Stabilizing Extrapolation in Looped Transformers via Learned Stochastic Stopping

Hsun-Yu Kuo [1] **El Mahdi Chayti** [*1] **Patrik Reizinger** [*2] **Wieland Brendel** [†2] **Martin Jaggi** [†1]

Abstract

Looped Transformers — which repeatedly apply a shared block — are an architecturally natural fit for variable-length algorithmic tasks. Although they can exhibit strong length generalization beyond the length of training sequences, this behavior is brittle, yielding high out-of-distribution (OOD) variance, even across well-performing in-distribution solutions. We trace this variance to the spurious correlation in simple algorithmic tasks between sequence length and number of loops. Introducing stochasticity into the number of loops during training sharply reduces OOD variance and stabilises predictions across inference-time loop counts. To improve upon heuristic randomisation schemes, we further analyse RL-Halting as a learned stochastic schedule and find it generally improves the accuracy–stability trade-off. We provide comparisons across binary addition, Dyck-1, Unique Set, and Copy, showing that learned stochastic stopping often improves this trade-off but can also stabilise a suboptimal computation. Overall, our results suggest that "when to stop" should be treated as a training-time design choice in looped architectures, not merely as an inference-time compute-allocation rule.

1. Introduction

Transformers often achieve strong in-distribution performance, but can fail sharply when evaluation requires *extrapolation*: inputs may be longer, harder, or more compositional than those seen during training (Reizinger et al., 2024; Abbe et al., 2024). A common example is *length generalisation*, where models are trained on short instances but tested on longer ones (Zhou et al., 2023; Cai et al., 2025; Lee et al., 2025). Algorithmic tasks such as multi-digit addition and planning-style tasks such as maze solving are representative cases: success requires executing a reusable procedure, not merely interpolating between familiar patterns.

A natural way to support such reusable computation is to add *iteration via weight sharing*. Looped Transformers and related recurrent-depth architectures repeatedly apply a shared block to an internal state before producing an answer (Dehghani et al., 2018; Schwarzschild et al., 2021; Bansal et al., 2022; Giannou et al., 2023; Fan et al., 2024; Geiping et al., 2025; Zhu et al., 2025). This gives a simple computational intuition: if longer or harder instances require more repeated computation, then running the same block for more iterations may enable extrapolation. Indeed, prior work has shown that looped models can represent algorithmic computations and can sometimes discover extrapolating solutions (Giannou et al., 2023; Yang et al., 2023; Fan et al., 2024). However, expressivity alone does not answer the training question: even if an extrapolating computation exists in the architecture, gradient-based training may not reliably select it.

We show that this gap between *architectural potential* and *training outcome* is central in practice. On length-generalisation tasks, Looped Transformers can contain computations that solve inputs far beyond the training-length regime, yet this potential is highly unstable across runs. Runs with similar in-distribution accuracy can exhibit qualitatively different out-of-distribution (OOD) behaviour: some extrapolate, while others collapse soon after leaving the training regime. This suggests that looping does not simply add a uniformly helpful inductive bias. Instead, it introduces a training-time *selection problem*: among the many computation trajectories available through repeated application of the same block, training must select one that is both stable and extrapolating.

We study this selection problem through the lens of *stopping schedules*, which determine how many loop iterations are used during training. A deterministic schedule supervises each input at a single prescribed depth, potentially making the learned computation brittle to that choice. A

[*]Equal mentorship contribution. [†]Equal supervision. [1]EPFL, Lausanne, Switzerland [2]Max Planck Institute for Intelligent Systems & ELLIS Institute, Tübingen, Germany. Correspondence to: Hsun-Yu Kuo <hsun-yu.kuo@epfl.ch>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

stochastic schedule instead exposes the model to a local neighbourhood of loop depths, reducing dependence on any one stopping time. Empirically, we find that stochastic schedules substantially reduce run-to-run OOD variability and make predictions more stable across inference-time loop counts. However, stability alone is not sufficient: a schedule can stabilise a computation that still fails to extrapolate. We therefore also study RL-Halting, a learned stochastic stopping schedule related to adaptive-computation methods (Banino et al., 2021; Zhu et al., 2025). RL-Halting often improves the accuracy–stability trade-off across sequence tasks, but it can also stabilise a suboptimal computation.

Overall, our results suggest that "when to stop" is not merely an inference-time compute-allocation question. For looped architectures, it is also a training-time design choice that affects which computation trajectory is learned. **Our contributions** are:

- **The extrapolation–stability gap.** We show that deterministic Looped Transformers can contain extrapolating computations, but that these computations are not reliably selected across runs (§ 4.1).
- **Stabilisation via stochastic schedules.** We show that stochastic stopping schedules reduce run-to-run OOD variability and stabilise predictions across inference-time loop counts (§ 4.3).
- **Learning stochastic stopping schedules.** We study RL-Halting as a learned stochastic schedule, finding that it often improves the accuracy–stability trade-off, while also showing that stable stopping does not always imply extrapolating stopping (§ 4.4).

2. Background

2.1. Looped Transformers

Looped Transformers implement iterative computation by repeatedly applying the same Transformer block across depth (Dehghani et al., 2018; Giannou et al., 2023; Fan et al., 2024; Geiping et al., 2025; Zhu et al., 2025). Rather than stacking K independent blocks with separate parameters, a looped model reuses a single block for multiple iterations. This weight tying keeps the parameter count fixed while allowing the amount of computation to vary with the number of loop steps.

In our experiments, we follow the looped architecture used by Fan et al. (2024). Let x be an input sequence and let $E(x) \in \mathbb{R}^{m\times d}$ denote its token embeddings, where m is the token-sequence length. The model maintains hidden states $H^{(t)} \in \mathbb{R}^{m\times d}$ for loop iteration t, initialised as $H^{(0)} = E(x)$. At each loop iteration, we apply the same Transformer block Block_θ:

$$H^{(t)} = \mathrm{Block}_\theta\left(H^{(t-1)} + H^{(0)}\right), \qquad t = 1, \dots, K.$$

The addition of $H^{(0)}$ implements input injection (Fan et al., 2024; Geiping et al., 2025): the original input representation is reintroduced at each iteration so that the loop can refine its state while retaining direct access to the input tokens.

After K loop iterations, a readout map produces token-level logits from $H^{(K)}$, and training uses standard token-level cross-entropy on the supervised output positions. Crucially, supervision is applied only at the chosen loop depth, rather than at every intermediate iteration. Thus, unlike a standard fixed-depth Transformer, a Looped Transformer requires specifying not only the model parameters θ, but also the loop depth at which the prediction is read out.

2.2. Stopping Schedules as Training-Time Choices

A stopping schedule determines how many loop iterations are used for an input. We write $K \sim \pi_{\mathrm{stop}}(K \mid x)$, where π_{stop} may be deterministic, stochastic, or learned. We use K for a generic loop count, and τ when emphasising a sampled stopping time, as in § 4.4.

For an example (x, y), a stopping schedule induces the objective

$$\mathcal{L}_\pi(\theta; x, y) = \mathbb{E}_{K\sim\pi_{\mathrm{stop}}(\cdot|x)}\ell(\theta; x, y, K),$$

where $\ell(\theta; x, y, K)$ is the prediction loss after K loop iterations. In practice, each optimisation step samples one loop depth per example. For deterministic schedules, this reduces to training at the prescribed depth. For stochastic schedules, the sampled-depth update is an unbiased estimate of the schedule-averaged objective.

This sampled-depth view is important because the stopping schedule affects training, not only inference. By choosing which loop depths receive supervision, the schedule can change which computation trajectory the model learns. Fixed schedules always train at a prescribed depth, length-matched schedules tie the depth to the input length, and adaptive-computation methods learn or infer when to stop (Saunshi et al., 2024; Fan et al., 2024; Bansal et al., 2021; Banino et al., 2021; Zhu et al., 2025). In the experiments, we compare deterministic schedules, hand-designed stochastic schedules, and a learned stochastic stopping rule.

3. Experimental Setup

Unless otherwise stated, our diagnostic experiments use binary addition; cross-task results are reported in § 4.4.

3.1. Tasks and length-generalisation protocol

We evaluate length generalisation under a controlled train–test length shift. Let n denote the problem length. For binary addition, n is the number of digits in each operand; for the other sequence tasks, n is the corresponding input length or problem size. All models are trained on lengths $n < 20$, which we call the in-distribution regime (ID),

and evaluated on lengths $n \geq 20$, which we call OOD. Unless otherwise stated, OOD summaries are computed over lengths $\{20, 25, 30, \ldots, 60\}$.

Our main task is binary addition. Each example contains two n-digit binary numbers, and the target is their sum followed by an end token; for example, $1011 + 0110$ maps to 10001 <eos>. Binary addition admits a linear-time algorithm, but vanilla Transformers struggle to length-generalise on this task without explicit index hints (Zhou et al., 2023; Fan et al., 2024).

We also report results on Dyck-1, Unique Set, and Copy. Dyck-1 tests generalisation in a simple formal-language setting with nested structure. Unique Set maps a sequence to the unique symbols in order of first appearance, e.g. $[a, b, a, c, b] \mapsto [a, b, c, \texttt{<eos>}]$. Copy maps the input sequence to itself followed by an end token. For all tasks, a prediction is correct only if the complete output sequence is correct, including the end token.

