SCALING ORDER LEARNING FOR DIFFUSION LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

The standard masked diffusion model (MDM) objective treats all unmasking orders uniformly, potentially allocating substantial training compute to orders poorly suited to individual examples. The resulting training inefficiency makes high-performing MDMs costly to train even when initialized from strong autoregressive models (ARMs). We introduce SOLDIFF, a scalable variational framework that learns data-dependent order distributions to train stronger MDMs under limited compute budgets. We model orders as latent permutations, with a learned posterior focusing token-prediction training on mask patterns induced by preferred orders and a learned prior reproducing these preferences at inference. To scale this formulation to MDM pretraining, we systematically identify key design choices for stable and effective optimization and introduce rank-weighted auxiliary objectives to support parallel decoding. Across 12 benchmarks, SOLDIFF-8B outperforms all other evaluated ARMs and MDMs of comparable size on average. We highlight that this result requires only 100B tokens of continued pretraining, less than 0.3% of the base model's pretraining tokens and substantially fewer than those used in existing ARM-to-MDM adaptations. In controlled comparisons, SOLDIFF outperforms standard block diffusion trained with $6.8\times$ as much compute. We further observe consistent gains across model sizes from 1.7B to 14B.

1 INTRODUCTION

Masked diffusion models (MDMs) support flexible generation orders and parallel token generation (Nie et al., 2025b; Li et al., 2026). Their strong performance across a wide range of tasks and recent scaling advances (Zhu et al., 2026; Bie et al., 2026) highlight their potential as an alternative to autoregressive models (ARMs) (Yang et al., 2025; AI@Meta, 2024). However, standard MDM training is considerably less compute-efficient than ARM training (Nie et al., 2025a; Sahoo et al., 2026; Ye et al., 2025), making ARM-competitive performance expensive to obtain even with strong ARM initialization. This inefficiency limits the performance attainable under practical compute budgets, making better use of training compute essential to developing stronger MDMs.

We argue that a key source of this inefficiency lies in how standard MDM training allocates compute across unmasking orders. ARMs optimize next-token prediction in a fixed left-to-right order, while the standard MDM objective averages token-prediction losses uniformly over all possible unmasking orders (Figure 2(a)). As a result, substantial training compute can be spent on orders poorly suited to individual examples, such as revealing a conclusion before its supporting reasoning.

To address this inefficiency, we introduce a **S**calable **O**rder **L**earning framework for **Diff**usion large language models (SOLDIFF) that learns a data-dependent order distribution to guide the allocation of training compute, enabling stronger MDMs under a given compute budget. As illustrated in Figure 2(b), we model the unmasking order as a latent permutation and formulate order learning via variational inference (Kingma & Welling, 2013; Wang et al., 2025). The resulting evidence lower bound (ELBO) uses a clean-sequence-conditioned posterior q_ϕ to focus token-prediction training on mask patterns induced by preferred orders, while aligning a masked-state-conditioned prior p_η with the posterior so that the learned preferences can guide inference. Scaling existing latent-order formulations to MDM pretraining poses substantial optimization challenges. To this end, we systematically identify the key design choices needed for stable and effective joint optimization of the order models and token predictor. Since the learned-order objective emphasizes the next unmasking step, we further introduce rank-weighted auxiliary objectives over the remaining masked positions to support parallel decoding.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

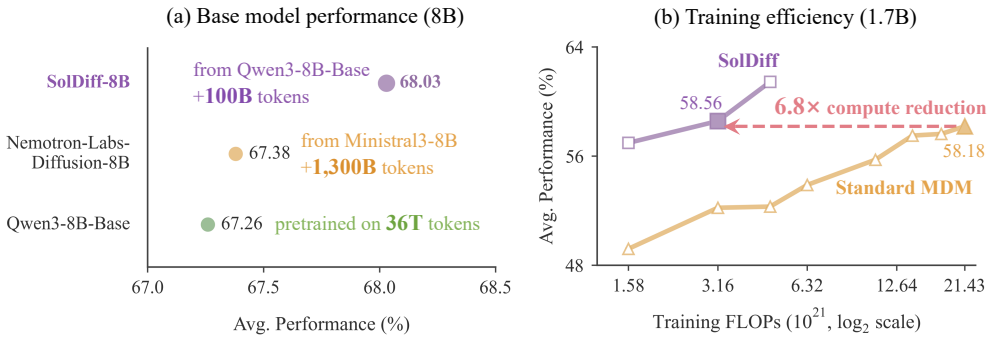

Figure 1: (a) Performance, 12-benchmark average; (b) training efficiency, 4-benchmark average.

Extensive experiments across general knowledge and reasoning, mathematics, and code generation demonstrate that SOLDIFF substantially improves MDM training compute efficiency and sets a new state of the art for MDMs. SOLDIFF-8B achieves the highest average score across 12 benchmarks among all evaluated ARMs and MDMs of comparable size (Figure 1(a)). While existing ARM-to-MDM methods often require extensive continued pretraining, we obtain this result with only 100B tokens, less than 0.3% of the 36T-token pretraining budget reported for Qwen3-8B-Base (Yang et al., 2025). SOLDIFF also outperforms standard block diffusion trained with $6.8\times$ as much compute (Figure 1(b)). FLOPs-matched comparisons attribute these gains over MDM baselines to learned-order training rather than differences in data or initialization. This advantage persists as we scale models from 1.7B to 14B parameters. With parallel-aware training, SOLDIFF achieves a better performance–efficiency trade-off than standard block diffusion under parallel decoding.

2 PRELIMINARY

Any-order Autoregressive Models and Masked Diffusion Models. Any-order autoregressive models (AO-ARMs) (Uria et al., 2014; Hoogeboom et al., 2021) minimize token-prediction losses averaged over uniformly sampled generation orders, with each token prediction conditioned on the tokens preceding it in the sampled order. Masked Diffusion Models (MDMs) (Nie et al., 2025b; Ye et al., 2025) instead sample a mask ratio t uniformly from $[0, 1]$, independently mask each token with probability t, and predict the masked tokens from the resulting partially observed context. Nie et al. (2025b) show that the MDM objective is equivalent to the uniform-order AO-ARM objective.

Learning-order Autoregressive Model. LO-ARM (Wang et al., 2025) models the generation order as a latent permutation over all variables and learns a state-dependent policy to select the next variable from the partial state. It introduces an amortized posterior conditioned on fully observed data, enabling variational training of the token predictor and order policy through an evidence lower bound (ELBO). LO-ARM is evaluated primarily on molecular graph generation using serial decoding. In MDM pretraining on natural language, the challenge is to achieve stable and effective scaling of latent-order learning while supporting parallel decoding.

3 METHODOLOGY

We present SOLDIFF, a block-wise variational framework for scaling order learning in MDM pretraining, with extended supervision to support parallel decoding (Figure 2).

3.1 VARIATIONAL FORMULATION

To scale latent-order modeling (Wang et al., 2025) to MDM pretraining, we adopt a block-wise variational formulation that mitigates the optimization difficulties of full-sequence order learning (§5.4). Detailed derivations are provided in Appendix A.1. We partition a length-L sequence $\boldsymbol{x}$ into B consecutive blocks $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^B)$. The inter-block order is fixed left-to-right, while the intra-block unmasking order is flexible. Attention is bidirectional within blocks and causal across blocks. We introduce a latent permutation $\boldsymbol{z}^b = (z_1^b, \ldots, z_{n_b}^b)$ to specify the unmasking order within each block b of length n_b. At step i, the prefix $\boldsymbol{z}_{<i}^b$, which lists the revealed token positions, induces a masked

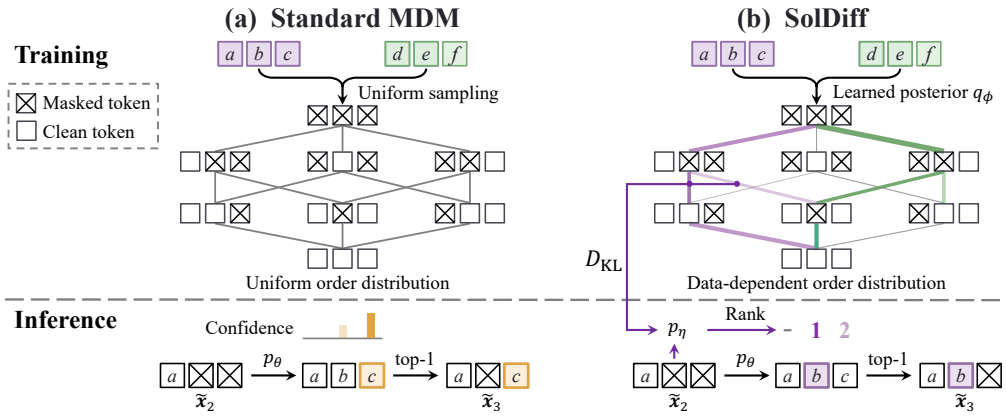

Figure 2: Training and inference in standard MDMs and SOLDIFF. (a) Standard MDMs train uniformly over unmasking orders and use a heuristic order at inference. (b) SOLDIFF focuses training on data-dependent orders induced by $q_\phi(\cdot \mid \boldsymbol{x})$, and aligns p_η with q_ϕ to reproduce these preferences at inference. Thick paths indicate higher-probability orders under q_ϕ for different examples.

