Recursive Scaling in Masked Diffusion Models

Alba Carballo-Castro ALBA.CARBALLOCASTRO@EPFL.CH
LTS4, EPFL, Lausanne, Switzerland

Julianna Piskorz JP2048@CAM.AC.UK
University of Cambridge, Cambridge, UK

Paulius Rauba PR501@CAM.AC.UK
University of Cambridge, Cambridge, UK

Mihaela van der Schaar MV472@DAMTP.CAM.AC.UK
University of Cambridge, Cambridge, UK

Pascal Frossard PASCAL.FROSSARD@EPFL.CH
LTS4, EPFL, Lausanne, Switzerland

Abstract

Masked diffusion models (MDMs) have recently emerged as a promising paradigm for sequence generation. Scaling MDMs is conventionally achieved by increasing the parameter count or the number of denoising steps. We introduce Recursive Masked Diffusion Models (R-MDMs), which add *recursive depth* as a third scaling axis by repeatedly applying the same denoising transformer within each diffusion step. Recursion enables iterative refinement of the output through parameter reuse, increasing effective model depth without increasing parameter count. Across structured generation tasks, including Sudoku and Countdown, we show that R-MDMs achieve substantially improved parameter efficiency: a model with L recursive iterations often matches the performance of non-recursive baselines with roughly $L\times$ more parameters. Moreover, recursive refinement can partially substitute for additional denoising steps, allowing recursive models to reach *the same generation quality with fewer forward passes* at inference time. These results suggest that recursive depth is a practically useful scaling mechanism for MDMs, improving both parameter efficiency and the allocation of test-time compute.

1. Introduction

Masked diffusion models (MDMs) have emerged as a compelling paradigm for discrete sequence generation: at each denoising step, the model attends bidirectionally to the entire (partially masked) sequence and predicts all masked tokens simultaneously, in contrast to autoregressive (AR) models which generate tokens one by one. Recent work has scaled MDMs from small models [1, 27] to billions of parameters [25] by increasing transformer depth and width, but parameter count is not the only axis along which a model can be made more capable. We argue that MDMs are particularly well-suited to benefit from *recursive depth*—repeatedly applying the same transformation to an intermediate representation [7, 14, 29]. First, each recursive loop in an MDM operates with full bidirectional attention, enabling global and parallel information exchange across the entire sequence, making recursion potentially far more effective per iteration than in the left-to-right AR setting. Second, MDM performance already scales with the number of denoising steps [1, 8], suggesting that recursive loops within a step and denoising steps across the diffusion trajectory may be *partially interchangeable* axes of computation. We organize our study around two questions:

© A. Carballo-Castro, J. Piskorz, P. Rauba, M.v.d. Schaar & P. Frossard.

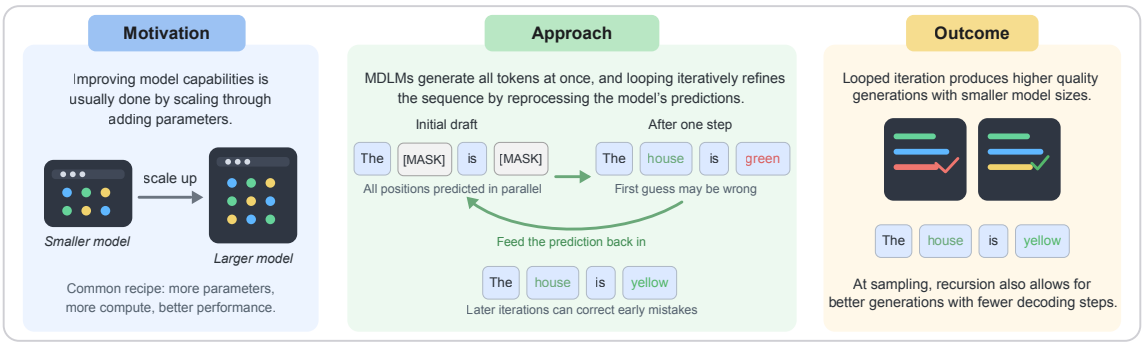

Figure 1: Standard capability gains often come from scaling model size, whereas our approach improves generation by looping an MDLM over its own predictions.

RQ$_1$ Can recursive depth substitute for parameter scaling in MDMs?
We ask whether explicitly training the denoising network to be applied recursively, with weights shared across loops, can allow recursive MDMs to match the performance of substantially larger non-recursive models. We study this across tasks of varying difficulty, considering alternative loss modes, loop-index embeddings, and loop-count scheduling, and by comparing against iso-parameter and iso-FLOP baselines.

RQ$_2$ Can recursion effectively reduce the number of denoising steps?
Given that recursion helps, we ask whether recursive loops can partially replace denoising steps: can a heavily-looped model reach the same generation quality as a single-pass model with *fewer* denoising steps, potentially leading to inference-time compute savings?

2. Related Work

Masked diffusion models have been scaled from small architectures [1, 27] to billions of parameters [25], with the number of denoising steps serving as the primary axis for trading inference speed against generation quality—but recursion as a complementary axis has not been explored. In the autoregressive setting, looped transformers [7, 14, 29] have shown that a k-layer block looped L times can match a kL-layer model at a fraction of the parameter cost; however, each loop remains a left-to-right pass, and propagating information across a length-n sequence may require up to $\mathcal{O}(n)$ sequential loops [9]. MDMs break this constraint: because each loop applies full bidirectional attention, all token positions interact simultaneously within a single pass, making each recursive loop a global constraint-propagation step rather than an incremental positional advance. We defer a comprehensive discussion of discrete diffusion variants, looped architectures, and latent reasoning in both autoregressive and diffusion models to Appendix A.

3. Method

In masked discrete diffusion, a forward process corrupts sequences by independently masking tokens according to a noise schedule, and a bidirectional transformer denoiser is trained to reconstruct the original tokens at masked positions. At inference time, T denoising steps progressively unmask the sequence; we treat T as an explicit experimental variable. We denote a K-layer shared block applied L times as $(K \otimes L)$: effective depth is $K \times L$, but parameter count equals that of a K-layer model.

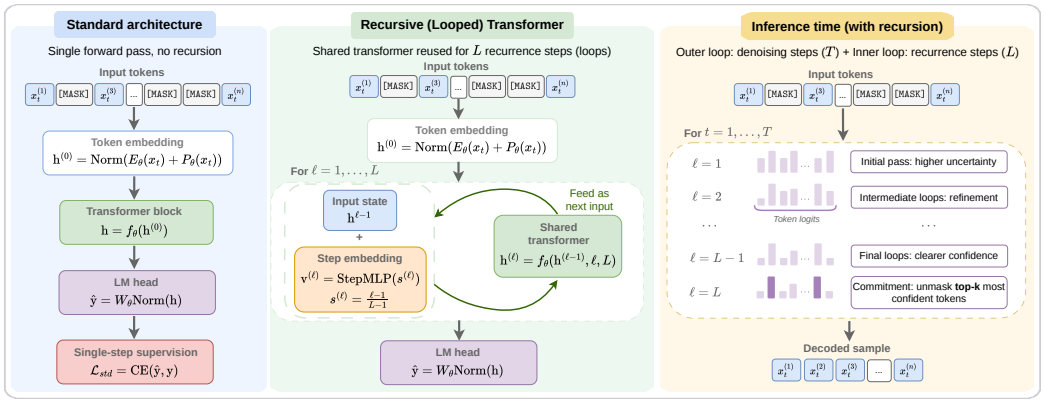

Figure 2: Comparison between the standard one-pass MDLM and the recursive variant. The recursive model reuses the same transformer blocks across refinement steps, optionally conditioned on a step embedding. At inference, there is a recursive loop within each denoising step to refine predictions.

Recursive architecture. Rather than applying a KL-layer transformer once, our model applies a K-layer block f_θ with *shared weights* L times in sequence, passing the output hidden state of each loop as input to the next (with RMS normalization between loops). The model therefore has exactly the same parameter count as a single-pass $(K \otimes 1)$ baseline. For iso-FLOP comparisons, a $(KL \otimes 1)$ baseline applies KL *distinct* layers once, performing the same floating-point operations but using $L\times$ more parameters. See Appendix B for further methodological details.

Step embedding. To signal loop position, we inject a learned embedding computed from the normalized loop progress $s_\ell = (\ell - 1)/(L - 1) \in [0, 1]$ into the hidden state before each loop. Using normalized rather than raw indices keeps the embedding well-conditioned at loop counts unseen during training, and allows early loops to specialize differently from late loops.

Training objectives. We consider two losses over the L sets of per-loop logits. The FINAL loss supervises only the last loop's output, relying on backpropagation to train earlier loops. The ALL loss averages cross-entropy across all L loop outputs, providing direct gradient signal at every loop [21, 43]; this increases training memory but not forward-pass compute. We additionally study WEIGHTED and TRUNCATED variants (full details in Appendix E.2), but adopt ALL as the default throughout, as it consistently outperforms the alternatives.

Loop-count scheduling. During training, the number of recursive iterations L_t is either kept fixed; deterministically varied over training according to a predefined progression from an initial to a final value; or sampled from a predefined distribution. This allows the model to experience varying computation depths during optimization (see Appendix E.1 for further details).

4. Experiments

4.1. Setup

We evaluate on three benchmarks of increasing structural diversity: **Sudoku** (9×9 and 25×25 constraint satisfaction, reporting Valid Puzzle Rate (VPR)), **Countdown** (synthetic arithmetic planning with $k \in \{3, 4, 5\}$ operands, reporting Reaches Target Rate (RTR)), and **Text8** (character-level language modelling, reporting NLL and Generative Perplexity). Full dataset construction, metric

definitions, and training hyperparameters are in Appendices C and D. Unless stated otherwise, all models use the ALL loss and step embedding; results are reported as means over 5 sampling runs of 100 samples each.

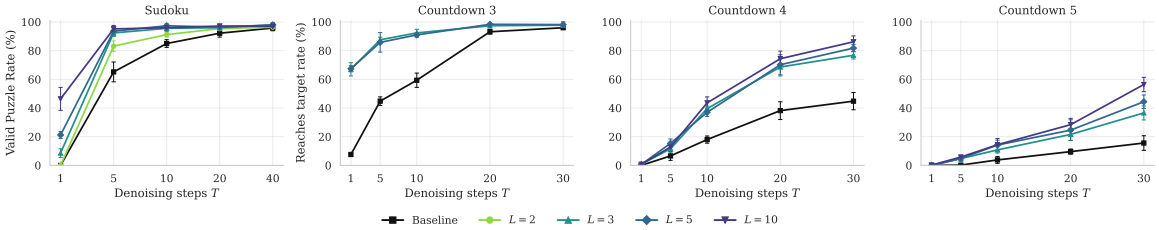

Figure 3: Effect of recursive refinement depth (L) on task performance for Sudoku 9×9 and Countdown across different target lengths (3, 4, and 5 digits). Increasing L consistently improves success rates, with the strongest gains for more difficult problems.

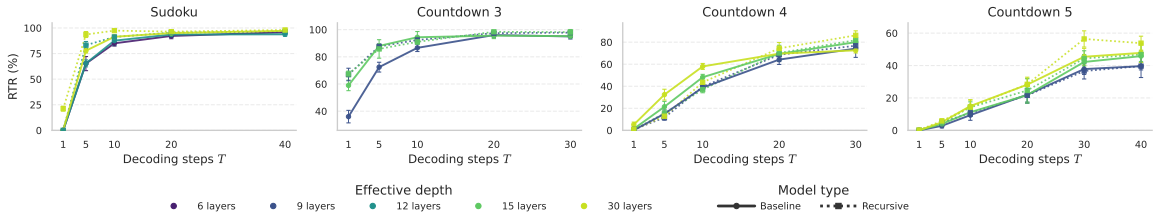

Figure 4: Effect of recursion at matched effective model depth (iso-FLOP) for Sudoku 9×9 and Countdown tasks of different target length (3, 4, and 5 digits).

4.2. RQ_1: Can Recursive Depth Substitute for Parameter Scaling?

Iso-parameter comparison. Figure 3 plots VPR/RTR as a function of T for loop counts $L \in \{1, 2, 3, 5, 10\}$, with all models sharing the same parameter budget. On Sudoku $9{\times}9$, even $L{=}2$ substantially outperforms the single-pass baseline, raising VPR from 65.2% to 83.2% at $T{=}5$; with $L{=}5$ the model reaches 97.4% at $T{=}10$, a quality level the baseline only achieves at $T{=}40$. Recursion provides even larger gains on the harder multi-step Countdown task: at $T{=}10$ the $L{=}3$ model reaches 92.4% RTR on Countdown-3 versus 59.4% for the baseline, and for Countdown-4 the gap grows further ($L{=}10$ at $T{=}30$: 86.2% vs. 44.8%). The benefit of recursion scales with task difficulty: harder operand counts and higher masking ratios yield larger absolute gains (full $25{\times}25$ Sudoku results in Appendix F.4).

Iso-FLOP comparison. Figure 4 compares $(K \otimes L)$ recursive models against $(KL \otimes 1)$ baselines that share the same effective depth but use $L\times$ more parameters. On Sudoku (Figure 4, left), the $(6 \otimes 5)$ model (10.6M) achieves 93.6% VPR at $T{=}5$, matching or exceeding all single-pass baselines including models $5\times$ larger; the gap narrows as T grows, confirming that additional denoising steps can eventually compensate for absent recursion, but at a substantially higher sampling-time cost. On Countdown-4 (medium difficulty), $(3 \otimes 5)$ and $(3 \otimes 10)$ reach 70.2% and 74.4% RTR at $T{=}20$, beating the iso-FLOP 15-layer model (69.4%) and matching the 30-layer model (69.6%) with $5\times$–$10\times$ fewer parameters. Full iso-FLOP tables are provided in Appendix F.

Generalisation across domains. On unstructured character-level modelling, recursive models underperform the baseline on NLL (4.742 vs. 5.429 for L=3 at T=18); however, qualitative samples suggest looped models produce more coherent text despite worse likelihood scores (Appendix F.6).

4.3. RQ$_2$: Can Recursion Reduce the Number of Denoising Steps?

Cross-recursion flexibility. Before investigating the compute trade-off between recursive loops and denoising steps, we first verify that a model trained with L_t loops can be effectively deployed with a different number of loops L_s at inference time. Figure 11 shows that this flexibility holds: for one-shot generation (T=1), increasing L_s above L_t consistently improves performance, and moderate extrapolation still helps at T=5. Performance degrades only when $L_s \gg L_t$ or $L_s \ll L_t$, suggesting that representations are calibrated to converge in approximately L_t loops. This decoupling of training and sampling loops is what enables the continuous compute trade-off we investigate next.

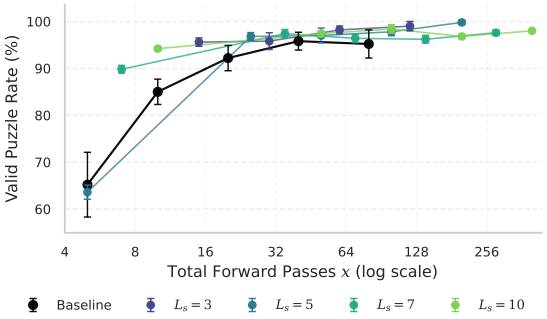

Figure 5: Valid Puzzle Rate versus total computational budget (forward passes $x = T \times L$) on Sudoku 9×9.

