}^{-}$ (visualized in Fig. 10). The superscript '-' indicates that the forward order is replaced by a random permutation for $\mathcal{P}_{\mathrm{r}}$ and a block-level random one for $\mathcal{P}_{\mathrm{b}}$. For $\mathcal{P}_{\mathrm{s}}^{-}$, a random permutation is sorted toward the *reverse* order. For $L \geq 20$, only the global stage is conducted as the local stage is costly. We tested four parameter combinations ($n_{\mathrm{layer}} \in \{1, 2\}, n_{\mathrm{head}} \in \{1, 2\}$) and report single-run results from the one with the highest success rate after final training in Tab. 3. Given the analysis in Fig. 11, we set $(d_{\mathrm{emb}}, d_{\mathrm{ffn}})$ to $(128, 512)$ for $\mathcal{P}_{\mathrm{r}}$ and $\mathcal{P}_{\mathrm{s}}$, and to $(512, 2048)$ for $\mathcal{P}_{\mathrm{b}}$.

Results. With $\mathcal{P}_{\mathrm{r}}^{-}$ and $L = 10$, the exploration successfully discovers the forward order with near-perfect accuracy for the RELU, SQUARE, and TRIANGLE tasks. Non-trivial accuracy is also achieved for MLP and SINE. The CUBIC task presents itself as the hardest. Note that $L = 10$ already requires searching among $10! > 3 \times 10^6$ permutations and thus is challenging.

Table 4: Best success rate (%) among 32 sampled orders.

Task	Pool	$L = 30$	$L = 40$
RELU	$\mathcal{P}_{\mathrm{s}}^{-}$	30.2	3.2
	$\mathcal{P}_{\mathrm{b}}^{-}$	86.1	64.6
SQUARE	$\mathcal{P}_{\mathrm{s}}^{-}$	0.1	0.2
	$\mathcal{P}_{\mathrm{b}}^{-}$	31.0	25.0

Structured initialization. Table 3 also shows that structured initializations $\mathcal{P}_{\mathrm{s}}^{-}$ and $\mathcal{P}_{\mathrm{b}}^{-}$ extend the success to larger L. Yet, Tab. 4 shows that simple random sampling from them is not effective (see Sec. F.4 for the setup and additional controls), showing that the structured pools alone do not guarantee a high-accuracy order among the sampled candidates. Note that 32 runs are more costly than running our exploration (cf. Tab. 30).

Additional three-seed experiments assess seed variability in 18 settings (Sec. F.5). Section F provides further breakdowns and analyses.

5.3 REDISCOVERING THE OPTIMAL ORDER FOR THE PROD TASK

We finally address the PROD task to rediscover the reverse order reported by Shen et al. (2023).

PROD. Let L be even. The input sequence is a concatenation of two left-zero-padded digit sequences, i.e., $X = [a_1, \ldots, a_{L/2}, b_1, \ldots, b_{L/2}]$, where a_k, b_k are the k-th most significant digits of integers a and b, respectively. The target sequence $[c_1, \ldots, c_L]$ is the left-zero-padded sequence of digits of ab. As Fig. 1 shows, the reverse order is known to be learning-friendly, while the forward order is not.

Setup. As multiplication involves complex carry propagation, the final training uses a larger GPT-2 (n_{layer}=12, n_{head}=12, $(d_{\mathrm{emb}}, d_{\mathrm{ffn}})$=$(768, 3072)$, following (Shen et al., 2023)) and an enlarged training set of 10^6 samples. The digit counts of a and b are sampled uniformly from $\{1, \ldots, L/2\}$ for training and from $\{L/2-2, L/2-1, L/2\}$ for evaluation, where hard cases concentrate (Fig. 1).

Results. Table 5 shows successful rediscovery of the reverse order for $\mathcal{P}_{\mathrm{r}}^{-}$ at $L = 10$ and for $\mathcal{P}_{\mathrm{s}}, \mathcal{P}_{\mathrm{b}}^{-}$ at $L = 20$. The search fails at larger L; learning is too difficult even with the reverse order, so it loses its early-stage edge over other orders. Table 6 shows that exploration is shorter than even a single full training run; exhaustive order ranking by retraining would require many full runs and is impractical.

Table 5: Success rates (%) on PROD using the discovered orders; **bold** marks runs that recovered the reverse order.

Initialization	Target Len.	Success Rate (%)	
		Discovered	Forward
$\mathcal{P}_{\mathrm{r}}^{-}$	$L = 10$	**100.0**	72.9
	$L = 12$	51.4	51.4
$\mathcal{P}_{\mathrm{s}}$	$L = 20$	**98.2**	9.7
$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 20$	**98.2**	9.7

Table 6: Runtime breakdown (in minutes) of the exploration and final training phases for the PROD task.

Params	Exploration		Final
	Global	Local	Training
$K = 6, L = 10$	116.7	78.8	470.2
$K = 6, L = 12$	121.7	102.2	511.6
$K = 7, L = 20$	599.7	—	639.5

6 DELAY DYNAMICAL SYSTEMS: A CASE STUDY

This section presents a case study on the effect of orderings on time-series prediction in delay dynamical systems. Unlike in the previous section, the recurrence is now multivariate, and the optimal order is unknown.

Example 6.1 (ReLU delay network)**.** The ReLU delay network has state $s(t) = (a(t), b(t), c(t), d(t))$. Given initial states $s(1), s(2)$, it evolves according to

$$a(t{+}2) = \mathrm{ReLU}(b(t{+}1) + d(t)), \qquad b(t{+}2) = a(t{+}1) - c(t{+}1),$$
$$c(t{+}2) = b(t) + d(t{+}1), \qquad d(t{+}2) = \mathrm{ReLU}(a(t{+}1) + c(t)).$$

Delay systems are common in various dynamical systems, where variables depend on their own lagged values or those of other variables, e.g., due to different timescales of biochemical reactions (Erneux, 2009). Real-world systems are often continuous, but naive autoregressive models suffer from error accumulation, so we focus on discrete delay systems.

6.1 CHOOSING FROM TOPOLOGICAL SORTS.

In applications, network topology is sometimes known. For example, interactions in gene regulatory networks can be inferred from gene knockout experiments, yet interaction strengths and nonlinearities remain unknown. This motivates selecting a learning-friendly order for time-series prediction from topological sorts, i.e., orderings of variables that respect the dependency structure.

Datasets. We introduce six delay systems (including the ReLU delay network), defined in Sec. G.1, with time horizon $T = 11$: the input is $(s(1), s(2))$ and the target comprises the 36 integers in $s(3), \ldots, s(11)$. The input states are uniformly sampled from bounded domains (Sec. G.2), and the subsequent states are computed according to the delay systems. We prepare $100{,}000/2{,}000/1{,}000$ training/validation/test trajectories for each system.

Setup. We prepare four manually designed topological orders and one reverse-order baseline (Sec. G.3). For each of these orders, GPT-2 models with six layers, embedding/FFN dimensions $128/512$, and one attention head are trained for 10 epochs. We also introduce a multi-head GPT-2 model that predicts all variables at each time step simultaneously (Sec. G.4). Loss profiling uses a 1-layer GPT-2 for one epoch. Both training and loss profiling are repeated with three seeds; full training and evaluation details are in Secs. G.2 and G.4.

Results. Table 7 summarizes the results. The correlation ρ measures agreement between profiling-loss ranks and final test-accuracy ranks across the five AR orders. The selection counts show that loss profiling selects high-accuracy orders for the first four systems, although the rank correlation is variable on COUPLED RING. Notably, multi-head simultaneous prediction shows limited performance on most tasks. This supports the value of studying learning-friendly orders in autoregressive generation; autoregressive generation can be effective for these discrete systems, but the default time-major variable order need not be the most learning-friendly. Larger-T results on RELU are in Sec. G.6.

Computational admissibility is not enough. The substantial variation in learnability among topological orders in Tab. 7 suggests an interesting gap: computational admissibility may be necessary for learning-friendliness, but it is not sufficient. This observation implies that knowing the dependency

Table 7: Delay-network exact match (%, mean±sample SD, three seeds). Bold/underline: best/second-best mean per row, including ties. Each asterisk marks one seed's loss profiling selection. ρ: profiling–accuracy rank correlation over the five AR orders (Sec. G.4).

Task	time_first	var_first_topo	dep_sorted	causal_depth	reverse	MH	ρ
ReLU	73.4±2.9	**99.6±0.2*****	79.6±10.4	69.0±9.0	0.0±0.0	1.9±3.3	0.81±0.10
Hidden-cause	54.6±10.8	**96.6±2.3****	79.6±20.6*	54.6±14.2	1.4±0.5	12.1±1.2	0.93±0.06
Coupled ring	98.2±2.6***	99.6±0.2	**99.7±0.1**	98.2±2.6	26.2±6.5	29.6±11.2	0.36±0.53
Sine	99.8±0.3	99.8±0.3	**100.0±0.0*****	99.4±0.3	0.0±0.0	99.1±0.2	0.87±0.04
Modular	66.8±57.6	0.0±0.0**	**74.4±44.4***	33.3±57.7	0.0±0.0	0.0±0.0	−0.10±0.38
Mixed Nl.	78.6±20.2	30.9±48.2***	32.3±47.3	**81.8±12.8**	0.0±0.0	25.6±9.9	0.10±0.20

structure is not enough to find a learning-friendly order, and additional constraints are needed. The success of the reverse order in integer multiplication (Shen et al., 2023) was attributed to its computational admissibility, but our observation suggests that additional conditions exist. Elucidating the conditions is left for future work.

6.2 No prior on topology.

We next run our hierarchical exploration without any prior on the topology.

Setup. We run only the global exploration stage as the local stage is costly for the target length 36. The global stage uses two initializations. `topo`: 100 random topological sorts. `no_topo`: $4! = 24$ time-aligned orderings with consistent permutations over four variables. The former assumes the dependency graph is known; the latter does not, only using the time-induced block initialization. The discovered order is retrained with the same 6-layer GPT-2 protocol as in the previous subsection. For each initialization and seed, we perform global search at four hierarchy depths, $K \in \{2, 3, 4, 5\}$, and select the discovered order with the lowest validation profiling loss for final training. We repeat the complete search and retraining pipeline with three seeds.

Table 8: Global-search success rates (%, mean±sample SD over three seeds).

Task	topo	no_topo
Hidden-cause	0.9 ±0.5	**29.4 ±7.7**
Coupled ring	59.7 ±12.2	**99.6 ±0.4**
Sine	99.2 ±0.5	**100.0 ±0.1**
Modular	0.0 ±0.0	**0.2 ±0.3**
Mixed Nl.	0.0 ±0.0	**58.6 ±51.5**

Results. Table 8 shows that the topology-agnostic initialization achieves higher mean accuracy on all five tasks. These results suggest that time-aligned initialization can be more effective for global search than random topological sorts, despite requiring no knowledge of the dependency graph. However, success remains task-dependent: the search nearly matches the best hand-designed orders on Coupled ring and Sine, but falls short on the other tasks (Tab. 33). Another experiment on random arithmetic DAGs can be found in Sec. H.

7 Conclusion

We studied how to find learning-friendly output orders for autoregressive sequential computation. Our method combines loss profiling with global–local exploration: early validation losses from lightweight models guide the search over candidate orders. Experiments on controlled recurrence tasks demonstrate the effectiveness of both components, and the multiplication experiment recovers the reverse-digit order reported in prior work. On delay dynamical systems, where the optimal order is unknown, learnability varies substantially even among valid topological sorts. Global exploration also finds effective orders without access to the dependency graph. These results motivate treating output order as a choice to be explored rather than fixed by computational dependencies alone.

Limitations and future work. Local refinement remains costly for long sequences, and early losses do not reliably distinguish learning-friendly orders in every task. More efficient refinement and more reliable profiling signals are therefore important directions. Understanding what makes an order learning-friendly beyond respecting computational dependencies, and extending the approach to natural-language tasks, remain open challenges.