state $\widetilde{\boldsymbol{x}}_i^b$ in which all preceding blocks $\boldsymbol{x}^{<b}$ and revealed tokens $\boldsymbol{x}^b_{\boldsymbol{z}^b_{<i}}$ are visible, while the remaining positions in block b are masked. Given this masked state, an order prior p_η selects the next position to unmask, and p_θ predicts the corresponding token. The likelihood of block b then factorizes as

$$p_{\theta,\eta}(\boldsymbol{x}^b \mid \boldsymbol{x}^{<b}) = \sum_{\boldsymbol{z}^b} p_{\theta,\eta}(\boldsymbol{z}^b, \boldsymbol{x}^b \mid \boldsymbol{x}^{<b}) = \sum_{\boldsymbol{z}^b} \prod_{i=1}^{n_b} p_\eta(z_i^b \mid \boldsymbol{z}^b_{<i}, \widetilde{\boldsymbol{x}}_i^b)\, p_\theta(x^b_{z_i^b} \mid \widetilde{\boldsymbol{x}}_i^b). \tag{1}$$

We introduce a variational posterior q_ϕ to make the likelihood tractable to optimize:

$$q_\phi(\boldsymbol{z}^b \mid \boldsymbol{x}^{\le b}) = \prod_{i=1}^{n_b} q_\phi(z_i^b \mid \boldsymbol{z}^b_{<i}, \boldsymbol{x}^{\le b}). \tag{2}$$

This yields the following evidence lower bound (ELBO), written as a sum of one-step terms:

$$\log p_{\theta,\eta}(\boldsymbol{x}) \ge \mathcal{E}_{\mathrm{ELBO}}(\boldsymbol{x}) = \sum_{b=1}^{B} \sum_{i=1}^{n_b} \mathbb{E}_{q_\phi(\boldsymbol{z}^b_{<i} \mid \boldsymbol{x}^{\le b})} F_i^b(\boldsymbol{z}^b_{<i}, \boldsymbol{x}^{\le b}). \tag{3}$$

Since the one-step term depends only on $\boldsymbol{z}^b_{<i}$ and z_i^b, the suffix $\boldsymbol{z}^b_{>i}$ can be marginalized out, while the expectation over z_i^b is computed exactly. For a fixed prefix $\boldsymbol{z}^b_{<i}$, we define

$$F_i^b = \underbrace{\mathbb{E}_{q_\phi(z_i^b \mid \boldsymbol{z}^b_{<i}, \boldsymbol{x}^{\le b})} \left[\log p_\theta \left(x^b_{z_i^b} \mid \widetilde{\boldsymbol{x}}_i^b \right) \right]}_{\text{posterior-weighted token reconstruction}} - \underbrace{D_{\mathrm{KL}} \left(q_\phi \left(z_i^b \mid \boldsymbol{z}^b_{<i}, \boldsymbol{x}^{\le b} \right) \,\|\, p_\eta \left(z_i^b \mid \boldsymbol{z}^b_{<i}, \widetilde{\boldsymbol{x}}_i^b \right) \right)}_{\text{posterior–prior order alignment}}. \tag{4}$$

The first term trains the token predictor p_θ on posterior-induced masked states, with posterior-weighted supervision over candidate next positions. The second encourages the masked-state-conditioned prior p_η to match the full-context posterior. For the posterior, the objective favors next positions with high token likelihood while ensuring that the learned order remains predictable from the masked state.

3.2 SCALABLE DESIGN FOR LEARNED-ORDER TRAINING

Scaling this formulation requires effective co-adaptation between the order models and token predictor while maintaining stable joint optimization. Whereas previous work compared architectural variants trained to convergence for autoregressive molecular generation (Wang et al., 2025), we systematically evaluate four key design choices for MDM pretraining under matched training FLOPs (§5.4):

(1) Posterior. To study posterior capacity and backbone coupling, we consider (i) a backbone-shared posterior with a SwiGLU scoring head (Shazeer, 2020) on hidden states from a separate clean-sequence backbone pass, with optional detachment; and (ii) an independent transformer posterior

Algorithm 1 One training step of SOLDIFF

Require: Dataset $\mathcal{D}$; posterior q_ϕ, token predictor p_θ, prior p_η

1: Sample a sequence $\boldsymbol{x} \sim \mathcal{D}$ and partition it into B blocks
2: Run q_ϕ once on clean $\boldsymbol{x}$ to obtain posterior order scores for all blocks
3: Sample $t \sim \mathrm{Uniform}(0,1)$; set $i_b = n_b - \lceil t n_b \rceil + 1, \forall b$
4: Sample $\boldsymbol{z}^{b,1}, \boldsymbol{z}^{b,2} \overset{\text{i.i.d.}}{\sim} q_\phi(\cdot \mid \boldsymbol{x}^{\le b})$ from the same posterior scores, $\forall b$
5: Mask the order suffix $\boldsymbol{z}^{b,j}_{\ge i_b}$ in block b to form $\tilde{\boldsymbol{x}}^{b,j}_{i_b}$, keeping $\boldsymbol{x}^{<b}$ and $\boldsymbol{x}^b_{\boldsymbol{z}^{b,j}_{<i_b}}$ visible, $\forall b,\ j = 1,2$
6: Run one p_θ forward pass per posterior path; reuse its hidden states for p_η
7: Compute $F^{b,j}$ by Eq. 4 and $\mathcal{L}_{\mathrm{LO}}$ by Eq. 5
8: Compute rank-weighted $\mathcal{L}_{\mathrm{rec}}$ and $\mathcal{L}_{\mathrm{ord}}$ using the same outputs and detached posterior suffixes
9: Update (θ, ϕ, η) using $\mathcal{L}_{\mathrm{train}} = \mathcal{L}_{\mathrm{LO}} + \lambda_{\mathrm{rec}}\mathcal{L}_{\mathrm{rec}} + \lambda_{\mathrm{ord}}\mathcal{L}_{\mathrm{ord}}$ in Eq. 9

with a configurable number of layers. **(2) Prior.** To study prior parameterization and backbone co-adaptation, we consider (i) a confidence-based prior derived from the token predictor's top-1 confidence; and (ii) a learned prior using a linear or SwiGLU scoring head on masked-state backbone hidden states, with optional detachment. **(3) Estimator.** Because the sampled prefix induces a high-variance score-function term, we compare (i) a single-sample REINFORCE estimator, (ii) REINFORCE with a moving-average baseline (Kool et al., 2018), and (iii) a two-sample REINFORCE leave-one-out (RLOO) estimator (Kool et al., 2019a). **(4) Order Scope.** We consider (i) full-sequence order learning with a single latent permutation over the sequence; and (ii) block-wise order learning with one latent permutation per block at varying block sizes, examining the trade-offs among optimization difficulty, order flexibility, and decoding parallelism.

All variants parameterize the posterior q_ϕ and prior p_η as Plackett–Luce models (Plackett, 1975) using position-wise scores (Appendix A.2). Unless otherwise specified, we use an independent transformer posterior, a SwiGLU prior head without backbone detachment, two-sample RLOO, and a block size of 32. Using a single-step ELBO estimator (Wang et al., 2025), we sample $t \sim \mathrm{Uniform}(0,1)$ once per sequence and set $i_b = n_b - \lceil t n_b \rceil + 1$ for each block b. Since $i_b \sim \mathrm{Uniform}\{1, \dots, n_b\}$, $n_b F^b_{i_b}$ is an unbiased estimator of the sum over i. To estimate the remaining expectation over $\boldsymbol{z}^b_{<i_b}$, for each block we draw $\boldsymbol{z}^{b,1}, \boldsymbol{z}^{b,2} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}^b \mid \boldsymbol{x}^{\le b})$ and define $F^{b,j} = F^b_{i_b}(\boldsymbol{z}^{b,j}_{<i_b}, \boldsymbol{x}^{\le b})$ for $j \in \{1,2\}$. Both samples reuse posterior scores from a single clean-sequence forward pass. Each $F^{b,j}$ serves as the other's leave-one-out baseline. The resulting learned-order surrogate loss is

$$\mathcal{L}_{\mathrm{LO}} = -\sum_{b=1}^{B} \frac{n_b}{2} \Big[\big(\log q_\phi(\boldsymbol{z}^{b,1}_{<i_b} \mid \boldsymbol{x}^{\le b}) - \log q_\phi(\boldsymbol{z}^{b,2}_{<i_b} \mid \boldsymbol{x}^{\le b}) \big)\, \mathrm{sg}(\Delta F^b) + F^{b,1} + F^{b,2} \Big], \tag{5}$$

where $\Delta F^b = F^{b,1} - F^{b,2}$ and $\mathrm{sg}(\cdot)$ denotes the stop-gradient operator.

3.3 Parallel-aware Training

Parallel decoding introduces an additional requirement for scaling order learning to MDM pretraining: accurate token predictions and a reliable ranking over multiple remaining positions from the same masked state. However, $\mathcal{L}_{\mathrm{LO}}$ increasingly focuses supervision on the preferred next position as the posterior concentrates. We therefore introduce two auxiliary objectives over the sampled posterior suffix, weighting positions by their posterior ranks to emphasize earlier reveals. For block b, let $m_b = n_b - i_b + 1$ denote the suffix length and define

$$w_r^{(m_b)} = \frac{\exp(-r/\tau)}{\sum_{s=0}^{m_b-1} \exp(-s/\tau)}, \qquad r = 0, \dots, m_b - 1, \tag{6}$$

where $r = 0$ corresponds to the next posterior position, and $\tau > 0$ is the rank-decay temperature.