Compute-efficient inference. Figure 5 shows that for a fixed total forward-pass budget $x = T \times L_s$, allocating computation to recursive depth can be more efficient than scaling denoising steps alone. At x=10, the baseline (T=10) achieves 85.0% VPR, whereas the looped model with L_s=10, T=1 reaches 94.2%—a 9.2% absolute improvement under identical compute. At x=15, the configuration L_s=3, T=5 achieves 95.6%, matching the baseline at x=40 (95.8%), corresponding to a 2.7× reduction in compute. Beyond efficiency, recursion also raises the performance ceiling: while the baseline saturates at ≈95.8% even at x=80, recursive models consistently exceed this limit, reaching 97.4%–98.2% at moderate depths.

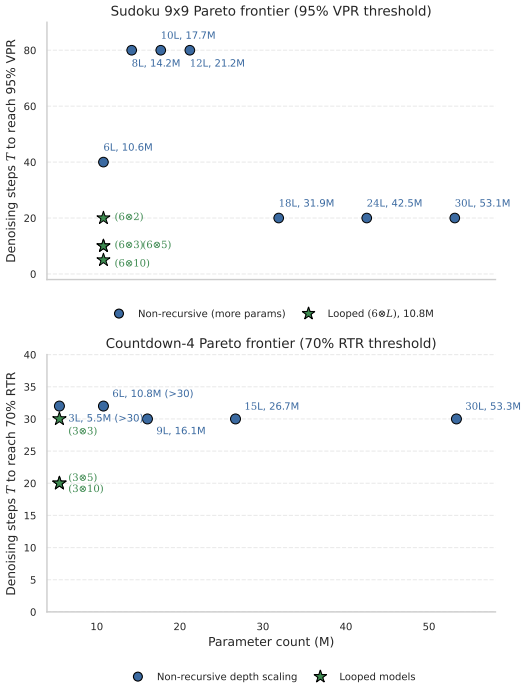

Figure 6: Parameter-step Pareto frontier between model parameter count and the number of decoding steps T needed to reach 95% VPR on Sudoku 9×9 and 70% RTR on Countdown-4.

Parameter–denoising step Pareto frontier. Figure 6 reports, for each model configuration, the minimum number of denoising steps T required to reach 95% VPR on Sudoku 9×9 and 70% RTR on Countdown-4, plotted against parameter count. On Sudoku, the $(6 \otimes 3)$ model (10.6M) crosses 95% VPR at T=10, while the non-recursive 6-layer baseline (10.6M) never reaches this threshold; achieving it without recursion requires at least 18 layers (31.9M, 3× more parameters). The iso-FLOP Pareto frontier is *flat* beyond 18 layers,

whereas the recursive frontier continues to improve at fixed T with a fraction of those parameters. $(6 \otimes 5)$ and $(6 \otimes 10)$ both reach 95% at $T{=}10$ and $T{=}5$ respectively, showing that more loops simultaneously reduce the required step count. On Countdown-4, the $(3 \otimes 3)$ model (5.5M) matches the 9-layer iso-FLOP baseline (16.1M, $3\times$ more parameters) at $T{=}30$, and with $L{=}5$ exceeds the 15-layer baseline (26.7M) at $T{=}20$.

4.4. Ablations

Below we present ablations on the main design choices of our recursive architecture. All experiments are run on Sudoku $9{\times}9$, with further details in Appendix E.

Recursive step choice. We compare fixed recursion depth, stochastic sampling, and curriculum-based schedules. As shown in Figure 7 (Appendix E.1), a reverse curriculum (*Schedule* $10 \rightarrow 1$) consistently outperforms all alternatives across evaluation depths and decoding budgets, with the largest gains in the low-compute regime.

Training objective. We compare the four supervision strategies for recursive training. As shown in Figure 9 (Appendix E.2), ALL consistently achieves the strongest and most robust performance and is selected as our default. This suggests that dense per-step supervision is important for aligning the recursive trajectory with the denoising objective.

Step embedding. Figure 10 (Appendix E.3) compares learned, fixed, and no step embeddings (Eq. 8). Learned embeddings provide the most consistent performance across evaluation depths and achieve the strongest overall results, becoming our default, while fixed embeddings generally underperform both alternatives.

5. Conclusions, Limitations, and Future Work

In this work, we explored the role of recursion in masked diffusion models, in which a shared transformer block is applied L times per denoising step without adding parameters and can be trained with different supervision signals. Across structured reasoning tasks, we find that recursive depth acts as a dual mechanism for compute allocation: it can substitute both parameter scaling and denoising-step scaling, yielding comparable or better performance than significantly larger non-recursive models while also reducing the number of denoising steps required to reach a given quality threshold. These gains grow with task difficulty, suggesting that recursion acts as a mechanism for iterative constraint propagation whose benefits compound in structured regimes. In contrast, for unstructured character-level language modeling, the benefit of recursion is still unclear and is left to future work.

The main limitations of this work are its small scale experiments (up to $\sim$50M parameters and 625-token sequences), the $L\times$ training compute overhead of the all-steps loss, and the tight calibration of learned representations to the training loop count, which limits flexibility at inference time. Along these lines, important directions for future work include scaling looped MDMs to larger-scale models to test whether the parameter–loop trade-off persists at the scale of modern language models; a mechanistic investigation of what iterative computation the shared block actually implements across loops; and a systematic study of whether recursive gains extend to tasks that lie between the fully constrained and fully unstructured regimes, such as code or mathematical reasoning, where exploitable structure is present but weaker than in Sudoku.

Use of Large language Models

Large Language Models (LLMs) were used as a coding and writing assistant tool in the preparation of this manuscript and its experiments. Specifically, they were employed to aid in writing, checking spelling, and editing for clarity. They were also used to assist in coding and plotting of the results. Nonetheless, all text and code produced with the assistance of LLMs was carefully reviewed, verified, and revised by the authors to ensure accuracy and appropriateness. The use of LLMs was limited to these support functions, and the authors take full responsibility for the final contents.

Impact Statement

The primary objective of this paper is to advance discrete generative models with a more parameter-efficient architecture. By reducing the model size and number of denoising steps required to achieve high-quality generation, recursive masked diffusion models could lower the computational cost of deploying sequence generation systems, with potential benefits in resource-constrained settings. While more capable and accessible generative models carry the general dual-use concerns associated with language generation research, we do not foresee any immediate societal concerns specific to the methodology proposed here.

References

[1] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://arxiv.org/abs/2107.03006.

[2] Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed Recursive Transformers: Effective Parameter Sharing with Layer-wise LoRA. *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.20672.

[3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://arxiv.org/abs/1909.01377.

[4] Xingjian Bai and Luke Melas-Kyriazi. Fixed Point Diffusion Models. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. URL http://arxiv.org/abs/2401.08741.

[5] Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.04997.

[6] Chen-Hao Chao, Wei-Fang Sun, Hanwen Liang, Chun-Yi Lee, and Rahul G. Krishnan. Beyond masked and unmasked: Discrete diffusion models via partial masking. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2505.18495.

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

[8] Justin Deschenaux and Caglar Gulcehre. Promises, outlooks and challenges of Diffusion Language Modeling. *arXiv preprint arXiv:2406.11473*, 2024. URL https://arxiv.org/abs/2406.11473.

[9] Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. *arXiv preprint arXiv:2409.15647*, 2024. URL https://arxiv.org/abs/2409.15647.

[10] David Fox, Sam Bowyer, Song Liu, Laurence Aitchison, Raul Santos-Rodriguez, and Mengyue Yang. Learning generation orders for MDLM via variational inference. *arXiv preprint arXiv:2602.23968*, 2026. URL https://arxiv.org/abs/2602.23968.

[11] Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (SoS): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024. URL https://arxiv.org/abs/2404.03683.

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

[13] Zhengyang Geng, Ashwini Pokle, and J. Zico Kolter. One-Step Diffusion Distillation via Deep Equilibrium Models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2401.08639.

[14] Angeliki Giannou, Shashank Rajput, Jy yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. *International Conference on Machine Learning (ICML)*, 2023. URL https://arxiv.org/abs/2301.13196.

[15] Andre He, Sean Welleck, and Daniel Fried. Reasoning with latent tokens in diffusion language models. *arXiv preprint arXiv:2602.03769*, 2026. URL https://arxiv.org/abs/2602.03769.

[16] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2006.11239.

[17] Hanzhang Hu, Debadeepta Dey, Martial Hebert, and J. Andrew Bagnell. Learning anytime predictions in neural networks via adaptive loss balancing. *arXiv preprint arXiv:1708.06832*, 2018. URL https://arxiv.org/abs/1708.06832.

[18] Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-Depth Looped Transformers for Latent Reasoning via Shortcut Modulation. *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2602.11451.

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

[20] Chanhyuk Lee, Jaehoon Yoo, Manan Agarwal, Sheel Shah, Jerry Huang, Aditi Raghunathan, Seunghoon Hong, Nicholas M. Boffi, and Jinwoo Kim. Flow map language models: One-step language modeling via continuous denoising. *arXiv preprint arXiv:2602.16813*, 2026. URL https://arxiv.org/abs/2602.16813.

[21] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014. URL https://arxiv.org/abs/1409.5185.

[22] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2310.16834.

[23] Matt Mahoney. Text8: Large text compression benchmark, 2006. URL http://mattmahoney.net/dc/textdata.

[24] Nesta Midavaine, Christian A. Naesseth, and Grigory Bartosh. Towards latent diffusion suitable for text. *EuRIPs 2025 Workshop on Principles of Generative Modeling*, 2025. URL https://arxiv.org/abs/2601.16220.

[25] Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2502.09992.

[26] Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y. Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026. URL https://arxiv.org/abs/2604.12946.

[27] Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2406.07524.

[28] Subham Sekhar Sahoo, Jean-Marie Lemercier, Zhihan Yang, Justin Deschenaux, Jingyu Liu, John Thickstun, and Ante Jukic. Scaling beyond masked diffusion language models. *arXiv preprint arXiv:2602.15014*, 2026. URL https://arxiv.org/abs/2602.15014.

[29] Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2502.17416.

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

[31] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015. URL http://arxiv.org/abs/1503.03585.

[32] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2011.13456.

[33] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2023. URL https://arxiv.org/abs/2104.09864.

[34] Anej Svete and Ashish Sabharwal. On the reasoning abilities of masked diffusion language models. *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2510.13117.

[35] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 billion parameter autoregressive language model, 2021. URL https://github.com/kingoflolz/mesh-transformer-jax.

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

[37] Kevin Xu and Issei Sato. A formal comparison between chain of thought and latent thought. *arXiv preprint arXiv:2509.25239*, 2025. URL https://arxiv.org/abs/2509.25239.

[38] Ruihan Xu, Yuting Gao, Lan Wang, Jianing Li, Weihao Chen, Qingpei Guo, Ming Yang, and Shiliang Zhang. Looping back to move forward: Recursive transformers for efficient and flexible large multimodal models. *arXiv preprint arXiv:2602.09080*, 2026. URL https://arxiv.org/abs/2602.09080.

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

[40] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.10601.

[41] Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.14157.

[42] Chengting Yu, Xiaobo Shu, Yadao Wang, Yizhen Zhang, Haoyi Wu, You Wu, Rujiao Long, Ziheng Chen, Yuchi Xu, Wenbo Su, and Bo Zheng. SpiralFormer: Looped Transformers Can Learn Hierarchical Dependencies via Multi-Resolution Recursion. *arXiv preprint arXiv:2602.11698*, 2026. URL http://arxiv.org/abs/2602.11698.

[43] Qifan Yu, Zhenyu He, Sijie Li, Xun Zhou, Jun Zhang, Jingjing Xu, and Di He. Enhancing autoregressive chain-of-thought through loop-aligned reasoning. *arXiv preprint arXiv:2502.08482*, 2025. URL https://arxiv.org/abs/2502.08482.

[44] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://arxiv.org/abs/1910.07467.

[45] Huangjie Zheng, Shansan Gong, Ruixiang Zhang, Tianrong Chen, Jiatao Gu, Mingyuan Zhou, Navdeep Jaitly, and Yizhe Zhang. CADD: Continuous-absorbing discrete diffusion. *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2510.01329.

[46] Cai Zhou, Chenxiao Yang, Yi Hu, Chenyu Wang, Chubin Zhang, Muhan Zhang, Lester Mackey, Tommi Jaakkola, Stephen Bates, and Dinghuai Zhang. Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner. *arXiv preprint arXiv:2510.03206*, 2025. URL https://arxiv.org/abs/2510.03206.

Appendix A. Extended Related Work

Section 2 covers the two threads most directly relevant to our contributions. This appendix broadens the context along four additional dimensions: the wider discrete-diffusion landscape that situates our choice of MDMs (§A.1), an introduction to looped and recurrent architectures (§A.2), and the growing literature on recurrent and latent reasoning in autoregressive models (§A.3) and in diffusion models themselves (§A.4).

A.1. The Landscape of Discrete Diffusion for Language

The discrete diffusion family has grown rapidly. The D3PM framework [1] provided the absorbing-state (masking), uniform, and nearest-neighbor transition variants inspired from continuous settings [16, 31, 32]. SEDD [22] proposed score entropy as a novel loss that generalizes score matching to discrete spaces, achieving competitive performance with autoregressive models on language benchmarks. Discrete Flow Models [5] showed that discrete diffusion arises as a special case of continuous-time Markov chains, unifying diffusion and flow matching and enabling flexible multimodal generation.

Among masked approaches, MDLM [27] demonstrated strong perplexity with a simplified Rao-Blackwellized objective. LLaDA [25] scaled to 8B parameters, matching LLaMA 3 8B on most benchmarks. Subsequent scaling law studies [28] challenge the assumption that masked diffusion is the uniquely best discrete noise type, finding that uniform diffusion may be more parameter-efficient at scale, raising open questions about which noise type is optimal in each regime. Continuous–discrete hybrids have also been proposed [20, 45, 46], but tend to suffer from trainability issues in the continuous space. Finally, Midavaine et al. [24] briefly explores latent diffusion for text, identifying trainability as the core bottleneck when operating in continuous embedding space. This supports our decision to remain in the masked discrete setting, where training is stable and the parallel denoising structure makes bidirectional recursion most natural.

Recent work has also studied generation *order* in masked models. Chao et al. [6] explore partial masking strategies that interpolate between absorbing and uniform noise, and Fox et al. [10] study how to learn optimal generation orders via variational inference. Our recursive architecture is orthogonal to these concerns: the same looped denoising network can be combined with any masking schedule or ordering policy.

A.2. Implicit Depth, Recurrent, and Looped Architectures

The search for parameter efficiency has led to an increased interest in architectures with *implicit depth*, where a single block is applied repeatedly rather than stacking distinct layers. Universal Transformers [7] introduced this for sequence modeling; more recent looped Transformer architectures [18, 26, 42], including variants with relaxed causal attention [2], leverage shared weights to simulate the capacity of much deeper models at a fraction of the parameter cost, the principle we import into the MDM setting.

Deep Equilibrium Models (DEQ) [3] formalize iterative refinement by finding the fixed point of a shared layer via implicit differentiation. In the generative setting, the Generative Equilibrium Transformer [13] and Fixed-Point Diffusion Models [4] embed this idea inside the generative process itself, a principle our work extends to discrete masked diffusion, where training stability and bidirectional structure make explicit looping more natural than implicit solvers.