AI USE STATEMENT

In this work, we used generative AI tools to implement and edit experiment code and analysis scripts. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate or prove mathematical claims, translate the manuscript, or clean datasets, and survey or interview instruments are not applicable to this work. Additionally, we used generative AI tools to draft and edit the manuscript for readability, to format tables and references, to search related work, and to brainstorm wording. We have reviewed all AI-assisted text, code, and analysis. Experimental results were produced by our own training runs and checked by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies Transformers trained from scratch on synthetically generated numerical sequences. It does not involve human subjects, personal data, or deployed decision-making systems. We do not foresee direct ethical risks beyond the general dual-use considerations of machine learning research.

REPRODUCIBILITY STATEMENT

Model architectures, optimizer settings, hierarchical search depth, candidate-set constructions, and dataset-generation rules (including random seeds) are specified in Sec. 5 and the appendix, notably Sec. D.1 and Sec. D. Source code is not included with this submission; we plan to release the code and data-generation scripts after acceptance.

REFERENCES

Alberto Alfarano, Francois Charton, and Amaury Hayat. Global lyapunov functions: a long-standing open problem in mathematics, with symbolic transformers. In *Advances in Neural Information Processing Systems*, 2024.

Devansh Arpit, Stanisław Jastrzundefinedbski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In *International Conference on Machine Learning*, 2017.

Hikaru Asano, Tadashi Kozuno, Kuniaki Saito, and Yukino Baba. Where-to-unmask: Ground-truth-guided unmasking order learning for masked diffusion language models, 2026. URL `https://arxiv.org/abs/2602.09501`.

Robert J. N. Baldock, Hartmut Maennel, and Behnam Neyshabur. Deep learning through the lens of example difficulty. In *Advances in Neural Information Processing Systems*, 2021.

François Charton. Linear algebra with transformers. *Transactions on Machine Learning Research*, 2022.

François Charton. Learning the greatest common divisor: explaining transformer predictions. In *International Conference on Learning Representations*, 2024.

François Charton. Int2int: a framework for mathematics with transformers. *arXiv preprint arXiv:2502.17513*, 2025.

Thomas Erneux. *Applied Delay Differential Equations*, volume 3 of *Surveys and Tutorials in the Applied Mathematical Sciences*. Springer, 2009.

Mahsa Forouzesh and Patrick Thiran. Differences between hard and noisy-labeled samples: An empirical study. In *Proceedings of the 2024 SIAM International Conference on Data Mining (SDM)*, pp. 91–99, 2024.

Michael Hahn and Mark Rofin. Why are sensitive functions hard for transformers? In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Samy Jelassi, Stéphane d'Ascoli, Carles Domingo-Enrich, Yuhuai Wu, Yuanzhi Li, and François Charton. Length generalization in arithmetic transformers, 2023.

Hiroshi Kera, Yuki Ishihara, Yuta Kambe, Tristan Vaccon, and Kazuhiro Yokoyama. Learning to compute gröbner bases. In *Advances in Neural Information Processing Systems*, 2024.

Hiroshi Kera, Nico Pelleriti, Yuki Ishihara, Max Zimmer, and Sebastian Pokutta. Computational algebra with attention: Transformer oracles for border basis algorithms. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=bA9rhgWOHk.

Juno Kim and Taiji Suzuki. Transformers provably solve parity efficiently with chain of thought. In *International Conference on Learning Representations*, 2025.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020.

Cathy Yuanchen Li, Jana Sotáková, Emily Wenger, Mohamed Malhou, Evrard Garcelon, François Charton, and Kristin Lauter. Salsapicante: A machine learning attack on lwe with binary secrets. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2023a.

Cathy Yuanchen Li, Emily Wenger, Zeyuan Allen-Zhu, Francois Charton, and Kristin E. Lauter. Salsa verde: a machine learning attack on learning with errors with sparse small secrets. In *Advances in Neural Information Processing Systems*, 2023b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, and Tom Goldstein. Transformers can do arithmetic with the right embeddings. In *Advances in Neural Information Processing Systems*, 2024.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI*, 2019.

Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *International Conference on Machine Learning*, 2017.

Ruoqi Shen, Sébastien Bubeck, Ronen Eldan, Yin Tat Lee, Yuanzhi Li, and Yi Zhang. Positional description matters for transformers arithmetic. *arXiv:2311.14737*, 2023.

Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A. Smith, and Yejin Choi. Dataset cartography: Mapping and diagnosing datasets with training dynamics. In *Proceedings of EMNLP*, 2020.

Sean Welleck, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. Non-monotonic sequential text generation. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6716–6726. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/welleck19a.html.

Emily Wenger, Mingjie Chen, François Charton, and Kristin E. Lauter. SALSA: attacking lattice cryptography with transformers. In *Advances in Neural Information Processing Systems*, 2022.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua M. Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? A study in length generalization. In *International Conference on Learning Representations*, 2024.

A Details on order-sensitive tasks

A.1 Order-sensitive tasks

Let Σ be the set of tokens. As defined in Sec. 3, we consider a task to be a deterministic mapping $F : \Sigma^* \to \Sigma^L$. For an input sequence $X \in \Sigma^*$, the target output is $Y = F(X) = [y_1, \ldots, y_L] \in \Sigma^L$. We denote the set of all permutations over indices $\{1, \ldots, L\}$ by S_L. For a fixed permutation $\pi \in S_L$, the reordered target sequence is denoted by $Y^{(\pi)} := [y_{\pi(1)}, \ldots, y_{\pi(L)}]$.

To quantify the computational complexity of generating Y under a specific order, let Ω be a fixed set of unit-cost primitive operators, including arithmetic operations, comparisons, and value-dependent selections. Each primitive acts on at most b scalar values, where b is independent of L. Reading or copying a scalar is free.

Autoregressive Context Constraint. We define a sequential generation algorithm respecting order π as a tuple $A_\pi = (\phi_1, \ldots, \phi_L)$, where each ϕ_t produces the next token $y_{\pi(t)}$ from the information available at autoregressive decoding step t. Specifically, for every $t \in \{1, \ldots, L\}$, the output must be computed from the input X and the previously generated tokens $Y^{(\pi)}_{<t} := [y_{\pi(1)}, \ldots, y_{\pi(t-1)}]$:

$$y_{\pi(t)} = \phi_t \left(X, Y^{(\pi)}_{<t} \right). \tag{4}$$

Each ϕ_t may use temporary storage, but starts its computation from X and the emitted token values alone. Computed summaries of X, intermediate states, and branch histories are not retained across output steps. Input and output tokens are supplied as individual scalars. This idealized model isolates the computational reuse provided by the output sequence.

The operation count of such an algorithm, denoted by $\mathrm{Ops}(A_\pi; X)$, is the total number of operators from Ω invoked across all functions $\phi_1, \ldots, \phi_L$. The minimal sequential cost is defined as:

$$\mathcal{C}_\pi(X) := \min_{A_\pi} \mathrm{Ops}(A_\pi; X), \tag{5}$$

where the minimization is over algorithms A_π satisfying the autoregressive context constraint and computing the exact output for every input in the support of $\mathcal{D}$.

Order-Sensitivity. We characterize the difficulty of a task based on the disparity in computational cost between different generation orders.

Definition A.1 (Order sensitivity)**.** Let $\mathcal{D}$ be a distribution over inputs X. We define the *order-sensitivity ratio* $\rho(L)$ for a task F as the ratio of expected minimal costs between the worst-case order σ and the best-case order π:

$$\rho(L) := \sup_{\pi, \sigma \in S_L} \frac{\mathbb{E}_{X \sim \mathcal{D}}[\mathcal{C}_\sigma(X)]}{\mathbb{E}_{X \sim \mathcal{D}}[\mathcal{C}_\pi(X)]}. \tag{6}$$

We classify the task F as **order-sensitive** if this ratio grows asymptotically with the sequence length L, i.e.,

$$\lim_{L \to \infty} \rho(L) = \infty. \tag{7}$$

A.2 Order sensitivity under recurrence evaluation

The forward–reverse gap in Tab. 1 motivates the following additional assumption for the six recurrence tasks, $y_1 = g(x_1)$ and $y_i = f(x_i, y_{i-1})$.

Assumption A.2 (Recurrence evaluation in reverse order)**.** Admissible reverse-order algorithms compute each y_i afresh only from $x_1, \ldots, x_i$ by evaluating its defining recurrence from the initial state. With fixed task parameters, evaluating g and f costs c_g and $c_f > 0$ operations, respectively, independently of L.

Write $\mathcal{C}^{\mathrm{rec}}_\pi$ and $\rho_{\mathrm{rec}}(L)$ for the minimal costs and order-sensitivity ratio of Def. A.1 when reverse-order algorithms are additionally restricted by Assumption A.2.

Proposition A.1 (Six recurrence tasks under the additional assumption)**.** Under Assumption A.2, the ReLU, Square, Triangle, MLP, Sine, and Cubic tasks are order-sensitive, with $\rho_{\mathrm{rec}}(L) = \Omega(L)$.

Proof. Let π_{fwd} and π_{rev} denote the forward and reverse orders. Forward generation can reuse each preceding state as an emitted token, so

$$\mathcal{C}^{\mathrm{rec}}_{\pi_{\mathrm{fwd}}}(X) \le c_g + (L-1)c_f = O(L). \tag{8}$$

Under Assumption A.2, producing y_i in reverse order requires one initialization and $i-1$ transitions. Thus

$$\mathcal{C}^{\mathrm{rec}}_{\pi_{\mathrm{rev}}}(X) = \sum_{i=1}^{L}\big(c_g + (i-1)c_f\big) = \Theta(L^2). \tag{9}$$

These bounds hold for every input and therefore also in expectation. Hence $\rho_{\mathrm{rec}}(L) \ge \mathbb{E}[\mathcal{C}^{\mathrm{rec}}_{\pi_{\mathrm{rev}}}]/\mathbb{E}[\mathcal{C}^{\mathrm{rec}}_{\pi_{\mathrm{fwd}}}] = \Omega(L) \to \infty$. □

A.3 Dataset examples

We present representative samples for the six recurrence tasks in Sec. 3.2 and the PROD task. Table 9 summarizes the correspondence between the input X and the target Y, where Y is shown in forward order for the recurrence tasks and in reverse-digit order for PROD. Note that for the PROD task, the input consists of two integers a and b. For the MLP task, we use

$$\mathrm{MLP}(u,v) = W_2\ \mathrm{ReLU}\big(W_1[u,v]^\top + b_1\big) + b_2, \tag{10}$$

where $W_1 \in \mathbb{R}^{h\times 2}$, $b_1 \in \mathbb{R}^h$, $W_2 \in \mathbb{R}^{1\times h}$, and $b_2 \in \mathbb{R}$ are initialized once and fixed throughout dataset generation. We use Kaiming uniform initialization (He et al., 2015) for W_1, Xavier uniform initialization for W_2, and independent $\mathrm{Uniform}(-1,1)$ initialization for both bias vectors.