Using these weights, we first introduce a parallel reconstruction loss over the remaining positions:

$$\mathcal{L}_{\mathrm{rec}} = -\frac{1}{2} \sum_{j=1}^{2} \sum_{b=1}^{B} m_b \sum_{r=0}^{m_b-1} w_r^{(m_b)} \log p_\theta \left(x^b_{z^{b,j}_{i_b+r}} \mid \widetilde{\boldsymbol{x}}^{b,j}_{i_b} \right). \tag{7}$$

We further introduce a suffix-order loss that trains the prior to rank the remaining masked positions according to the sampled posterior suffix. For each masked state $\widetilde{\boldsymbol{x}}_{i_b}^{b,j}$, we keep the prior scores fixed and progressively restrict the Plackett–Luce normalization to the remaining candidates along the suffix, excluding the final position whose conditional probability is one.

$$\mathcal{L}_{\text{ord}} = -\frac{1}{2}\sum_{j=1}^{2}\sum_{\substack{b=1\\ m_b>1}}^{B}(m_b-1)\sum_{r=0}^{m_b-2} w_r^{(m_b-1)} \log p_\eta\left(z_{i_b+r}^{b,j} \mid \boldsymbol{z}_{<i_b+r}^{b,j}, \widetilde{\boldsymbol{x}}_{i_b}^{b,j}\right). \tag{8}$$

The complete training loss is

$$\mathcal{L}_{\text{train}} = \mathcal{L}_{\text{LO}} + \lambda_{\text{rec}}\mathcal{L}_{\text{rec}} + \lambda_{\text{ord}}\mathcal{L}_{\text{ord}}. \tag{9}$$

Here, λ_{rec} and λ_{ord} weight the two auxiliary objectives, which reuse the forward passes for $\mathcal{L}_{\text{LO}}$ without additional transformer computation. Posterior-sampled orders are detached in both terms, so q_ϕ is optimized only through $\mathcal{L}_{\text{LO}}$. The auxiliary objectives are not part of the learned-order ELBO. We refer to the variants trained with $\mathcal{L}_{\text{LO}}$ alone and $\mathcal{L}_{\text{train}}$ as SOLDIFF$_{\text{LO}}$ and SOLDIFF, respectively. Algorithm 1 summarizes the training procedure. Ablations are provided in Appendix C.6.

3.4 INFERENCE

At inference, we discard the posterior q_ϕ and decode blocks from left to right using the token predictor p_θ and learned prior p_η. When decoding block b, all previous blocks are visible, while the current block starts fully masked and is progressively unmasked. At each step, a single token-predictor forward pass produces token distributions for all remaining masked positions, while p_η reuses the backbone hidden states to assign them order scores. These scores define a Plackett–Luce distribution, which can then be used to produce an ordering, e.g., via ranking or sampling. Serial decoding reveals only the first position in the resulting ordering, while parallel decoding reveals a prefix. The prefix length can be fixed or determined by a heuristic. For example, with a confidence threshold γ, we reveal positions in order until reaching the first position whose top-1 confidence falls below γ, while always revealing at least one position. Selected tokens are predicted from the same masked state and revealed simultaneously, while the others remain masked. This process repeats until the block is fully decoded. Thus, the prior p_η defines state-dependent ordering preferences, while the decoding rule determines the realized unmasking schedule and degree of parallelism.

4 RELATED WORK

Standard MDM training is equivalent to averaging over a uniform distribution of unmasking orders, while inference requires deciding which masked positions to reveal at each step. Recent work has therefore explored how unmasking orders should be handled at both training and inference time.

Inference-time Methods. One line of work improves MDM decoding by modifying only the inference-time unmasking policy. Heuristic methods prioritize positions using top probability, top-probability margin (Kim et al., 2025), or low entropy (Ben-Hamu et al., 2025). Other methods use planners or learned policies for position selection (Peng et al., 2026a; Asano et al., 2026; Hong et al., 2026a; Jazbec et al., 2025; Liu et al., 2026). These approaches can improve generation quality, but their gains remain constrained by the fixed MDM under a given training budget. SOLDIFF instead learns ordering preferences during pretraining to shape both token-prediction training and decoding.

Training-time Methods. Another line of work modifies MDM training through non-uniform masking or learned unmasking preferences. Some alter masking distributions through bell-shaped time sampling (Hong et al., 2026b), position-dependent masking (Fu et al., 2025), or token-dependent schedules (Shi et al., 2024). PUMA (Kim et al., 2026) constructs masks through teacher-forced progressive unmasking guided by the model's policy. PAPL (Peng et al., 2026b) and AOP (Mohamud et al., 2026) reweight denoising losses using planner- or policy-based preferences. LoMDM (Hong et al., 2026c) jointly learns context-dependent, position-wise diffusion schedules with the denoiser, while Fox et al. (2026) model per-position reveal decisions as latent Bernoulli variables optimized through variational inference. Among these approaches, heuristic masking relies on prescribed selection criteria, whereas loss reweighting alone leaves the mask-sampling distribution unchanged. Although some methods explicitly learn this distribution, their empirical evaluation has largely focused on smaller-scale or task-specific settings. SOLDIFF advances this line of work by establishing a framework for stable and effective scaling of learned-order pretraining to billion-parameter MDMs.

Table 1: Comparison of base models. Best results are **bold**; second-best are underlined. SOLDIFF-8B achieves the highest average score with only 100B continued-pretraining tokens ($< 0.3\%$ of its base model's pretraining tokens), establishing an efficient recipe for training state-of-the-art MDMs.

Type	AR-Base Model		MDM-Base Model			SOLDIFF-Base (Ours)	
Model	Llama-3.1-8B	Qwen3-8B	LLaDA-8B	Dream-7B	Nemotron-Labs-Diffusion-8B	SOLDIFF$_{\text{LO}}$-8B	SOLDIFF-8B
Knowledge & Reasoning							
ARC-C	81.02	**91.86**	82.71	90.17	90.51	89.15	90.85
ARC-E	89.77	**97.53**	92.59	95.06	94.89	94.89	97.18
MMLU	61.11	75.81	64.34	70.38	**79.29**	74.98	77.26
MMLU-Pro	33.66	55.47	39.32	45.54	**62.74**	55.05	57.23
Mathematics							
GSM8K	58.15	**91.43**	75.21	78.62	90.75	88.10	89.39
MATH	20.38	55.46	31.06	41.02	**62.12**	59.92	60.90
OlympiadBench	6.00	**20.30**	9.00	11.80	18.80	19.30	**20.30**
Code Generation							
HumanEval	43.90	85.37	34.15	59.15	78.66	**88.41**	87.20
HumanEval+	39.02	78.66	28.05	52.44	71.34	**79.88**	**79.88**
MBPP	46.40	**69.00**	39.40	54.40	67.00	68.00	68.40
MBPP+	52.12	**71.69**	44.44	59.52	70.37	69.58	70.63
LiveCodeBench v6	7.93	14.54	5.73	9.91	**22.03**	18.06	17.18
Avg.	44.96	67.26	45.50	55.67	67.38	67.11	**68.03**

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Implementation Details. We continue pretraining SOLDIFF from Qwen3-8B-Base (Yang et al., 2025) on a mixture of web, knowledge, code, mathematics, and reasoning data. SOLDIFF$_{\text{LO}}$-8B and SOLDIFF-8B are trained for 20B and 100B tokens, respectively, with $\tau = 8$ for SOLDIFF. The τ ablation, SFT formulation, and implementation details are in Appendices C.6, A.3, and B.1.

Evaluation. We evaluate 12 benchmarks across knowledge and reasoning (ARC-C, ARC-E (Clark et al., 2018), MMLU (Hendrycks et al., 2020), and MMLU-Pro (Wang et al., 2024)), mathematics (GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), and OlympiadBench (He et al., 2024)), and code generation (HumanEval (Chen et al., 2021), HumanEval+ (Liu et al., 2023), MBPP (Austin et al., 2021), MBPP+ (Liu et al., 2023), and LiveCodeBench v6 (Jain et al., 2025)). All models are evaluated with serial decoding. We report pass@1 for code generation and accuracy otherwise.

Baselines. We compare SOLDIFF against two groups of baselines: (1) ARMs, including Llama-3.1-8B (AI@Meta, 2024) and Qwen3-8B-Base (Yang et al., 2025); and (2) MDMs, including LLaDA-8B-Base (Nie et al., 2025b), Dream-v0-Base-7B (Ye et al., 2025), and Nemotron-Labs-Diffusion-8B-Base (Fu et al., 2026). For each baseline, we follow its official hyperparameter settings.

5.2 MAIN RESULTS

Table 1 shows that SOLDIFF-8B achieves the highest average performance across 12 diverse benchmarks among all evaluated ARMs and MDMs of comparable size, with only 100B continued-pretraining tokens (less than 0.3% of its base model's pretraining tokens). The transition from autoregressive to masked diffusion modeling can require extensive continued pretraining even to recover the base model's performance. Our continued-pretraining budget is substantially smaller than those of Dream and Nemotron-Labs-Diffusion, which use 580B and 1.3T tokens, respectively. Trained on only 20B tokens, SOLDIFF$_{\text{LO}}$-8B achieves a HumanEval pass@1 of 88.41%, exceeding Nemotron-Labs-Diffusion by 9.75 points. **These results establish SOLDIFF as an efficient training recipe for high-performing MDMs with substantially reduced continued-pretraining budgets.**

5.3 CONTROLLED COMPARISON

To isolate the source of SOLDIFF's gains, we compare MDM training methods using the same data and Qwen3-1.7B-Base initialization. We train both SOLDIFF variants for 20B tokens and adjust each baseline's budget to match their total training FLOPs. We refer to the baseline using the standard block-diffusion objective of BD3-LMs (Arriola et al., 2025) as *Standard MDM*. We also evaluate

Table 2: FLOPs-matched comparison of MDMs with shared training data, Qwen3-1.7B-Base initialization, and comparable inference-time parameter counts. Both SOLDIFF variants outperform all baselines on average with only 20B tokens, versus approximately 44B for the Standard MDM. Avg. (12) denotes the 12-benchmark average. Best results are **bold**; second-best are underlined.