Closest in spirit to our RQ_2 are HRM [36] and TRM [19], which demonstrate that recursive computation can unlock capabilities far beyond what parameter count alone would predict. Both operate in discriminative or sequence-to-sequence regimes, however, and neither addresses whether inner loops can substitute for outer denoising iterations, a trade-off that only arises in a generative framework with an outer iterative axis alongside the inner recursive one.

Our recursive MDM diverges from all these approaches along three dimensions. Unlike the causal or relaxed-causal attention used in the looped architectures above, each loop applies full bidirectional attention, enabling global constraint propagation across all positions simultaneously. Unlike continuous fixed-point solvers, our architecture must handle the non-uniform token states of discrete diffusion (masked, newly decoded, or clean) as they evolve across steps. Most distinctively, we treat the recursive loop count and the denoising step count as two *explicit, interchangeable axes of compute*, a joint design space that none of the above works explores.

A.3. Recurrent and Latent Reasoning in Autoregressive Models

While the previous section focused on the architectural mechanics of looping, we explore how these loops function as a substrate for latent reasoning and survey the AR looped-transformer literature more closely.

Geiping et al. [12] train a 3.5B-parameter recurrent-depth AR model that improves at test time by using additional loops, reaching effective compute equivalent to a 50B-parameter single-pass model. One key finding from their work translates directly to ours: harder tasks saturate at higher loop counts. We observe the same in the MDM setting: harder Countdown variants and higher Sudoku masking ratios both demand more loops before performance plateaus. Notably, [12] also demonstrate flexible test-time compute scaling by varying loops after training, which directly motivates our cross-recursion trade-off analysis (Figure 11), where we decouple training loop count L_t from sampling loop count L_s.

On the other hand, Fan et al. [9] demonstrate superior length generalization in looped transformers for tasks with known iterative structure, observing that required loops scale with the complexity of the iterative algorithm and longer sequences require more loops, matching algorithm iteration counts. This suggests a principled mapping between problem difficulty and optimal loop count—a connection we observe empirically across Sudoku board sizes and Countdown operand counts.

Yang et al. [39] established empirical evidence that looped transformers can match standard in-context learners with under 10% of the parameters. Additionally, Yu et al. [43] (RELAY) showed that per-iteration supervision enables looped AR models to generalize beyond their training sequence lengths. Both inform our design: the parameter efficiency of looping motivates RQ_2, while RELAY's per-step supervision directly inspires our all-steps loss $\mathcal{L}_{\text{all}}$ (Eq. 16), which provides direct gradient signal at every loop rather than relying entirely on backpropagation through the full loop chain.

Finally, Xu and Sato [37] provide the formal basis for understanding why latent thought is more efficient than CoT for parallelizable problems. Their separation result—latent thought in looped transformers enables efficient parallel computation for directed acyclic graph evaluation—maps directly onto the kind of global constraint propagation that is natural in an MDM with bidirectional attention. It also supports our hypothesis that MDM loops are qualitatively more powerful per iteration than their AR counterparts.

A.4. Reasoning in Diffusion Models and the Latent Token View

Ye et al. [41] (MGDM) show that masked diffusion substantially outperforms AR models on structured tasks including Countdown, Sudoku, and 3-SAT. They attribute this to the model learning which subgoals are hard via the masking objective, and propose a multi-granularity loss that prioritizes harder positions. This finding provides the empirical baseline on which our work builds: if MDMs already excel at structured tasks relative to AR models, and recursion further amplifies their capacity for constraint propagation, then structured generation is precisely where we expect the largest gains, a prediction our experiments confirm. The difficulty-dependent gap between recursive and non-recursive models (larger for Countdown-5 than for Countdown-3, larger at 90% masking than at 70%) directly echoes [41] intuition that harder subgoals benefit most from richer computation.

He et al. [15] identify masked tokens as latent computational states in MDM generation. Their central finding is that joint prediction over undecoded tokens—which MDMs perform naturally—acts as implicit parallel reasoning. They introduce a semi-causal diffusion model (SCDM) that interpolates between independent and joint prediction, enabling control over the quality–speed trade-off. This is complementary to our approach along a distinct axis: while SCDM controls *which* token positions participate in joint prediction, we control *how many times* the shared transformer block processes the full joint hidden state. A natural direction for future work is to combine both mechanisms, tuning the scope of joint prediction and the depth of iterative refinement, within a single model.

Finally, Svete and Sabharwal [34] provide the theoretical anchor for both RQ_1 and RQ_2. As discussed in Section 2, their MDM–PLT equivalence shows that MDMs with $O(\log N)$ steps can match chain-of-thought transformers requiring $O(N)$ tokens, by replacing sequential generation with parallel computation. Our recursive loops extend this picture: within each denoising step, L loops of a k-layer transformer implement the representational capacity of kL layers, and because each loop uses full bidirectional attention, this capacity is applied globally rather than locally. The combination of a logarithmically efficient outer diffusion trajectory and a deeply recursive inner denoiser suggests a route to highly capable MDMs that remain tractable in both parameter count and sampling budget.

Appendix B. Methodological Details

This appendix provides the full formal treatment of the recursive MDM architecture, expanding on the condensed presentation in Section 3.

B.1. Masked Discrete Diffusion

Diffusion models generate data by learning to iteratively reverse a noising process. In the discrete setting, this process corrupts sequences by masking tokens, and generation proceeds by progressively denoising a fully masked sequence. Let $x_0 = (x_0^{(1)}, \ldots, x_0^{(n)}) \in \mathcal{V}^n$ be a sequence of n tokens from a finite vocabulary $\mathcal{V}$. The absorbing-state forward process independently masks each token:

$$q(x_t \mid x_0) = \prod_{i=1}^{n} \text{Cat}\big(x_t^{(i)}; \alpha_t\, x_0^{(i)} + (1-\alpha_t)\, \mathbf{e}_{\texttt{[MASK]}}\big), \tag{1}$$

where $t \in [0,1]$, and $\alpha_t \in [0,1]$ is a monotone noise schedule with $\alpha_0 = 1$ (no noise) and $\alpha_1 = 0$ (fully masked).

The denoising network $p_\theta(x_0 \mid x_t)$ is parameterized by a bidirectional transformer and trained to maximize the ELBO

$$\mathcal{L}_{\text{ELBO}}(\theta) = \mathbb{E}_{t,x_0,x_t}\left[\sum_{i:\, x_t^{(i)} = \texttt{[MASK]}} \log p_\theta\Big(x_0^{(i)} \mid x_t\Big)\right], \tag{2}$$

where the sum runs over the masked positions in x_t, and the expectation is over $t \sim \mathcal{U}[0,1]$, data $x_0 \sim p_{\text{data}}$, and the forward process $x_t \sim q(\cdot \mid x_0)$. At inference time, the generative process iterates the denoising network for T denoising steps, progressively unmasking the sequence from x_1 (fully masked) to x_0 (fully unmasked). We treat T as an explicit experimental variable throughout (**RQ$_2$**).

Notation summary. We use n to denote sequence length, d the hidden dimension of the transformer, H the number of attention heads, K the number of transformer layers in the shared block, L the number of recursive loops applied to that block per denoising network call, and T the number of denoising steps at sampling time. The effective depth of our network is therefore $K \times L$, while its parameter count equals that of a K-layer model. We use the shorthand $(K \otimes L)$ for this configuration, mirroring the notation of Saunshi et al. [29].

B.2. Recursive Architecture

The denoising network of a standard MDM applies a KL-layer transformer once to x_t and decodes logits from the final hidden state. Our model instead uses a K-layer transformer block $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ with shared weights, applying it L times in sequence. Concretely, given a noisy input $x_t \in \mathcal{V}^n$, the forward pass is:

$$\mathbf{h}^{(0)} = \text{Norm}(E_\theta(x_t) + P_\theta(x_t))\,, \tag{3}$$

$$\mathbf{h}^{(\ell)} = f_\theta\Big(\mathbf{h}^{(\ell-1)},\ \ell,\ L\Big)\,, \qquad \ell = 1,\ldots,L, \tag{4}$$

$$\mathbf{h}_{\text{out}}^{(\ell)} = \text{Norm}\Big(\mathbf{h}^{(\ell)}\Big)\,, \tag{5}$$

$$\hat{\mathbf{y}}^{(\ell)} = W_\theta\, \mathbf{h}_{\text{out}}^{(\ell)}, \tag{6}$$

where $E_\theta : \mathcal{V}^n \to \mathbb{R}^{n\times d}$ is the token embedding, P_θ is the positional encoding (rotary or 2-D Sudoku-specific; see Appendix C), Norm denotes RMS normalization [44], $W_\theta \in \mathbb{R}^{d\times|\mathcal{V}|}$ is the shared language model head, and $\hat{\mathbf{y}}^{(\ell)} \in \mathbb{R}^{n\times|\mathcal{V}|}$ are the logits produced after loop ℓ.

The shared block f_θ is a stack of K standard pre-norm transformer layers (each consisting of multi-head self-attention followed by a position-wise MLP), with full bidirectional attention and no causal masking. Critically, the weights of f_θ and W_θ are *identical* across all L iterations: the model has exactly the same number of parameters as a single-pass $(K \otimes 1)$ baseline. The output of loop ℓ is passed as the input hidden state to loop $\ell + 1$ without any additional projection or gating; the only transformation between loops is RMS normalization of the hidden state, and therefore the only information flow between loops is through $\mathbf{h}^{(\ell)}$.

Comparison to a non-looped baseline. A $(KL \otimes 1)$ iso-FLOP baseline applies KL *distinct* transformer layers once. A model $(K \otimes L)$ applies the K-layer transformer block f_θ L times with shared parameters. Both perform the same number of floating-point operations per forward pass; only the parameter count differs by a factor of L. This is the comparison regime used throughout our experiments.

B.3. Recursive Step Embedding

In their basic form, Eqs. (3)–(4) apply f_θ repeatedly with shared parameters, where each loop operates on the evolving hidden state but otherwise receives the same conditioning inputs. The model has no explicit signal indicating which loop is currently being executed; it must infer this from the evolving statistics of $\mathbf{h}^{(\ell)}$ alone. To give the model direct access to its position in the recursive computation, we introduce a *step embedding*.

Let $s_\ell = (\ell - 1)/(L - 1) \in [0, 1]$ be the normalized loop progress (with $s_1 = 0$ for $L = 1$). We map this scalar to a d-dimensional vector via a two-layer MLP:

$$\mathbf{v}^{(\ell)} = W_2\, \sigma(W_1\, s_\ell + b_1)\,, \qquad W_1 \in \mathbb{R}^{d\times 1},\ W_2 \in \mathbb{R}^{d\times d}, \tag{7}$$

where σ denotes the SiLU activation. The step vector $\mathbf{v}^{(\ell)}$ is broadcast over the sequence and added to the hidden state before each loop application, replacing Eq. (4) with:

$$\mathbf{h}^{(\ell)} = f_\theta\Big(\mathrm{Norm}\Big(\mathbf{h}^{(\ell-1)} + \mathbf{v}^{(\ell)}\Big)\,,\ \ell,\ L\Big)\,. \tag{8}$$

The step embedding parameters (W_1, b_1, W_2) are *shared* across loops (they depend on ℓ only through s_ℓ) and add $\mathcal{O}(d^2)$ parameters to the model, which is a negligible overhead relative to the transformer block.

The step embedding serves two purposes. First, it allows the model to *specialize* its computation by loop index: early loops, which operate on a representation with high uncertainty, may learn to behave differently from late loops, which refine an already coherent prediction. Second, using the *normalized* progress s_ℓ rather than the raw index ℓ makes the embedding well-conditioned at any loop count L, including values of L not seen during training. Whether the step embedding helps is one of the training design factors we ablate (Section §4).

Appendix C. Dataset and Metrics

We evaluate our models on a diverse set of datasets spanning structured combinatorial reasoning (Sudoku), symbolic arithmetic planning (Countdown), and natural language modeling (Text8). This selection allows us to test different aspects of sequence modeling: constraint satisfaction, multi-step reasoning with intermediate state tracking, and distributional language understanding. For each dataset, we construct train and validation splits and report both training objectives and task-specific evaluation metrics. Further information on split sizes, vocabulary sizes, and sequence lengths are detailed in Table 1.

For all datasets during training, we monitor the average masked cross-entropy loss over mini-batches, computed for both the train and validation splits by averaging over `eval_iters` randomly sampled batches at the end of each evaluation interval. For a batch $\{(x_t^{(i)}, x_0^{(i)}, m^{(i)})\}_{i=1}^{B}$ of size B, the estimated validation loss is

$$\hat{\mathcal{L}}_{\text{val}} = \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}(y^{(i)};\, x_0^{(i)},\, m^{(i)}), \tag{9}$$

where $\mathcal{L}$ is whichever training objective is active.

Table 1: Summary statistics for all datasets used in our experiments. Sequence length refers to the fixed context length used during training. Vocabulary size includes special tokens such as masks where applicable.

Dataset	Train Size	Val Size	Vocab Size	Seq Length
Sudoku 9×9	1.8M	100k	10	81
Sudoku 36×36	100k	20k	26	625
Countdown ($k = 2$)	100k	20k	18	32
Countdown ($k = 3$)	100k	20k	18	48
Countdown ($k = 4, 5$)	100k	20k	18	64
Text8	90M chars	5M chars	27	256

C.1. Sudoku

Sudoku 9×9 For the standard Sudoku, we adapt the dataset from [30], which consists of pre-existing puzzle files containing solved 9×9 encoded as move sequences. Each sample in the raw data is a sequence of 81 moves, where each move records a quadruple (r, c, v, s) that indicates the row index, column index, digit value, and a strategy tag respectively. Since we are interested on training on the raw Sudokus, we preprocess the data and store the boards row-wise as flattened sequences of length 81 with digits in $\{1, \ldots, 9\}$. The vocabulary consists of digits $\{0, 1, \ldots, 9\}$, where 0 serves as the mask token.

Extended sudoku For experiments at larger scales, we construct a synthetic dataset of $n \times n$ Sudoku boards, where n is chosen such that a valid rectangular block decomposition exists. Specifically, we require $n = b_r \times b_c$ for integers $b_r, b_c \geq 2$ chosen as the factor pair closest to $\sqrt{n}$. Each board is a completed, valid Sudoku with digits in $\{1, \ldots, n\}$, stored as a flattened sequence of length n^2.

Boards are generated by first constructing a single deterministic base solution using the closed-form pattern

$$G_{r,c} = (b_c \cdot (r \bmod b_r) + \lfloor r/b_r \rfloor + c) \bmod n + 1,$$

which yields a valid completed $n \times n$ grid for any compatible block shape. Each subsequent sample is derived from this base by applying a random symmetry-preserving permutation: row-bands and rows within each band are independently shuffled, column-stacks and columns within each stack are independently shuffled, and digit labels are remapped via a random permutation of $\{1, \ldots, n\}$. All three operations preserve Sudoku validity. The vocabulary consists of digits $\{0, \ldots, n\}$, with 0 again serving as the mask token, giving a vocabulary size of $n + 1$.

We generate 100,000 boards for training and 20,000 for validation with $n = 25$. A fixed set of binary blank masks is pre-generated for the validation split, where each mask is drawn i.i.d. with a per-cell blank probability of 0.15, subject to the constraint that at least one cell is blank and at least one is given.