3.2. Models and stopping schedules

We compare fixed-depth Transformers with Looped Transformers. The fixed-depth baselines are a 3-layer Transformer, matching the parameter count of one looped block, and a 60-layer Transformer, serving as a higher-compute baseline. The Looped Transformer reuses a shared 3-layer block across loop iterations.

For Looped Transformers, we evaluate three families of stopping schedules.

Deterministic schedules. We consider a fixed schedule, $K = 20$, as in fixed-depth latent-reasoning settings (Saunshi et al., 2024), and a length-matched schedule, $K = n$, where n is the input length (Fan et al., 2024).

Hand-designed stochastic schedules. Starting from the length-matched schedule, we sample $\Delta \sim \mathrm{Unif}\{-w, \ldots, w\}$ and train at $K = \mathrm{clip}(n + \Delta, 1, T_{\max})$. Unless otherwise stated, stochastic schedules sample one loop depth per example per optimisation step.

Learned stochastic schedules. We study RL-Halting as a learned stochastic schedule. It samples a stopping time during training and updates the stopping policy using the realised task loss. The full parameterisation and optimisation rule are introduced in § 4.4.

Additional implementation details are given in Appx. B.

3.3. Evaluation metrics

Our primary diagnostic metric is *oracle-over-iterations accuracy*. For examples of length n, we define

$$\mathrm{Acc}_{\mathrm{oracle}}(n) = \mathbb{E}_{(x,y)\sim\mathcal{D}_n}\left[\max_{K\in\mathcal{K}_{\mathrm{eval}}} \mathrm{acc}(\hat{y}_K(x), y)\right],$$

where $\hat{y}_K(x)$ is the prediction after K loop iterations. For fixed-depth Transformer baselines, which have no loop depth to select, we evaluate the final prediction at the model's fixed depth.

Oracle-over-iterations accuracy is not an inference policy: it uses the ground-truth answer to select the best loop depth. We use it as a diagnostic upper bound on the *extrapolation potential* of a trained looped model, separating whether an extrapolating computation exists at some depth from whether a practical stopping rule can select it. We make this distinction explicit on Unique Set in Appx. D.3.

For summary tables, we report the following metrics. *OOD* is the average oracle accuracy over lengths $\{20, 25, 30, \ldots, 60\}$, and *Near* is the average over near-OOD lengths 20–40. *Max@90* is the largest OOD length at which each run reaches at least 90% accuracy, averaged across runs. *Front.@90* is the mean-accuracy frontier: the largest evaluated OOD length at which the mean accuracy across runs reaches at least 90%. *Std.* is the standard deviation of per-run average OOD accuracy.

For length-wise figures, we additionally show the mean and standard deviation across independent runs at each input length.

4. Experimental Results

4.1. Deterministic Looping Reveals Extrapolation Potential but Is Unstable

We first ask whether deterministic looped models reliably learn computations that extrapolate beyond the training-length regime. This isolates the effect of looping itself before introducing stochastic or learned stopping schedules. The answer is mixed: looped models can contain extrapolating computations, but these computations are not reliably selected across training runs.

4.2. Effect of fixed loop budget

Looping reveals extrapolation potential, but not reliable OOD performance.

Fig. 1 compares fixed-depth Transformers with Looped Transformers on binary addition. The fixed-depth Transformer baselines fit the ID regime but degrade sharply beyond the training-length boundary. Looped Transformers, in contrast, show clear extrapolation potential: for both fixed $K = 20$ and length-matched $K = n$, the best runs remain accurate on substantially longer inputs.

This potential is not reflected in reliable average performance. The individual run curves in Fig. 1 show large OOD variation: some looped runs extrapolate far beyond the training lengths, while others collapse soon after entering the OOD regime. This is also visible quantitatively in Fig. 5: the OOD standard deviation is only 0.009 and 0.005 for

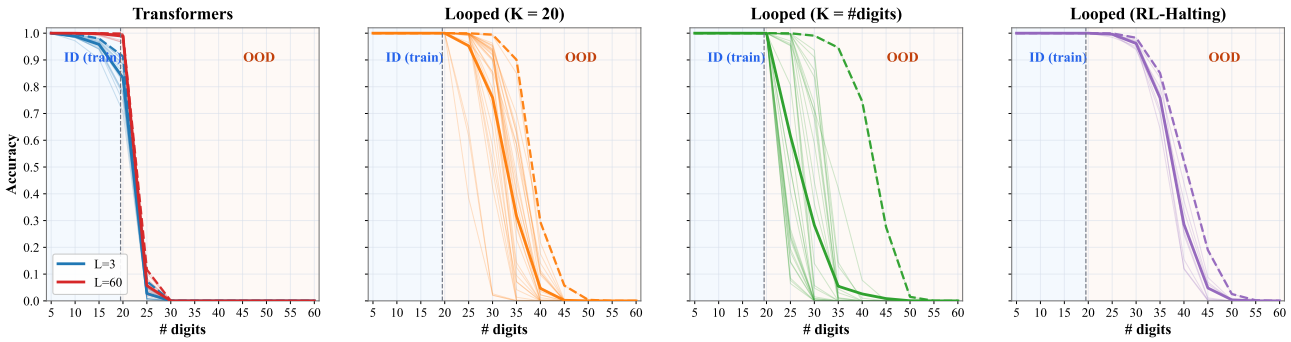

Figure 1. **Oracle-over-iterations accuracy on binary addition as a function of input length for standard Transformers and Looped Transformers trained with deterministic stopping schedules.** The shaded region marks the ID regime, and the right-hand side corresponds to longer OOD inputs. Thin solid lines denote individual runs, the bold solid line denotes the mean across runs, and the dashed line denotes the per-length maximum across runs.

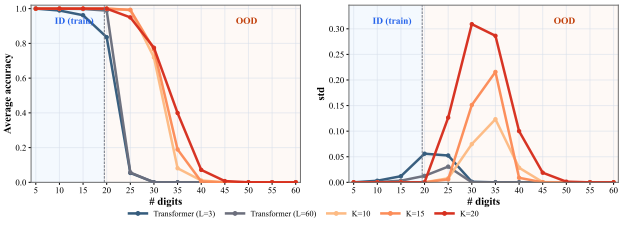

Figure 2. **Mean accuracy and run-to-run variability on binary addition across input lengths. Left:** mean oracle-over-iterations accuracy across runs for standard Transformer baselines and fixed-K Looped Transformers with $K \in \{10, 15, 20\}$. **Right:** standard deviation across runs at each digit length. The shaded region marks the ID regime, while digit lengths from 20 to 60 correspond to the OOD regime.

the 3-layer and 60-layer Transformer baselines, but rises to 0.087 for Looped Transformers with fixed $K = 20$ and 0.122 for Looped Transformers with $K = n$. Controlled variants, where either the initialisation or the data order is fixed, show the same qualitative instability; see Appx. D.1 for details.

Larger loop budgets widen the gap between potential and reliability.

We next vary the fixed loop budget. As shown in Fig. 2, increasing K shifts the mean accuracy curve toward longer inputs, suggesting that additional loop computation can extend the range over which some models extrapolate. However, the same increase also raises run-to-run variability in the OOD regime, especially at intermediate lengths where some runs still solve the task while others have already collapsed.

Thus, larger loop budgets increase extrapolation potential, but also make OOD behaviour less reliable. This supports the view that looping expands the set of computation trajectories available to training: some trajectories extrapolate, while others fit the ID regime but fail OOD.

This supports the view that looping creates a selection problem: the architecture may contain extrapolating trajectories, but training does not reliably select them.

4.3. Stochastic Schedules Stabilise Looped Models

The previous subsection showed that deterministic looped models can contain extrapolating computations, but do not reliably select them across runs. We now test whether this selection problem can be regularised by randomising the loop depth during training. Starting from the length-matched schedule $K = n$, we train stochastic variants that sample loop depths from a local window around n. The mechanism behind this stabilisation is discussed separately in Appx. E, where we give a toy illustration of how randomising over nearby loop depths can average out depth-specific effects.

We evaluate stochastic schedules along two axes. First, we ask whether they reduce run-to-run variability in OOD performance. Second, we ask whether they make each trained model less sensitive to the precise loop depth used at inference time.

Randomised schedules reduce run-to-run variability

Fig. 3 shows that stochastic schedules substantially reduce the spread of OOD outcomes across runs. The deterministic length-matched schedule has a wide min–max range, indicating that different runs select very different OOD behaviours. By contrast, the ± 5 and ± 10 windows produce much tighter ranges and smaller standard deviations.

The effect is primarily one of stabilisation, not a monotonic improvement in accuracy. The mean OOD accuracy increases from the deterministic schedule to the ± 5 window, but decreases again for the ± 10 window. Thus, randomisation can reduce OOD variance, but too much randomness need not further improve extrapolation. A toy gradient-averaging view that may explain this stabilising effect is given in Appx. E.1.

Randomised schedules stabilise predictions across loop depths.