Benchmark	Standard MDM	Bell-TS	PosMask	PUMA-1S	R2LMask	SOLDIFF$_{\mathrm{LO}}$-1.7B	SOLDIFF-1.7B
GSM8K	70.51	68.54	70.28	74.83	73.54	**75.66**	74.07
MATH	34.02	34.82	36.10	38.14	41.16	**43.78**	43.64
HumanEval	54.27	47.56	53.05	59.76	59.15	**62.20**	59.15
MBPP	38.00	39.60	39.20	45.20	45.00	**51.20**	51.00
Avg. (12)	47.44	47.25	47.86	50.25	50.92	**52.75**	52.37

Table 3: Design-space ablations for learned-order training. The reference configuration is trained on 20B tokens; each variant changes one design choice and uses $20\mathrm{B}/r$ tokens to match total training FLOPs. The posterior q_ϕ is used only during training; the SwiGLU prior p_η adds only $\approx 0.2\%$ to the parameter count of p_θ at inference. Abbreviations: Indep/Shared=independent/backbone-shared posterior; D/ND=detached/non-detached hidden states; Conf=confidence-based; RF-1/RF-MA/RLOO-2=single-sample REINFORCE / moving-average REINFORCE / two-sample RLOO.

Variant	Extra training params. q_ϕ/p_η (M)	Rel. FLOPs per token (r)	HumanEval	MBPP	GSM8K	MATH	Avg.
Reference: Indep-14L q_ϕ, RLOO-2 SwiGLU-ND p_η, Block 32	1016 / 4	1.00×	62.20	51.20	75.66	43.78	58.21
Posterior q_ϕ Variants							
Shared-ND q_ϕ	4 / 4	1.07×	68.90	52.80	75.13	44.04	60.22
Shared-D q_ϕ	4 / 4	0.96×	16.46	9.40	33.97	9.98	17.45
Indep-2L q_ϕ	412 / 4	0.94×	57.32	50.40	72.55	43.84	56.03
Indep-28L q_ϕ	1721 / 4	1.07×	68.29	52.60	75.21	44.60	60.18
Prior p_η Variants							
Linear-ND p_η	1016 / < 0.01	1.00×	62.20	52.00	74.15	42.76	57.78
SwiGLU-D p_η	1016 / 4	1.00×	27.44	18.80	41.02	14.90	25.54
Conf-ND p_η	1016 / 0	1.00×	51.22	39.00	67.93	34.24	48.10
Estimator Variants							
RF-1 estimator	1016 / 4	0.55×	40.24	30.80	51.25	20.54	35.71
RF-MA estimator	1016 / 4	0.55×	51.83	41.80	66.87	29.46	47.49
Order Scope Variants							
Block size 8	1016 / 4	1.00×	67.68	51.40	77.18	46.18	60.61
Block size 128	1016 / 4	1.00×	59.76	45.20	74.53	41.62	55.28
Full-sequence scope	1016 / 4	0.48×	7.93	14.60	6.52	6.68	8.93

alternative masking strategies: bell-shaped time sampling (Bell-TS) (Hong et al., 2026b), which emphasizes training in the middle-context region; position-dependent token masking (PosMask) (Fu et al., 2025), which assigns higher masking probabilities to later positions; PUMA-1S, a one-shot variant of PUMA (Kim et al., 2026) that constructs masks from token confidence; and R2LMask, which deterministically masks tokens from right to left. See Appendix B for detailed settings.

As shown in Table 2, both SOLDIFF variants outperform all baselines on average across the 12 benchmarks, using fewer training tokens than the Standard MDM at matched FLOPs. These controlled results show that the gains arise from learned-order training rather than differences in data or initialization. Full results are provided in Appendix C.1.

5.4 DESIGN CHOICES ABLATION

We examine how the design choices in §3.2 affect optimization stability and downstream performance under matched training FLOPs. All variants are trained with $\mathcal{L}_{\mathrm{LO}}$, evaluated with serial decoding, and initialize the token predictor p_θ from Qwen3-1.7B-Base. See Appendix B.1 for details.

Table 3 highlights four key design principles. **(1) Posterior capacity and parameterization are both important.** With detached backbone features, the shared posterior has limited trainable capacity and performs poorly. Allowing posterior gradients to update the backbone recovers performance but leads to less stable optimization (Appendix C.2). Independent posteriors provide a more stable

alternative, with average performance improving as depth increases. **(2) Prior learning benefits from backbone co-adaptation.** Both linear and SwiGLU priors outperform the confidence-based heuristic, while detaching the backbone hidden states passed to the prior substantially degrades performance, highlighting the importance of prior–backbone co-adaptation. **(3) Variance reduction is critical for latent-order optimization.** Two-sample RLOO substantially outperforms both single-sample REINFORCE and REINFORCE with a moving-average baseline, despite requiring an additional token-predictor pass. **(4) Order scope trades off optimization difficulty against flexibility and parallelism.** Full-sequence order learning is difficult to optimize. Smaller blocks improve performance under serial decoding but restrict order flexibility and parallel decoding (Appendix C.3).

These ablations identify four key choices for stable and effective training: an independent posterior, a learned prior without backbone detachment, two-sample RLOO, and an appropriate block size.

5.5 Training Compute Efficiency and Model Scaling

We first assess training compute efficiency by tracking average downstream performance on GSM8K, MATH, HumanEval, and MBPP against cumulative training FLOPs. All methods are initialized from Qwen3-1.7B-Base. As shown in Figure 3 (left), both SolDiff variants achieve substantially better performance with markedly less training compute than the standard block-diffusion baseline. Specifically, with only 3.16×10^{21} training FLOPs, SolDiff outperforms Standard MDM trained with $6.8\times$ as much compute (21.43×10^{21} FLOPs). These results show that SolDiff translates training compute into downstream performance more efficiently.

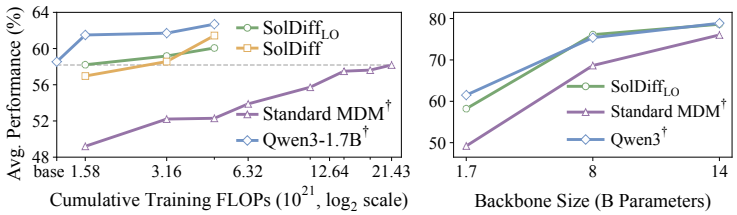

Figure 3: Downstream performance versus (left) cumulative training FLOPs and (right) backbone size under FLOPs-matched training. † denotes models continued-pretrained under the same setting as SolDiff.

We next examine whether this advantage persists with model scale. We compare SolDiff$_{LO}$, Qwen3 ARMs, and Standard MDM at 1.7B, 8B, and 14B. At each scale, SolDiff$_{LO}$ is trained for 20B tokens, with the baselines matched in total FLOPs. In Figure 3 (right), SolDiff$_{LO}$ consistently outperforms Standard MDM while remaining comparable to Qwen3. The gap between Standard MDM and the other models narrows with model size, suggesting that larger models better accommodate uniform-order training. Nevertheless, SolDiff$_{LO}$ retains a clear advantage even at 14B.

5.6 Parallel Decoding Analysis

We assess parallel-aware training by comparing SolDiff$_{LO}$-1.7B-Base and SolDiff-1.7B-Base. Figure 4(a) shows faster early improvement for SolDiff$_{LO}$ under serial decoding, consistent with $\mathcal{L}_{LO}$ concentrating supervision on the immediate next reveal. The parallel-aware objectives instead extend supervision to later suffix positions, resulting in slower early improvement under serial decoding. This initial gap closes as training progresses.

Figures 4(b) and (c) compare performance on GSM8K and MBPP across tokens-per-forward (TPF) values. We compare SolDiff$_{LO}$ and SolDiff after 60B training tokens against FLOPs-matched Standard MDM and Qwen3-1.7B baselines. As TPF increases, SolDiff$_{LO}$ degrades rapidly, whereas SolDiff achieves a better performance–efficiency trade-off than Standard MDM† on both benchmarks and also surpasses Qwen3-1.7B† on GSM8K in accuracy and decoding efficiency. Parallel-aware training thus enables SolDiff to retain strong performance under parallel decoding.

5.7 What Orders Does the Model Learn?

Our analyses show that the learned order is shaped by the data rather than fixed a priori.