Metrics and evaluation

- **Valid Puzzle Rate (VPR)**: A generated $n \times n$ board $y \in \{1, \ldots, n\}^{n^2}$ is valid if and only if every row, every column, and every $b_r \times b_c$ block contains each digit in $\{1, \ldots, n\}$ exactly once. Any out-of-range digit, repeated digit in a unit, or board of incorrect length counts as invalid.

- **Soft Constraint Loss (SCL)**: This metric is adapted from [15] and allows a more fine-grained evaluation of the quality of the generated sudokus. For a board $y \in \mathbb{Z}^{n^2}$ with vocabulary $\mathcal{V} = \{1, \ldots, n\}$, it measures the fractional coverage of the required digit set for each constraint unit:
$$\ell(u; y) = 1 - \frac{|\{y_j : j \in u\} \cap \mathcal{V}|}{n}, \tag{10}$$
where u ranges over all $3n$ units (rows, columns, blocks). A perfectly valid board has $\ell(u; y) = 0$ for all u; a board where every unit contains only one distinct valid digit has $\ell(u; y) = (n-1)/n$ for all u. Out-of-vocabulary digits do not contribute to the covered set and therefore increase the loss. The maximum value $3(n-1)$ corresponds to a board where every unit contains a single repeated in-vocabulary digit.

C.2. Countdown

Dataset The Countdown dataset [11, 40] is a synthetic arithmetic reasoning task designed to evaluate a model's ability to perform multi-step planning, maintain a dynamic state (the "pool"), and execute precise arithmetic operations. The task requires generating a sequence of operations that transform a set of initial numbers into a specific target value.

Each sample is generated procedurally to ensure the existence of at least one valid solution. The generation follows a reverse-construction logic:

1. **Initialization:** A set of k initial operands $\mathcal{N} = \{n_1, n_2, \ldots, n_k\}$ is sampled uniformly from the range $[1, 100]$. In our experiments, $k = 2, 3, 4, 5$.
2. **Chain Construction:** The algorithm iteratively reduces the pool of numbers until a single value remains. In each step, two numbers a and b are sampled from the current pool and combined using an operator $\odot \in \{+, -, *, /\}$.
3. **Validation:** Operations must result in positive integers. For division, it is required that $b \neq 0$ and $a \equiv 0 \pmod{b}$.
4. **Target Assignment:** The final remaining number in the pool after $k-1$ steps is designated as the target τ.

Each example is represented as a fixed-length character sequence of length $L = 32$ for $k = 2$; $L = 48$ for $k = 3$; and $L = 64$ for $k = 4, 5$. The dataset uses a restricted vocabulary of 18 tokens: digits 0–9, operators $+$, $-$, $*$, $/$, and delimiters $=$, ,, and $\backslash n$. The mask token is _.

Metrics and evaluation

- **Reaches Target Rate (RTR):** We consider a solution *valid* if and only if: (i) each step is arithmetically correct ($a \odot b = c$ evaluates correctly); (ii) at each step both operands a and b are available in the current pool, which is initialized to $\mathcal{N}$ and updated by removing a and b and inserting c after each step; and (iii) after all steps the pool contains exactly one element equal to τ. The Reaches Target Rate is then:
$$\text{RTR} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\left[y^{(i)} \text{ is a valid Countdown solution}\right]. \tag{11}$$

- **Local Arithmetic Fraction (LAF):** LAF measures the fraction of generated steps whose arithmetic is locally correct, regardless of whether the operands were available in the pool at that point in the solution:

$$\mathrm{LAF}(y) = \frac{1}{|S|} \sum_{s \in S} \mathbf{1}[\text{parse}(s) \neq \varnothing \ \text{ and } \ a \odot b = c \text{ is numerically correct}] \,, \tag{12}$$

  where S is the set of step strings in y that match the expected format $a \odot b = c$. LAF distinguishes between arithmetic errors (low LAF, low RTR) and correct arithmetic with pool-management errors (high LAF, low RTR).

- **Pool Prefix Fraction (PPF):** PPF measures how far through a *pool-consistent* simulation the model gets before its first failure, normalized by the expected number of steps $k-1$ where $k = |\mathcal{N}|$:

$$\mathrm{PPF}(y) = \frac{\max\{j : \text{steps } s_1, \ldots, s_j \text{ are all valid and pool-consistent}\}}{k-1} \ \in \ [0, 1]. \tag{13}$$

  A model that always fails on the first step has PPF $= 0$; RTR $= 1$ implies PPF $= 1$. PPF tracks progress toward the solution and is particularly informative at intermediate loop counts.

- **Target Residual Norm (TRN):** After following the longest valid pool-consistent prefix of the generated solution, TRN measures how close the remaining pool values are to the target τ:

$$\mathrm{TRN}(y) = \frac{\min_{x \in \mathcal{P}^*} |x - \tau|}{\max(1, |\tau|)} \ \in \ [0, +\infty), \tag{14}$$

  where $\mathcal{P}^*$ is the pool state after the longest pool-consistent prefix. TRN $= 0$ means the correct answer appears somewhere in the pool (a necessary condition for RTR $= 1$); TRN > 0 quantifies how far the best available value is from the target, scaled by the magnitude of the target to be comparable across instances. An unparseable or entirely invalid chain gives TRN $= 1.0$ by convention.

C.3. Text8

Dataset Text8 [23] is a standard character-level language modeling benchmark derived from a cleaned subset of Wikipedia text. The corpus consists of 100 million characters, restricted to lowercase English letters and spaces, resulting in a vocabulary of size 27. We follow the conventional split, using the first 90 million characters for training and the next 5 million for validation (with the remainder typically reserved for testing, though not used here).

The data is segmented into contiguous non-overlapping sequences of fixed length (256 tokens in our experiments). Each sequence is treated as a standalone training example. For masked modeling, mask positions are sampled independently per sequence according to a predefined masking probability, with mask token replacing the original character at those positions.

Metrics and evaluation We evaluate models on Text8 using standard language modeling metrics. All metrics are computed using a frozen pretrained evaluation model (GPT-J-6B, Wang and Komatsuzaki [35]), which scores generated sequences to provide a consistent external measure of fluency and likelihood.

- **Negative Log-Likelihood (NLL):** The average token-level negative log-likelihood on the validation set under the model, computed with teacher forcing.
- **Generative Perplexity (Gen PPL):** Perplexity is computed as $\exp(\text{NLL})$, measuring the effective branching factor of the model when generating sequences autoregressively.
- **Entropy:** We additionally report the entropy of the model's predictive distribution, averaged over validation tokens. This captures the model's uncertainty and provides insight into calibration, especially when comparing masked vs autoregressive objectives.

Appendix D. Training and Sampling Algorithms

D.1. Training Procedure

Training uses a masked denoising objective [27]. We randomly mask positions within the completion region, and the model is trained to minimize the cross-entropy loss at those positions. Further details are described in Algorithm 1.

Algorithm 1 Training Iteration with Recursive Refinement

Input: Dataset $\mathcal{D}$, Model f_θ, Optimizer $\mathcal{O}$, Accumulation Steps A
for iteration $it = 1$ **to** I_{max} **do**
  $\mathcal{O}.zero_grad()$
  $L_{accum} = 0$
  for micro-step $a = 1$ **to** A **do**
    Sample batch $(\mathbf{x}, \mathbf{y}, \mathbf{mask})$ from $\mathcal{D}$
    Resolve recursive steps N (Fixed, Sample, or Schedule)
    logits, $\mathcal{L}, \{\mathcal{L}_k\} \leftarrow f_\theta(\mathbf{x}, \mathbf{y}, \mathbf{mask}, \text{step_idx} = it)$
    Calculate $\mathcal{L}_{scaled} = \mathcal{L}/A$
    Compute gradients: $\mathcal{L}_{scaled}.backward()$
    $L_{accum} \leftarrow L_{accum} + \mathcal{L}.item()$
  end for
  $\mathcal{O}.step()$
  Update Learning Rate Scheduler
end for

All models were trained on a single a single NVIDIA A100-SXM4-80GB GPU. Across all runs, the base optimizer settings are shared: AdamW with learning rate $\text{LR} = 3 \times 10^{-4}$ and a cosine scheduler with warmup (`warmup_steps`=2000, `min_lr_ratio`=0.1). Further details on training hyperparameters are detailed in Table 2.

Positional encodings The model uses two positional-encoding regimes. For non-Sudoku datasets, we employ standard 1D rotary positional embeddings (RoPE, Su et al. [33]). For Sudoku datasets, we explore a 2D variant: each flattened token index is mapped back to grid coordinates (r, c), rotary frequencies are computed separately for row and column components, and then concatenated so the attention mechanism is aware of both horizontal and vertical structure. In addition, we add a learned *block embedding* identifying each subgrid (e.g., 3×3 blocks in 9×9 Sudoku), which explicitly encodes Sudoku block constraints beyond pure sequence order.

Table 2: Architecture and optimization settings, plus effective training epochs, for each dataset.

Dataset	n_{layer}	n_{head}	n_{embd}	Params	Epochs
Sudoku 9×9	6	6	384	10.8M	10
Sudoku 25×25	6	6	384	10.8M	75
Countdown	3	12	384	5.5M	300
text8	6	6	384	10.8M	10

D.2. Iterative Parallel Decoding

At inference time, we fill masked regions using an iterative process. We implement three schedules for token commitment:

- **Steps:** A deterministic schedule where the K most confident tokens are "unmasked" (fixed) at each step following the method in [25].
- **Steps Random:** Similar to the steps schedule, but positions are chosen randomly rather than by confidence, serving as a stochastic baseline, also similar to the option in [25].
- **Confidence-based:** An adaptive schedule in which all tokens that exceed a probability threshold γ are committed in each iteration.

Algorithm 2 details the `Steps` decoding procedure, which we found most effective and used for our tasks, as it allowed us to compare the role of recursiveness on the effective number of denoising steps.

Algorithm 2 Iterative Parallel Decoding (Steps Method)

Input: model f_θ, input sequence $\mathbf{x}$ with masked positions $\mathbf{M}$, total steps T, temperature τ
$N = \sum \mathbf{M}$ {Total tokens to unmask}
Calculate schedule $S = [s_1, s_2, \ldots, s_T]$ such that $\sum s_t = N$
for $t = 1$ **to** T **do**
  Compute logits $z = f_\theta(\mathbf{x})$
  Sample tokens $\hat{\mathbf{x}}$ from z/τ using Top-K multinomial sampling
  Compute confidence scores $c = \text{softmax}(z/\tau)_{max}$
  Mask scores: $v_i = c_i$ if M_i is true, else $-\infty$
  $k = S[t]$
  Identify indices $\mathcal{I}$ of the k largest values in v
  Update $\mathbf{x}[i] = \hat{\mathbf{x}}[i]$ for all $i \in \mathcal{I}$
  Update $\mathbf{M}[i] =$ false for all $i \in \mathcal{I}$
end for
Output: Completed sequence $\mathbf{x}$

Appendix E. Ablations

E.1. Recursive step choice

We study several strategies for selecting the number of recursive computation steps T during training. These strategies control the amount of iterative refinement applied by the model and can be broadly categorized into *fixed*, *stochastic*, and *scheduled* schemes.

Fixed-depth recursion In the fixed-depth setting, the number of recursive iterations is held constant $T = L$, where L is a predefined hyperparameter. This approach provides stable optimization and predictable computational cost, and is commonly used in looped or recurrent transformer architectures. Notably, Yang et al. [39] demonstrate that Transformers trained with a fixed number of recurrent loops can improve algorithmic generalization when unrolled for multiple iterations. Similarly, RecursiveVLM [38] trains with a fixed recursion depth and evaluates the model at different sampling recursive steps.

Random-depth sampling In the stochastic depth setting, the recursion depth is sampled per example or batch $T \sim \Lambda$, where Λ denotes a predefined probability distribution over positive integers. This formulation exposes the model to a distribution over computational budgets during training, improving robustness to variable inference-time compute. This follows the line of work from [12], which samples recurrent iteration counts using log-normal or Poisson-like distributions combined with truncated backpropagation through time. This enables the model to allocate computation adaptively while maintaining efficiency on average. In our implementation, this category includes:

- Uniform distribution: $T \sim \mathcal{U}(T_{\min}, T_{\max})$
- Poisson distribution: $T \sim \text{Poisson}(\lambda) + 1$, where λ is drawn from a log-normal prior with parameters chosen to match a desired mean depth

Curriculum-based scheduling In curriculum-based approaches, the recursion depth is deterministically varied over the course of training $T = f(t)$ where t denotes the training iteration and $f(\cdot)$ is a monotonic schedule mapping training progress to computation depth. This idea has been widely used in recurrent and looped architectures [7, 9]. Typically, T is gradually increased or decreased from an initial value T_{start} to a final value T_{end}, enabling a curriculum over looping depth:

$$T(t) = \text{round}\left(T_{\text{start}} + \alpha(t)(T_{\text{end}} - T_{\text{start}})\right),$$

with $\alpha(t) \in [0, 1]$.

In our implementation, we evaluate both **increasing** and **decreasing linear schedules**, corresponding respectively to $T_{\text{start}} < T_{\text{end}}$ and $T_{\text{start}} > T_{\text{end}}$. This allows us to study whether progressively allocating more or less recursive steps affects downstream performance.

Figure 7 evaluates the impact of different training schemes for the number of recursive steps on the final performance across evaluation depths $L_s \in \{1, 3, 5, 7, 10, 15, 20\}$ and decoding budgets $T \in \{1, 5, 20\}$. We compare a constant baseline (*Fixed* L=5), stochastic sampling methods (*Uniform* and *Poisson*), a standard linear curriculum that increases recursive steps during training (*Schedule* $1 \to 10$), and a reverse linear curriculum (*Schedule* $10 \to 1$). The reverse curriculum (*Schedule* $10 \to 1$) emerges as the superior training strategy by a substantial margin, particularly at small-to-moderate evaluation depths and under tight decoding budgets.

For instance, at T=1 and L_s=5, *Schedule* $10 \to 1$ achieves a 63.6% valid puzzle rate, outperforming the constant baseline *Fixed* L=5 (21.2%), stochastic *Poisson* (21.0%), and the forward curriculum *Schedule* $1 \to 10$ (8.0%). This performance gap is even more pronounced at L=7 under T=1, where the reverse curriculum reaches 89.8% compared to just 61.6% for *Fixed* L=5 and 33.2% for *Schedule* $1 \to 10$. Even when more decoding steps are allowed, the reverse curriculum retains its advantage: at T=5 and L=3, it reaches 95.6% validity, whereas *Fixed* L=5 and *Schedule* $1 \to 10$ lag behind at 79.4% and 73.8%, respectively. These results suggest that introducing deep recursion early in training forces the model to learn high-capacity, multi-step coordination, while progressively reducing the depth down to 1 fine-tunes its ability to execute highly efficient, single-step updates. Conversely, starting with shallow steps (*Schedule* $1 \to 10$) or sampling them randomly (*Uniform*, *Poisson*) fails to establish the same level of systematic multi-step alignment, leading to sub-optimal performance across all evaluation depths.