Table 9: Representative input–output samples for each task

Task	Input	Target
RELU, $L=50$	$X=$ (4, −7, −7, −3, 8, 1, −8, −9, 8, 6 0, −9, 5, −9, 6, 5, −5, −9, 7, −5 8, −6, −7, −2, −7, 6, 7, −2, 0, −6 −3, −8, −7, −8, 3, −1, −6, 1, −4, −9 2, −7, 1, 4, 9, −5, 6, 2, 3, −3)	$Y=$ (4, 0, 0, 0, 8, 9, 1, 0, 8, 14 14, 5, 10, 1, 7, 12, 7, 0, 7, 2 10, 4, 0, 0, 0, 6, 13, 11, 11, 5 2, 0, 0, 0, 3, 2, 0, 1, 0, 0 2, 0, 1, 5, 14, 9, 15, 17, 20, 17)
SQUARE, $L=50$	$X=$ (−5, −9, 8, 7, 8, −7, 5, −9, −6, 9 −2, −8, 6, −7, 2, −7, −6, −5, −5, 7 3, 6, −9, 1, 7, 0, −7, 7, −5, 0 −2, 6, −1, −9, −6, −7, 0, 2, 7, −1 1, −2, −6, −7, 5, 1, 9, −6, −3, −3)	$Y=$ (−5, 2, 2, 6, −4, −1, −2, 0, 8, 3 4, −5, −5, 8, 2, 6, 6, −5, 3, −8 7, 0, −4, 8, 9, −4, −1, 3, 6, 8 2, −7, 3, 5, −5, 8, −2, −1, 3, 1 −7, 6, 6, 0, −3, 1, −3, −2, 4, −3)
TRIANGLE, $L=50$	$X=$ (1, 17, 11, 18, 18, 5, 10, 12, 20, 11 9, 12, 6, 11, 3, 13, 20, 12, 2, 16 11, 12, 12, 9, 0, 11, 19, 3, 14, 7 5, 10, 19, 14, 19, 8, 6, 17, 2, 20 16, 19, 18, 1, 2, 13, 20, 19, 1, 0)	$Y=$ (19, 16, 7, 5, 3, 12, 2, 6, 6, 3 8, 0, 14, 5, 12, 5, 5, 3, 15, 11 2, 6, 2, 9, 11, 2, 1, 16, 10, 3 12, 2, 1, 5, 4, 8, 6, 3, 15, 15 11, 10, 8, 11, 7, 0, 0, 1, 18, 2)
MLP, $L=50$	$X=$ (1, 0, 16, 12, 12, 10, 2, 30, 24, 7 22, 10, 9, 9, 7, 3, 14, 1, 1, 27 6, 8, 19, 14, 27, 10, 10, 13, 21, 29 9, 11, 23, 13, 8, 9, 4, 9, 3, 2 27, 26, 25, 30, 24, 28, 3, 28, 17, 7)	$Y=$ (1, 30, 16, 21, 19, 21, 0, 21, 14, 24 14, 22, 23, 23, 26, 0, 26, 2, 29, 11 25, 25, 15, 20, 13, 22, 22, 19, 16, 12 23, 20, 15, 21, 24, 23, 30, 24, 0, 30 11, 17, 14, 13, 15, 13, 29, 11, 20, 25)
SINE, $L=50$	$X=$ (9, 1, 28, 31, 28, 27, 4, 20, 13, 25 21, 5, 27, 10, 22, 28, 4, 15, 5, 1 5, 2, 29, 29, 11, 9, 7, 17, 7, 30 11, 24, 24, 20, 5, 1, 13, 21, 10, 27 17, 3, 27, 16, 22, 31, 11, 15, 28, 3)	$Y=$ (5, 5, 31, 31, 5, 31, 0, 31, 26, 26 0, 0, 0, 31, 31, 5, 5, 31, 5, 5 31, 31, 31, 31, 31, 31, 0, 0, 0, 31 31, 0, 26, 5, 31, 31, 26, 0, 31, 0 0, 31, 0, 5, 0, 26, 0, 0, 0, 31)
CUBIC, $L=50$	$X=$ (12, 10, 18, 15, 15, 7, 6, 5, 8, 18 13, 3, 12, 11, 0, 18, 18, 13, 3, 5 12, 1, 14, 16, 6, 12, 16, 8, 1, 10 12, 11, 2, 4, 11, 11, 5, 4, 5, 4 18, 4, 11, 5, 0, 0, 13, 11, 6, 1)	$Y=$ (12, 9, 6, 3, 4, 14, 14, 13, 1, 0 13, 15, 5, 3, 8, 17, 10, 6, 10, 17 4, 8, 13, 9, 13, 5, 8, 7, 2, 18 11, 12, 1, 5, 3, 0, 5, 15, 17, 15 11, 5, 3, 13, 12, 18, 12, 10, 18, 0)
PROD, $L=10$	$a=$ (0, 0, 2, 0, 3) $b=$ (0, 2, 6, 3, 7)	$Y=$ (1, 1, 3, 5, 3, 5, 0, 0, 0, 0)

A.4 SUCCESS RATES AT L=20 AND L=50

Table 1 in the main text reports the success rates at L=20. Table 10 extends the comparison to L=50 and shows the same trend across all six tasks: the forward order remains near 100%, while the reverse order collapses to single-digit accuracy or below. The forward-order success rates are slightly higher than at L=20 in some tasks because longer sequences provide more training tokens per sample under the same dataset size.

Table 10: Success rates (%) of Transformers trained with a fixed target order (forward or reverse), evaluated on validation sets at both L=20 and L=50. Across tasks and lengths, the forward order is substantially more learning-friendly. The L=20 rows are duplicated from Tab. 1 for completeness.

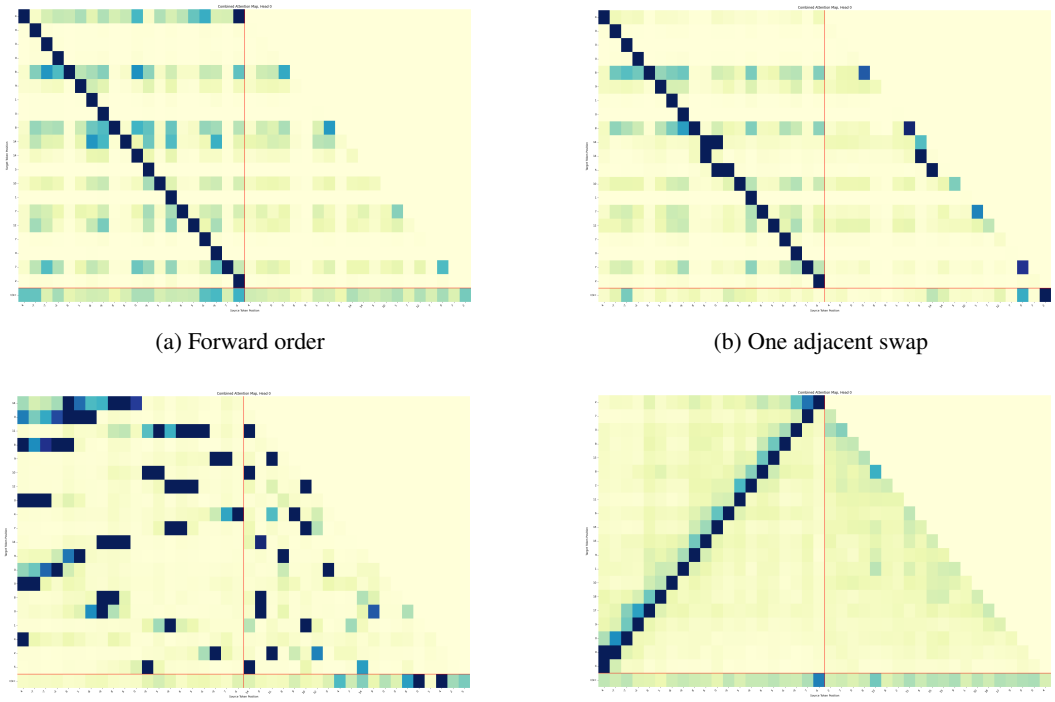
Task	**Recurrence** ($i \geq 2$)	**Length**	**Success Rate (%)**	
			Forward	**Reverse**
RELU	$y_i = \mathrm{ReLU}(x_i + y_{i-1})$	$L = 20$	**99.6**	5.6
		$L = 50$	**99.9**	0.6
SQUARE	$y_i = ((x_i^2 + y_{i-1}^2) \bmod N) - \lfloor N/2 \rfloor$	$L = 20$	**100**	0.1
		$L = 50$	**100**	0.0
TRIANGLE	$y_i = \big\lvert((y_{i-1} + x_i) \bmod 2N) - N\big\rvert$	$L = 20$	**100**	0.2
		$L = 50$	**100**	0.0
MLP	$y_i = \mathrm{round}\big(\mathrm{MLP}(x_i, y_{i-1})\big) \bmod N$	$L = 20$	**100**	9.4
		$L = 50$	**100**	0.6
SINE	$y_i = \big\lfloor A \sin\big(\frac{2\pi}{P}(y_{i-1} + x_i)\big)\big\rfloor \bmod N$	$L = 20$	**100**	6.2
		$L = 50$	**100**	0.3
CUBIC	$y_i = (y_{i-1}^3 + x_i) \bmod N$	$L = 20$	**100**	1.6
		$L = 50$	**100**	0.0

A.5 ATTENTION MAPS

We present the attention maps obtained when training a Transformer on our proposed RELU task with datasets reordered in different ways. For this analysis, we use a GPT-2 model with a single layer and a single attention head. Figure 4 shows the attention maps for target length $L = 20$ under four target orders. The forward and reverse orders are defined in Sec. 3.2. The one-permuted order swaps exactly one pair of adjacent target tokens, whereas the random order is a random permutation of the forward sequence. Figure 5 illustrates how the attention maps change as the target length increases.

Table 11: Attention sparsity S across target orders. A smaller value of S indicates greater sparsity.

Task	**Target length**	**Sparsity**	
		Forward	**Reverse**
RELU	$L = 20$	**1.160**	1.640
	$L = 50$	**1.462**	4.319
	$L = 100$	**1.687**	3.195
SQUARE	$L = 20$	**1.117**	1.531
	$L = 50$	**1.773**	1.914
	$L = 100$	**1.407**	1.990



(a) Forward order

(b) One adjacent swap

(c) Random order

(d) Reverse order

Figure 4: Attention matrices from models trained with four different target orders in the RELU task.

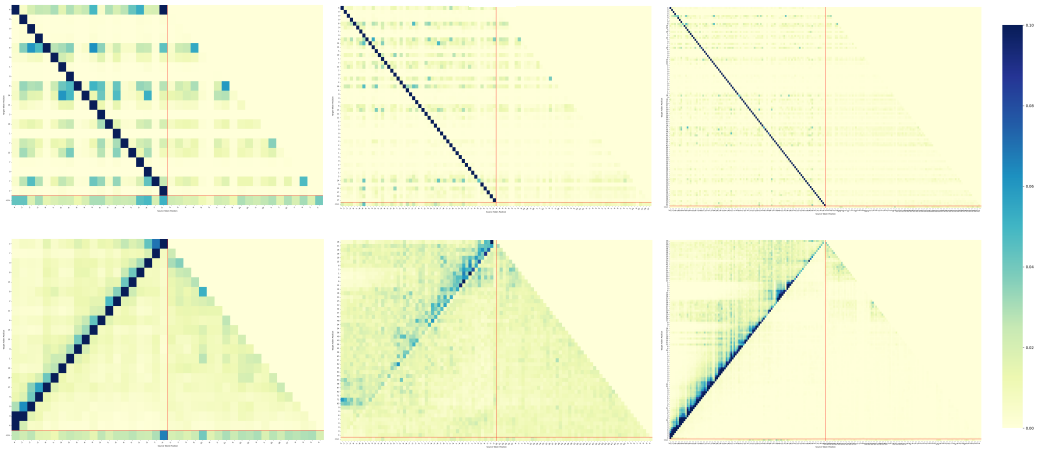


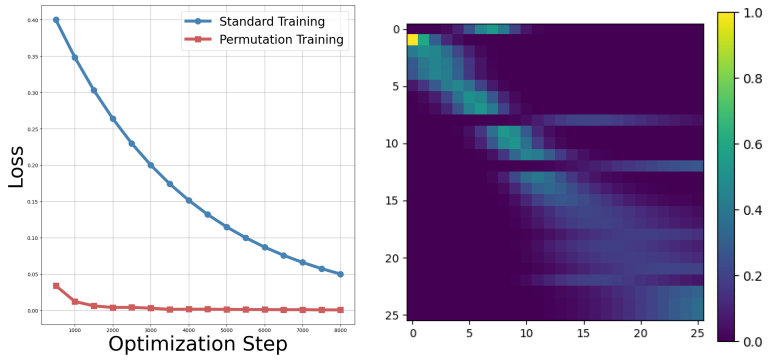
Figure 5: Differences in the attention matrices for the RELU task between forward and reverse orderings. The top three matrices correspond to models trained with forward order, and the bottom three with reverse order. Each pair of matrices shows results for input lengths $n = 20$, 50, and 100, respectively.