Controlled Partial-order Tasks. To isolate order learning from pretrained initialization and uncontrolled natural-language dependencies, we introduce *Parallel Chains* and train SolDiff$_{LO}$ from scratch. Each answer consists of S chains of K tokens, with sequential dependencies within each

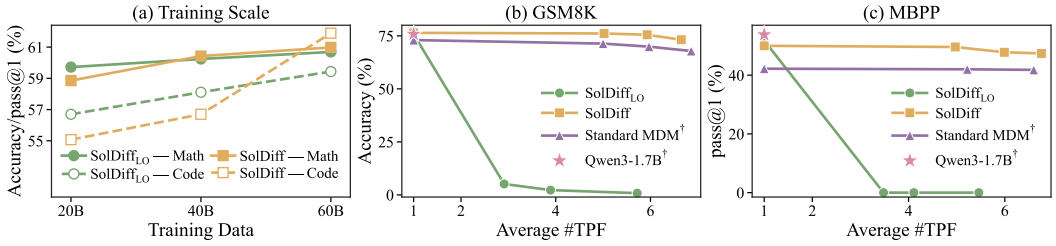

Figure 4: Effect of parallel-aware training on decoding performance and efficiency. (a) Performance under serial decoding across training scales (Math: GSM8K/MATH; Code: HumanEval/MBPP). (b,c) Performance–efficiency trade-off on GSM8K and MBPP versus average TPF.

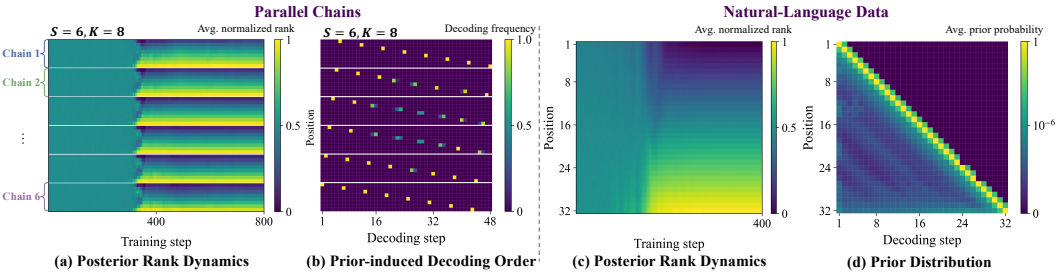

Figure 5: Learned ordering behavior on *Parallel Chains* (a,b) and natural-language data (c,d). Natural-language panels use SOLDIFF$_{LO}$-8B; (d) is evaluated on MBPP.

chain and no dependencies across chains. Any interleaving that preserves within-chain order therefore respects the task's dependency structure. Figure 5(a) shows that the posterior evolves from no clear ordering preference to increasing ranks within each chain, with positions at the same depth receiving similar ranks across chains. At inference, the prior reproduces this structure (Figure 5(b)), interleaving chains while preserving all within-chain precedence relations (0% violation rate). Together, these results show that SOLDIFF$_{LO}$ learns ordering preferences consistent with the task's underlying partial-order structure and reproduces this structure at inference. Appendix E provides task details and additional analyses, including mixed-pattern order learning.

Learned Orders in Natural Language. Figure 5(c) shows that the posterior rapidly develops a near left-to-right (LTR) preference on natural-language data, assigning lower ranks to earlier within-block positions. This behavior also emerges under MDM initialization, indicating that the preference is not merely inherited from ARM initialization (Appendix C.5). At inference, the learned prior distribution on MBPP exhibits a similar overall LTR structure with local variation (Figure 5(d)), consistent with observations in previous work (Gong et al., 2026). To further assess the benefit of learning a distribution over orders, we construct a fixed LTR variant by replacing the learned posterior with a deterministic LTR posterior while keeping all other training objectives unchanged. SOLDIFF consistently outperforms this variant across training budgets (Appendix C.4), indicating that the learned distribution captures useful ordering structure beyond its dominant LTR pattern.

6 CONCLUSION

We present SOLDIFF, a scalable variational framework for compute-efficient MDM pretraining through data-dependent order learning. The posterior focuses token-prediction training on mask patterns induced by preferred orders, while the prior reproduces these preferences at inference. A systematic design study identifies key choices that enable the stable and effective optimization required to scale learned-order training, while rank-weighted auxiliary objectives support parallel decoding. With only 100B tokens of continued pretraining, SOLDIFF-8B achieves the highest average performance among evaluated ARMs and MDMs of comparable size. In controlled comparisons, SOLDIFF outperforms standard block diffusion trained with $6.8\times$ as much compute. Together, these results show that our method improves MDM training compute efficiency, enabling ARM-competitive performance with substantially fewer continued-pretraining tokens than existing ARM-to-MDM adaptations, while supporting parallel decoding. Future work could study compute-optimal trade-offs among token-predictor scale, posterior capacity, and training data.

AI USE STATEMENT

In this work, we used generative AI tools to assist with writing and language polishing and with implementing parts of the proposed methods in code. All AI-assisted content was reviewed by the authors, and all AI-assisted code was manually reviewed and tested for correctness. We did not use generative AI for research ideation. We take responsibility for the final content of this work, including all AI-assisted text and code.

ETHICS STATEMENT

This work studies training methods for masked diffusion language models and does not involve human-subject research or the collection of new personal or sensitive data. Models trained with our method may inherit biases and privacy risks from their training data and base models and, like other general-purpose language models, may be subject to misuse. We encourage responsible use consistent with the licenses and intended uses of the underlying datasets and models.

REPRODUCIBILITY STATEMENT

The methodology is presented in §3, with detailed derivations and model parameterizations in Appendix A. The experimental setup is described in §5.1, with the corresponding implementation details for training, inference, and FLOP accounting provided in Appendix B. Appendix E details the construction and evaluation protocol for the controlled partial-order tasks. Anonymized source code is provided in the supplementary material.

REFERENCES

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.

Hikaru Asano, Tadashi Kozuno, Kuniaki Saito, and Yukino Baba. Where-to-unmask: Ground-truth-guided unmasking order learning for masked diffusion language models. *arXiv preprint arXiv:2602.09501*, 2026.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking, 2025. URL https://arxiv.org/abs/2505.24857.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, et al. Llada2. 1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Shuang Cheng, Yihan Bian, Dawei Liu, Yuhua Jiang, Yihao Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 22058–22075, 2026.

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

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

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels, 2024. URL https://arxiv.org/abs/2412.05496.

David Fox, Sam Bowyer, Song Liu, Laurence Aitchison, Raul Santos-Rodriguez, and Mengyue Yang. Learning generation orders for masked discrete diffusion models via variational inference. *arXiv preprint arXiv:2602.23968*, 2026.

Yonggan Fu, Lexington Whalen, Zhifan Ye, Xin Dong, Shizhe Diao, Jingyu Liu, Chengyue Wu, Hao Zhang, Enze Xie, Song Han, et al. Efficient-dlm: From autoregressive to diffusion language models, and beyond in speed. *arXiv preprint arXiv:2512.14067*, 2025.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, Chenhan Yu, Ye Yu, Weijia Chen, Sajad Norouzi, Jingyu Liu, Shiyi Lan, Ligeng Zhu, Jin Wang, Jindong Jiang, Morteza Mardani, Mehran Maghoumi, Song Han, Ante Jukic, Nima Tajbakhsh, Jan Kautz, and Pavlo Molchanov. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. Technical report, NVIDIA, 2026. Technical report.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. In *International Conference on Learning Representations*, volume 2026, pp. 15723–15750, 2026.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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

Chunsan Hong, Seonho An, Min-Soo Kim, and Jong Chul Ye. Improving discrete diffusion unmasking policies beyond explicit reference policies. In *International Conference on Learning Representations*, volume 2026, pp. 108226–108260, 2026a.

Chunsan Hong, Sanghyun Lee, Chieh-Hsin Lai, Satoshi Hayakawa, Yuhta Takida, Yuki Mitsufuji, Seungryong Kim, and Jong Chul Ye. Understanding and accelerating the training of masked diffusion language models. *arXiv preprint arXiv:2605.13026*, 2026b.

Chunsan Hong, Sanghyun Lee, and Jong Chul Ye. Unifying masked diffusion models with various generation orders and beyond. *arXiv preprint arXiv:2602.02112*, 2026c.

Emiel Hoogeboom, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. *arXiv preprint arXiv:2110.02037*, 2021.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in neural information processing systems*, 36: 62991–63010, 2023.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Metod Jazbec, Theo X Olausson, Louis Béthune, Pierre Ablin, Michael Kirchhof, João Monteiro, Victor Turrisi, Jason Ramapuram, and Marco Cuturi. Learning unmasking policies for diffusion language models. *arXiv preprint arXiv:2512.09106*, 2025.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv:2502.06768*, 2025.

Jaeyeon Kim, Jonathan Geuter, David Alvarez-Melis, Sham Kakade, and Sitan Chen. Stop training for the worst: Progressive unmasking accelerates masked diffusion training. *arXiv preprint arXiv:2602.10314*, 2026.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! In *DeepRLStructPred@ICLR*, 2019a. URL https://api.semanticscholar.org/CorpusID:198489118.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International conference on machine learning*, pp. 3499–3508. PMLR, 2019b.

Jia-Nan Li, Jian Guan, Wei Wu, and Chongxuan Li. Refusion: A diffusion large language model with parallel autoregressive decoding. In *International Conference on Learning Representations*, volume 2026, pp. 53846–53869, 2026.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2961–2984, 2024.

Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, et al. Torchtitan: One-stop pytorch native solution for production ready llm pretraining. In *International Conference on Learning Representations*, volume 2025, pp. 92161–92181, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Xuejie Liu, Yap Vit Chun, Yitao Liang, and Anji Liu. Lookahead path likelihood optimization for diffusion llms. *arXiv preprint arXiv:2602.03496*, 2026.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

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

Kaijing Ma, Xeron Du, Yunran Wang, Haoran Zhang, Zhoufutu Wen, Xingwei Qu, Jian Yang, Jiaheng Liu, Minghao Liu, Xiang Yue, et al. Kor-bench: Benchmarking language models on knowledge-orthogonal reasoning tasks. In *International Conference on Learning Representations*, volume 2025, pp. 80062–80161, 2025.