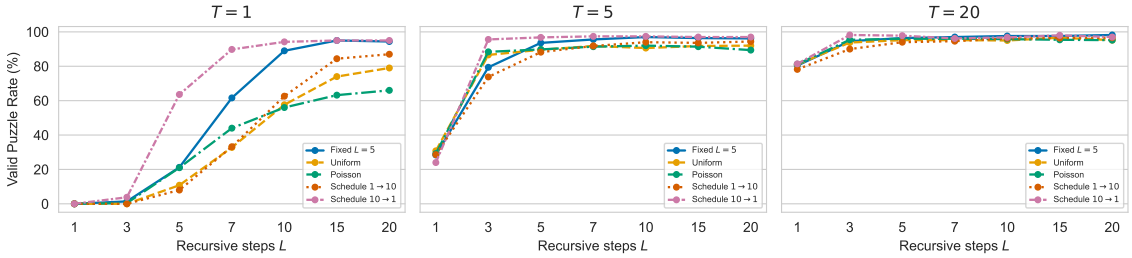

Figure 7: Performance comparison between models trained with different recursive steps choices on 9×9 Sudoku puzzles. We report the Valid Puzzle Rate across varying loop counts at sampling L_s and denoising steps T.

E.2. Training objectives

Here we extend the discussion on the different supervision methods, and report the results from the ablation study.

Final-step loss (FINAL). Only the logits produced at the last loop, $\hat{\mathbf{y}}^{(L)}$, are supervised:

$$\mathcal{L}_{\text{final}}(\theta) = \mathbb{E}_{t,\,x_0,\,x_t}\left[\frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}} \mathrm{CE}\Big(\hat{\mathbf{y}}_i^{(L)},\ x_0^{(i)}\Big)\right], \tag{15}$$

where $\mathcal{M} = \{i : x_t^{(i)} = \texttt{[MASK]}\}$ is the set of masked positions, $|\mathcal{M}|$ is its cardinality, and CE denotes the cross-entropy loss. This is the natural extension of the standard single-pass MDM objective to the looped setting: the intermediate loop outputs are used only as hidden-state refinements, not directly supervised. The gradient signal reaches early loops only through backpropagation through the full chain of shared blocks.

All-steps loss (ALL). All L sets of logits are supervised, with the loss averaged across loops:

$$\mathcal{L}_{\text{all}}(\theta) = \mathbb{E}_{t,\,x_0,\,x_t}\left[\frac{1}{L}\sum_{\ell=1}^{L}\frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}} \mathrm{CE}\Big(\hat{\mathbf{y}}_i^{(\ell)},\ x_0^{(i)}\Big)\right]. \tag{16}$$

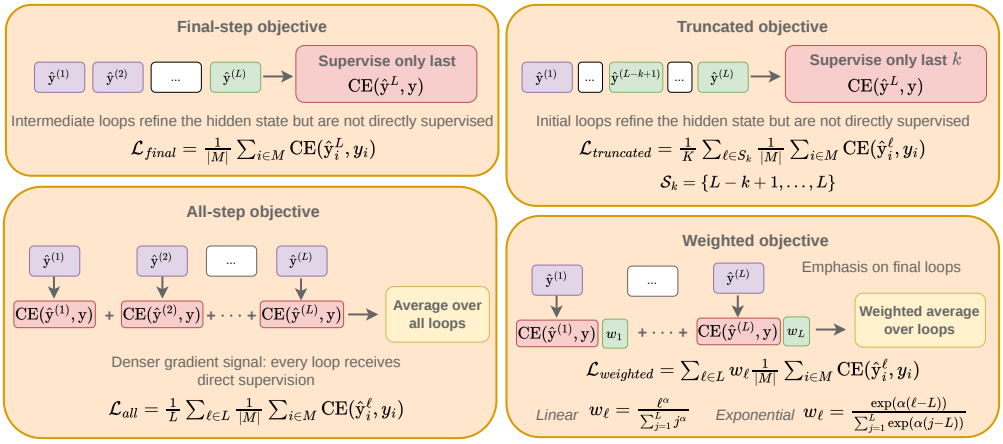

Figure 8: Schematic of the different supervision modes considered.

This provides a denser gradient signal: every loop receives direct supervision rather than relying entirely on backpropagation through subsequent loops. This idea is closely related to deeply supervised learning, where intermediate representations are explicitly trained to be predictive [21]. It also regularizes the model to produce semantically meaningful intermediate predictions, not only a high-quality final output, the MDM analogue of the per-iteration supervision in RELAY for AR looped models [43]. The all-steps loss increases training memory due to storing intermediate activations for backpropagation through all loops, but it does not increase forward-pass compute.

Weighted loss (WEIGHTED). Rather than supervising all loops equally, the weighted objective assigns a non-uniform weight w_ℓ to the loss at each loop:

$$\mathcal{L}_{\text{weighted}}(\theta) = \mathbb{E}_{t,\, x_0,\, x_t}\left[\sum_{\ell=1}^{L} w_\ell \frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}} \mathrm{CE}\Big(\hat{\mathbf{y}}_i^{(\ell)},\ x_0^{(i)}\Big)\right], \tag{17}$$

where $w_\ell \geq 0$ and $\sum_{\ell=1}^{L} w_\ell = 1$. In our implementation, we consider two weighting schemes. For *linear* weighting,

$$w_\ell = \frac{\ell^\alpha}{\sum_{j=1}^{L} j^\alpha}, \tag{18}$$

while for *exponential* weighting,

$$w_\ell = \frac{\exp\big(\alpha(\ell - L)\big)}{\sum_{j=1}^{L}\exp\big(\alpha(j-L)\big)}, \tag{19}$$

where $\alpha > 0$ controls how strongly supervision is concentrated toward later loops. Larger values of α place increasing emphasis on the final recursive iterations while still providing direct supervision to earlier loops. This objective is related to weighted auxiliary-loss training in anytime and deeply supervised neural networks, where intermediate predictions are optimized with unequal importance to balance early predictive capability against final performance [17].

Truncated loss (TRUNCATED). The truncated objective supervises only the final k recursive loops. Let

$$\mathcal{S}_k = \{L - k + 1, \ldots, L\} \tag{20}$$

denote the set of the last k loops (or all loops if $k > L$). The loss is then

$$\mathcal{L}_{\text{truncated}}(\theta) = \mathbb{E}_{t,\, x_0,\, x_t}\left[\frac{1}{k}\sum_{\ell \in \mathcal{S}_k} \frac{1}{|\mathcal{M}|}\sum_{i \in \mathcal{M}} \text{CE}\Big(\hat{\mathbf{y}}_i^{(\ell)},\ x_0^{(i)}\Big)\right], \tag{21}$$

where $k \leq L$. This objective provides direct supervision only to the later stages of the recursive computation, which are expected to be closer to the final prediction. FINAL corresponds to truncated with $k = 1$. In the more general setting, multiple late-loop predictions receive explicit supervision, improving gradient flow, while compared to ALL, early loops are not constrained to produce accurate predictions and can instead focus on building useful intermediate representations. This strategy is similar in spirit to the truncated step-range averaging used in looped transformer architectures, where training emphasizes a subset of later recurrent iterations rather than supervising the entire trajectory equally [39].

Figure 9 (with full results in Table 3 in Appendix F.1) reports Sudoku 9×9 Valid Puzzle Rate performance for models trained with the different losses and 5 loops at training. We show the results sampling recursive steps $L_s \in \{1, 3, 5, 7, 10, 15, 20\}$ and denoising steps $T \in \{1, 5, 20\}$.

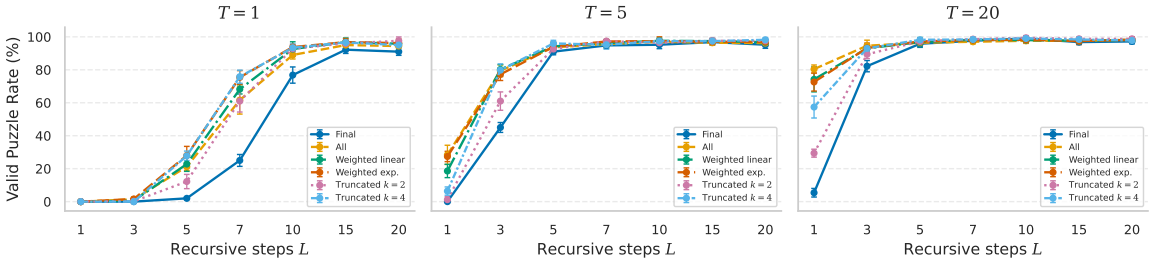

Figure 9: Performance comparison between supervision losses on 9×9 Sudoku puzzles for the model trained with $L_t = 5$ recursive steps. We report the Valid Puzzle Rate across varying loop counts at sampling L_s and denoising steps T.

We observe that cross panels, ALL yields the strongest and most robust performance, particularly in the regime with less recursion (L_s small to moderate) and where inference is budget-constrained (T small). At L_s=1, the gap between ALL and FINAL is notable. This supports the view that dense, per-step supervision is necessary to align every recursive layer with the denoising objective, rather than relying on error signals that propagate only through the terminal state. As L_s increases, all methods improve and the curves compress, yet ALL remains at or near the top through moderate depth. We therefore adopt ALL as the default supervision strategy in subsequent experiments.

E.3. Step Embedding Ablation

Finally, we investigate the role of the recursive step embedding introduced in Equation 8. The purpose of this embedding is to provide the model with information about its current position within the recursion process, allowing different iterations to specialize their computation rather than applying an identical transformation at every step.

To evaluate its impact, we compare three variants: (i) a model with a *learned* step embedding, where the normalized recursion progress is projected into the hidden space through a small trainable network; (ii) a model with a *fixed* step embedding, constructed directly from the normalized recursion progress (ranging from 0 to 1) using a deterministic encoding; and (iii) a model with *no step embedding*, in which all recursive iterations share exactly the same computation and receive no explicit indication of the current recursion depth.

Figure 10 shows a relatively close comparison between learned embeddings and the no-embedding baseline, with both outperforming fixed embeddings in the most important low-depth regimes. Learned embeddings perform strongly across the sweep and give the model an explicit, trainable signal for the current recursion step, which makes them a natural general-purpose choice. However, the no-embedding baseline is also very competitive. The advantage of learned embeddings is therefore not absolute; rather, they offer a balanced tradeoff: more flexible and robust than fixed embeddings, while remaining close to the best observed performance across the full grid.

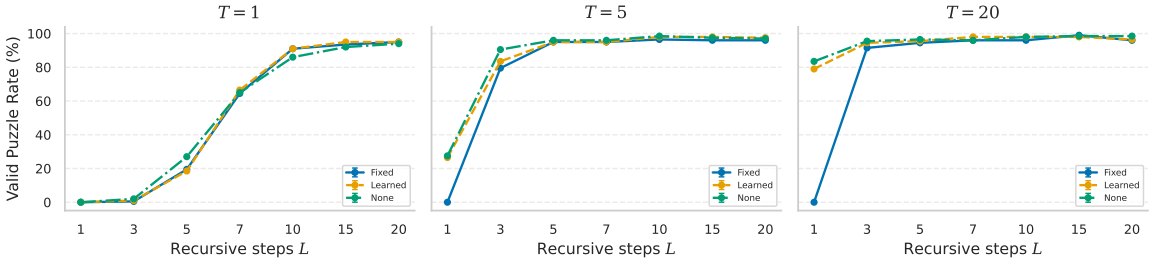

Figure 10: Performance comparison between models trained with learned, fixed and without step-embedding on 9×9 Sudoku puzzles. We report the Valid Puzzle Rate across varying sampling loop counts L_s and denoising steps T.

Appendix F. Extended Results

F.1. Sudoku 9×9: Full Tables

Table 3 presents the comprehensive Sudoku 9×9 results for all combinations of recursion depth, loss mode, and denoising step budget. Latency and throughput figures are measured at batch size 1 on a single A100. Rows labelled *Baseline* correspond to $(6 \otimes 1)$ without any recursion. Rows labelled k *rec steps* use a $(6 \otimes k)$ model. Within each recursion depth, *Final* uses $\mathcal{L}_{\text{final}}$ and *All* uses $\mathcal{L}_{\text{all}}$. Latency grows linearly in L at fixed T; throughput (samples per second) falls accordingly. The highlighted cells mark configurations that exceed 97% VPR.

Table 3: Sudoku Performance: comparison across recursion depths and loss types (ALL vs FINAL). We report mean and standard deviation across 5 sampling runs of 100 samples each.

Model	Loss type	Steps	VPR %	SCL	Latency (s)	Throughput
Baseline	-	1	0.0% ± 0.0	3.077 ± 0.039	0.007	149.6
		5	65.2% ± 6.9	0.226 ± 0.052	0.022	44.9
		10	85.0% ± 2.7	0.088 ± 0.024	0.043	23.3
		20	92.2% ± 2.7	0.031 ± 0.017	0.084	11.9
		40	95.8% ± 1.9	0.011 ± 0.006	0.169	5.9
2 rec steps	FINAL	1	0.0% ± 0.0	2.935 ± 0.073	0.010	98.6
		5	67.2% ± 4.1	0.248 ± 0.028	0.041	24.3
		10	86.2% ± 1.9	0.082 ± 0.019	0.082	12.2
		20	92.4% ± 1.1	0.040 ± 0.007	0.161	6.2
		40	93.0% ± 1.2	0.025 ± 0.006	0.321	3.1
	ALL	1	0.0% ± 0.0	2.443 ± 0.080	0.010	102.1
		5	83.2% ± 3.6	0.120 ± 0.026	0.039	25.8
		10	91.2% ± 1.6	0.055 ± 0.017	0.079	12.7
		20	95.4% ± 2.8	0.024 ± 0.015	0.153	6.6
		40	96.4% ± 1.8	0.016 ± 0.009	0.298	3.4
3 rec steps	FINAL	1	0.4% ± 0.6	2.523 ± 0.063	0.025	40.7
		5	79.6% ± 0.9	0.166 ± 0.024	0.095	10.5
		10	93.0% ± 3.2	0.044 ± 0.018	0.184	5.4
		20	93.6% ± 4.2	0.030 ± 0.018	0.314	3.2
		40	97.4% ± 1.5	0.010 ± 0.007	0.541	1.8
	ALL	1	8.6% ± 3.0	1.645 ± 0.051	0.020	50.5
		5	92.4% ± 2.5	0.056 ± 0.018	0.072	13.9
		10	95.6% ± 2.1	0.023 ± 0.015	0.131	7.7
		20	96.0% ± 2.2	0.024 ± 0.016	0.234	4.3
		40	98.0% ± 0.7	0.008 ± 0.005	0.456	2.2
5 rec steps	FINAL	1	2.0% ± 0.7	2.147 ± 0.068	0.081	12.3
		5	91.0% ± 1.9	0.071 ± 0.018	0.218	4.6
		10	95.0% ± 1.4	0.029 ± 0.013	0.355	2.8
		20	95.8% ± 2.2	0.020 ± 0.008	0.648	1.5
		40	97.6% ± 2.0	0.011 ± 0.009	1.181	0.8
	ALL	1	21.2% ± 2.4	1.093 ± 0.113	0.084	11.9
		5	93.6% ± 2.9	0.050 ± 0.026	0.210	4.8
		10	97.4% ± 1.3	0.016 ± 0.014	0.362	2.8
		20	96.4% ± 1.5	0.016 ± 0.007	0.696	1.4
		40	98.0% ± 1.6	0.008 ± 0.006	1.119	0.9
10 rec steps	FINAL	1	9.8% ± 2.2	1.568 ± 0.109	0.064	15.6
		5	91.2% ± 1.8	0.065 ± 0.011	0.275	3.6
		10	94.6% ± 2.0	0.032 ± 0.013	0.541	1.8
		20	97.8% ± 1.3	0.011 ± 0.006	1.546	0.6
		40	97.6% ± 1.5	0.011 ± 0.007	3.435	0.3
	ALL	1	46.4% ± 8.0	0.670 ± 0.156	0.036	27.8
		5	95.2% ± 2.4	0.034 ± 0.016	0.171	5.8
		10	96.0% ± 1.6	0.026 ± 0.016	0.504	2.0
		20	97.2% ± 1.3	0.017 ± 0.012	1.232	0.8
		40	97.0% ± 1.9	0.014 ± 0.010	3.457	0.3

Table 4 presents the Sudoku 9×9 results for the (*Schedule*10 $\rightarrow$ 1) training recursion depth and for different evaluated sampling recursion depths (L_s). We observe equally that Latency grows linearly in L at fixed T.