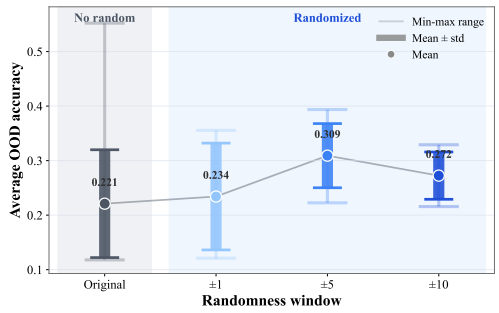

Figure 3. **OOD oracle-over-iterations accuracy for stochastic variants of the length-matched schedule on binary addition.** "Original" denotes the deterministic schedule $K = n$. For window size $w \in \{1, 5, 10\}$, we sample $\Delta \sim \mathrm{Unif}\{-w, \ldots, w\}$ and set $K = \mathrm{clip}(n + \Delta, 1, T_{\max})$. For each run, OOD accuracy is averaged over digit lengths $\{20, 25, 30, \ldots, 60\}$. Circles indicate the mean across runs, thick vertical bars show mean $\pm$ standard deviation, and light horizontal bars show the minimum–maximum range across runs.

Task	Method	OOD ↑	Near ↑	Max@90 ↑	Front.@90 ↑	Std. ↓
Addition	Fix $K = 20$	34.2	61.5	27.3	25	7.4
	$K = n$	22.1	39.6	23.0	20	9.9
	$K = n, \pm 5$	30.9	55.6	27.5	25	5.9
	RL-Halt	**45.0**	**80.0**	**30.0**	**30**	**2.7**
Dyck-1	Fix $K = 20$	84.6	93.3	49.5	30	23.8
	$K = n$	66.6	79.8	40.8	25	33.3
	$K = n, \pm 5$	88.2	97.4	52.0	35	18.7
	RL-Halt	**97.5**	**100.0**	**55.6**	**60**	**3.6**
Unique	Fix $K = 20$	66.1	78.0	39.0	20	34.0
	$K = n$	78.5	96.6	**44.1**	35	17.5
	$K = n, \pm 5$	73.2	98.8	40.7	**40**	9.3
	RL-Halt	**82.8**	**98.8**	41.2	**40**	**3.3**
Copy	Fix $K = 20$	62.3	82.0	36.2	25	27.7
	$K = n$	**68.7**	**94.3**	**36.7**	**35**	13.2
	$K = n, \pm 5$	55.9	89.4	33.8	30	8.5
	RL-Halt	43.2	76.1	28.0	25	**2.7**

Table 1. **OOD performance across tasks and stopping schedules.** OOD and Near denote average accuracy over lengths 20–60 and 20–40, respectively. Max@90 is the per-run maximum solved length averaged across runs, Front.@90 is the mean-accuracy frontier at 90%, and Std. is the standard deviation of per-run average OOD accuracy.

We next test whether stochastic schedules also stabilise the loop trajectory within a trained model. For each model, we sweep the inference-time loop count K and measure accuracy at each depth. We also measure the prediction flip rate, $\Pr_x[\hat{y}_{K+1}(x) \neq \hat{y}_K(x)]$, which records how often the predicted sequence changes between adjacent loop depths.

Fig. 4 shows that deterministic schedules are sensitive to the inference-time stopping depth. They often achieve high accuracy only over a narrow range of loop counts, and their predictions can continue changing across many iterations. Randomised training reduces this sensitivity: the randomised length-matched schedule gives a broader high-accuracy region across K and a faster decrease in prediction flip rate. This is consistent with the loop-consistency view in Appx. E.2, where supervising nearby depths encourages predictions to stabilise along the loop trajectory.

Together, Figs. 3 and 4 show that stochastic schedules stabilise looped models across both training runs and inference-time loop depths, while also highlighting that stabilisation alone does not guarantee extrapolation.

4.4. Learned Stopping Improves the Accuracy–Stability Trade-off

The stochastic schedules above stabilise looped models, but their sampling distribution is hand-designed and the window size matters. We therefore ask whether the stopping distribution itself can be learned. We study this with RL-Halting, a learned stochastic stopping schedule that samples one loop depth during training and updates the stopping policy from the realised task loss. This places RL-Halting within our main goal: understanding whether adaptive depth selection can reduce the training-time selection problem in looped models.

RL-Halting parameterises a distribution over stopping times using a hazard head on the loop hidden states. During training, it samples a stopping depth, applies the task loss only at that realised depth, and updates the stopping policy with a score-function estimator using negative task loss as reward. This differs from weighted-loss adaptive computation methods such as PonderNet, and keeps the comparison aligned with the sampled-depth schedules in Sec. 4.3. We give the full formulation in Appx. C. A direct comparison with a PonderNet-style baseline on binary addition is given in Appx. D.5.

Learned schedules can improve the accuracy–stability trade-off. Tab. 1 compares deterministic schedules, hand-designed stochastic schedules, and RL-Halting across four tasks. On Addition and Dyck-1, RL-Halting achieves both the highest OOD accuracy and the lowest OOD standard deviation. On Unique Set, it also improves average and near-OOD accuracy while substantially reducing variance, although the length-matched schedule attains a slightly larger Avg. Max Len@90; an oracle–policy gap analysis on this task is given in Appx. D.3. These results suggest that adaptive depth selection can preserve the stabilising effect of sampled-depth training while improving extrapolation on several tasks.

Stability does not guarantee the best extrapolating schedule. The Copy task shows the limitation. RL-Halting reduces OOD standard deviation from 13.2 to 2.7, but its average OOD accuracy and extrapolation frontier are worse than the length-matched schedule. Thus, a learned stochastic schedule can stabilise the model around a suboptimal computation. Taken together, these results show that adaptive stopping can improve the accuracy–stability trade-off, but it does not fully solve the trajectory-selection problem.

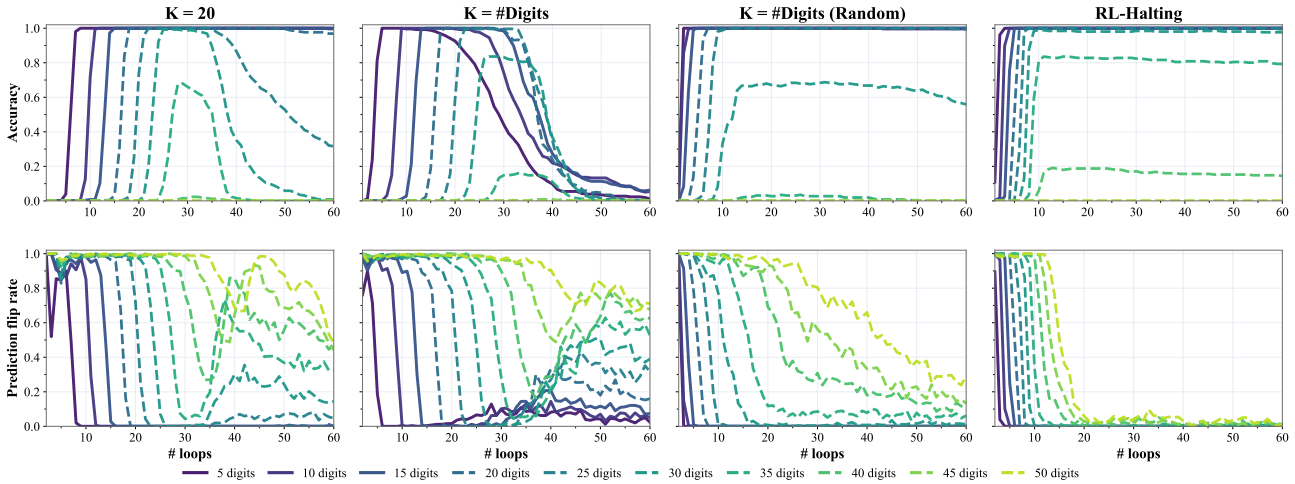

Figure 4. **Accuracy and prediction dynamics across inference-time loop counts on binary addition.** Columns compare models trained with fixed $K = 20$, length-matched $K = \#\text{Digits}$, randomised length-matched $K = \#\text{Digits}$ (Random), and RL-Halting. Here $K = \#\text{Digits}$ (Random) denotes training with $K = \text{clip}(n + \Delta, 1, T_{\max})$, where $\Delta \sim \text{Unif}\{-5, \ldots, 5\}$. **Top row:** accuracy as a function of inference-time loop count K for different input lengths. **Bottom row:** prediction flip rate between adjacent loop counts. Lower flip rates indicate more stable loop trajectories. RL-Halting is shown for reference and discussed in § 4.4.

Conclusion

We studied length generalisation in Looped Transformers through the lens of stopping schedules. Our results show that looping creates a training-time selection problem over computation trajectories: deterministic schedules can reveal strong extrapolation potential, but this potential is unstable across runs. Stochastic schedules reduce this fragility by training across nearby loop depths, improving run-to-run stability and robustness to the inference-time loop count. However, stability alone does not guarantee extrapolation: RL-Halting often improves the accuracy–stability trade-off, but can also stabilise a suboptimal computation. These findings suggest that "when to stop" should be treated as a central training-time design choice in looped architectures.

Author Contributions

H.-Y.K. developed the experimental framework, implemented the methods, conducted the experiments, analysed the results, and led the writing of the manuscript. The project originated in W.B.'s group, with P.R. and W.B. playing central roles in the initial brainstorming, defining the research direction, and shaping the experimental agenda. P.R. provided close mentoring throughout the project, including guidance on method development, experimental design, interpretation of results, and manuscript preparation. After the project continued in M.J.'s group at EPFL, E.M.C. also provided close mentoring and regular guidance on the research direction, experimental design, manuscript preparation, and possible methodological extensions, including reinforcement-learning-based improvements. M.J. provided supervisory feedback, as well as institutional and computational support at EPFL.