Jama Hussein Mohamud, Mohsin Hasan, Mirco Ravanelli, and Yoshua Bengio. Adaptive order policies for masked diffusion. *arXiv preprint arXiv:2606.00295*, 2026.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. In *International Conference on Learning Representations*, volume 2025, pp. 82974–82997, 2025a.

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, 2025b. URL https://arxiv.org/abs/2502.09992.

Fred Zhangzhi Peng, Zachary Bezemek, Sawan Patel, Jarrid Rector-Brooks, Sherwood Yao, Avishek Joey Bose, Alexander Tong, and Pranam Chatterjee. Path planning for masked diffusion model sampling, 2026a. URL https://arxiv.org/abs/2502.03540.

Zhangzhi Peng, Zachary Bezemek, Jarrid Rector-Brooks, Shuibai Zhang, Michael Bronstein, Anru Zhang, Joey Bose, and Alexander Tong. Planner aware path learning in diffusion language models training. In *International Conference on Learning Representations*, volume 2026, pp. 126799–126844, 2026b.

Robin L Plackett. The analysis of permutations. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 24(2):193–202, 1975.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

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.

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

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024.

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning*, pp. 467–475. PMLR, 2014.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37: 95266–95290, 2024.

Zhe Wang, Jiaxin Shi, Nicolas Heess, Arthur Gretton, and Michalis K Titsias. Learning-order autoregressive models with application to molecular graph generation. *arXiv preprint arXiv:2503.05979*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

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

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Fengqi Zhu, Shaoxuan Xu, Jingyang Ou, Zebin You, Yipeng Xing, Huabin Liu, Xiaolu Zhang, Jun Zhou, Zhenzhong Lan, Yankai Lin, et al. Llada moe v2: Scaling mixture-of-experts diffusion language models. *arXiv preprint arXiv:2608.03457*, 2026.

A METHODOLOGICAL DETAILS

A.1 DETAILED DERIVATION OF THE LEARNED-ORDER OBJECTIVE

We provide a complete derivation of the learned-order training objective of SOLDIFF.

Block Construction and Latent Unmasking Order. Let $\boldsymbol{x} = (x_1, \ldots, x_L)$ be a sequence of length L. We partition it into B consecutive blocks,

$$\boldsymbol{x} = (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^B), \quad \boldsymbol{x}^b = (x_1^b, \ldots, x_{n_b}^b), \quad \sum_{b=1}^{B} n_b = L. \tag{10}$$

The blocks are generated in the fixed left-to-right order $\boldsymbol{x}^1 \to \cdots \to \boldsymbol{x}^B$, while the generation order within each block is unconstrained. When $B = 1$, the formulation reduces to a full-sequence bidirectional MDM; when $B > 1$, it corresponds to a block-diffusion model with bidirectional intra-block attention and causal inter-block attention.

For each block b, let

$$\boldsymbol{z}^b = (z_1^b, \ldots, z_{n_b}^b) \tag{11}$$

denote a permutation of $\{1, \ldots, n_b\}$ that specifies the within-block unmasking order. At step i, we define the unmasked prefix and the set of remaining masked positions as

$$\boldsymbol{z}_{<i}^b = (z_1^b, \ldots, z_{i-1}^b), \qquad R_i^b = \{1, \ldots, n_b\} \setminus \{z_1^b, \ldots, z_{i-1}^b\}. \tag{12}$$

The corresponding partially observed state is

$$\widetilde{\boldsymbol{x}}_i^b := \widetilde{\boldsymbol{x}}(\boldsymbol{x}, b, \boldsymbol{z}_{<i}^b). \tag{13}$$

In $\widetilde{\boldsymbol{x}}_i^b$, all preceding blocks $\boldsymbol{x}^{<b} = (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^{b-1})$ and the tokens $\boldsymbol{x}^b_{\boldsymbol{z}^b_{<i}} = (x^b_{z_1^b}, \ldots, x^b_{z_{i-1}^b})$ are visible, while the positions in R_i^b are masked. Subsequent blocks $\boldsymbol{x}^{>b} = (\boldsymbol{x}^{b+1}, \ldots, \boldsymbol{x}^B)$ are not attended to under block-wise causal attention.

The data-dependent order policy for block b is

$$p_\eta^x(\boldsymbol{z}^b) = \prod_{i=1}^{n_b} p_\eta(z_i^b \mid \boldsymbol{z}_{<i}^b, \widetilde{\boldsymbol{x}}_i^b), \tag{14}$$

where each conditional factor is a categorical distribution supported on the remaining position set R_i^b. Given the selected position z_i^b, the token predictor models $p_\theta(x^b_{z_i^b} \mid \widetilde{\boldsymbol{x}}_i^b)$. The sequence likelihood therefore factorizes as

$$p_{\theta,\eta}(\boldsymbol{x}) = \prod_{b=1}^{B} p_{\theta,\eta}(\boldsymbol{x}^b \mid \boldsymbol{x}^{<b}), \tag{15}$$

$$p_{\theta,\eta}(\boldsymbol{x}^b \mid \boldsymbol{x}^{<b}) = \sum_{\boldsymbol{z}^b} p_{\theta,\eta}(\boldsymbol{z}^b, \boldsymbol{x}^b \mid \boldsymbol{x}^{<b}) = \sum_{\boldsymbol{z}^b} \prod_{i=1}^{n_b} p_\eta(z_i^b \mid \boldsymbol{z}_{<i}^b, \widetilde{\boldsymbol{x}}_i^b)\, p_\theta(x^b_{z_i^b} \mid \widetilde{\boldsymbol{x}}_i^b), \tag{16}$$

where the sum ranges over all permutations of $\{1, \ldots, n_b\}$.

Variational Posterior and Evidence Lower Bound. Since the within-block permutations are latent, exact evaluation of the marginal likelihood requires summing over all $n_b!$ possible permutations for each block, which is computationally intractable. We therefore introduce an amortized variational posterior over the latent orders. Writing $\boldsymbol{z} = (\boldsymbol{z}^1, \ldots, \boldsymbol{z}^B)$, we define

$$q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = \prod_{b=1}^{B} q_\phi(\boldsymbol{z}^b \mid \boldsymbol{x}^{\le b}), \quad q_\phi(\boldsymbol{z}^b \mid \boldsymbol{x}^{\le b}) = \prod_{i=1}^{n_b} q_\phi(z_i^b \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b}), \tag{17}$$

where $\boldsymbol{x}^{\le b} = (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^b)$. Unlike the prior, which is conditioned on the masked state, the posterior has access to the complete contents of block b and all preceding blocks.

Applying Jensen's inequality to the marginal likelihood of each block yields

$$\begin{aligned}
\log p_{\theta,\eta}(\boldsymbol{x}) &= \sum_{b=1}^{B} \log p_{\theta,\eta}(\boldsymbol{x}^b \mid \boldsymbol{x}^{<b}) \\
&= \sum_{b=1}^{B} \log \mathbb{E}_{q_\phi(\boldsymbol{z}^b \mid \boldsymbol{x}^{\le b})} \left[\frac{p_{\theta,\eta}(\boldsymbol{z}^b, \boldsymbol{x}^b \mid \boldsymbol{x}^{<b})}{q_\phi(\boldsymbol{z}^b \mid \boldsymbol{x}^{\le b})} \right] \\
&\ge \sum_{b=1}^{B} \mathbb{E}_{q_\phi(\boldsymbol{z}^b \mid \boldsymbol{x}^{\le b})} \left[\sum_{i=1}^{n_b} \log \frac{p_\eta\left(z_i^b \mid \boldsymbol{z}_{<i}^b, \widetilde{\boldsymbol{x}}_i^b\right) p_\theta\left(x_{z_i^b}^b \mid \widetilde{\boldsymbol{x}}_i^b\right)}{q_\phi\left(z_i^b \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b}\right)} \right] \\
&= \sum_{b=1}^{B} \sum_{i=1}^{n_b} \mathbb{E}_{q_\phi(\boldsymbol{z}_{<i}^b \mid \boldsymbol{x}^{\le b})} \left[\mathbb{E}_{q_\phi(z_i^b \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b})} \left[\log \frac{p_\eta\left(z_i^b \mid \boldsymbol{z}_{<i}^b, \widetilde{\boldsymbol{x}}_i^b\right) p_\theta\left(x_{z_i^b}^b \mid \widetilde{\boldsymbol{x}}_i^b\right)}{q_\phi\left(z_i^b \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b}\right)} \right] \right] \\
&= \sum_{b=1}^{B} \sum_{i=1}^{n_b} \mathbb{E}_{q_\phi(\boldsymbol{z}_{<i}^b \mid \boldsymbol{x}^{\le b})} \left[F_i^b(\boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b}) \right] =: \mathcal{E}_{\mathrm{ELBO}}(\boldsymbol{x}).
\end{aligned} \tag{18}$$

Since the i-th term depends on the permutation only through $\boldsymbol{z}_{<i}^b$ and z_i^b, the suffix $\boldsymbol{z}_{>i}^b$ can be marginalized out. Given $\boldsymbol{z}_{<i}^b$, we compute the expectation over z_i^b exactly by summing over all remaining positions in R_i^b, while the outer expectation over $\boldsymbol{z}_{<i}^b$ is estimated using samples from $q_\phi(\boldsymbol{z}_{<i}^b \mid \boldsymbol{x}^{\le b})$.