Table 4: Sudoku Performance: comparison across sampling recursion depths (L_s) and decoding steps (T). We report mean and standard deviation across 5 sampling runs of 100 samples each.

Recursion L_s	**Steps**	**VPR %**	**SCL**	**Latency (s)**	**Throughput**
1 rec step	1	0.0% ± 0.0	3.729 ± 0.034	0.018	55.0
	5	24.0% ± 2.5	0.587 ± 0.033	0.045	22.3
	10	63.6% ± 2.5	0.210 ± 0.034	0.087	11.5
	20	81.4% ± 2.5	0.084 ± 0.008	0.143	7.0
	40	85.2% ± 0.8	0.050 ± 0.003	0.286	3.5
3 rec steps	1	3.8% ± 1.3	1.791 ± 0.046	0.024	41.8
	5	95.6% ± 0.9	0.025 ± 0.009	0.081	12.3
	10	95.8% ± 1.8	0.021 ± 0.007	0.161	6.2
	20	98.2% ± 0.8	0.012 ± 0.011	0.276	3.6
	40	99.0% ± 1.0	0.004 ± 0.004	0.608	1.6
5 rec steps	1	63.6% ± 1.5	0.426 ± 0.019	0.032	30.9
	5	96.8% ± 0.8	0.022 ± 0.007	0.120	8.3
	10	97.0% ± 1.6	0.020 ± 0.007	0.235	4.2
	20	97.8% ± 0.8	0.012 ± 0.006	0.446	2.2
	40	99.8% ± 0.4	0.001 ± 0.002	0.940	1.1
7 rec steps	1	89.8% ± 0.8	0.113 ± 0.007	0.040	25.1
	5	97.4% ± 0.9	0.021 ± 0.004	0.159	6.3
	10	96.4% ± 0.9	0.015 ± 0.004	0.339	3.0
	20	96.2% ± 0.8	0.018 ± 0.004	0.573	1.7
	40	97.6% ± 0.5	0.011 ± 0.002	1.279	0.8
10 rec steps	1	94.2% ± 0.4	0.082 ± 0.013	0.049	20.4
	5	97.4% ± 0.5	0.021 ± 0.002	0.212	4.7
	10	98.4% ± 0.9	0.005 ± 0.003	0.435	2.3
	20	96.8% ± 0.4	0.014 ± 0.004	0.806	1.2
	40	98.0% ± 0.0	0.007 ± 0.000	1.565	0.6
15 rec steps	1	95.0% ± 0.0	0.062 ± 0.010	0.053	18.9
	5	97.0% ± 0.0	0.016 ± 0.001	0.326	3.1
	10	97.2% ± 0.4	0.014 ± 0.002	0.613	1.6
	20	98.0% ± 0.0	0.010 ± 0.002	1.183	0.8
	40	98.0% ± 0.0	0.007 ± 0.000	1.822	0.5
20 rec steps	1	95.0% ± 0.0	0.060 ± 0.006	0.071	14.0
	5	97.0% ± 0.0	0.018 ± 0.003	0.373	2.7
	10	97.6% ± 0.5	0.007 ± 0.002	0.726	1.4
	20	97.0% ± 0.0	0.018 ± 0.003	1.579	0.6
	40	98.0% ± 0.0	0.007 ± 0.000	2.416	0.4

F.2. Sudoku 9×9: Depth Scaling Without Recursion

Table 5 reports Valid Puzzle Rate and Soft Constraint Loss metrics for non-recursive baselines with 6, 8, 10, 12, 18, 24, and 30 transformer layers. Parameter counts range from 10.8M to 53.1M. All models share the same d=384, H=6 architecture and are trained for the same number of gradient steps as the recursive models.

Notably, increasing depth beyond 18 layers produces only marginal improvements on VPR (96.6%→97.2%→96.8% for 18→24→30 layers at T=40), whereas increasing L from 1 to 5 at fixed 6-layer parameter count produces a larger gain (95.8%→98.0%).

Table 5: Sudoku performance across different depths (4–30 layers). We report mean and standard deviation across 5 sampling runs of 100 samples each.

Model	Steps	VPR %	SCL	Latency (s)	Throughput
6 layers (10.6M)	1	0.0% ± 0.0	3.077 ± 0.039	0.007	149.6
	5	65.2% ± 6.9	0.226 ± 0.052	0.022	44.9
	10	85.0% ± 2.7	0.088 ± 0.024	0.043	23.3
	20	92.2% ± 2.7	0.031 ± 0.017	0.084	11.9
	40	95.8% ± 1.9	0.011 ± 0.006	0.169	5.9
8 layers (14.2M)	1	0.0% ± 0.0	3.209 ± 0.053	0.010	104.8
	5	58.6% ± 6.4	0.291 ± 0.038	0.033	30.6
	10	83.0% ± 3.7	0.099 ± 0.018	0.072	13.9
	20	90.2% ± 3.5	0.045 ± 0.016	0.151	6.6
	40	94.4% ± 2.0	0.020 ± 0.007	0.370	2.7
10 layers (17.7M)	1	0.0% ± 0.0	3.100 ± 0.098	0.010	102.4
	5	60.4% ± 5.9	0.302 ± 0.055	0.046	22.0
	10	83.8% ± 3.0	0.098 ± 0.016	0.094	10.6
	20	91.4% ± 4.3	0.047 ± 0.028	0.211	4.7
	40	94.6% ± 1.1	0.024 ± 0.004	0.470	2.1
12 layers (21.2M)	1	0.0% ± 0.0	3.025 ± 0.057	0.010	101.5
	5	65.4% ± 3.4	0.262 ± 0.034	0.040	24.8
	10	87.6% ± 4.8	0.064 ± 0.022	0.082	12.1
	20	93.6% ± 2.9	0.031 ± 0.015	0.276	3.6
	40	93.8% ± 1.9	0.018 ± 0.006	1.005	1.0
18 layers (31.9M)	1	0.0% ± 0.0	2.622 ± 0.083	0.023	43.0
	5	80.6% ± 3.2	0.144 ± 0.021	0.085	11.7
	10	95.2% ± 2.4	0.028 ± 0.017	0.187	5.4
	20	94.8% ± 1.5	0.027 ± 0.008	0.535	1.9
	40	96.6% ± 0.6	0.014 ± 0.005	1.112	0.9
24 layers (42.5M)	1	0.2% ± 0.5	2.710 ± 0.056	0.048	21.1
	5	82.8% ± 3.6	0.127 ± 0.025	0.167	6.0
	10	92.2% ± 2.8	0.042 ± 0.019	0.365	2.7
	20	95.0% ± 2.5	0.023 ± 0.013	0.724	1.4
	40	97.2% ± 0.8	0.008 ± 0.002	1.435	0.7
30 layers (53.1M)	1	0.2% ± 0.5	2.624 ± 0.093	0.050	19.9
	5	77.8% ± 2.2	0.159 ± 0.018	0.160	6.2
	10	91.4% ± 1.3	0.044 ± 0.010	0.284	3.5
	20	95.0% ± 2.0	0.021 ± 0.006	0.547	1.8
	40	97.2% ± 0.8	0.012 ± 0.004	1.059	0.9

F.3. Sudoku 9×9: Cross-Recursion Trade-off

Table 6 extends the cross-recursion analysis by reporting both Valid Puzzle Rate and Soft Constraint Loss across training depths $L_t \in \{1, 2, 3, 5, 10\}$ and a range of sampling depths L_s. The $(L_t = 2, L_s = 3)$ setting, which uses one more loop at sampling than training, achieves 91.6% at T=5—already surpassing the L_t=2 baseline at T=5 (83.2%) and comparable to the L_t=3 model at T=5 (92.4%). This suggests that moderate extrapolation is possible, likely because the step embedding is parameterized by normalized progress $s_\ell \in [0, 1]$ rather than the raw loop index.

Table 6: Sudoku Performance: Cross-Recursion Trade-off. Training Recursion Depth vs. Sampling Recursion Depth across various decoding steps (T). We report mean and standard deviation across 5 sampling runs of 100 samples each.

Tr. Rec	Sa. Rec	$T = 1$		$T = 5$		$T = 10$		$T = 20$		$T = 40$	
Steps	Steps	VPR %	SCL	VPR %	SCL	VPR %	SCL	VPR %	SCL	VPR %	SCL
Baseline	1	0.0 ± 0.0	3.077 ± 0.039	65.2 ± 6.9	0.226 ± 0.052	85.0 ± 2.7	0.088 ± 0.024	92.2 ± 2.7	0.031 ± 0.017	95.8 ± 1.9	0.011 ± 0.006
Rec 2	1	0.0 ± 0.0	3.355 ± 0.069	43.6 ± 3.0	0.415 ± 0.048	77.6 ± 2.5	0.129 ± 0.017	88.2 ± 3.8	0.052 ± 0.024	92.6 ± 3.2	0.023 ± 0.011
	2	0.0 ± 0.0	2.433 ± 0.080	83.2 ± 3.6	0.120 ± 0.026	91.2 ± 1.6	0.055 ± 0.017	95.4 ± 2.8	0.024 ± 0.015	96.4 ± 1.8	0.016 ± 0.009
	3	8.2 ± 3.1	1.625 ± 0.042	91.6 ± 2.7	0.060 ± 0.025	96.2 ± 2.2	0.026 ± 0.015	96.2 ± 1.1	0.021 ± 0.008	98.4 ± 0.9	0.007 ± 0.004
	5	52.8 ± 5.0	0.658 ± 0.110	67.2 ± 5.6	0.153 ± 0.023	64.6 ± 3.2	0.160 ± 0.027	67.6 ± 2.7	0.137 ± 0.019	67.0 ± 6.5	0.130 ± 0.019
Rec 3	1	0.0 ± 0.0	4.485 ± 0.058	4.0 ± 3.9	0.884 ± 0.020	24.2 ± 3.1	0.457 ± 0.029	38.6 ± 7.1	0.277 ± 0.028	50.8 ± 9.3	0.165 ± 0.043
	3	8.6 ± 3.0	1.645 ± 0.051	92.4 ± 2.5	0.056 ± 0.018	95.6 ± 2.1	0.023 ± 0.015	96.0 ± 2.2	0.024 ± 0.016	98.0 ± 0.7	0.008 ± 0.005
	9	38.8 ± 5.5	1.304 ± 0.188	86.2 ± 3.0	0.174 ± 0.040	91.6 ± 4.4	0.073 ± 0.035	93.6 ± 3.4	0.044 ± 0.025	95.2 ± 1.3	0.033 ± 0.021
	15	38.0 ± 3.7	1.796 ± 0.285	70.0 ± 6.6	0.817 ± 0.325	80.0 ± 5.4	0.424 ± 0.168	84.6 ± 1.1	0.260 ± 0.067	82.8 ± 2.7	0.290 ± 0.037
Rec 5	1	0.0 ± 0.0	3.645 ± 0.060	28.4 ± 5.8	0.605 ± 0.054	61.6 ± 4.6	0.219 ± 0.035	80.4 ± 2.5	0.097 ± 0.022	83.4 ± 3.3	0.061 ± 0.014
	5	21.2 ± 2.4	1.093 ± 0.113	93.6 ± 2.9	0.050 ± 0.026	97.4 ± 1.3	0.016 ± 0.014	96.4 ± 1.5	0.016 ± 0.007	98.0 ± 1.6	0.008 ± 0.006
	10	89.0 ± 2.5	0.127 ± 0.033	97.0 ± 1.4	0.022 ± 0.011	95.8 ± 2.5	0.025 ± 0.013	97.6 ± 0.9	0.012 ± 0.004	98.4 ± 0.9	0.008 ± 0.007
	20	94.4 ± 2.9	0.080 ± 0.052	96.2 ± 1.9	0.027 ± 0.018	97.4 ± 1.3	0.018 ± 0.011	98.2 ± 0.8	0.011 ± 0.007	97.4 ± 2.4	0.012 ± 0.012
	25	96.4 ± 2.1	0.050 ± 0.027	97.4 ± 1.1	0.018 ± 0.013	98.4 ± 1.8	0.009 ± 0.012	97.4 ± 1.7	0.014 ± 0.010	99.2 ± 0.8	0.002 ± 0.002
Rec 10	1	0.0 ± 0.0	3.935 ± 0.044	16.0 ± 3.1	0.686 ± 0.035	51.6 ± 5.0	0.259 ± 0.041	70.4 ± 3.8	0.132 ± 0.012	73.8 ± 4.6	0.081 ± 0.015
	5	1.2 ± 0.8	2.186 ± 0.097	79.4 ± 0.9	0.163 ± 0.025	88.4 ± 2.7	0.069 ± 0.027	92.4 ± 1.5	0.048 ± 0.010	94.8 ± 1.9	0.026 ± 0.005
	10	46.4 ± 8.0	0.670 ± 0.156	95.2 ± 2.4	0.034 ± 0.016	96.0 ± 1.6	0.026 ± 0.016	97.2 ± 1.3	0.017 ± 0.012	97.0 ± 1.9	0.014 ± 0.010
	20	90.6 ± 2.1	0.120 ± 0.050	95.0 ± 2.1	0.041 ± 0.019	96.4 ± 3.1	0.026 ± 0.018	98.4 ± 0.9	0.006 ± 0.003	97.6 ± 1.5	0.013 ± 0.009
	30	95.2 ± 2.5	0.098 ± 0.053	95.6 ± 1.5	0.026 ± 0.009	95.6 ± 2.0	0.021 ± 0.009	97.0 ± 1.0	0.018 ± 0.007	98.4 ± 1.1	0.007 ± 0.006
	50	92.8 ± 2.5	0.122 ± 0.042	96.0 ± 1.0	0.024 ± 0.004	97.2 ± 1.9	0.018 ± 0.012	97.8 ± 2.3	0.012 ± 0.013	98.8 ± 0.8	0.005 ± 0.003

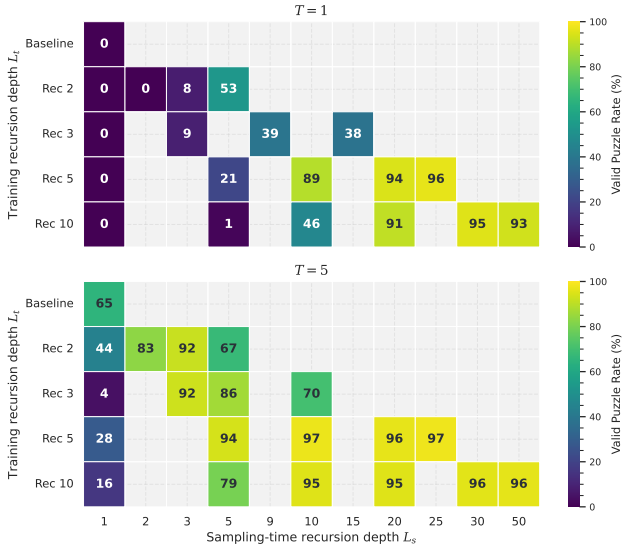

Figure 11: Cross-recursion trade-off between training recursion depth (L_t) and sampling recursion depth (L_s) for 9 × 9 Sudoku tasks. The top heatmap shows one-step decoding, while the bottom shows 5 decoding steps.