B ALTERNATIVE APPROACHES TO PERMUTATION SEARCH

B.1 SOFT-PERMUTATION OPTIMIZATION VIA ATTENTION SPARSITY

B.1.1 ANALYSIS OF INFORMATION LEAKAGE IN NAIVE SOFT-PERMUTATION OPTIMIZATION

We address the fundamental limitation of optimizing a soft permutation matrix $\tilde{P} \in [0, 1]^{L \times L}$ jointly with the model parameters. We formulate the naive optimization objective as the minimization of the



(a) Eval loss curve (b) Optimized Permutation

Figure 6: Failure mode of naive soft-permutation optimization. (a) Training loss drops immediately due to leakage. (b) The learned soft permutation has off-diagonal weights near the main diagonal, indicating access to future tokens.

empirical risk:

$$\min_{\theta,\tilde{P}} \frac{1}{m}\sum_{i=1}^{m} \ell(\mathcal{T}_\theta, X_i, Y_i\tilde{P}). \tag{11}$$

However, as shown in Fig. 6, such an approach leads to an immediate loss drop at the early stage of training. This creates a trivial solution where the soft permutation $\tilde{P}$ causes information leakage from future tokens. Specifically, each token in the reordered target sequence $Y_i\tilde{P}$ becomes a soft mixture of all the tokens in the original sequence Y_i. In an autoregressive setting, this mixture allows the model to access the ground truth of the next token (which is contained in the future positions of Y_i) through the non-zero off-diagonal entries of $\tilde{P}$, thereby undermining the next-token prediction task.

Figure 6(b) visualizes the learned soft permutation matrix obtained from this naive training. The matrix exhibits sparse off-diagonal weights clustered around the main diagonal. These weights indicate that the optimization process finds shortcuts to "peek" at the immediate future tokens rather than discovering a meaningful global causal ordering. This phenomenon motivates the need for alternative approaches, such as attention sparsity regularization or the proposed discrete search pipeline, to prevent such leakage.

B.1.2 Analysis of attention sparsity.

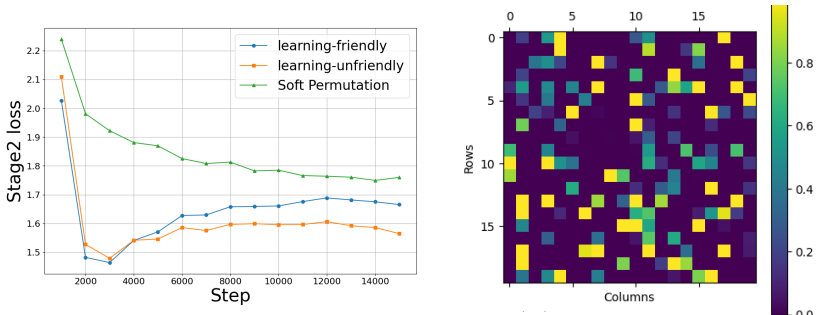
To address the challenges of permutation optimization, we undertake a more detailed analysis of the attention mechanism. Intuitively, when the target order is learning-unfriendly, the local causal structure of the sequence is broken, meaning that more input and output tokens become relevant to predicting the next token. Conversely, for a learning-friendly order, we expect the attention map to be *sparser*, as the model can rely on specific local dependencies.

Let the query and key matrices be $Q, K \in \mathbb{R}^{L' \times d_{\text{emb}}}$, where L' is the decoder-input length and d_{emb} is the embedding dimension. The self-attention weights are given by

$$A = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_{\text{emb}}}}\right) \in \mathbb{R}^{L' \times L'}, \tag{12}$$

where $\text{Softmax}(\cdot)$ is applied row-wise. Because each row of $A = (a_{ij})_{ij}$ is a probability vector, we define the mean sparsity S using the Shannon entropy:

$$S = -\frac{1}{L'}\sum_{i=1}^{L'}\sum_{j=1}^{L'} a_{ij}\log a_{ij}. \tag{13}$$



(a) stage2 loss (b) permutation matrix

Figure 7: (a) Comparison of Stage 2 loss under a fixed learning-friendly order, a fixed learning-unfriendly order, and a learned soft permutation. (b) shows a visualization of the learned soft permutation.

We compute S for models trained on both the forward (learning-friendly) and reverse (learning-unfriendly) orders of the recurrence tasks (Sec. 3.2). Table 11 shows that the forward order consistently yields lower S. Since a smaller S directly corresponds to lower entropy (i.e., higher sparsity), this confirms that learning-friendly orders produce sparser attention maps. Representative heat maps are provided in Sec. A.5.

Because S is derived solely from the learned attention weights, it is independent of the language-model loss and can serve as an orthogonal diagnostic metric. We explored optimizing permutations using an additional sparsity regularizer that explicitly rewards low-entropy attention. However, even with this bias, the optimizer failed to discover the learning-friendly order and instead converged to interleaved permutations, suggesting that sparsity guidance alone is insufficient to solve the permutation search in difficult regimes.

Specifically, we formulated a soft-permutation optimization method based on attention sparsity. In our two-stage strategy, we first optimize the Transformer parameters θ by minimizing the standard sequence-modeling loss over the training set:

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \ell\big(\mathcal{T}_\theta,\, X_i,\, Y_i\big). \tag{14}$$

Next, denoting by $A(\tilde{P}) = [a_{ij}]$ the attention map produced when the target sequence is fed as $Y_i\tilde{P}$ into the Transformer, we optimize the soft permutation $\tilde{P}$ by minimizing the total attention entropy:

$$\min_{\tilde{P}}\; S(A(\tilde{P})) = \min_{\tilde{P}} \left(-\frac{1}{L'} \sum_{i=1}^{L'} \sum_{j=1}^{L'} a_{ij} \log a_{ij} \right). \tag{15}$$

In our experiments, we alternated between these two optimization stages. Figure 7 compares the Stage 2 loss (entropy) under three conditions: the fixed learning-friendly order, the fixed learning-unfriendly (reverse) order, and the learned soft permutation. We observe that the soft permutation optimization does not significantly reduce the entropy-based loss relative to the fixed orders, nor does it converge to a genuinely structured ordering. This indicates that attention sparsity, while a useful diagnostic metric, is difficult to use as a direct, reliable objective for permutation optimization due to optimization landscapes or trivial solutions.

B.2 Permutation search via evolutionary strategy

This section summarizes the evolutionary strategy (ES) baseline that we ran in parallel with our proposed method to search the permutation space. Each individual is a permutation P; its fitness is the (negative) early-stage training loss of a Transformer trained with that order, so that permutations that are easier to learn receive higher scores. The ES is controlled by the population size N_p, crossover probability N_c, mutation probability N_m, number of generations N_g, tournament size N_t, and elitism ratio N_r, and proceeds as follows:

Table 12: Success rate obtained when the Transformer is retrained on the permutations discovered by the ES.

Task	Input length	ES-discovered order	Success rate (%)	
			Retrain	Reverse
RELU	$L = 5$	[2, 1, 0, 4, 3]	26.9	10.4
	$L = 10$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	100	3.5
	$L = 20$	[6, 7, 9, 8, 12, 11, 13, 18, 17, 14, 16, 15, 19, 5, 10, 1, 0, 3, 4, 2]	9.2	0.7
SQUARE	$L = 5$	[1, 2, 3, 4, 0]	100	21.5
	$L = 10$	[3, 4, 5, 6, 7, 8, 9, 1, 0, 2]	99.9	6.5
	$L = 20$	[9, 10, 11, 12, 13, 14, 2, 3, 4, 5, 6, 15, 16, 17, 18, 19, 0, 1, 7, 8]	5.2	0.1

(1) **Population initialization:** sample N_p random permutations.

(2) **Selection:** pick parents via tournament selection with size N_t.

(3) **Crossover:** with probability N_c, apply partially mapped crossover to each selected pair.

(4) **Mutation:** with probability N_m, swap two positions in the offspring permutation.

(5) **Elitism:** evaluate every individual by

$$\text{fitness}(P) = -\frac{1}{m'} \sum_{i=1}^{m'} \ell\big(\mathcal{T}_\theta,\, X_i,\, Y_i P\big),$$

and copy the top N_r fraction to the next generation.

(6) **Termination:** stop when N_g generations have been processed.

Table 12 lists the permutation identified by the evolution strategy (ES) and the performance obtained when the model is retrained using that order.

C VISUALIZING THE GLOBAL–LOCAL PIPELINE

Figure 8 illustrates the hierarchical exploration procedure described in Sec. 4.

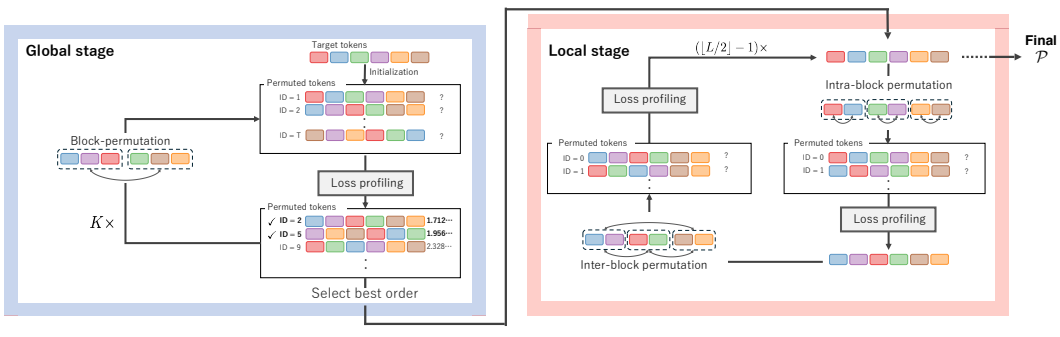


Figure 8: Overview of the exploration. **Global stage** (left panel) generates T candidate permutations by swapping the sequence at the macro-level, exchanging token blocks to quickly spot coarse, learning-friendly orders. **Local stage** (right panel) further permutes the tokens for each target block, refining the sequence to discover a final permutation that maximizes learning ease.

D EXPERIMENTAL SETTINGS AND CANDIDATE POOLS

D.1 EXPERIMENTAL PARAMETERS AND SETTINGS

We summarize the specific parameters used for the recurrence tasks and the model training in Tabs. 13 and 14, respectively.

For RELU, $y_1 = \max(0, x_1)$; for SQUARE, $y_1 = x_1$ and $y_i = ((x_i^2 + y_{i-1}^2) \bmod 19) - 9$ for $i \geq 2$. For TRIANGLE, $y_0 = 0$ and the recurrence applies from $i = 1$, giving $y_1 = N - x_1$ for $x_1 \in \{0, \ldots, N\}$. For MLP, $y_1 = x_1$; round rounds to the nearest integer, with ties to even. For SINE, we define $y_0 = 0$ and apply $y_i = \lfloor A \sin(2\pi(y_{i-1} + x_i)/P) \rfloor \bmod N$ from $i = 1$, with $A = 100$, $P = 10$, and $N = 32$.

In addition to the parameters listed in the tables, the following configurations apply to the experiments in Sec. 5, unless stated otherwise. Optimization is performed using AdamW (Loshchilov & Hutter, 2019) with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We employ a linearly decaying learning rate schedule starting from 5.0×10^{-5}. The dropout rate is set to 0.1, and positional embeddings are randomly initialized and optimized during training.

Table 13: Parameters for recurrence tasks

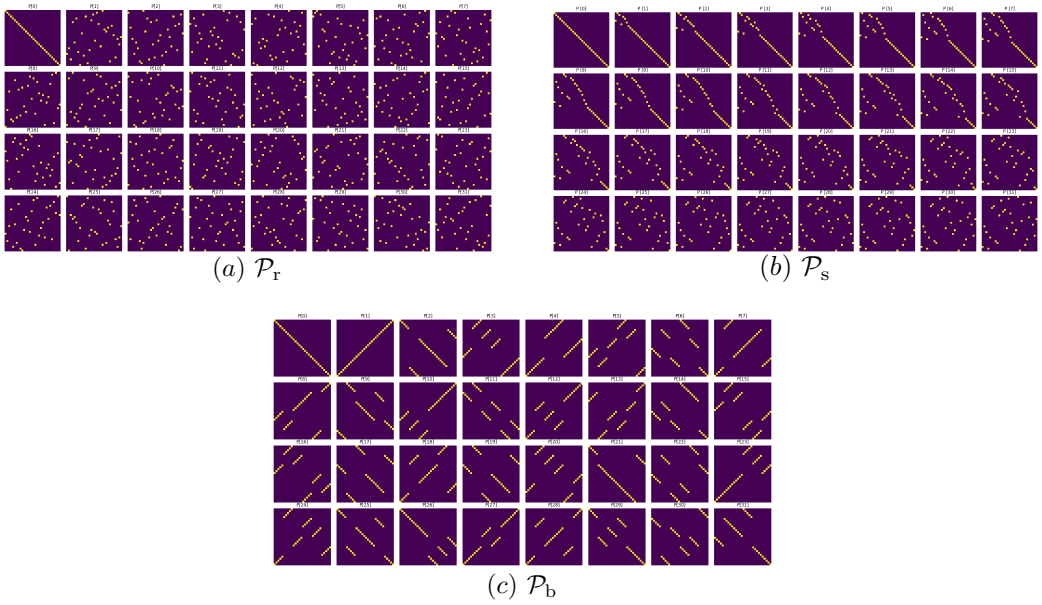
Task	**Modulus (N)**	**Input Range (x_i)**	**Other Constants**
RELU	N/A	$\{-9, \ldots, 9\}$	—
SQUARE	19	$\{-9, \ldots, 9\}$	—
CUBIC	19	$\{0, \ldots, 18\}$	—
TRIANGLE	20	$\{0, \ldots, 20\}$	Amplitude $N = 20$
SINE	32	$\{0, \ldots, 31\}$	$A = 100, P = 10$
MLP	31	$\{0, \ldots, 30\}$	$d_{\text{hidden}} = 64$

Table 14: Transformer training configuration.