Acknowledgements

H.-Y.K. initiated this project during an internship in W.B.'s group at the Max Planck Institute for Intelligent Systems and the ELLIS Institute, Tübingen, and continued it as a project student in M.J.'s group at EPFL. We gratefully acknowledge the EPFL Research Computing Platform (RCP) for providing computational resources through M.J.'s group. P.R. acknowledges his membership in the European Laboratory for Learning and Intelligent Systems (ELLIS) PhD program and thanks the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for its support. This work was supported by the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center, FKZ: 01IS18039A, and by a grant from Coefficient Giving to Aaron Mueller. W.B. acknowledges financial support from an Emmy Noether Grant funded by the German Research Foundation (DFG) under grant no. BR 6382/1-1, and from the Open Philanthropy Foundation, funded by the Good Ventures Foundation. W.B. is a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645. This research also used compute resources at the Tübingen Machine Learning Cloud, DFG FKZ INST 37/1057-1 FUGG. Finally, we thank the organisers and participants of the Fourth Bellairs Workshop on Causality, held at the McGill University Bellairs Research Institute from 14–21 February 2025, for helpful discussions that contributed to the development of the project idea, in particular discussions with Kartik Ahuja.

References

Abbe, E., Bengio, S., Lotfi, A., and Rizk, K. Generalization on the Unseen, Logic Reasoning and Degree Curriculum, November 2024. URL http://arxiv.org/abs/2301.13105.

Anil, C., Pokle, A., Liang, K., Treutlein, J., Wu, Y., Bai, S., Kolter, Z., and Grosse, R. Path Independent Equilibrium Models Can Better Exploit Test-Time Computation.

Banino, A., Balaguer, J., and Blundell, C. PonderNet: Learning to Ponder, September 2021. URL http://arxiv.org/abs/2107.05407. arXiv:2107.05407 [cs].

Bansal, A., Schwarzschild, A., Borgnia, E., Emam, Z., Huang, F., Goldblum, M., and Goldstein, T. Thinking Deeper With Recurrent Networks: Logical Extrapolation Without Overthinking. October 2021. URL https://openreview.net/forum?id=kDF4Owotj5j.

Bansal, A., Schwarzschild, A., Borgnia, E., Emam, Z., Huang, F., Goldblum, M., and Goldstein, T. End-to-end Algorithm Synthesis with Recurrent Networks: Logical Extrapolation Without Overthinking, October 2022. URL http://arxiv.org/abs/2202.05826. arXiv:2202.05826 [cs].

Cai, Z., Lee, N., Schwarzschild, A., Oymak, S., and Papailiopoulos, D. Extrapolation by Association: Length Generalization Transfer in Transformers, August 2025. URL http://arxiv.org/abs/2506.09251. arXiv:2506.09251 [cs].

Cho, H., Cha, J., Awasthi, P., Bhojanapalli, S., Gupta, A., and Yun, C. Position Coupling: Improving Length Generalization of Arithmetic Transformers Using Task Structure. November 2024. URL https://openreview.net/forum?id=5cIRdGM1uG&referrer=%5Bthe%20profile%20of%20Pranjal%20Awasthi%5D(%2Fprofile%3Fid%3D~Pranjal_Awasthi3).

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, . Universal Transformers, July 2018. URL https://arxiv.org/abs/1807.03819v3.

Fan, A., Grave, E., and Joulin, A. Reducing Transformer Depth on Demand with Structured Dropout, September 2019. URL http://arxiv.org/abs/1909.11556. arXiv:1909.11556 [cs].

Fan, Y., Du, Y., Ramchandran, K., and Lee, K. Looped Transformers for Length Generalization. October 2024. URL https://openreview.net/forum?id=2edigk8yoU.

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., and Goldstein, T. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach, February 2025. URL http://arxiv.org/abs/2502.05171. arXiv:2502.05171 [cs].

Giannou, A., Rajput, S., Sohn, J.-y., Lee, K., Lee, J. D., and Papailiopoulos, D. Looped Transformers as Programmable Computers. *arXiv*, 2023. doi: 10.48550/arxiv.2301.13196. TLDR: It is demonstrated that a constant number of encoder layers can emulate basic computing blocks, including embedding edit operations, non-linear functions, function calls, program counters, and conditional branches, and using these building blocks to emulate a small instruction-set computer.

Graves, A. Adaptive Computation Time for Recurrent Neural Networks, February 2017. URL http://arxiv.org/abs/1603.08983. arXiv:1603.08983 [cs].

Hacohen, G. and Weinshall, D. On The Power of Curriculum Learning in Training Deep Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2535–2544. PMLR, May 2019. URL https://proceedings.mlr.press/v97/hacohen19a.html.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Deep Networks with Stochastic Depth, July 2016. URL http://arxiv.org/abs/1603.09382. arXiv:1603.09382 [cs].

Kazemnejad, A., Padhi, I., Ramamurthy, K. N., Das, P., and Reddy, S. The Impact of Positional Encoding on Length Generalization in Transformers.

Kuo, H.-Y., Liao, Y.-H., Chao, Y.-C., Ma, W.-Y., and Cheng, P.-J. Not All LLM-Generated Data Are Equal: Rethinking Data Weighting in Text Classification. In *The Thirteenth International Conference on Learning Representations*, March 2025. doi: 10.48550/arXiv.2410.21526. URL http://arxiv.org/abs/2410.21526.

Lee, N., Cai, Z., Schwarzschild, A., Lee, K., and Papailiopoulos, D. Self-Improving Transformers Overcome Easy-to-Hard and Length Generalization Challenges. June 2025. URL https://openreview.net/forum?id=ZtX0MBT6mf.

Li, R. and Boduljak, G. ON VANISHING VARIANCE IN TRANSFORMER LENGTH GENERALIZATION. 2025.

Mészáros, A., Ujváry, S., Brendel, W., Reizinger, P., and Huszár, F. Rule extrapolation in language modeling: A study of compositional generalization on ood prompts.

Advances in Neural Information Processing Systems, 37: 34870–34899, 2024.

Mészáros, A., Reizinger, P., and Huszár, F. Out-of-distribution evaluation of rule-based and strategic reasoning in chess transformers. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=JFGUsQ68rG.

Ramasinghe, S., Macdonald, L. E., Farazi, M., Saratchandran, H., and Lucey, S. How much does Initialization Affect Generalization? In *Proceedings of the 40th International Conference on Machine Learning*, pp. 28637–28655. PMLR, July 2023. URL https://proceedings.mlr.press/v202/ramasinghe23a.html.

Reizinger, P., Ujváry, S., Mészáros, A., Kerekes, A., Brendel, W., and Huszár, F. Position: Understanding LLMs Requires More Than Statistical Generalization. *arXiv*, 2024. doi: 10.48550/arxiv.2405.01964. URL https://arxiv.org/abs/2405.01964.

Ruoss, A., Deltang, G., Genewein, T., Grau-Moya, J., Csords, R., Bennani, M., Legg, S., and Veness, J. Randomized Positional Encodings Boost Length Generalization of Transformers, May 2023. URL http://arxiv.org/abs/2305.16843. arXiv:2305.16843 [cs].

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. Reasoning with Latent Thoughts: On the Power of Looped Transformers. October 2024. URL https://openreview.net/forum?id=din0lGfZFd.

Schwarzschild, A., Borgnia, E., Gupta, A., Huang, F., Vishkin, U., Goldblum, M., and Goldstein, T. Can You Learn an Algorithm? Generalizing from Easy to Hard Problems with Recurrent Networks, November 2021. URL http://arxiv.org/abs/2106.04537. arXiv:2106.04537 [cs].

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting.

Wang, G., Li, J., Sun, Y., Chen, X., Liu, C., Wu, Y., Lu, M., Song, S., and Yadkori, Y. A. Hierarchical Reasoning Model, July 2025. URL http://arxiv.org/abs/2506.21734. arXiv:2506.21734 [cs].

Yang, L., Lee, K., Nowak, R., and Papailiopoulos, D. Looped Transformers are Better at Learning Learning Algorithms. *arXiv*, 2023. doi: 10.48550/arxiv.2311.12424.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J., Bengio, S., and Nakkiran, P. What Algorithms can Transformers Learn? A Study in Length Generalization, October 2023. URL http://arxiv.org/abs/2310.16028. arXiv:2310.16028.

Zhou, Y., Alon, U., Chen, X., Wang, X., Agarwal, R., and Zhou, D. Transformers Can Achieve Length Generalization But Not Robustly, February 2024. URL http://arxiv.org/abs/2402.09371. arXiv:2402.09371 [cs].

Zhu, R.-J., Wang, Z., Hua, K., Zhang, T., Li, Z., Que, H., Wei, B., Wen, Z., Yin, F., Xing, H., Li, L., Shi, J., Ma, K., Li, S., Kergan, T., Smith, A., Qu, X., Hui, M., Wu, B., Min, Q., Huang, H., Zhou, X., Ye, W., Liu, J., Yang, J., Shi, Y., Lin, C., Zhao, E., Cai, T., Zhang, G., Huang, W., Bengio, Y., and Eshraghian, J. Scaling Latent Reasoning via Looped Language Models, October 2025. URL http://arxiv.org/abs/2510.25741. arXiv:2510.25741 [cs].