F.4. Sudoku 25×25: Effect of Recursion and Masking

Table 7 shows Valid Puzzle Rate and Soft Constraint Loss for the baseline and recursive models on the 25×25 Sudoku task under three masking regimes: 70%, 80%, and 90% of cells masked at inference time.

At 70% masking, the task is tractable for all models at high T: the baseline reaches 95.2% VPR at T=100. However, both L=3 and L=5 achieve 100% at T=1—a single denoising step—demonstrating that a single forward pass of the looped model can solve a task that requires at least 50 passes for the baseline. At 80% masking, the baseline degrades severely (69.6% VPR at T=100), while L=3 achieves 100% VPR at T=100 and 99.6% at T=10. At 90% masking, all models struggle, but the gap between baseline (2.0% at T=100) and L=3 (52.2%) reveals the qualitative benefit of recursive loops for highly under-determined constraint satisfaction.

These results suggest that the benefits of recursive depth are not simply additive with problem size but compound with masking difficulty: as fewer clues are provided, the model's need to propagate global constraints increases, and recursive loops (which implement exactly one round of full-bidirectional attention per iteration) provide increasingly valuable additional computation.

Table 7: Sudoku 25 × 25 generative performance: Comparison between different percentages of masks (M). We report mean and standard deviation across 5 sampling runs of 100 samples each.

		M = 70%		M = 80%		M = 90%	
Model	**Steps**	**VPR %**	**SCL**	**VPR %**	**SCL**	**VPR %**	**SCL**
Baseline	1	50.6% ± 3.3	0.189 ± 0.021	0.0% ± 0.0	4.587 ± 0.127	0.0% ± 0.0	24.978 ± 0.226
	5	52.0% ± 1.6	0.225 ± 0.008	6.6% ± 1.8	1.031 ± 0.080	0.0% ± 0.0	12.286 ± 0.453
	10	52.8% ± 3.4	0.221 ± 0.023	9.8% ± 0.8	0.719 ± 0.034	0.0% ± 0.0	6.363 ± 0.240
	25	68.6% ± 3.9	0.116 ± 0.026	28.6% ± 5.2	0.335 ± 0.032	0.2% ± 0.5	3.272 ± 0.188
	50	90.8% ± 2.5	0.026 ± 0.006	52.6% ± 4.2	0.142 ± 0.007	0.2% ± 0.5	2.060 ± 0.058
	100	95.2% ± 2.4	0.008 ± 0.004	69.6% ± 5.6	0.432 ± 0.040	2.0% ± 1.6	1.428 ± 0.120
3 rec. steps	1	100.0% ± 0.0	0.000 ± 0.000	98.8% ± 1.1	0.018 ± 0.019	1.4% ± 1.3	10.426 ± 0.323
	5	95.8% ± 2.2	0.007 ± 0.004	70.6% ± 5.6	0.082 ± 0.029	17.4% ± 3.7	2.803 ± 0.243
	10	100.0% ± 0.0	0.000 ± 0.000	99.6% ± 0.6	0.001 ± 0.001	31.4% ± 5.9	0.256 ± 0.050
	25	100.0% ± 0.0	0.000 ± 0.000	99.8% ± 0.5	0.001 ± 0.001	42.8% ± 1.5	0.680 ± 0.056
	50	100.0% ± 0.0	0.000 ± 0.000	99.6% ± 0.6	0.002 ± 0.003	47.2% ± 6.8	0.480 ± 0.058
	100	100.0% ± 0.0	0.000 ± 0.000	100.0 % ± 0.0	0.000 ± 0.000	52.2% ± 1.6	0.430 ± 0.051
5 rec. steps	1	100.0% ± 0.0	0.000 ± 0.000	93.6% ± 2.8	0.083 ± 0.035	1.4% ± 1.7	9.298 ± 0.754
	5	100.0% ± 0.0	0.000 ± 0.000	91.8% ± 2.3	0.118 ± 0.030	18.2% ± 4.4	2.891 ± 0.187
	10	100.0% ± 0.0	0.000 ± 0.000	91.0% ± 2.5	0.101 ± 0.027	24.8% ± 3.5	1.691 ± 0.158
	25	100.0% ± 0.0	0.000 ± 0.000	93.6% ± 1.3	0.039 ± 0.012	36.2% ± 1.1	0.936 ± 0.093
	50	100.0% ± 0.0	0.000 ± 0.000	91.8% ± 2.6	0.030 ± 0.013	35.8% ± 4.8	0.835 ± 0.168
	100	100.0% ± 0.0	0.000 ± 0.000	95.4% ± 5.5	0.017 ± 0.001	40.0% ± 3.7	0.600 ± 0.088

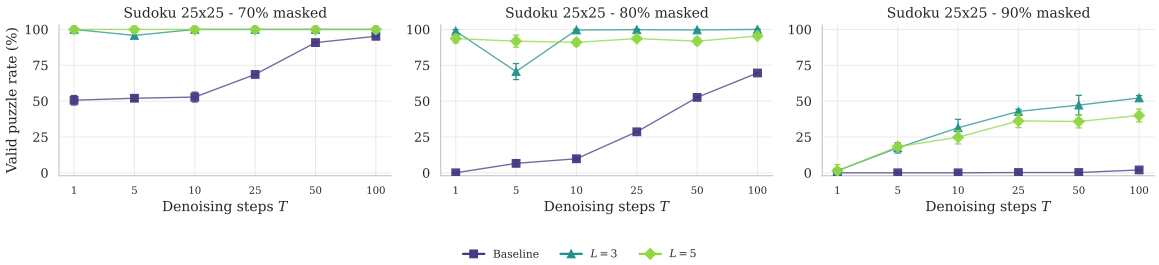

Figure 12: Effect of recursive refinement depth (L) on 25 × 25 Sudoku reconstruction under different masking conditions. Across all masking regimes, recursion improves valid puzzle recovery.

F.5. Countdown: Full Tables

Table 8 reports Countdown performance for recursive models and the baseline across $k \in \{3, 4, 5\}$ operands, varying both recursion depth and denoising step budget. Table 9 reports the equivalent depth-scaling analysis for non-recursive models with 3, 6, 9, 15, and 30 layers.

For Countdown-3, the task is tractable even for the baseline at high T (96.0% RTR at $T{=}30$), but recursion substantially accelerates convergence: $L{=}3$ reaches 92.4% at $T{=}10$, a step budget at which the baseline achieves only 59.4%. For Countdown-4 and -5, the baseline saturates at substantially lower accuracy than the recursive models, suggesting that the difficulty of multi-step arithmetic planning grows faster than what additional denoising steps alone can resolve.

For the depth-scaling analysis (Table 9), results show that a 15-layer non-recursive model (26.7M) on Countdown-4 reaches 80.0% RTR at $T{=}30$, while the $(3 \otimes 5)$ model (5.5M) achieves 81.8%, comparable performance at less than one-fifth the parameter count.

Table 8: Countdown performance with different recursive steps. We report mean and standard deviation across 5 sampling runs of 100 samples each as well as the total sampling time.

Model	Dec. steps	RTR %	PPF %	LAF %	TRN	Sampling time (s)
Countdown 3						
Baseline	1	7.6% ± 1.1	19.0% ± 1.2	26.2% ± 2.7	1.046 ± 0.300	2.79
	5	44.8% ± 3.0	51.0% ± 3.4	64.5% ± 4.1	0.520 ± 0.027	8.27
	10	59.4% ± 5.0	63.5% ± 4.2	75.8% ± 3.9	0.346 ± 0.061	7.32
	20	93.2% ± 1.6	94.0% ± 1.7	94.3% ± 1.4	0.068 ± 0.033	41.51
	30	96.0% ± 0.7	96.3% ± 0.5	96.7% ± 0.6	0.037 ± 0.011	56.41
3 recursive steps	1	67.0% ± 4.6	70.1% ± 5.6	76.9% ± 4.6	0.307 ± 0.053	3.52
	5	87.6% ± 1.8	89.2% ± 2.0	92.9% ± 1.9	0.145 ± 0.086	14.43
	10	92.4% ± 2.5	93.2% ± 2.7	95.5% ± 2.0	0.061 ± 0.017	28.24
	20	97.4% ± 1.5	97.6% ± 1.6	98.0% ± 1.2	0.021 ± 0.012	52.85
	30	97.8% ± 1.6	98.2% ± 1.5	98.3% ± 1.3	0.030 ± 0.024	55.96
5 recursive steps	1	67.4% ± 2.1	70.3% ± 2.0	79.5% ± 1.5	0.317 ± 0.088	5.57
	5	85.8% ± 6.8	86.6% ± 6.0	93.0% ± 3.7	0.122 ± 0.059	23.32
	10	91.0% ± 1.6	91.5% ± 1.5	95.4% ± 0.9	0.071 ± 0.015	46.97
	20	98.4% ± 0.9	98.7% ± 0.8	98.7% ± 0.8	0.012 ± 0.011	86.52
	30	98.2% ± 1.9	98.5% ± 1.5	98.8% ± 0.9	0.016 ± 0.016	91.50
Countdown 4						
Baseline	1	0.0% ± 0.0	1.9% ± 0.6	4.1% ± 1.1	1.054 ± 0.169	2.55
	5	6.6% ± 3.1	16.7% ± 3.2	36.2% ± 1.2	1.001 ± 0.231	5.67
	10	18.0% ± 2.5	31.6% ± 3.5	49.3% ± 3.6	0.817 ± 0.140	10.82
	20	38.2% ± 6.2	52.3% ± 4.4	64.5% ± 3.8	0.648 ± 0.206	21.36
	30	44.8% ± 6.0	59.5% ± 5.2	71.5% ± 3.3	0.626 ± 0.046	31.88
3 recursive steps	1	0.8% ± 0.8	5.5% ± 1.2	15.0% ± 1.7	1.243 ± 0.401	4.51
	5	11.6% ± 2.4	28.4% ± 2.3	53.2% ± 2.2	0.908 ± 0.180	16.52
	10	39.6% ± 1.8	56.9% ± 1.7	72.6% ± 1.0	0.593 ± 0.143	32.83
	20	68.6% ± 6.2	77.6% ± 4.0	85.1% ± 3.0	0.364 ± 0.150	63.77
	30	76.8% ± 2.6	84.4% ± 1.6	89.6% ± 1.0	0.256 ± 0.131	83.95
5 recursive steps	1	0.6% ± 0.6	6.7% ± 2.6	17.2% ± 1.9	1.021 ± 0.145	7.46
	5	15.4% ± 3.0	30.9% ± 2.0	54.2% ± 1.6	0.802 ± 0.075	28.37
	10	37.0% ± 2.9	53.7% ± 3.1	69.9% ± 2.0	0.580 ± 0.088	50.52
	20	70.2% ± 7.0	78.3% ± 5.2	85.6% ± 3.3	0.269 ± 0.055	93.76
	30	81.8% ± 2.5	86.9% ± 1.0	90.4% ± 1.1	0.212 ± 0.116	137.65
10 recursive steps	1	0.8% ± 0.8	5.6% ± 2.4	14.8% ± 3.2	0.876 ± 0.043	10.50
	5	12.8% ± 2.2	27.6% ± 1.2	48.1% ± 1.9	0.848 ± 0.118	46.15
	10	43.6% ± 4.2	56.7% ± 4.1	70.2% ± 3.5	0.676 ± 0.317	91.65
	20	74.4% ± 5.3	81.1% ± 4.6	86.1% ± 2.8	0.261 ± 0.102	179.72
	30	86.2% ± 4.1	89.7% ± 3.0	91.7% ± 2.5	0.211 ± 0.184	264.44
Countdown 5						
Baseline	1	0.0% ± 0.0	0.3% ± 0.3	1.3% ± 0.6	0.925 ± 0.112	2.75
	5	0.2% ± 0.5	11.1% ± 1.3	33.7% ± 0.9	1.023 ± 0.192	5.68
	10	3.8% ± 2.4	20.9% ± 2.8	47.0% ± 2.9	0.909 ± 0.069	11.23
	20	9.6% ± 1.8	28.3% ± 2.3	57.4% ± 1.7	0.886 ± 0.161	21.74
	30	15.6% ± 5.1	37.6% ± 4.1	63.3% ± 3.0	0.865 ± 0.229	32.22
	40	17.0% ± 2.6	40.5% ± 4.4	65.1% ± 3.4	0.889 ± 0.168	166.18
3 recursive steps	1	0.0% ± 0.0	0.5% ± 0.4	1.6% ± 0.6	1.036 ± 0.229	3.50
	5	4.8% ± 1.8	20.9% ± 1.2	47.6% ± 1.7	0.971 ± 0.169	14.00
	10	10.8% ± 2.2	31.0% ± 1.6	60.9% ± 1.9	0.758 ± 0.068	28.00
	20	21.6% ± 4.2	42.6% ± 5.5	70.9% ± 2.5	0.913 ± 0.153	55.50
	30	36.6% ± 4.9	57.5% ± 2.2	79.6% ± 1.0	0.746 ± 0.160	83.00
	40	39.6% ± 7.0	60.3% ± 6.0	80.8% ± 3.7	0.782 ± 0.125	107.00
5 recursive steps	1	0.0% ± 0.0	0.7% ± 0.4	2.6% ± 0.6	1.136 ± 0.123	6.42
	5	4.8% ± 1.9	23.6% ± 1.9	47.0% ± 4.5	1.051 ± 0.275	24.82
	10	14.2% ± 3.4	34.0% ± 2.1	59.6% ± 2.1	0.864 ± 0.177	48.76
	20	24.6% ± 7.2	44.6% ± 6.4	69.7% ± 3.8	0.822 ± 0.303	97.16
	30	44.4% ± 4.7	62.1% ± 4.8	80.4% ± 2.1	0.585 ± 0.122	141.88
	40	46.8% ± 6.5	64.3% ± 4.3	81.3% ± 1.3	0.586 ± 0.270	174.04
10 recursive steps	1	0.2% ± 0.5	1.2% ± 1.0	3.5% ± 0.9	1.107 ± 0.330	10.50
	5	5.8% ± 1.3	22.4% ± 1.9	44.2% ± 2.8	0.856 ± 0.204	43.51
	10	14.4% ± 4.3	35.1% ± 4.1	58.2% ± 4.7	0.880 ± 0.166	86.57
	20	28.4% ± 4.4	49.3% ± 4.2	69.4% ± 2.5	0.654 ± 0.033	173.24
	30	56.4% ± 5.0	71.4% ± 4.2	82.0% ± 2.0	0.671 ± 0.355	260.25
	40	53.8% ± 4.3	69.6% ± 3.4	80.4% ± 2.0	0.538 ± 0.208	330.43

Table 9: Countdown performance with models of different depth. We report mean and standard deviation across 5 sampling runs of 100 samples each as well as the total sampling time.