Hyperparameter	**Phase**		
	Exploration (Small)	**Final training (Large)**	**Final training (PROD)**
Layers (n_{layer})	$1 \sim 4$	6	12
Attention heads (n_{head})	$1 \sim 4$	8	12
Embedding dim (d_{emb})	$128 \sim 512$	512	768
FFN dim (d_{ffn})	$512 \sim 2048$	2048	3072
Training samples	10^5	10^5	10^6
Batch size	$32 \sim 128$	128	128
Training epochs	1	10	30

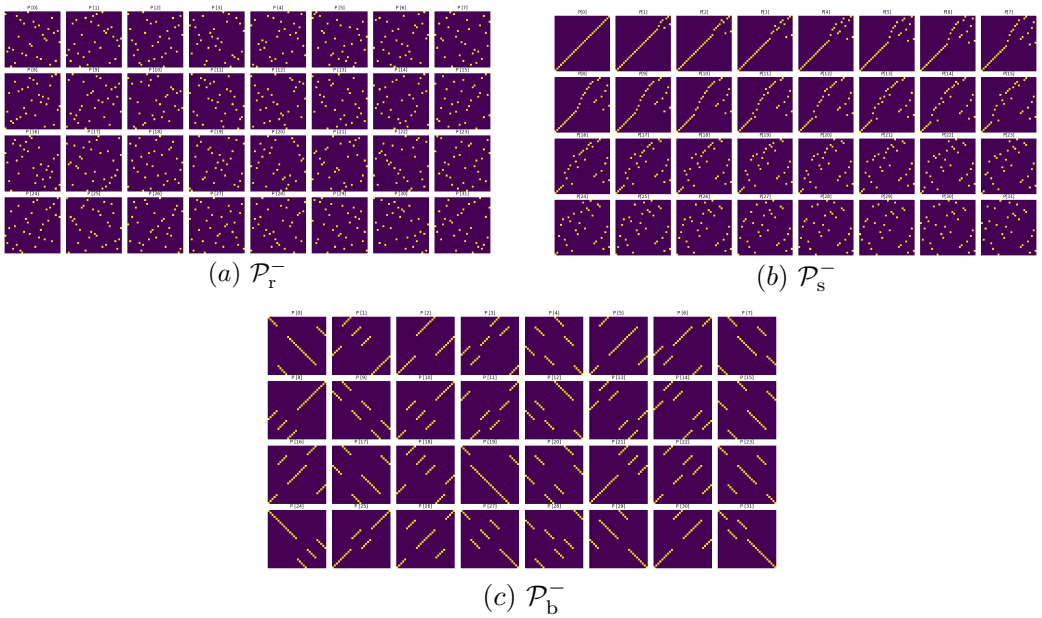
D.2 PERMUTATION SET VISUALIZATION

Next, we visualize the three permutation sets ($\mathcal{P}_{\text{r}}$, $\mathcal{P}_{\text{s}}$, $\mathcal{P}_{\text{b}}$) introduced in Sec. 5. Figure 9 displays the sets used in the loss profiling experiments (Sec. 5.1), which include the forward order (Ground Truth) at ID=0. Figure 10 illustrates the sets used in the global–local search experiments (Sec. 5.2), which do not include the forward order in the initial candidate pool.



(a) $\mathcal{P}_{\mathrm{r}}$ (b) $\mathcal{P}_{\mathrm{s}}$ (c) $\mathcal{P}_{\mathrm{b}}$

Figure 9: Visualization of the elements in the three permutation sets used in Sec. 5.1. ID=0 corresponds to the forward (learning-friendly) order.



(a) $\mathcal{P}_{\mathrm{r}}^{-}$ (b) $\mathcal{P}_{\mathrm{s}}^{-}$ (c) $\mathcal{P}_{\mathrm{b}}^{-}$

Figure 10: Visualization of the elements in the permutation sets used in Sec. 5.2. Note that the forward order is not included in the initial candidates.

E Loss profiling: detailed results and analyses

E.1 Forward-order identification

In this section, we report the full results of the experiments conducted in Sec. 5.1. Please refer to Sec. 5 and Sec. 5.1 for the experimental setup and methodology. Tables 15 to 20 present the success rates of loss profiling on the proposed tasks. In these tables, "Target Length" denotes the target

sequence length, and "Permutation" indicates the specific permutation set used in the experiment. "Top-k Acc" indicates the proportion of trials where the optimal forward order is found within the top k ranked candidates. "Med. Rank" represents the median rank of the forward order among the 128 candidates based on the validation loss.

Table 15: Performance evaluation on the RELU task.

Target Length	Permutation	Top-1 Acc	Top-5 Acc	Top-10 Acc	Med. Rank
$L = 30$	$\mathcal{P}_{\mathrm{r}}$	85.19%	92.59%	96.30%	1.00
	$\mathcal{P}_{\mathrm{b}}$	77.78%	96.30%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	96.30%	100.00%	100.00%	1.00
$L = 50$	$\mathcal{P}_{\mathrm{r}}$	100.00%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{b}}$	81.48%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	62.96%	77.78%	100.00%	1.00
$L \in \{20, 21 \ldots, 30\}$	$\mathcal{P}_{\mathrm{r}}$	62.96%	81.48%	88.89%	1.00
	$\mathcal{P}_{\mathrm{b}}$	77.78%	85.19%	88.89%	1.00
	$\mathcal{P}_{\mathrm{s}}$	70.37%	96.30%	100.00%	1.00
$L \in \{40, 41 \ldots, 50\}$	$\mathcal{P}_{\mathrm{r}}$	96.30%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{b}}$	66.67%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	66.67%	85.19%	96.30%	1.00

Table 16: Performance evaluation on the SQUARE task.

Target Length	Permutation	Top-1 Acc	Top-5 Acc	Top-10 Acc	Med. Rank
$L = 30$	$\mathcal{P}_{\mathrm{r}}$	77.78%	85.19%	85.19%	4.00
	$\mathcal{P}_{\mathrm{b}}$	92.31%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	55.56%	66.67%	100.00%	1.00
$L = 50$	$\mathcal{P}_{\mathrm{r}}$	96.30%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{b}}$	62.96%	66.67%	66.67%	1.00
	$\mathcal{P}_{\mathrm{s}}$	33.33%	40.74%	96.30%	6.00
$L \in \{20, 21 \ldots, 30\}$	$\mathcal{P}_{\mathrm{r}}$	70.37%	92.59%	96.30%	1.00
	$\mathcal{P}_{\mathrm{b}}$	96.30%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	11.11%	11.11%	81.48%	9.00
$L \in \{40, 41 \ldots, 50\}$	$\mathcal{P}_{\mathrm{r}}$	100.00%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{b}}$	66.67%	66.67%	66.67%	1.00
	$\mathcal{P}_{\mathrm{s}}$	33.33%	44.44%	92.59%	6.00

Table 17: Performance evaluation on the SINE task.

Target Length	Permutation	Top-1 Acc	Top-5 Acc	Top-10 Acc	Med. Rank
$L = 30$	$\mathcal{P}_{\mathrm{r}}$	37.04%	44.44%	48.15%	14.00
	$\mathcal{P}_{\mathrm{b}}$	88.89%	92.59%	96.30%	1.00
	$\mathcal{P}_{\mathrm{s}}$	11.11%	66.67%	70.37%	3.00
$L = 50$	$\mathcal{P}_{\mathrm{r}}$	55.56%	77.78%	77.78%	1.00
	$\mathcal{P}_{\mathrm{b}}$	33.33%	51.85%	55.56%	2.00
	$\mathcal{P}_{\mathrm{s}}$	59.26%	59.26%	59.26%	1.00
$L \in \{20, 21 \ldots, 30\}$	$\mathcal{P}_{\mathrm{r}}$	18.52%	25.93%	44.44%	11.00
	$\mathcal{P}_{\mathrm{b}}$	96.30%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	0.00%	18.52%	66.67%	7.00
$L \in \{40, 41 \ldots, 50\}$	$\mathcal{P}_{\mathrm{r}}$	62.96%	66.67%	77.78%	1.00
	$\mathcal{P}_{\mathrm{b}}$	48.15%	66.67%	70.37%	2.00
	$\mathcal{P}_{\mathrm{s}}$	48.15%	55.56%	55.56%	2.00

Table 18: Performance evaluation on the CUBIC task.

Target Length	Permutation	Top-1 Acc	Top-5 Acc	Top-10 Acc	Med. Rank
$L = 30$	$\mathcal{P}_{\mathrm{r}}$	18.52%	22.22%	22.22%	83.00
	$\mathcal{P}_{\mathrm{b}}$	55.56%	85.19%	85.19%	1.00
	$\mathcal{P}_{\mathrm{s}}$	0.00%	0.00%	59.26%	8.00
$L = 50$	$\mathcal{P}_{\mathrm{r}}$	22.22%	44.44%	59.26%	8.00
	$\mathcal{P}_{\mathrm{b}}$	55.56%	66.67%	66.67%	1.00
	$\mathcal{P}_{\mathrm{s}}$	0.00%	0.00%	11.11%	25.00
$L \in \{20, 21 \ldots, 30\}$	$\mathcal{P}_{\mathrm{r}}$	3.70%	7.41%	11.11%	55.00
	$\mathcal{P}_{\mathrm{b}}$	74.07%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	0.00%	0.00%	48.15%	15.00
$L \in \{40, 41 \ldots, 50\}$	$\mathcal{P}_{\mathrm{r}}$	0.00%	22.22%	25.93%	39.00
	$\mathcal{P}_{\mathrm{b}}$	74.07%	77.78%	77.78%	1.00
	$\mathcal{P}_{\mathrm{s}}$	0.00%	0.00%	0.00%	31.00

Table 19: Performance evaluation on the MLP task.

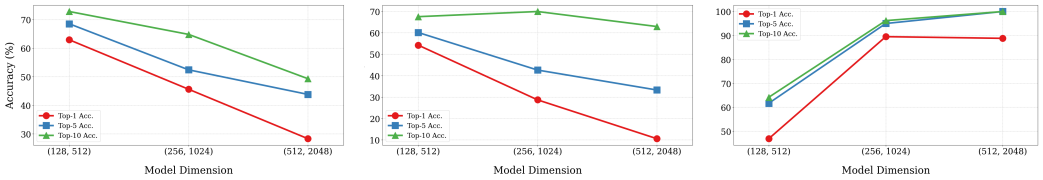
Target Length	Permutation	Top-1 Acc	Top-5 Acc	Top-10 Acc	Med. Rank
$L = 30$	$\mathcal{P}_{\mathrm{r}}$	22.22%	25.93%	37.04%	26.00
	$\mathcal{P}_{\mathrm{b}}$	100.00%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	0.00%	18.52%	85.19%	7.00
$L = 50$	$\mathcal{P}_{\mathrm{r}}$	85.19%	96.30%	96.30%	1.00
	$\mathcal{P}_{\mathrm{b}}$	88.89%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	0.00%	0.00%	0.00%	25.00
$L \in \{20, 21 \ldots, 30\}$	$\mathcal{P}_{\mathrm{r}}$	18.52%	25.93%	40.74%	13.00
	$\mathcal{P}_{\mathrm{b}}$	100.00%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	25.93%	96.30%	100.00%	2.00
$L \in \{40, 41 \ldots, 50\}$	$\mathcal{P}_{\mathrm{r}}$	18.52%	29.63%	51.85%	10.00
	$\mathcal{P}_{\mathrm{b}}$	66.67%	85.19%	88.89%	1.00
	$\mathcal{P}_{\mathrm{s}}$	37.04%	37.04%	37.04%	15.00

Table 20: Performance evaluation on the TRIANGLE task.