A. Related Works

Stability of length generalisation. Prior work has shown that Transformer length generalisation can be possible but fragile. Most directly, Zhou et al. (2024) show that extrapolation on integer addition is highly sensitive to random initialisation and training data order, even with strong in-distribution performance. Other work connects length-generalisation failures to positional-encoding distribution shift (Ruoss et al., 2023; Cho et al., 2024; Kazemnejad et al.), attention-variance mechanisms (Li & Boduljak, 2025), and broader effects of initialisation or data ordering on optimisation (Ramasinghe et al., 2023; Hacohen & Weinshall, 2019; Schwarzschild et al., 2021; Kuo et al., 2025). Mészáros et al. (2024) studies OOD extrapolation performance in formal languages across many architectures (not including Looped Transformers), and Mészáros et al. (2026) conducts a similar study across multiple chess variants. Our work studies a complementary instability specific to looped architectures: the loop count used during training is not merely a compute budget, but affects which computation trajectory is selected and how stable length extrapolation becomes.

Extrapolation and recurrent structure. Recurrent and weight-tied architectures support extrapolation by reusing computation across multiple steps, as studied in recurrent networks, Universal Transformers, deep equilibrium models, and Looped Transformers (Schwarzschild et al., 2021; Bansal et al., 2022; Dehghani et al., 2018; Anil et al.; Giannou et al., 2023; Yang et al., 2023; Fan et al., 2024; Saunshi et al., 2024; Geiping et al., 2025; Zhu et al., 2025; Wang et al., 2025). In particular, Fan et al. (2024) show that Looped Transformers can achieve strong length generalisation on arithmetic tasks, demonstrating the extrapolation potential of repeated computation. Our work studies a complementary question: whether this potential is selected reliably during training. We show that looped models can exhibit high run-to-run OOD instability, and that stochastic loop schedules can regularise this instability by reducing dependence on a single prescribed loop depth.

Learning when to stop. Adaptive-computation methods learn to allocate computation across inputs, from Adaptive Computation Time (Graves, 2017) to probabilistic halting methods such as PonderNet (Banino et al., 2021). In recurrent reasoning, the stopping rule matters because too few iterations may under-compute, while too many can cause overthinking and hurt extrapolation (Bansal et al., 2021; 2022). Recent Looped Language Models also use learned depth allocation and relate it to PonderNet-style objectives (Zhu et al., 2025). Our RL-Halting variant is related to these methods, but matches our sampled-depth protocol: each update samples one stopping depth and supervises only that realised depth. This lets us study learned stopping as an adaptive form of stochastic loop-depth regularisation, rather than primarily as compute saving.

Randomisation as regularisation. Randomisation is widely used to regularise neural networks by preventing over-reliance on a single computation path. Dropout randomly removes units (Srivastava et al.), stochastic depth randomly skips residual layers (Huang et al., 2016), and LayerDrop applies structured dropout to Transformer layers (Fan et al., 2019). Randomisation has also been used to improve length generalisation, for example through randomised positional encodings (Ruoss et al., 2023), and to encourage path-independent behaviour in equilibrium models through randomised depth training (Anil et al.). Our work studies a different axis of randomisation: the loop depth at which a recurrent-depth model is supervised. This can be viewed as regularising the learned computation trajectory by averaging over nearby stopping depths, thereby reducing sensitivity to a single prescribed loop count.

B. Implementation Details

Architecture and training. Our Looped Transformer implementation follows Fan et al. (Fan et al., 2024). One looped block consists of three Transformer layers and is reused across loop iterations with input injection at each step. Following the length-generalisation setup of Fan et al. (Fan et al., 2024), we use no positional embeddings (NoPE). The 3-layer Transformer baseline matches the parameter count of one looped block, while the 60-layer Transformer baseline provides a higher-compute fixed-depth comparison. All models are trained with AdamW using batch size 64 for 100k gradient steps. The learning rate is set to 10^{-4} and decayed to 0 with a cosine schedule after the curriculum reaches the maximum training length. For stability evaluation, we run 32 independent seeds on Addition and 18 independent seeds on the other tasks.

Stopping schedules. For stochastic schedules, each optimisation step samples one loop count per example. For the randomised length-matched schedules, we sample

$$\Delta \sim \mathrm{Unif}\{-w, \ldots, w\}, \qquad K = \mathrm{clip}(n + \Delta, 1, T_{\max}).$$

Evaluation. Unless otherwise stated, OOD summaries are computed over lengths $\{20, 25, 30, \ldots, 60\}$, and a prediction is correct only if the complete output sequence is correct, including the end token.

Controlled variance experiments. For the "Original" setting, both random initialisation and data order vary across runs. For "Fixed init.", all runs use the same initialisation and vary only the data order. For "Fixed data", all runs use the same data order and vary only the initialisation.

C. RL-Halting Details

We describe the learned stochastic stopping rule used in Sec. 4.4. RL-Halting parameterises a distribution over loop depths, samples one depth during training, and updates the stopping policy from the realised task loss. This matches the sampled-depth protocol used by the hand-designed stochastic schedules, while allowing the stopping distribution to adapt during training.

Stopping distribution. Let $H^{(t)} \in \mathbb{R}^{m \times d}$ denote the hidden states after t loop iterations, and let $T_{\max}$ be the maximum allowed number of loop iterations. After each iteration, we average-pool the hidden states to obtain a summary vector

$$h_t = \mathrm{Pool}(H^{(t)}). \tag{1}$$

A learned stopping head q_ϕ maps this summary to a stopping hazard

$$r_t = \sigma(q_\phi(h_t)), \tag{2}$$

where σ is the sigmoid function. The hazard r_t is the probability of stopping at iteration t conditioned on not having stopped earlier. This induces the stopping distribution

$$\pi_{\mathrm{stop},\phi}(\tau = t \mid x) = r_t \prod_{i=1}^{t-1}(1 - r_i), \qquad t < T_{\max}. \tag{3}$$

The remaining probability mass is assigned to the final iteration:

$$\pi_{\mathrm{stop},\phi}(\tau = T_{\max} \mid x) = \prod_{i=1}^{T_{\max}-1}(1 - r_i). \tag{4}$$

Thus, the model is forced to stop by $T_{\max}$.

Sampled-depth supervision. For each training example (x, y), we sample a stopping time

$$\tau \sim \pi_{\mathrm{stop},\phi}(\cdot \mid x). \tag{5}$$

The Looped Transformer is then supervised only at the sampled depth τ. Let $\hat{y}_\tau(x)$ denote the prediction after τ loop iterations. The task loss is

$$\ell(\theta; x, y, \tau) = \mathrm{CE}(\hat{y}_\tau(x), y), \tag{6}$$

where CE denotes the task cross-entropy loss. The model parameters θ are updated using the ordinary supervised gradient of this sampled-depth loss.

Stopping-policy update. The stopping policy is trained with a score-function estimator. We define the reward as the negative task loss:

$$R(x, y, \tau) = -\ell(\theta; x, y, \tau). \tag{7}$$

The objective for the stopping policy is to maximise the expected reward under the stopping distribution:

$$J(\phi) = \mathbb{E}_{\tau \sim \pi_{\mathrm{stop},\phi}(\cdot|x)}[R(x, y, \tau)]. \tag{8}$$

Using the score-function identity, the policy-gradient estimator is

$$\nabla_\phi J(\phi) = \mathbb{E}_{\tau \sim \pi_{\mathrm{stop},\phi}(\cdot|x)}[(R(x, y, \tau) - b)\nabla_\phi \log \pi_{\mathrm{stop},\phi}(\tau \mid x)], \tag{9}$$

where b is a moving-average baseline used to reduce gradient variance. In practice, we also add an entropy regulariser over the stopping distribution to encourage exploration over loop depths:

$$J_{\mathrm{ent}}(\phi) = J(\phi) + \lambda_{\mathrm{ent}} \mathcal{H}(\pi_{\mathrm{stop},\phi}(\cdot \mid x)). \tag{10}$$

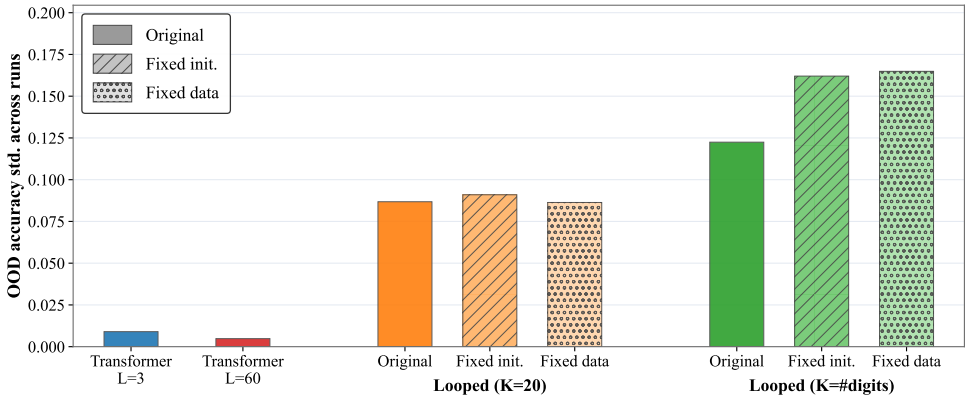

Figure 5. OOD variability on binary addition across models and controlled settings. Bars show the mean standard deviation of oracle-over-iterations accuracy across runs, averaged over OOD digit lengths from 20 to 60. For Looped Transformers, "Original" varies both initialisation and data order across runs, "Fixed init." varies only data order, and "Fixed data" varies only initialisation. Hatching marks the controlled settings.