Model (params)	Dec. steps	RTR %	PPF %	LAF %	TRN	Sampling time (s)
Countdown 3						
3 layers (5.5M)	1	7.6% ± 1.1	19.0% ± 1.2	26.2% ± 2.7	1.046 ± 0.300	2.79
	5	44.8% ± 3.0	51.0% ± 3.4	64.5% ± 4.1	0.520 ± 0.027	8.27
	10	59.4% ± 5.0	63.5% ± 4.2	75.8% ± 3.9	0.346 ± 0.061	7.32
	20	93.2% ± 1.6	94.0% ± 1.7	94.3% ± 1.4	0.068 ± 0.033	41.51
	30	96.0% ± 0.7	96.3% ± 0.5	96.7% ± 0.6	0.037 ± 0.011	56.41
6 layers (10.8M)	1	20.0% ± 4.1	28.5% ± 3.5	39.9% ± 4.8	0.865 ± 0.201	2.52
	5	56.8% ± 6.5	59.9% ± 6.4	73.2% ± 4.9	0.413 ± 0.093	9.36
	10	74.4% ± 5.8	76.4% ± 5.9	85.4% ± 4.0	0.341 ± 0.186	18.48
	20	97.2% ± 1.8	97.4% ± 1.5	97.6% ± 1.4	0.025 ± 0.014	34.41
	30	97.0% ± 1.2	97.3% ± 1.2	97.8% ± 1.2	0.028 ± 0.013	35.78
9 layers (16.1M)	1	36.0% ± 4.6	43.1% ± 3.4	54.8% ± 2.6	0.558 ± 0.043	4.89
	5	72.4% ± 3.6	75.0% ± 3.5	83.2% ± 2.6	0.389 ± 0.261	13.17
	10	86.6% ± 2.7	87.6% ± 2.1	92.1% ± 2.0	0.121 ± 0.013	26.12
	20	96.0% ± 1.9	96.5% ± 1.7	97.2% ± 1.8	0.036 ± 0.015	48.75
	30	95.0% ± 1.9	95.7% ± 1.4	96.2% ± 1.4	0.131 ± 0.208	51.51
15 layers (26.7M)	1	59.0% ± 3.9	64.3% ± 4.0	73.4% ± 3.0	0.405 ± 0.090	5.67
	5	88.0% ± 1.4	90.0% ± 0.7	91.6% ± 0.8	0.119 ± 0.049	20.54
	10	94.4% ± 4.2	95.1% ± 3.6	96.1% ± 2.8	0.052 ± 0.037	40.93
	20	95.6% ± 2.2	96.4% ± 2.0	96.8% ± 1.6	0.036 ± 0.025	76.25
	30	95.2% ± 1.3	96.0% ± 1.2	96.6% ± 1.1	0.038 ± 0.013	83.87
Countdown 4						
3 layers (5.5M)	1	0.0% ± 0.0	1.9% ± 0.6	4.1% ± 1.1	1.054 ± 0.169	2.55
	5	6.6% ± 3.1	16.7% ± 3.2	36.2% ± 1.2	1.001 ± 0.231	5.67
	10	18.0% ± 2.5	31.6% ± 3.5	49.3% ± 3.6	0.817 ± 0.140	10.82
	20	38.2% ± 6.2	52.3% ± 4.4	64.5% ± 3.8	0.648 ± 0.206	21.36
	30	44.8% ± 6.0	59.5% ± 5.2	71.5% ± 3.3	0.626 ± 0.046	31.88
6 layers (10.8M)	1	0.4% ± 0.6	3.8% ± 1.0	10.7% ± 1.2	1.003 ± 0.180	3.33
	5	10.2% ± 3.6	24.0% ± 4.8	47.9% ± 1.2	0.915 ± 0.129	13.44
	10	34.8% ± 4.3	48.9% ± 5.0	65.1% ± 3.2	0.755 ± 0.269	26.62
	20	59.8% ± 3.1	71.7% ± 3.7	79.2% ± 3.3	0.385 ± 0.058	52.90
	30	68.4% ± 4.0	77.6% ± 2.3	83.1% ± 1.3	0.445 ± 0.255	76.06
9 layers (16.1M)	1	0.2% ± 0.5	5.7% ± 1.7	14.1% ± 2.1	1.023 ± 0.108	5.19
	5	15.0% ± 3.5	28.0% ± 2.8	51.3% ± 2.8	0.837 ± 0.140	20.07
	10	38.8% ± 2.7	51.9% ± 1.5	68.2% ± 1.9	0.591 ± 0.071	42.87
	20	64.2% ± 4.4	73.0% ± 3.5	80.0% ± 2.1	0.359 ± 0.091	84.29
	30	74.2% ± 8.0	81.3% ± 5.7	86.1% ± 4.3	0.227 ± 0.086	106.25
15 layers (26.7M)	1	1.4% ± 1.1	9.1% ± 2.1	24.2% ± 2.2	0.999 ± 0.303	5.65
	5	21.4% ± 4.9	33.9% ± 3.1	60.1% ± 3.6	0.781 ± 0.124	22.96
	10	48.2% ± 2.6	59.6% ± 2.2	75.5% ± 1.8	0.453 ± 0.046	41.23
	20	69.4% ± 4.3	78.3% ± 2.6	84.7% ± 1.8	0.269 ± 0.066	81.46
	30	80.0% ± 2.1	85.7% ± 1.9	89.5% ± 2.3	0.183 ± 0.042	121.79
30 layers (53.3M)	1	5.2% ± 2.3	15.2% ± 2.1	32.4% ± 3.5	0.886 ± 0.075	8.69
	5	32.4% ± 4.9	42.6% ± 3.7	67.2% ± 3.7	0.616 ± 0.047	40.17
	10	58.0% ± 2.7	68.7% ± 2.8	80.4% ± 1.3	0.356 ± 0.051	79.95
	20	69.6% ± 2.6	78.2% ± 1.8	85.4% ± 2.0	0.362 ± 0.152	159.88
	30	72.6% ± 2.4	81.3% ± 2.0	86.3% ± 1.6	0.300 ± 0.144	231.88
Countdown 5						
3 layers (5.5M)	1	0.0% ± 0.0	0.3% ± 0.3	1.3% ± 0.6	0.925 ± 0.112	2.75
	5	0.2% ± 0.5	11.1% ± 1.3	33.7% ± 0.9	1.023 ± 0.192	5.68
	10	3.8% ± 2.4	20.9% ± 2.8	47.0% ± 2.9	0.909 ± 0.069	11.23
	20	9.6% ± 1.8	28.3% ± 2.3	57.4% ± 1.7	0.886 ± 0.161	21.74
	30	15.6% ± 5.1	37.6% ± 4.1	63.3% ± 3.0	0.865 ± 0.229	32.22
	40	17.0% ± 2.6	40.5% ± 4.4	65.1% ± 3.4	0.889 ± 0.168	166.18
6 layers (10.8M)	1	0.0% ± 0.0	0.9% ± 0.4	2.1% ± 1.1	0.976 ± 0.162	2.51
	5	1.8% ± 0.8	17.5% ± 2.3	45.7% ± 3.3	0.923 ± 0.150	9.51
	10	8.6% ± 0.6	27.9% ± 1.3	54.4% ± 1.2	1.047 ± 0.397	18.40
	20	15.4% ± 2.9	36.9% ± 5.4	64.3% ± 2.6	1.017 ± 0.176	36.99
	30	28.4% ± 3.8	51.2% ± 4.0	71.9% ± 2.8	0.708 ± 0.121	54.75
	40	29.4% ± 3.8	53.6% ± 3.1	73.3% ± 1.6	0.845 ± 0.279	72.69
9 layers (16.1M)	1	0.0% ± 0.0	1.3% ± 0.4	2.7% ± 0.9	1.128 ± 0.232	4.01
	5	2.8% ± 1.3	18.5% ± 2.2	43.4% ± 1.3	0.953 ± 0.201	13.15
	10	9.4% ± 3.2	28.6% ± 3.4	56.0% ± 3.3	0.992 ± 0.322	25.85
	20	22.0% ± 5.3	40.6% ± 4.3	65.0% ± 3.5	0.718 ± 0.060	56.03
	30	37.8% ± 3.3	57.4% ± 2.6	75.0% ± 2.3	0.714 ± 0.226	87.30
	40	39.6% ± 2.3	60.7% ± 1.6	77.4% ± 1.5	0.604 ± 0.090	99.92
15 layers (26.7M)	1	0.0% ± 0.0	1.2% ± 0.6	3.0% ± 0.7	0.977 ± 0.170	5.81
	5	4.0% ± 1.7	20.3% ± 3.4	44.4% ± 3.2	0.909 ± 0.076	23.30
	10	11.2% ± 2.4	31.7% ± 2.3	60.8% ± 0.9	1.091 ± 0.381	43.96
	20	21.8% ± 5.0	44.6% ± 3.7	69.7% ± 2.2	0.764 ± 0.109	83.75
	30	42.2% ± 4.6	63.5% ± 2.9	81.2% ± 1.6	0.587 ± 0.079	125.12
	40	45.8% ± 3.3	64.9% ± 2.0	82.2% ± 2.0	0.530 ± 0.073	161.05
30 layers (53.3M)	1	0.0% ± 0.0	1.7% ± 0.3	3.6% ± 0.8	0.912 ± 0.047	8.57
	5	5.0% ± 1.6	21.5% ± 1.7	47.1% ± 3.3	0.859 ± 0.117	39.80
	10	15.0% ± 3.9	34.8% ± 3.3	60.3% ± 2.5	0.927 ± 0.347	79.44
	20	28.2% ± 3.9	47.5% ± 3.1	70.0% ± 1.4	0.698 ± 0.165	158.83
	30	45.4% ± 5.2	65.2% ± 4.2	79.6% ± 2.6	0.479 ± 0.048	237.60
	40	47.8% ± 6.0	66.6% ± 3.4	80.6% ± 3.2	0.494 ± 0.149	304.52

F.6. Text8 Results

Table 10 reports generative perplexity (Gen-PPL), NLL, entropy, and bits-per-dimension (BPD) on the Text8 validation set, for denoising budgets $T \in \{18, 32, 64, 96\}$. The results show that recursive models underperform the single-pass baseline across all metrics and step budgets. The gap narrows at high T (T=96), where additional denoising steps seem to partially compensate for the weaker per-step predictions. On the other hand, entropy is marginally higher for recursive models.

However, this gap may partly reflect limitations of the evaluation metrics rather than the recursive mechanism itself. Likelihood-based metrics can be misleading in this regime: lower entropy or NLL may arise from overconfident but degenerate predictions, rather than genuinely coherent text. Our qualitative samples (Table 11) suggest that recursive models produce more coherent generations despite their worse likelihood scores. We leave further exploration of this as future work.

Table 10: text8 Evaluation Results. We report mean and standard deviation across 5 sampling runs of 100 samples each.

Model	Dec. steps	Entropy	NLL	Gen-PPL	BPD	Time (s)
Baseline	18	2.237 ± 0.011	4.705 ± 0.064	126.43 ± 6.85	6.788 ± 0.092	35.24
	32	2.325 ± 0.009	5.190 ± 0.043	210.05 ± 11.86	7.488 ± 0.063	58.53
	64	2.395 ± 0.017	5.528 ± 0.074	299.93 ± 23.47	7.975 ± 0.107	116.64
	96	2.444 ± 0.010	5.694 ± 0.063	346.10 ± 22.44	8.215 ± 0.091	175.54
3 recursive steps	18	2.389 ± 0.005	5.422 ± 0.050	249.80 ± 12.38	7.823 ± 0.073	89.16
	32	2.470 ± 0.016	5.776 ± 0.046	356.01 ± 16.09	8.332 ± 0.066	157.80
	64	2.514 ± 0.013	5.880 ± 0.074	412.92 ± 33.01	8.484 ± 0.107	314.93
	96	2.554 ± 0.015	5.875 ± 0.085	408.70 ± 31.23	8.476 ± 0.123	473.29
5 recursive steps	18	2.412 ± 0.006	5.539 ± 0.066	277.97 ± 17.17	7.991 ± 0.094	148.26
	32	2.464 ± 0.016	5.747 ± 0.050	345.82 ± 15.54	8.292 ± 0.071	261.17
	64	2.501 ± 0.019	5.782 ± 0.063	374.53 ± 23.02	8.342 ± 0.090	526.03
	96	2.535 ± 0.017	5.795 ± 0.081	379.96 ± 25.36	8.360 ± 0.117	785.92

Appendix G. Visualizing Text8 samples

Table 11: **Qualitative samples at 64 decoding steps.** Conditional generations (prompt continuation) for baseline (no recursion) and with 3 and 5 recursive steps and 64 decoding steps. Recursion tends to produce continuations with more topical anchoring and more coherent phrase-level structure, even when the text remains noisy.

Setting	Prompt	Model output (64 steps)
Baseline	and humans to build a more perfect world utopia has un sanction	and humans to build a more perfect world utopia has un sanctions were life was tee ed the time of the refeate and the the has hade the sayed the less toes like the goes ofe ohe sarans ane hans lee me derne s s dee she are erene e i re me a le le mene e
	d forward to his companion a reporter and said they re wishing m	d forward to his companion a reporter and said they re wishing more tod late the tne toen the marines and other and lake tad and so lone the same lame and ohe lone more me and water der see lie an eye de teened ohe la mes rente re and me men yo sand le
	n is not one to one and makes an effective hash function a typic	n is not one to one and makes an effective hash function a typically none i int tne the nine are line the n one n is are tne one a a a a n a l n n inen and a let ln i n n int ne left in t in ana a i i i a llet l a e al in a na l ia i ina l li in in i i
3 rec. steps	and humans to build a more perfect world utopia has un sanction	and humans to build a more perfect world utopia has un sanctions the oo the engthes on the trame games aod the voted on the mone on age thst one in thme is the two sert there an e carter name de see masthe site in site the name of prace on the mantre de
	d forward to his companion a reporter and said they re wishing m	d forward to his companion a reporter and said they re wishing many oe weres coverted to the degree of an end ahd made the age of the fire sore in sone of the of thre dhd lide line the tne th pragment in de stee to stone one on the horse de dees the de
	n is not one to one and makes an effective hash function a typic	n is not one to one and makes an effective hash function a typically decline a seit was some of the game aad an an anticant thereal game eriors the time nated to be one of thee the readers on the game game an effective game function gun ips and te game the
5 rec. steps	and humans to build a more perfect world utopia has un sanction	and humans to build a more perfect world utopia has un sanctions the east time same on the nine of the roman s names of ohe han connerted on the of the renties in the name as toe sons on the lone secares concring the ahe of the sentence one severe is the
	d forward to his companion a reporter and said they re wishing m	d forward to his companion a reporter and said they re wishing many the date of dates on the parties ted aater thes made th the waiting tender to oeers on the power of date he tne reaches a rarations of largers of date they have the tones on aelites o
	n is not one to one and makes an effective hash function a typic	n is not one to one and makes an effective hash function a typically fnr the time in some days there is the border ins the sea lone sea es with the sea same in the same a styge if ths has to set the state where one with the laster s is line is like the se