Target Length	Permutation	Top-1 Acc	Top-5 Acc	Top-10 Acc	Med. Rank
$L = 30$	$\mathcal{P}_{\mathrm{r}}$	48.15%	51.85%	51.85%	2.00
	$\mathcal{P}_{\mathrm{b}}$	96.30%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	59.26%	85.19%	88.89%	1.00
$L = 50$	$\mathcal{P}_{\mathrm{r}}$	96.30%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{b}}$	66.67%	66.67%	66.67%	1.00
	$\mathcal{P}_{\mathrm{s}}$	37.04%	37.04%	37.04%	12.00
$L \in \{20, 21 \ldots, 30\}$	$\mathcal{P}_{\mathrm{r}}$	48.15%	62.96%	85.19%	2.00
	$\mathcal{P}_{\mathrm{b}}$	100.00%	100.00%	100.00%	1.00
	$\mathcal{P}_{\mathrm{s}}$	7.41%	59.26%	85.19%	4.00
$L \in \{40, 41, \ldots, 50\}$	$\mathcal{P}_{\mathrm{r}}$	70.37%	85.19%	92.59%	1.00
	$\mathcal{P}_{\mathrm{b}}$	74.07%	81.48%	88.89%	1.00
	$\mathcal{P}_{\mathrm{s}}$	33.33%	33.33%	37.04%	13.00

E.2 Identification accuracy across model dimensions

Figure 11 presents the impact of model dimensions $(d_{\text{emb}}, d_{\text{ffn}})$ on the top-1 identification accuracy of loss profiling, complementing the brief summary in Sec. 5.1. For $\mathcal{P}_{\text{r}}$ and $\mathcal{P}_{\text{s}}$, smaller model dimensions yield higher identification accuracy, suggesting that models with limited expressivity are more sensitive to the difficulty of the order and adapt more easily to learning-friendly sequences. Conversely, for $\mathcal{P}_{\text{b}}$, higher model dimensions correlate with better identification accuracy: when candidates are locally similar (as in block permutations), a larger capacity is required to distinguish global structural differences.



(a) $\mathcal{P}_{\text{r}}$ (b) $\mathcal{P}_{\text{s}}$ (c) $\mathcal{P}_{\text{b}}$

Figure 11: Identification accuracy of the forward order across various model dimensions, defined by pairs of $(d_{\text{emb}}, d_{\text{ffn}})$. Subplots (a), (b), and (c) show the results for the initialization sets $\mathcal{P}_{\text{r}}$, $\mathcal{P}_{\text{s}}$, and $\mathcal{P}_{\text{b}}$, respectively.

E.3 Correlation analysis

The loss-profiling results in Sec. E evaluate whether the known forward order is ranked near the top. Here, we additionally ask whether the *entire* profiling ranking is predictive of the accuracy obtained after retraining with each candidate order fixed. For each task and target length, we fix the retraining accuracies from a reference run and pair them with the profiling loss of the same permutation index. We analyze CUBIC, MLP, and RELU at length $L = 30$. The profiling side contains the same 27 GPT-2 configurations used in Sec. 5.1: $n_{\text{layer}} \in \{1, 2, 4\}$, $n_{\text{head}} \in \{1, 2, 4\}$, and $(d_{\text{emb}}, d_{\text{ffn}}) \in \{(128, 512), (256, 1024), (512, 2048)\}$.

We report four metrics. Spearman correlation is computed between the profiling rank and the retraining-accuracy rank, so larger values indicate stronger agreement. Pearson correlation is computed between the raw profiling loss and the raw test accuracy; a useful profiling signal should produce a negative value because lower profiling loss should correspond to higher accuracy. Top-5 overlap is the fraction of candidates shared by the profiling top-5 and the retraining-accuracy top-5. Mean absolute rank gap is the average absolute difference between the two ranks, where smaller values are better.

Table 21: Correlation between loss-profiling rankings and retraining accuracy rankings at $L = 30$.

Task	Runs	Spearman	Pearson	Top-5 overlap	Rank gap
RELU	27	0.9241	-0.8090	0.6889	10.0877
MLP	27	0.8977	-0.8058	0.5852	12.4363
CUBIC	27	0.1999	-0.0928	0.4370	31.1681

Table 22: Correlation metrics grouped by profiling-model embedding dimension. Each row aggregates all three tasks and the corresponding profiling configurations.

Dim.	Runs	Spearman	Pearson	Top-5 overlap	Rank gap
$D = 128$	27	0.6094	-0.5281	0.2815	18.8854
$D = 256$	27	0.7060	-0.5852	0.7111	17.3938
$D = 512$	27	0.7064	-0.5944	0.7185	17.4129

Table 21 shows that the profiling ranking strongly agrees with the retraining-accuracy ranking on RELU and MLP. CUBIC is the main exception: the correlation is weak, consistent with the difficulty

of obtaining high retraining accuracy on this task even under favorable orders. Thus, loss profiling is best interpreted as a screening signal for candidate orders rather than a perfect predictor of final accuracy. Table 22 further shows that $D = 256$ and $D = 512$ yield substantially higher top-5 overlap than $D = 128$, suggesting that moderately larger profiling models can be useful when the goal is to select a small pool of promising orders for retraining.

E.4 Profiling budget and pool size

We additionally analyze a fully matched pool of 128 orders for ReLU at $L = 30$ from the sort family $\mathcal{P}_{\mathrm{s}}$ (forward included), in which every profiled permutation has a corresponding final retraining accuracy. The profiling model is a 1-layer, 1-head GPT-2 with embedding dimension 128.

Budget. Evaluating all 128 candidates at $50/100/200/400/782$ optimizer steps along one profiling run, the order selected at one epoch is already ranked first at step 100 (0.13 epoch) and never changes. The step-100 ranking has Spearman correlation $\rho = 0.99$ with the one-epoch ranking, and its selected order reaches 99.9% after full retraining. Even the step-50 selection reaches 99.8% (regret 0.1 point).

Fewer candidates. From the same pool we sampled 100 random subsets at each size 16, 32, and 64. When the full-pool best order was included, profiling selected it in all 100 subsets at each size. Without forcing its inclusion, the selected orders achieved mean final accuracies of $85.6/92.7/99.1\%$. These subsets reuse the scores from the same profiling run; they do not represent independent training runs with smaller candidate pools.

F Global–local exploration: detailed results and controls

This section supplements Sec. 5.2 with local-stage ablations, reverse-order baselines, discovered orders, no-selection controls, seed variability, and runtime analysis.

F.1 Ablation of the local stage

To quantify the contribution of the local stage, we compare the order obtained after the global stage alone with the order obtained after running the local stage as well. For each discovered order, we retrain the model with that order fixed and report the resulting sequence-level exact-match accuracy. Table 23 shows that the local stage can substantially improve performance, especially on ReLU and on the $L = 10$ Square and Triangle settings. The effect is task-dependent: on Cubic, and on Square at $L = 13$, the local stage does not improve the final accuracy, and the final accuracy remains low. This suggests that local refinement is useful when the global stage has found a promising coarse structure, but it does not by itself solve settings where the overall search remains difficult.

Table 23: Ablation of the local stage across all tasks with available $L = 10$ and $L = 13$ results. Accuracy is measured after retraining with the order discovered by either the global stage alone or the full global–local pipeline. The change columns report Global+Local minus Global-only in percentage points.

Task	$L = 10$			$L = 13$		
	Global	**Global+Local**	**Change**	**Global**	**Global+Local**	**Change**
ReLU	30.9	99.6	+68.7	6.6	99.2	+92.6
Square	49.7	100.0	+50.3	8.1	5.6	-2.5
Triangle	11.4	100.0	+88.6	2.9	7.4	+4.5
MLP	22.1	38.2	+16.1	3.1	10.8	+7.7
Sine	37.9	54.2	+16.3	23.2	26.3	+3.1
Cubic	6.4	0.3	-6.1	0.7	0.2	-0.5
Prod	100.0	100.0	+0.0	—	—	—

Table 24 gives the $L = 10$ cases where both the global-stage order and the final local-stage order are available. On ReLU, the local stage turns a coarse, mostly reversed block structure into the forward order. On Square, the local stage recovers the forward order; on Prod, it preserves the reverse-digit

order. On MLP, it produces a different partially structured order and yields a smaller accuracy gain. On CUBIC, the local stage moves to the reverse order, matching the accuracy degradation in Tab. 23. Tables 25 and 26 further show the longer structured-initialization cases. These longer runs do not include a local stage, but the global stage often recovers a coarse order close to the forward sequence under $\mathcal{P}_{\mathrm{b}}^{-}$ or $\mathcal{P}_{\mathrm{s}}^{-}$.

Table 24: Discovered orders before and after local refinement at $L = 10$ with $\mathcal{P}_{\mathrm{r}}^{-}$ initialization. Orders use zero-based indices relative to the forward sequence for recurrence tasks and the reverse-digit sequence for PROD.

Task	Initialization	Target Length	Global-stage order	After local stage	Accuracy change
RELU	$\mathcal{P}_{\mathrm{r}}^{-}$	$L = 10$	[6, 7, 8, 9, 5, 4, 2, 3, 1, 0]	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	$30.9 \rightarrow 99.6$
SQUARE	$\mathcal{P}_{\mathrm{r}}^{-}$	$L = 10$	[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	$49.7 \rightarrow 100.0$
MLP	$\mathcal{P}_{\mathrm{r}}^{-}$	$L = 10$	[9, 8, 5, 3, 2, 1, 0, 4, 6, 7]	[0, 1, 3, 2, 4, 9, 8, 7, 6, 5]	$22.1 \rightarrow 38.2$
CUBIC	$\mathcal{P}_{\mathrm{r}}^{-}$	$L = 10$	[0, 1, 5, 6, 7, 8, 9, 2, 3, 4]	[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]	$6.4 \rightarrow 0.3$
PROD	$\mathcal{P}_{\mathrm{r}}^{-}$	$L = 10$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	$100.0 \rightarrow 100.0$

Table 25: Global-stage orders at $L = 20$ from structured initializations $\mathcal{P}_{\mathrm{s}}^{-}$ and $\mathcal{P}_{\mathrm{b}}^{-}$. The local stage was not run in these longer settings.

Task	Initialization	Target Length	Global-stage order
RELU	$\mathcal{P}_{\mathrm{s}}^{-}$	$L = 20$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
RELU	$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 20$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
MLP	$\mathcal{P}_{\mathrm{s}}^{-}$	$L = 20$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 19, 18, 17, 16, 15]
MLP	$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 20$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
CUBIC	$\mathcal{P}_{\mathrm{s}}^{-}$	$L = 20$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
CUBIC	$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 20$	[19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Table 26: Global-stage orders at $L = 40$ from structured initialization $\mathcal{P}_{\mathrm{b}}^{-}$. The local stage was not run in these longer settings.

Task	Initialization	Target Length	Global-stage order
RELU	$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 40$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
SQUARE	$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 40$	[39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
TRIANGLE	$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 40$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20]
MLP	$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 40$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
SINE	$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 40$	[39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
CUBIC	$\mathcal{P}_{\mathrm{b}}^{-}$	$L = 40$	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20]

F.2 REVERSE-ORDER BASELINES

Table 27: Success rates of final training when the target sequence is fixed to the reverse order, evaluated at the same lengths as the discovery experiments in Tab. 3. Accuracy is generally low, although some shorter-sequence settings retain non-negligible success rates.