Training loss. Equivalently, when using gradient descent, we minimise the stopping-policy loss

$$\mathcal{L}_{\text{halt}}(\phi) = -\text{sg}(R - b) \log \pi_{\text{stop},\phi}(\tau \mid x) - \lambda_{\text{ent}} \mathcal{H}(\pi_{\text{stop},\phi}(\cdot \mid x)), \tag{11}$$

where $\text{sg}(\cdot)$ denotes stop-gradient. The total update consists of the sampled-depth task loss for the Looped Transformer and the policy-gradient loss for the stopping head:

$$\mathcal{L} = \ell(\theta; x, y, \tau) + \mathcal{L}_{\text{halt}}(\phi). \tag{12}$$

Relation to stochastic schedules. The hand-designed stochastic schedules in Sec. 4.3 sample loop depths from a fixed distribution, such as a local window around the input length. RL-Halting instead learns this distribution from the realised task loss. Both approaches expose the model to sampled stopping depths during training, but RL-Halting makes the sampling distribution adaptive.

D. Additional Results

D.1. Controlled variance experiments

To test whether OOD instability is caused mainly by random initialisation or by data order, we run two controlled variants of the looped experiments. In the "Fixed init." setting, all runs use the same initialisation and differ only in data order. In the "Fixed data" setting, all runs use the same data order and differ only in initialisation.

Fig. 5 shows that the high OOD variance of Looped Transformers does not disappear under either control. For fixed $K = 20$, the OOD standard deviation remains around 0.09 in both controlled settings, compared with 0.087 in the original setting. For the length-matched schedule $K = n$, the controlled standard deviations are even higher, around 0.16, compared with 0.122 in the original setting. By contrast, the fixed-depth Transformer baselines have much lower OOD standard deviation, 0.009 for the 3-layer model and 0.005 for the 60-layer model.

These results suggest that OOD instability is not an artefact of varying all sources of randomness simultaneously. Both initialisation and data order can steer looped training toward different extrapolation behaviours, even when models achieve similar in-distribution performance.

Increasing the fixed loop budget shifts the mean accuracy curve toward longer inputs, suggesting that additional loop computation can extend extrapolation potential. However, it also raises run-to-run variability in the OOD regime, especially at intermediate lengths where some runs still solve the task while others have already collapsed.

D.2. Randomising around a fixed loop budget

We also test whether stochasticity helps when the centre of the schedule is fixed rather than length-matched. Starting from the fixed schedule $K = 20$, we sample $\Delta \sim \text{Unif}\{-5, \ldots, 5\}$ and train at $K = 20 + \Delta$.

Table 2. **Fixed-budget randomisation and learned-halting baselines on binary addition.** OOD is average oracle-over-iterations accuracy over lengths 20–60, Front.@90 is the mean-accuracy frontier at 90%, and Std. is the standard deviation across runs.

Setting	OOD ↑	Front.@90 ↑	Std. ↓
Fix $K = 20$	34.2	25	7.4
$K = 20 \pm 5$	34.2	25	4.7
PonderNet	29.4	20	15.1
RL-Halting	**45.0**	**30**	**2.7**

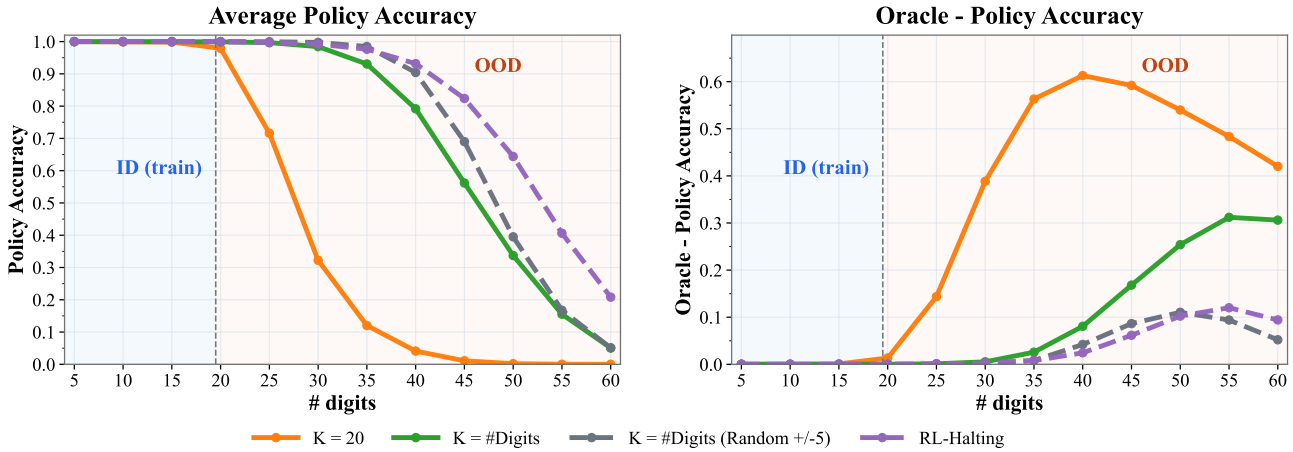

Figure 6. **Policy accuracy and oracle–policy gap on Unique Set.** Left: policy accuracy, evaluating each method using its prescribed stopping rule. Right: oracle-over-iterations accuracy minus policy accuracy. A large gap indicates that a correct output exists at some loop depth but is not selected by the policy. Randomised and learned stochastic schedules reduce this gap, suggesting that they stabilise predictions across loop depths and make performance less sensitive to the exact stopping time.

Tab. 2 shows that randomising around the fixed budget leaves the mean OOD accuracy and extrapolation frontier unchanged, but reduces the standard deviation across runs from 7.4 to 4.7. This suggests that the stabilising effect of stochastic schedules is not specific to length-matched schedules: exposing the model to nearby loop depths can reduce run-to-run variability even when the schedule is centred at a fixed compute budget.

D.3. Oracle–policy gap on Unique Set

Our main results use oracle-over-iterations accuracy to diagnose whether a trained looped model produces a correct output at some evaluated depth. However, oracle accuracy does not measure whether a practical stopping rule selects that depth. To make this distinction explicit, we compare oracle accuracy with *policy accuracy* on Unique Set. Policy accuracy evaluates each method using its inference-time stopping rule: $K = 20$ for $K = 20$, $K = n$ for $K = \#\text{Digits}$, and the centre depth $K = n$ for $K = \#\text{Digits}$ *(Random* ± 5*)*. For *RL-Halting*, we choose the stopping depth with the highest learned probability.

We define the oracle–policy gap at length n as

$$\text{Gap}(n) = \text{Acc}_{\text{oracle}}(n) - \text{Acc}_{\text{policy}}(n).$$

A large gap means that a correct output exists at some loop depth, but the stopping rule often fails to select it. Thus, the gap measures a failure of depth selection rather than the absence of an extrapolating computation.

Fig. 6 shows that deterministic schedules can have a large oracle–policy gap in the OOD regime. For fixed $K = 20$, policy accuracy collapses soon after the training-length boundary, while oracle accuracy remains substantially higher, indicating that the model can still produce correct outputs at some other loop depths. The length-matched schedule performs better, but still leaves a growing gap at longer OOD lengths.

By contrast, the randomised length-matched schedule and RL-Halting have much smaller oracle–policy gaps. This is consistent with the trajectory-stability results in Fig. 4: when predictions are more stable across nearby loop depths, the

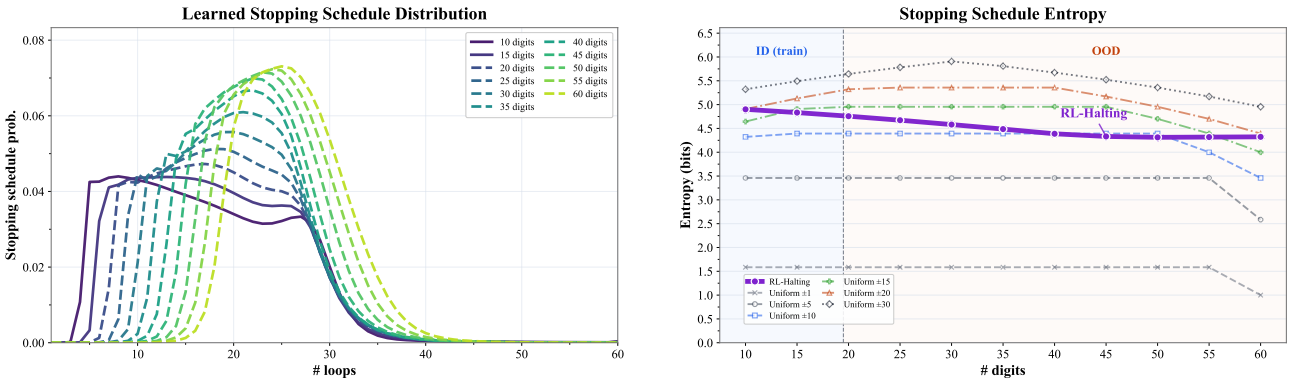

Figure 7. **Learned stopping schedule and stopping entropy on binary addition.** For this visualisation, the stopping distribution is truncated at $T_{\text{vis}} = 60$. **Left**: learned stopping distributions $\pi^{(T_{\text{vis}})}_{\text{stop},\phi}(\tau = k \mid x)$ for RL-Halting, averaged over binary-addition examples with the same digit length. As the number of digits increases, the probability mass shifts toward larger loop depths, indicating that the learned policy adapts its stopping distribution to problem length. **Right**: conditional stopping entropy as a function of digit length, compared with uniform local-window schedules. RL-Halting remains stochastic across both ID and OOD digit lengths, but has lower entropy than wide uniform schedules, suggesting that it learns a structured stochastic schedule rather than a maximally diffuse one.

exact stopping time matters less, and policy accuracy approaches oracle accuracy. Thus, stochastic schedules reduce not only run-to-run variability, but also the mismatch between the computation available somewhere along the loop and the computation selected by the stopping rule.