For a fixed prefix $\boldsymbol{z}_{<i}^b$, the exact one-step contribution is

$$\begin{aligned}
F_i^b(\boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b}) &:= \mathbb{E}_{q_\phi(z_i^b \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b})} \left[\log \frac{p_\eta(z_i^b \mid \boldsymbol{z}_{<i}^b, \widetilde{\boldsymbol{x}}_i^b)\, p_\theta(x_{z_i^b}^b \mid \widetilde{\boldsymbol{x}}_i^b)}{q_\phi(z_i^b \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b})} \right] \\
&= \sum_{u \in R_i^b} q_\phi(z_i^b = u \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b}) \log \frac{p_\eta(z_i^b = u \mid \boldsymbol{z}_{<i}^b, \widetilde{\boldsymbol{x}}_i^b)\, p_\theta(x_u^b \mid \widetilde{\boldsymbol{x}}_i^b)}{q_\phi(z_i^b = u \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b})}.
\end{aligned} \tag{19}$$

Equivalently, this term decomposes as

$$F_i^b = \underbrace{\mathbb{E}_{q_\phi(z_i^b \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b})} \left[\log p_\theta\left(x_{z_i^b}^b \mid \widetilde{\boldsymbol{x}}_i^b\right) \right]}_{\text{posterior-weighted token reconstruction}} - \underbrace{D_{\mathrm{KL}}\left(q_\phi\left(z_i^b \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b}\right) \,\middle\|\, p_\eta\left(z_i^b \mid \boldsymbol{z}_{<i}^b, \widetilde{\boldsymbol{x}}_i^b\right) \right)}_{\text{posterior–prior order alignment}}. \tag{20}$$

The first term trains the token predictor through posterior-weighted reconstruction over candidate next positions. The second term minimizes the KL divergence between the full-context posterior and the masked-state-conditioned prior, enabling the prior to recover the learned ordering preferences at inference. Together, the two terms encourage the posterior to favor positions with high token likelihood while remaining predictable by the prior.

Unbiased Single-step Estimator. Exact evaluation of the ELBO requires summing the one-step contributions over all $i \in \{1, \dots, n_b\}$ for every block b. To avoid enumerating all steps, we sample $t \sim \mathrm{Uniform}(0, 1)$ once for each sequence and set

$$i_b = n_b - \lceil t n_b \rceil + 1, \qquad b \in \{1, \dots, B\}. \tag{21}$$

Since $\lceil t n_b \rceil$ is uniformly distributed over $\{1, \dots, n_b\}$, i_b is also uniform over this set. For each block, we draw a posterior permutation $\boldsymbol{z}^b \sim q_\phi(\boldsymbol{z}^b \mid \boldsymbol{x}^{\le b})$ and retain the corresponding prefix $\boldsymbol{z}_{<i_b}^b$. This gives the single-step estimator

$$\widehat{\mathcal{E}}_{\mathrm{ELBO}}(\boldsymbol{x}) = \sum_{b=1}^{B} n_b F_{i_b}^b \left(\boldsymbol{z}_{<i_b}^b, \boldsymbol{x}^{\le b} \right). \tag{22}$$

Averaging over the uniformly sampled step i_b recovers the sum over all steps, and linearity of expectation then gives

$$\begin{aligned}
\mathbb{E}_{\substack{t \sim \mathrm{Uniform}(0,1) \\ \boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}} \left[\widehat{\mathcal{E}}_{\mathrm{ELBO}}(\boldsymbol{x}) \right] &= \sum_{b=1}^{B} n_b \sum_{i=1}^{n_b} \frac{1}{n_b} \mathbb{E}_{q_\phi(\boldsymbol{z}_{<i}^b \mid \boldsymbol{x}^{\le b})} \left[F_i^b(\boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b}) \right] \\
&= \mathcal{E}_{\mathrm{ELBO}}(\boldsymbol{x}).
\end{aligned} \tag{23}$$

Therefore, $\widehat{\mathcal{E}}_{\mathrm{ELBO}}(\boldsymbol{x})$ is an unbiased estimator of the ELBO.

Single-sample ELBO Gradient Estimator. Because the conditional expectation over z_i^b in Eq. 18 is computed exactly, the one-step contribution $F_i^b(\boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\leq b})$ can be differentiated directly through the finite sum over R_i^b. Thus, for a fixed prefix $\boldsymbol{z}_{<i}^b$, no score-function estimator is needed for z_i^b, and the gradients of F_i^b with respect to ϕ, θ, and η are computed exactly.

The only remaining expectation in Eq. 18 that requires stochastic estimation is over the prefix $\boldsymbol{z}_{<i}^b$. For any function $G_\phi(\boldsymbol{z})$, the score-function identity is

$$\nabla_\phi \mathbb{E}_{q_\phi(\boldsymbol{z})}\left[G_\phi(\boldsymbol{z})\right] = \mathbb{E}_{q_\phi(\boldsymbol{z})}\left[G_\phi(\boldsymbol{z})\nabla_\phi \log q_\phi(\boldsymbol{z}) + \nabla_\phi G_\phi(\boldsymbol{z})\right]. \tag{24}$$

Applying this identity to each term in Eq. 18 yields

$$\nabla_\phi \mathcal{E}_{\mathrm{ELBO}}(\boldsymbol{x}) = \sum_{b=1}^{B}\sum_{i=1}^{n_b} \mathbb{E}_{q_\phi(\boldsymbol{z}_{<i}^b \mid \boldsymbol{x}^{\leq b})}\left[\underbrace{F_i^b(\boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\leq b})\nabla_\phi \log q_\phi(\boldsymbol{z}_{<i}^b \mid \boldsymbol{x}^{\leq b})}_{\text{score-function gradient}} + \underbrace{\nabla_\phi F_i^b(\boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\leq b})}_{\text{direct gradient}}\right]. \tag{25}$$

The first term is the score-function contribution from sampling the prefix, while the second captures the direct dependence of F_i^b on ϕ.

Assuming disjoint parameter sets for q_ϕ, p_θ, and p_η, the prefix distribution is independent of θ and η. Their gradients therefore contain only the direct derivatives of F_i^b:

$$\nabla_\theta \mathcal{E}_{\mathrm{ELBO}}(\boldsymbol{x}) = \sum_{b=1}^{B}\sum_{i=1}^{n_b} \mathbb{E}_{q_\phi(\boldsymbol{z}_{<i}^b \mid \boldsymbol{x}^{\leq b})}\left[\nabla_\theta F_i^b(\boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\leq b})\right], \tag{26}$$

$$\nabla_\eta \mathcal{E}_{\mathrm{ELBO}}(\boldsymbol{x}) = \sum_{b=1}^{B}\sum_{i=1}^{n_b} \mathbb{E}_{q_\phi(\boldsymbol{z}_{<i}^b \mid \boldsymbol{x}^{\leq b})}\left[\nabla_\eta F_i^b(\boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\leq b})\right]. \tag{27}$$

We then replace the sum over i with the uniform step subsampling introduced above. Because the sampling distribution of i_b is independent of the model parameters, this subsampling introduces no additional score-function term. The resulting single-sample gradient estimators are

$$\widehat{\boldsymbol{g}}_\phi^{\mathrm{1S}}(\boldsymbol{x}) = \sum_{b=1}^{B} n_b\Big[F_{i_b}^b(\boldsymbol{z}_{<i_b}^b, \boldsymbol{x}^{\leq b})\nabla_\phi \log q_\phi(\boldsymbol{z}_{<i_b}^b \mid \boldsymbol{x}^{\leq b}) + \nabla_\phi F_{i_b}^b(\boldsymbol{z}_{<i_b}^b, \boldsymbol{x}^{\leq b})\Big], \tag{28}$$

$$\widehat{\boldsymbol{g}}_\theta^{\mathrm{1S}}(\boldsymbol{x}) = \sum_{b=1}^{B} n_b \nabla_\theta F_{i_b}^b(\boldsymbol{z}_{<i_b}^b, \boldsymbol{x}^{\leq b}), \tag{29}$$

$$\widehat{\boldsymbol{g}}_\eta^{\mathrm{1S}}(\boldsymbol{x}) = \sum_{b=1}^{B} n_b \nabla_\eta F_{i_b}^b(\boldsymbol{z}_{<i_b}^b, \boldsymbol{x}^{\leq b}). \tag{30}$$

Taking the expectation over the sampled t and posterior permutations recovers the corresponding ELBO gradients:

$$\begin{aligned}
\mathbb{E}\left[\widehat{\boldsymbol{g}}_\phi^{\mathrm{1S}}(\boldsymbol{x})\right] &= \nabla_\phi \mathcal{E}_{\mathrm{ELBO}}(\boldsymbol{x}),\\
\mathbb{E}\left[\widehat{\boldsymbol{g}}_\theta^{\mathrm{1S}}(\boldsymbol{x})\right] &= \nabla_\theta \mathcal{E}_{\mathrm{ELBO}}(\boldsymbol{x}),\\
\mathbb{E}\left[\widehat{\boldsymbol{g}}_\eta^{\mathrm{1S}}(\boldsymbol{x})\right] &= \nabla_\eta \mathcal{E}_{\mathrm{ELBO}}(\boldsymbol{x}).
\end{aligned} \tag{31}$$

Two-sample RLOO Gradient Estimator and Surrogate Loss. The single-sample gradient estimator in Eq. 28 consists of two terms: a score-function term, which typically has high variance, and a direct-gradient term. To reduce the variance of the score-function term, we use a two-sample REINFORCE leave-one-out (RLOO) estimator (Kool et al., 2019a), where each sample's one-step ELBO contribution serves as a baseline for the other.