Target Length	RELU	SQUARE	TRIANGLE	MLP	SINE	CUBIC
$L = 10$	2.7	6.5	3.5	10.4	21.8	0.6
$L = 13$	1.9	3.2	3.7	8.2	15.2	1.4
$L = 20$	5.6	0.1	0.2	9.4	6.2	1.6
$L = 30$	1.1	0.1	0.2	3.8	1.1	0.0
$L = 40$	0.9	0.0	0.0	0.8	0.7	0.0

F.3 DISCOVERED ORDERS

Table 28 lists the orders found at each stage.

Table 28: The orders discovered by the proposed method in its global and local stages. Depth denotes the hierarchy level K reached in the global stage. Orders use zero-based indices relative to the forward sequence for recurrence tasks and the reverse-digit sequence for PROD. Bold marks recovery of these respective reference orders.

Task	**Target Length**	**Depth**	**Order after global stage**	**Discovered final order**
RELU	$L = 7$	$K = 4$	$[6, 0, 5, 2, 3, 4, 1]$	$[2, 3, 4, 5, 0, 6, 1]$
	$L = 8$	$K = 4$	$[0, 2, 1, 3, 4, 5, 6, 7]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}]$
	$L = 9$	$K = 5$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}]$
	$L = 10$	$K = 6$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}]$
	$L = 11$	$K = 6$	$[8, 9, 10, 7, 6, 5, 4, 3, 2, 1, 0]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}, \mathbf{10}]$
	$L = 12$	$K = 6$	$[6, 7, 8, 9, 10, 11, 5, 4, 2, 3, 1, 0]$	$[1, 2, 3, 4, 0, 5, 6, 7, 8, 9, 10, 11]$
	$L = 13$	$K = 6$	$[11, 12, 10, 9, 8, 7, 6, 5, 4, 2, 3, 1, 0]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}, \mathbf{10}, \mathbf{11}, \mathbf{12}]$
SQUARE	$L = 7$	$K = 4$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}]$
	$L = 8$	$K = 4$	$[1, 2, 4, 5, 0, 6, 7, 3]$	$[1, 2, 4, 5, 0, 6, 7, 3]$
	$L = 9$	$K = 5$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}]$
	$L = 10$	$K = 6$	$[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}]$
	$L = 11$	$K = 6$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}, \mathbf{10}]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}, \mathbf{10}]$
	$L = 12$	$K = 6$	$[1, 2, 3, 4, 5, 6, 7, 11, 10, 9, 0, 8]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}, \mathbf{10}, \mathbf{11}]$
	$L = 13$	$K = 6$	$[0, 1, 2, 3, 12, 11, 10, 4, 5, 6, 7, 8, 9]$	$[8, 9, 0, 1, 2, 3, 4, 10, 11, 12, 5, 6, 7]$
PROD	$L = 10$	$K = 6$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}]$	$[\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}]$

F.4 STRUCTURED POOLS WITHOUT SELECTION

Table 4 reports no-selection controls for RELU and SQUARE at $L \in \{30, 40\}$. Before observing accuracy, we fix candidate IDs 0–31 from each structured family $\mathcal{P}_{\mathrm{s}}^{-}$ and $\mathcal{P}_{\mathrm{b}}^{-}$. Each order is trained independently for 10 epochs with the full 6-layer, 8-head GPT-2 of embedding dimension 512, using training seed 42 and the same sequence-level exact-match evaluation as the main experiment. The 32 candidates measure variation across orders, not across training seeds.

None of the 256 structured-order runs summarized in the main table reaches the 90% threshold. The largest observed accuracies are 86.1% for RELU at $L = 30$ and 64.6% at $L = 40$, both from $\mathcal{P}_{\mathrm{b}}^{-}$. Architecture-matched forward controls reach 100% in all four task–length settings. We also ran the same fixed candidates with the lightweight profiling models; these controls similarly do not yield a 90% candidate. Because this experiment fixes the number of candidates rather than matching every search configuration by measured runtime, we use it as a no-selection control rather than an exact equal-compute comparison.

F.5 SEED VARIABILITY

We evaluate all six tasks under $\mathcal{P}_{\mathrm{r}}^{-}$ at $L = 10$ and $\mathcal{P}_{\mathrm{s}}^{-}/\mathcal{P}_{\mathrm{b}}^{-}$ at $L = 20$, using training/search seeds $42, 43, 44$. These 18 settings cover the first row of each initialization in Tab. 3. We hold the dataset splits and initial candidate pools fixed across seeds. All six tasks use datasets generated from

the definitions in Sec. D: 100,000 training, 1,000 validation, and 1,000 test examples, with split-generation seeds $42, 84, 123$, respectively. The MLP recurrence uses one fixed network, initialized with seed 42, across all splits.

For each setting and seed, we run four exploration architectures with $n_{\text{layer}} \in \{1, 2\}$ and $n_{\text{head}} \in \{1, 2\}$. Model dimensions and search depths follow Sec. 5.2: $K = 6$ with local refinement for $\mathcal{P}_{\text{r}}^{-}$ at $L = 10$, and $K = 7$ without local refinement for the two structured initializations at $L = 20$. Each discovered order is trained for 10 epochs with the six-layer, eight-head final model in Sec. 5. We use AdamW with learning rate 5×10^{-5}, batch size 128, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, zero weight decay, gradient clipping at 1, and linear learning-rate decay without warmup. We select among the four final models by validation exact-match accuracy, with ties resolved by validation loss and then configuration ID. Test data are not used for selection.

Table 29: Test success rates (%, mean±sample SD over three seeds) with fixed data and candidate pools and validation-based model selection.

Task	$\mathcal{P}_{\text{r}}^{-}, L = 10$	$\mathcal{P}_{\text{s}}^{-}, L = 20$	$\mathcal{P}_{\text{b}}^{-}, L = 20$
RELU	100.0 ± 0.0	99.9 ± 0.0	90.8 ± 7.9
SQUARE	18.5 ± 5.2	2.7 ± 0.7	55.6 ± 4.9
TRIANGLE	86.0 ± 12.1	3.5 ± 0.9	33.4 ± 11.3
MLP	94.2 ± 1.3	80.2 ± 0.8	96.7 ± 0.8
SINE	64.2 ± 31.0	16.7 ± 5.4	47.4 ± 8.4
CUBIC	54.5 ± 40.8	1.3 ± 0.6	46.4 ± 1.3

Table 29 shows that seed variability depends on the task and initialization, with particularly large SDs for SINE and CUBIC under $\mathcal{P}_{\text{r}}^{-}$.

F.6 RUNTIME ANALYSIS

Table 30 compares exploration and final-training times. Exploration is shorter than one full training run for PROD, but can exceed it for the recurrence tasks.

Number of local training runs. The local implementation profiles one block at a time and updates the current permutation before proceeding to the next block. For block length l, let $q_l = \lfloor L/l \rfloor$ and $r_l = L \bmod l$. There are q_l full blocks and, if $r_l > 0$, one remainder block. Each block of length at least two requires one profiling training run; singleton blocks are skipped. One additional run profiles the block-reordering candidates after the within-block updates. Including one initial training run to score the starting permutation, the number of local training runs in a completed search is

$$N_{\text{train}}^{\text{local}}(L) = 1 + \sum_{l=2}^{\lfloor L/2 \rfloor} (q_l + \mathbf{1}\{r_l \geq 2\} + 1) = O(L \log L). \tag{16}$$

The bound follows from $\sum_{l=2}^{\lfloor L/2 \rfloor} \lfloor L/l \rfloor = O(L \log L)$; the remaining terms contribute $O(L)$. For $L = 10$, the counts for $l = 2, 3, 4, 5$ are $6, 4, 4, 3$, respectively, giving 18 runs including the initial run. For $L = 13$, the corresponding counts are $7, 5, 4, 4, 3$, giving 24 runs in total. These counts apply to one local search with one model configuration and seed, excluding retries and final retraining. They do not bound total runtime: with exhaustive enumeration, each block length also entails evaluating $q_l l! + \mathbf{1}\{r_l \geq 2\} r_l!$ within-block candidates and $\lceil L/l \rceil !$ block-reordering candidates.

Table 30: Detailed runtime analysis including total exploration time and the proportion of time spent on model training within the exploration phase. Exploration/final is total exploration time divided by final-training time (in multiples of one final-training run). For the six recurrence tasks, representative timings are measured on RELU, since runtime is almost identical across tasks under the shared training setup. For RELU, Total spans the start of Global through the end of the last exploration stage, whereas Global and Local are subprocess wall times. Their residual is orchestration time for reading the Global result, constructing the Local configuration, and launching the Local process; without Local, Total equals Global. Times are rounded to the nearest second.

Search Params	Exploration Time [sec]			Training Ratio	Final training	Exploration/
	Total	Global	Local	in Exploration (%)	[sec]	final
Prod Task						
$K = 6, L = 10$	11,729	7,001	4,448	67.6	28,213	0.42×
$K = 6, L = 12$	13,436	7,301	5,935	67.9	30,695	0.44×
$K = 7, L = 20$	35,981	35,981	—	11.7	38,369	0.94×
Other (Order-sensitive) Tasks						
$K = 6, L = 10$	2,823	2,331	492	13.7	423	6.67×
$K = 6, L = 13$	7,366	3,485	3,881	7.8	645	11.42×
$K = 7, L = 20$	127	127	—	53.6	792	0.16×
$K = 7, L = 30$	2,200	2,200	—	6.2	1,371	1.60×
$K = 7, L = 40$	18,358	18,358	—	0.9	1,326	13.84×

G DELAY DYNAMICAL SYSTEMS

In Sec. 6 we evaluated the loss-profiling pipeline on six delay network variants. This appendix gives the full transition rules for each variant and the candidate-order definitions.

G.1 TRANSITION RULES

Original RELU.

$$a(t+2) = \mathrm{ReLU}(b(t+1) + d(t)), \quad b(t+2) = a(t+1) - c(t+1),$$
$$c(t+2) = b(t) + d(t+1), \quad d(t+2) = \mathrm{ReLU}(a(t+1) + c(t)).$$

HIDDEN-CAUSE. With $h(1) = h(2) = 0$,

$$h(t+2) = \mathrm{ReLU}(a(t+1) - d(t)), \quad a(t+2) = \mathrm{ReLU}(b(t+1) + h(t+1)),$$
$$b(t+2) = a(t+1) - c(t+1), \quad c(t+2) = b(t) + d(t+1),$$
$$d(t+2) = \mathrm{ReLU}(h(t+2) + c(t)).$$

The latent variable h is not exposed to the model; only (a, b, c, d) appear in the target.