A small oracle–policy gap does not by itself imply strong extrapolation. A model can have a small gap because it is stably correct, but also because it is stably wrong. For this reason, we report both policy accuracy and oracle–policy gap: policy accuracy measures the quality of the selected computation, while the gap measures how much performance is lost due to imperfect depth selection.

D.4. Learned stopping schedule on binary addition

We further inspect the stopping distribution learned by RL-Halting on binary addition. The goal is to understand whether the learned schedule collapses to an almost deterministic stopping rule, or whether it remains genuinely stochastic during length extrapolation.

For this diagnostic, we evaluate the learned stopping head over a visualisation horizon $T_{\text{vis}} = 60$. This horizon is used only to inspect the learned distribution and compare it with length-matched uniform schedules. For any horizon T, the same hazard rule defines a truncated stopping distribution $\pi^{(T)}_{\text{stop},\phi}(\tau = k \mid x)$, with the remaining tail probability assigned to the final depth T.

We quantify the randomness of the learned schedule using the conditional entropy

$$H_T(\tau \mid x) = -\sum_{k=1}^{T} \pi^{(T)}_{\text{stop},\phi}(\tau = k \mid x) \log_2 \pi^{(T)}_{\text{stop},\phi}(\tau = k \mid x). \tag{13}$$

In this appendix, we set $T = T_{\text{vis}} = 60$ and report the entropy averaged over binary-addition examples of the same digit length,

$$H(n) = \mathbb{E}_{x:\#\text{digits}(x)=n} \left[H_{T_{\text{vis}}}(\tau \mid x) \right]. \tag{14}$$

This measures randomness after conditioning on the input, and therefore distinguishes genuinely stochastic stopping from deterministic input-dependent schedules such as $K = n$, which have zero conditional entropy.

Fig. 7 shows that RL-Halting learns a structured length-dependent distribution on binary addition. For short inputs, the distribution places substantial mass on smaller loop counts. As the number of digits increases, the mass shifts smoothly to the right, assigning higher probability to larger loop counts. Thus, the learned stopping rule is not simply a fixed-depth policy: it adapts the range of likely stopping times to the input length.

At the same time, the learned schedule does not collapse to a deterministic stopping depth. The entropy curve remains clearly above zero across both ID and OOD lengths, showing that RL-Halting continues to sample from a non-degenerate distribution. However, its entropy is lower than that of wide uniform schedules around the input length. This suggests that RL-Halting learns a middle ground: it preserves stochasticity over loop depths, but concentrates probability on a length-dependent range of useful depths.

This behaviour is consistent with the role of RL-Halting in the binary-addition experiments. Unlike deterministic schedules, it does not force every update to use a single prescribed depth. Unlike hand-designed random schedules, it does not spread probability uniformly over a fixed window. Instead, it learns an adaptive stochastic schedule whose support shifts with digit length while maintaining moderate entropy. This provides a possible explanation for why RL-Halting reduces run-to-run OOD variability on addition while improving the accuracy–stability trade-off.

D.5. Comparison with PonderNet-style halting

We also include a PonderNet-style learned halting baseline in Tab. 2. PonderNet and RL-Halting use a similar hazard-based parameterisation of the stopping distribution, but differ in how this distribution is trained. A PonderNet-style objective optimises a distribution-weighted loss over loop depths, so depths with larger stopping probability also receive larger effective training weight. By contrast, RL-Halting samples a single stopping depth for each example, supervises the model only at that realised depth, and updates the stopping head with a policy-gradient estimator using the negative task loss as reward.

In our binary-addition experiments, the PonderNet-style baseline is substantially less stable than RL-Halting. As shown in Tab. 2, PonderNet obtains lower OOD accuracy and a higher standard deviation across runs, while RL-Halting achieves both the best OOD accuracy and the lowest variance. Empirically, we find that the PonderNet-style objective is sensitive to data order and often collapses toward a nearly deterministic halting distribution, even with entropy regularisation coefficient 0.01.

This is consistent with the broader observation that learned halting or gating mechanisms require explicit regularisation to avoid degenerate stopping behaviour: PonderNet uses a KL prior partly to assign non-zero probability to all halting steps and promote exploration, while recent learned-depth models also report collapse or instability in learned gates (Banino et al., 2021; Zhu et al., 2025).

One possible explanation is a self-reinforcing feedback loop: early batches can assign slightly higher stopping probability to some depths, those depths then receive larger effective loss weight, and the stopping distribution further concentrates around them.

E. Why Does Randomisation Stabilise Looping?

The previous section showed that randomising the training loop depth stabilises Looped Transformers in two ways: it reduces run-to-run OOD variability, and it makes predictions less sensitive to the inference-time loop count. We now give an informal explanation for these effects. The goal is not to prove that randomisation improves extrapolation, but to identify two simple mechanisms by which it can reduce fragility. Formal toy statements and proofs are given in Appxs. F and G.

E.1. Why can randomisation reduce fragility? A toy gradient-averaging view

Although each SGD step samples only one loop depth, the expected update corresponds to a depth-averaged objective. Let $L_k(\theta; z)$ be the loss on example $z = (x, y)$ after k loop iterations, and define

$$J_w(\theta; z) = \frac{1}{|S_w(x)|} \sum_{k \in S_w(x)} L_k(\theta; z),$$

where $S_w(x)$ is a local window of loop depths around the input length. If $K \sim \mathrm{Unif}(S_w(x))$, then

$$\mathbb{E}_K \big[\nabla_\theta L_K(\theta; z)\big] = \nabla_\theta J_w(\theta; z).$$

Thus, randomisation changes the expected update from a single-depth update to an average over nearby loop depths.

To interpret this averaging effect, write each depth-specific gradient as $G_k = \bar{G} + \xi_k$, where $\bar{G}$ is the component shared across nearby depths and ξ_k is a depth-specific residual. If the residuals are bounded and weakly aligned, averaging over

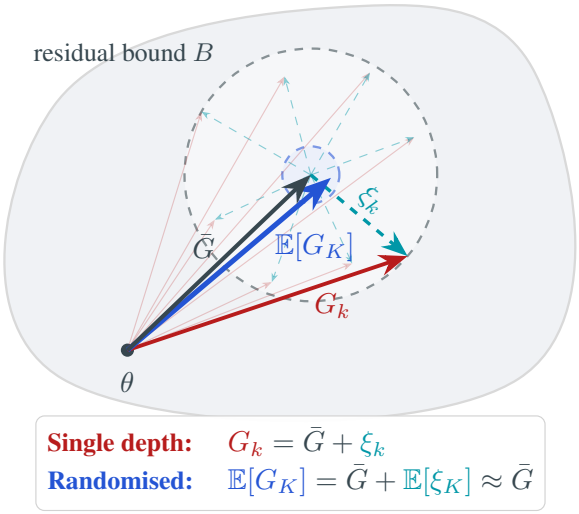

Figure 8. Toy geometry of gradient averaging. A single-depth update G_k can deviate from the shared direction $\bar{G}$ by a depth-specific residual ξ_k. When randomised schedules sample several nearby loop depths, weakly aligned residuals partially cancel in expectation, moving the expected update $\mathbb{E}[G_K]$ closer to the shared component $\bar{G}$. This provides a possible stabilising mechanism, but does not imply that $\bar{G}$ is extrapolating.

loop depths suppresses their contribution, so the expected update moves closer to the shared direction. This gives a possible mechanism for the reduced run-to-run spread in Fig. 3. A formal version of this toy argument is given in Prop. F.2.

This mechanism explains stabilisation, not extrapolation. The shared direction $\bar{G}$ need not be the direction that leads to the best OOD solution, so lower variance need not imply higher mean OOD accuracy.

E.2. Why can randomisation stabilise the loop trajectory?

A complementary view is that randomised schedules supervise the same input at multiple nearby loop depths. A deterministic schedule only asks the model to be correct at one prescribed depth, whereas a randomised schedule asks it to produce compatible answers across a local window of depths.

This can stabilise the loop trajectory. If nearby loop states are trained to support the same target output, then predictions are less likely to change abruptly from one iteration to the next. This gives an interpretation of the lower prediction flip rates and broader useful-K regions in Fig. 4. Again, this is only a stabilising mechanism: it encourages consistency across nearby depths, but does not guarantee that the consistent computation is the extrapolating one. A formal sufficient-condition argument is given in Prop. G.2.

F. Toy Analysis for Randomised Loop Schedules

This appendix formalises the gradient-averaging intuition used in § 4.3. The result should be read as a local mechanism for stabilisation, not as a guarantee that randomisation always improves extrapolation.