COUPLED RING.

$$a(t+2) = \mathrm{ReLU}(b(t+1) - d(t)), \quad b(t+2) = \mathrm{ReLU}(c(t+1) - a(t)),$$
$$c(t+2) = \mathrm{ReLU}(d(t+1) - b(t)), \quad d(t+2) = \mathrm{ReLU}(a(t+1) - c(t)).$$

SINE. Define $\mathrm{sine}_{32}(z) = \big(\lfloor 15 \sin(2\pi z/10) \rfloor + 16\big) \bmod 32$. Then

$$a(t+2) = \mathrm{sine}_{32}(b(t+1) + d(t)), \quad b(t+2) = \mathrm{sine}_{32}(a(t+1) - c(t+1)),$$
$$c(t+2) = \mathrm{sine}_{32}(b(t) + d(t+1)), \quad d(t+2) = \mathrm{sine}_{32}(a(t+1) + c(t)).$$

MODULAR. With all operations taken modulo 19,

$$a(t+2) = (b(t+1) + d(t)) \bmod 19, \quad b(t+2) = (a(t+1) - c(t+1)) \bmod 19,$$
$$c(t+2) = (b(t) \cdot d(t+1)) \bmod 19, \quad d(t+2) = (a(t+1) + c(t)) \bmod 19.$$

MIXED NONLINEAR. Define $\mathrm{sq}_{19}(x) = (x^2 \bmod 19) - 9$ and $\mathrm{mod}_{19}(x) = x \bmod 19$. Then

$$\begin{aligned}
a(t+2) &= \mathrm{ReLU}\big(b(t+1) - d(t) + \mathrm{sq}_{19}(c(t))\big),\\
b(t+2) &= \mathrm{mod}_{19}\big(a(t+1)^2 + c(t+1) - d(t)\big),\\
c(t+2) &= \mathrm{sq}_{19}(b(t) + d(t+1)) - \mathrm{ReLU}(a(t)),\\
d(t+2) &= \mathrm{ReLU}\big(a(t+1) + c(t) - \mathrm{mod}_{19}(b(t+1))\big).
\end{aligned}$$

G.2 COMMON PROTOCOL

For all variants, the state at time t is $s(t) = (a(t), b(t), c(t), d(t))$, the input is $(s(1), s(2))$, and the target is the time-major flattening of $s(3), \ldots, s(T)$, which has length $4(T-2)$. The eight initial-state components are sampled independently and uniformly from a task-specific integer domain: $\{-25, \ldots, 25\}$ for RELU, HIDDEN-CAUSE, COUPLED RING, and MIXED NONLINEAR; $\{0, \ldots, 31\}$ for SINE; and $\{0, \ldots, 18\}$ for MODULAR. The comparisons in Sec. 6 use $100{,}000/2{,}000/1{,}000$ training/validation/test samples generated with seeds $42/84/1337$, respectively; all model-training seeds share these fixed splits. Inputs and targets are tokenized as space-separated integers; loss is computed only on the target tokens and the end-of-sequence token. Loss profiling uses a 1-layer GPT-2 with embedding dimension 128 and one attention head, trained for one epoch with batch size 32 and learning rate 5×10^{-4}, applying each candidate permutation in a per-sample round-robin schedule during training and ranking candidates on the validation split. Retraining uses a 6-layer GPT-2 with the same embedding dimension and one head, trained for 10 epochs with batch size 128 and the same learning rate, with the target order fixed throughout.

G.3 CANDIDATE ORDERS

Each variant induces its own dependency graph, so the actual permutations associated with a candidate name are task-specific. The five candidate names are kept consistent across variants:

- `time_first`: the natural forward flattening $(a(3), b(3), c(3), d(3), \ldots, a(T), b(T), c(T), d(T))$.
- `variable_first_topological`: a topological order of the per-task dependency graph with tie-breaking $a > b > c > d$. Components from different time steps may be interleaved when the dependency graph permits.
- `dependency_sorted`: keeps the time-major structure but reorders within each time step according to a task-specific within-time order (e.g., (b, a, d, c) for the variants used here).
- `causal_depth`: keeps the time-major structure but uses a depth-based within-time order ((a, d, b, c) for the variants used here).
- `reverse`: complete reversal of the target sequence.

The shorter names `var_first_topo` and `dep_sorted` in Tab. 7 denote `variable_first_topological` and `dependency_sorted`, respectively.

G.4 MULTI-HEAD COMPARISON AND RANK CORRELATION

The comparison in Tab. 7 uses $T = 11$ and training seeds $42, 1, 2$ for both profiling and final training. Every final model has six layers, embedding dimension 128, FFN dimension 512, and one attention head and is trained for 10 epochs (7,820 updates) with batch size 128 and learning rate 5×10^{-4}. For MH, four channel-specific integer embeddings are summed into one state embedding at each time step. A causal decoder-only Transformer processes these state embeddings along the time axis, and four output heads predict the next state's components simultaneously. Each output head is tied to its corresponding input embedding; during free generation, all four predictions are fed back together. The AR objective additionally predicts an end-of-sequence token, whereas MH uses the known fixed horizon; this token is excluded from the shared exact-match metric over 36 target integers. The comparison matches data presentations and backbone configuration, not parameters or FLOPs. On the four tasks using the fixed 40k integer vocabulary, AR and MH have 6.33M and 21.70M parameters, respectively, because MH has four channel-specific embedding/output matrices. For each seed, Spearman ρ compares validation profiling-loss ranks (lower loss is better) with final-accuracy

ranks (higher accuracy is better) over the five fixed AR orders, including reverse; the table reports mean and sample SD across the three seeds.

G.5 PER-CANDIDATE VALIDATION LOSSES FROM LOSS PROFILING

Table 31 reports the validation losses obtained at the end of the 1-epoch profiling phase (1-layer GPT-2, $d_{\text{emb}} = 128$) for each of the four reported candidate orders. These supplementary single-run measurements are separate from the three-seed results in Tab. 7 and should not be interpreted as their underlying per-seed losses. Two patterns can be read directly from this supplementary run: on MODULAR, all four reported candidates lie within a 0.02-loss band, which is the saturation/tie issue; on MIXED NONLINEAR, the rank-1 (`variable_first_topological`, 2.46) and rank-2 (`dependency_sorted`, 2.48) candidates are nearly indistinguishable at the early stage despite substantially different retrain accuracies (1.4% vs. 64.5%).

Table 31: Validation loss after the 1-epoch profiling run (1-layer GPT-2, $d_{\text{emb}} = 128$, $T = 11$) for each of the four reported candidate orders on each delay network variant. **Bold** marks the rank-1 (lowest-loss) candidate per task in this single run.

Task	`time_first`	`var_first_topo`	`dep_sorted`	`causal_depth`
RELU	3.634	**3.492**	3.507	3.564
HIDDEN-CAUSE	2.706	**2.659**	2.710	2.748
COUPLED RING	**0.842**	1.266	0.879	0.842
SINE	1.441	1.460	**1.320**	1.492
MODULAR	2.843	2.829	**2.824**	2.829
MIXED NONLINEAR	2.479	**2.458**	2.475	2.560

G.6 SCALING WITH THE NUMBER OF TIME STEPS ON THE ORIGINAL RELU DELAY NETWORK

In a separate single-run experiment, we extend the RELU delay network to larger numbers of time steps, $T \in \{11, 13, 15, 17\}$, with corresponding target lengths $4(T-2) \in \{36, 44, 52, 60\}$. The order picked by loss profiling, `variable_first_topological`, retains $\geq 94.5\%$ accuracy throughout, whereas the natural `time_first` order degrades from 73.3% to 0.3% (Tab. 32). Together with the variant comparison in Sec. 6, this indicates that loss profiling identifies a learning-friendly order in delay networks as the prediction horizon T increases.

Table 32: Success rate (%) on the RELU delay network at varying T. The order picked by loss profiling, `variable_first_topological`, remains effective at all tested T values, while `time_first` degrades sharply.

Target order	$T = 11$	$T = 13$	$T = 15$	$T = 17$
`variable_first_topological`	**100.0**	**99.6**	**96.4**	**94.5**
`dependency_sorted`	84.6	75.9	70.5	46.3
`causal_depth`	63.0	65.4	61.7	39.3
`time_first`	73.3	58.5	51.8	0.3

G.7 MULTI-SEED GLOBAL SEARCH

This appendix records the three-seed global-search protocol used in Sec. 6. The two initializations are `topo` (100 random topological sorts) and `no_topo` (all 24 within-time variable orderings). Global search uses a 1-layer, 1-head GPT-2 with embedding dimension 128 and FFN dimension 512, trained for one epoch per level with batch size 32 and learning rate 5×10^{-4}. Final training uses the 6-layer protocol in Sec. G.4. For each variant, initialization, and training seed, K is selected from $\{2, 3, 4, 5\}$ by validation loss, with ties preferring smaller K and then the lower candidate ID. Test data are used only for final evaluation. The candidate pools and data splits are fixed, while model initialization and data order vary over seeds 42, 1, and 2.

Table 33 reports, for each variant and initialization, the selected validation loss, the success rate after fixed-order retraining, and the per-seed best accuracy over the original five fixed AR orders.

Table 33: Detailed three-seed results of the global search on delay network variants with $T = 11$ (mean±sample SD). "Global loss" is the validation loss used to select K. "Discovered" is sequence exact match after retraining the chosen order. "Best of 5" is the per-seed post-hoc maximum over the original five fixed AR orders (including reverse).

Task	Init	Global loss	Discovered (%)	Best of 5 (%)
HIDDEN-CAUSE	`topo`	3.199±0.148	0.9±0.5	96.6±2.3
	`no_topo`	2.793±0.247	29.4±7.7	96.6±2.3
COUPLED RING	`topo`	1.534±0.016	59.7±12.2	99.8±0.1
	`no_topo`	1.081±0.014	99.6±0.4	99.8±0.1
SINE	`topo`	1.628±0.013	99.2±0.5	100.0±0.0
	`no_topo`	1.293±0.004	100.0±0.1	100.0±0.0
MODULAR	`topo`	2.848±0.000	0.0±0.0	100.0±0.0
	`no_topo`	2.819±0.003	0.2±0.3	100.0±0.0
MIXED NONLINEAR	`topo`	2.862±0.036	0.0±0.0	91.0±4.9
	`no_topo`	2.530±0.001	58.6±51.5	91.0±4.9

A few patterns are visible in Tab. 33. First, `no_topo` nearly reaches the post-hoc hand-designed upper bound on COUPLED RING and SINE. Second, both initializations fail on MODULAR, where small early-loss differences do not identify the orders that later reach 100%. Third, MIXED NONLINEAR is strongly seed-sensitive under `no_topo`: the three final accuracies are 98.9%, 76.2%, and 0.6%, despite selected validation losses lying in a narrow range.

The selected depths for seeds $(42, 1, 2)$ are: HIDDEN-CAUSE, $(4, 4, 4)$ for `topo` and $(3, 3, 3)$ for `no_topo`; COUPLED RING, $(4, 4, 4)$ and $(2, 2, 2)$; SINE, $(2, 4, 2)$ and $(3, 3, 3)$; MODULAR, $(2, 4, 2)$ and $(3, 3, 3)$; and MIXED NONLINEAR, $(4, 4, 4)$ and $(2, 2, 3)$.

H RANDOM ARITHMETIC DAG

The delay-system experiments already provide a practical setting in which a natural, computationally admissible order need not be learning-friendly. To complement that case with a fully synthetic task whose good order is not forward or reverse, we constructed a random arithmetic DAG.

Task. The input consists of eight independent variables in $\mathbb{F}_{19}$. Ten output nodes $z_0, \ldots, z_9$ form a fixed random DAG (graph seed 20261113) with six latent layers. Every node applies the same explicit rule $f(a, b) = (ab + a + 2b + 1) \bmod 19$. Node IDs are randomized, so the canonical index order $[0, \ldots, 9]$ is unrelated to the latent computational order. Every permutation defines a valid target sequence, because each node is a function of the original inputs; parent-first generation, however, exposes lower-complexity intermediate computations in the prefix. The profiler and the learner receive only input/output examples and candidate permutations—not DAG edges, layers, or a topological order.

Protocol. We sample 60 unconstrained random permutations (candidate seed 1337), excluding duplicates and the four diagnostic controls (canonical, reverse, and two oracle topological sorts). A shared 2-layer, 4-head GPT-2 (embedding 128) is trained for one epoch on a mixture of these 60 orders, then ranks them by validation loss. Selected orders are retrained with a 6-layer, 8-head GPT-2 (embedding 512) for 10 epochs over three seeds, and evaluated by sequence-level exact match on $2{,}000$ test examples.

Results. The canonical and reverse orders reach only $19.78 \pm 0.14\%$ and $17.72 \pm 2.06\%$, respectively, while an oracle topological order reaches 100%. The order reaching $99.62 \pm 0.40\%$ ranks in the top four under all three profiling seeds $(42/43/44)$, although top-1 selection is seed-sensitive (Tab. 34).

The reported ranks are a post-hoc analysis of an order identified by its final accuracy, not an independently evaluated top-4 selection protocol. Retraining the top four candidates after one

Table 34: Random arithmetic DAG: final exact-match accuracy and profiling ranks for a pool of 60 unconstrained random permutations.

Evaluation	**Final exact match**	**Rank of the 99.62% order**
Canonical order	$19.78 \pm 0.14\%$	—
Reverse order	$17.72 \pm 2.06\%$	—
Profiling seed 42 (top-1)	36.97%	3
Profiling seed 43 (top-1)	99.62%	1
Profiling seed 44 (top-1)	36.97%	4
Oracle topological	100%	—

profiling run is estimated to use about 12–14$\times$ less GPU time than fully training all 60 candidates. We present this as a single illustrative synthetic case, not as average-case performance over random DAGs.