Let $z = (x, y)$ be a training example and let $L_k(\theta; z)$ denote the loss after running the looped model for k iterations. Assume L_k is differentiable in θ, and define the loop-k gradient $g_k(\theta; z) := \nabla_\theta L_k(\theta; z)$. For a randomised schedule π supported on a finite set of loop depths $S \subseteq \{1, \dots, T\}$, define the schedule-averaged objective

$$J_\pi(\theta; z) := \mathbb{E}_{K\sim\pi}\left[L_K(\theta; z)\right] = \sum_{k\in S} \pi(k) L_k(\theta; z).$$

Its gradient is

$$g_\pi(\theta; z) := \nabla_\theta J_\pi(\theta; z) = \sum_{k\in S} \pi(k) g_k(\theta; z).$$

Equivalently, if one samples $K \sim \pi$ at each update and uses the single-depth gradient $g_K(\theta; z)$, then

$$\mathbb{E}_{K\sim\pi}\left[g_K(\theta; z)\right] = g_\pi(\theta; z).$$

Thus, sampling one loop count per update gives an unbiased estimator of the gradient of the schedule-averaged objective.

Assumption F.1 (Local shared-gradient model)**.** Fix θ and $z = (x, y)$. For loop depths $k \in S$, assume that the gradients admit the decomposition

$$g_k(\theta; z) = \bar{g}(\theta; z) + \xi_k(\theta; z),$$

where $\bar{g}(\theta; z)$ is a shared update direction and $\xi_k(\theta; z)$ is a depth-specific residual. Assume there exist constants $B > 0$ and $\rho \in [0, 1]$ such that

$$\|\xi_k(\theta; z)\|^2 \le B^2 \qquad \text{for all } k \in S,$$

and

$$\langle \xi_i(\theta; z), \xi_j(\theta; z) \rangle \le \rho B^2 \qquad \text{for all } i \ne j,\ i, j \in S.$$

Proposition F.2 (Randomised schedules average depth-specific updates)**.** *Under Assumption F.1, the schedule-averaged gradient satisfies*

$$\|g_\pi(\theta; z) - \bar{g}(\theta; z)\|^2 \le B^2 \left[\rho + (1 - \rho) \sum_{k \in S} \pi(k)^2 \right].$$

In particular, if π is uniform on S and $m = |S|$, then

$$\|g_\pi(\theta; z) - \bar{g}(\theta; z)\|^2 \le B^2 \left(\rho + \frac{1 - \rho}{m} \right).$$

Thus, when the depth-specific residuals are weakly aligned across loop depths, increasing the number of loop depths in the randomised window reduces the remaining depth-specific component of the expected update.

Proof. Using the decomposition $g_k = \bar{g} + \xi_k$ and $\sum_{k \in S} \pi(k) = 1$, we have

$$g_\pi - \bar{g} = \sum_{k \in S} \pi(k) g_k - \bar{g} = \sum_{k \in S} \pi(k)(\bar{g} + \xi_k) - \bar{g} = \sum_{k \in S} \pi(k) \xi_k.$$

Therefore,

$$\|g_\pi - \bar{g}\|^2 = \left\| \sum_{k \in S} \pi(k) \xi_k \right\|^2.$$

Expanding the squared norm gives

$$\|g_\pi - \bar{g}\|^2 = \sum_{k \in S} \pi(k)^2 \|\xi_k\|^2 + \sum_{\substack{i, j \in S \\ i \ne j}} \pi(i) \pi(j) \langle \xi_i, \xi_j \rangle.$$

By Assumption F.1,

$$\|g_\pi - \bar{g}\|^2 \le B^2 \sum_{k \in S} \pi(k)^2 + \rho B^2 \sum_{\substack{i, j \in S \\ i \ne j}} \pi(i) \pi(j).$$

Since

$$\sum_{\substack{i, j \in S \\ i \ne j}} \pi(i) \pi(j) = \left(\sum_{k \in S} \pi(k) \right)^2 - \sum_{k \in S} \pi(k)^2 = 1 - \sum_{k \in S} \pi(k)^2,$$

we obtain

$$\|g_\pi - \bar{g}\|^2 \le B^2 \sum_{k \in S} \pi(k)^2 + \rho B^2 \left(1 - \sum_{k \in S} \pi(k)^2 \right).$$

Rearranging gives

$$\|g_\pi - \bar{g}\|^2 \le B^2 \left[\rho + (1 - \rho) \sum_{k \in S} \pi(k)^2 \right].$$

If π is uniform on S and $m = |S|$, then $\pi(k) = 1/m$ for all $k \in S$, so

$$\sum_{k \in S} \pi(k)^2 = m \cdot \frac{1}{m^2} = \frac{1}{m}.$$

Substituting this into the bound gives

$$\|g_\pi - \bar{g}\|^2 \leq B^2 \left(\rho + \frac{1-\rho}{m} \right).$$

□

Interpretation. For the schedules used in § 4.3, S corresponds to a window of nearby loop depths around the input length, e.g. $K \sim \#\text{digits} + \text{Unif}(-w, w)$. The proposition shows that if nearby depths share a common useful update direction and differ mainly through weakly aligned depth-specific residuals, then averaging over a larger window suppresses the depth-specific part of the expected update. This provides a possible mechanism for reduced OOD variance across runs.

The assumption is intentionally local. It is not expected to hold for arbitrary loop depths or arbitrarily large windows. Moreover, the proposition only explains stabilisation of the expected update around $\bar{g}$; it does not imply that $\bar{g}$ is an extrapolating direction. Therefore, lower variance under stochastic schedules need not imply higher mean OOD accuracy.

G. Toy Analysis for Loop-Consistent Representations

This appendix formalises the intuition that randomised schedules can encourage hidden states at different loop depths to become consistent. The result is a sufficient-condition argument: it does not claim that low loss always identifies a unique hidden representation, but shows that if successful predictions at different depths require closeness to a shared representation, then training across multiple depths forces those states to be close to one another.

Let $H_k^\theta(x) \in \mathbb{R}^{n \times d}$ be the full hidden state after k loop iterations, and let $L_k(\theta; x, y) = \ell(\text{Readout}(H_k^\theta(x)), y)$ be the prediction loss at depth k. Let $S \subseteq \{1, \ldots, T\}$ be a finite set of loop depths, and consider the randomised-depth objective

$$J_S(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\frac{1}{|S|} \sum_{k \in S} L_k(\theta; x, y) \right].$$

Assumption G.1 (Shared target representation)**.** For every input x, there exists a representation $Z^\star(x) \in \mathbb{R}^{n \times d}$ and a constant $\mu > 0$ such that, for every $k \in S$,

$$L_k(\theta; x, y) \geq \frac{\mu}{2} \|H_k^\theta(x) - Z^\star(x)\|_F^2.$$

Proposition G.2 (Randomised schedules encourage loop-consistent states)**.** *Under Assumption G.1, if $J_S(\theta) \leq \varepsilon$, then*

$$\mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\frac{1}{|S|^2} \sum_{i,j \in S} \|H_i^\theta(x) - H_j^\theta(x)\|_F^2 \right] \leq \frac{8\varepsilon}{\mu}.$$

Proof. By Assumption G.1,

$$J_S(\theta) = \mathbb{E}_{(x,y)} \left[\frac{1}{|S|} \sum_{k \in S} L_k(\theta; x, y) \right] \geq \frac{\mu}{2} \mathbb{E}_{(x,y)} \left[\frac{1}{|S|} \sum_{k \in S} \|H_k^\theta(x) - Z^\star(x)\|_F^2 \right].$$

Therefore, if $J_S(\theta) \leq \varepsilon$, then

$$\mathbb{E}_{(x,y)} \left[\frac{1}{|S|} \sum_{k \in S} \|H_k^\theta(x) - Z^\star(x)\|_F^2 \right] \leq \frac{2\varepsilon}{\mu}.$$

For any $i, j \in S$,

$$\|H_i^\theta(x) - H_j^\theta(x)\|_F^2 = \|(H_i^\theta(x) - Z^\star(x)) - (H_j^\theta(x) - Z^\star(x))\|_F^2$$

and therefore

$$\|H_i^\theta(x) - H_j^\theta(x)\|_F^2 \le 2\|H_i^\theta(x) - Z^\star(x)\|_F^2 + 2\|H_j^\theta(x) - Z^\star(x)\|_F^2.$$

Averaging over all pairs $(i, j) \in S \times S$ gives

$$\frac{1}{|S|^2} \sum_{i,j \in S} \|H_i^\theta(x) - H_j^\theta(x)\|_F^2 \le \frac{4}{|S|} \sum_{k \in S} \|H_k^\theta(x) - Z^\star(x)\|_F^2.$$

Taking expectation over $(x, y) \sim \mathcal{D}$ and using the previous bound yields

$$\mathbb{E}_{(x,y)} \left[\frac{1}{|S|^2} \sum_{i,j \in S} \|H_i^\theta(x) - H_j^\theta(x)\|_F^2 \right] \le 4 \cdot \frac{2\varepsilon}{\mu} = \frac{8\varepsilon}{\mu}.$$

□

Interpretation. The proposition shows that, under a shared-representation assumption, low loss across a stochastic window implies consistency of hidden states within that window. This provides a toy explanation for why stochastic schedules can reduce prediction changes across adjacent loop counts. The assumption is intentionally strong: cross-entropy loss does not generally identify a unique hidden representation. The result should therefore be interpreted as a sufficient mechanism, not as a universal guarantee.