For each block b, we draw two independent posterior permutations:

$$\boldsymbol{z}^{b,1}, \boldsymbol{z}^{b,2} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}^b \mid \boldsymbol{x}^{\leq b}). \tag{32}$$

For the sampled step i_b and $j \in \{1, 2\}$, define

$$F^{b,j} := F^b_{i_b}(\boldsymbol{z}^{b,j}_{<i_b}, \boldsymbol{x}^{\leq b}), \qquad \Delta F^b := F^{b,1} - F^{b,2}. \tag{33}$$

For the first sample, $F^{b,2}$ is independent of $\boldsymbol{z}^{b,1}_{<i_b}$ and can therefore serve as a baseline. Using the zero-mean property of the score function,

$$\mathbb{E}_{q_\phi(\boldsymbol{z}^b_{<i_b}|\boldsymbol{x}^{\leq b})}\left[\nabla_\phi \log q_\phi(\boldsymbol{z}^b_{<i_b} \mid \boldsymbol{x}^{\leq b})\right] = 0, \tag{34}$$

we obtain

$$\mathbb{E}\left[\nabla_\phi \log q_\phi(\boldsymbol{z}^{b,1}_{<i_b} \mid \boldsymbol{x}^{\leq b})\left(F^{b,1} - F^{b,2}\right)\right] = \mathbb{E}\left[F^{b,1}\nabla_\phi \log q_\phi(\boldsymbol{z}^{b,1}_{<i_b} \mid \boldsymbol{x}^{\leq b})\right]. \tag{35}$$

The same argument applies to the second sample with $F^{b,1}$ as its baseline. Averaging the two estimates gives

$$\widehat{\boldsymbol{g}}^b_{\phi,\text{score}} = \frac{1}{2}\Big(\nabla_\phi \log q_\phi(\boldsymbol{z}^{b,1}_{<i_b} \mid \boldsymbol{x}^{\leq b}) - \nabla_\phi \log q_\phi(\boldsymbol{z}^{b,2}_{<i_b} \mid \boldsymbol{x}^{\leq b})\Big)\Delta F^b, \tag{36}$$

whose expectation equals the score-function component of the block-wise posterior gradient. The direct-gradient component is estimated by

$$\widehat{\boldsymbol{g}}^b_{\phi,\text{direct}} = \frac{1}{2}\left(\nabla_\phi F^{b,1} + \nabla_\phi F^{b,2}\right). \tag{37}$$

The gradient estimators for θ and η are obtained analogously by averaging the corresponding direct gradients over the two posterior paths. Because the prefix distribution q_ϕ is independent of θ and η, neither estimator contains a score-function term. Combining both components with the uniform-step estimator yields

$$\widehat{\boldsymbol{g}}^{\text{RLOO}}_\phi(\boldsymbol{x}) = \sum_{b=1}^{B} \frac{n_b}{2}\Big[\big(\nabla_\phi \log q_\phi(\boldsymbol{z}^{b,1}_{<i_b} \mid \boldsymbol{x}^{\leq b}) - \nabla_\phi \log q_\phi(\boldsymbol{z}^{b,2}_{<i_b} \mid \boldsymbol{x}^{\leq b})\big)\Delta F^b + \nabla_\phi F^{b,1} + \nabla_\phi F^{b,2}\Big]. \tag{38}$$

The resulting estimator is unbiased:

$$\mathbb{E}\left[\widehat{\boldsymbol{g}}^{\text{RLOO}}_\phi(\boldsymbol{x})\right] = \nabla_\phi \mathcal{E}_{\text{ELBO}}(\boldsymbol{x}). \tag{39}$$

To implement the RLOO gradient estimator in Eq. 38 with automatic differentiation, we define a surrogate loss. Backpropagating through $-\sum_b \frac{n_b}{2}(F^{b,1} + F^{b,2})$ recovers the direct-gradient component. However, because the sampled prefixes are discrete and treated as constants during backpropagation, it does not capture the dependence of their sampling distribution on ϕ. We therefore add a REINFORCE surrogate term using $\text{sg}(\Delta F^b)$ as the RLOO learning signal.

Let $\text{sg}(\cdot)$ denote the stop-gradient operator. The corresponding RLOO surrogate term satisfies

$$\begin{aligned}&\nabla_\phi\Big[\big(\log q_\phi(\boldsymbol{z}^{b,1}_{<i_b} \mid \boldsymbol{x}^{\leq b}) - \log q_\phi(\boldsymbol{z}^{b,2}_{<i_b} \mid \boldsymbol{x}^{\leq b})\big)\,\text{sg}(\Delta F^b)\Big] \\ &\quad = \big(\nabla_\phi \log q_\phi(\boldsymbol{z}^{b,1}_{<i_b} \mid \boldsymbol{x}^{\leq b}) - \nabla_\phi \log q_\phi(\boldsymbol{z}^{b,2}_{<i_b} \mid \boldsymbol{x}^{\leq b})\big)\Delta F^b.\end{aligned} \tag{40}$$

We therefore define the learned-order surrogate loss

$$\mathcal{L}_{\text{LO}} = -\sum_{b=1}^{B} \frac{n_b}{2}\left[\Big(\log q_\phi(\boldsymbol{z}^{b,1}_{<i_b} \mid \boldsymbol{x}^{\leq b}) - \log q_\phi(\boldsymbol{z}^{b,2}_{<i_b} \mid \boldsymbol{x}^{\leq b})\Big)\,\text{sg}(\Delta F^b) + F^{b,1} + F^{b,2}\right]. \tag{41}$$

The surrogate term recovers the RLOO score-function component of the posterior gradient, while $F^{b,1} + F^{b,2}$ provides the direct-gradient component. The stop-gradient on ΔF^b prevents unintended differentiation through the RLOO learning signal. Therefore,

$$\mathbb{E}\left[\nabla_\phi \mathcal{L}_{\text{LO}}\right] = -\nabla_\phi \mathcal{E}_{\text{ELBO}}(\boldsymbol{x}). \tag{42}$$

Under the disjoint parameterization assumed above, the score-function surrogate contributes only to the gradient with respect to ϕ, whereas $F^{b,1} + F^{b,2}$ provides the direct gradients for all parameters. Consequently,

$$\mathbb{E}\left[\nabla \mathcal{L}_{\text{LO}}\right] = -\nabla \mathcal{E}_{\text{ELBO}}(\boldsymbol{x}). \tag{43}$$

A.2 Model Parameterization Details

We specify the neural parameterizations of the token predictor p_θ, variational posterior q_ϕ, and prior p_η. Both order distributions, q_ϕ and p_η, are parameterized as Plackett–Luce models (Plackett, 1975). Given position-wise scalar scores $\boldsymbol{s} = (s_1, \ldots, s_{n_b})$ and a remaining candidate set R_i^b, we define the corresponding conditional distribution as

$$\mathrm{PL}\left(u \mid R_i^b, \boldsymbol{s}\right) = \frac{e^{s_u}}{\sum_{u' \in R_i^b} e^{s_{u'}}}, \quad u \in R_i^b. \tag{44}$$

For a fixed conditioning state, each order model produces scalar scores for all positions in a single forward pass. The step-i conditional distribution is then obtained by restricting the normalization to the current remaining set R_i^b.

Token Predictor p_θ. For our natural-language continued-pretraining experiments, we initialize the token predictor p_θ from an existing transformer-based language model (Yang et al., 2025), retaining the architecture and weights of the selected base checkpoint. The resulting architecture therefore depends on the model family and scale used for initialization. In these experiments, we consider base models of multiple sizes, including Qwen3-1.7B-Base, Qwen3-8B-Base, and Qwen3-14B-Base. We adapt the attention mask of the inherited transformer to the corresponding MDM formulation. Under block-wise modeling, attention is bidirectional within each block and causal across blocks. In the full-sequence setting, attention is bidirectional over the entire sequence. Given a partially observed masked state $\widetilde{\boldsymbol{x}}_i^b$, the backbone f_θ produces a hidden state at every sequence position:

$$\boldsymbol{h}_\theta = f_\theta\left(\widetilde{\boldsymbol{x}}_i^b\right), \quad \boldsymbol{h}_{\theta,u} \in \mathbb{R}^{d_\theta}, \tag{45}$$

where d_θ denotes the hidden dimension of the selected base model. The inherited vocabulary prediction head then defines

$$p_\theta\left(y \mid \widetilde{\boldsymbol{x}}_i^b, u\right) = \mathrm{softmax}\left(g_{\mathrm{tok}}\left(\boldsymbol{h}_{\theta,u}\right)\right)_y, \tag{46}$$

where $g_{\mathrm{tok}} : \mathbb{R}^{d_\theta} \to \mathbb{R}^{|\mathcal{V}|}$ denotes the inherited vocabulary prediction head and $|\mathcal{V}|$ is the vocabulary size. A single forward pass on the masked state produces token distributions at all positions, allowing the reconstruction terms for all candidate positions $u \in R_i^b$ to be computed in parallel. The resulting hidden states are also reused by the learned prior parameterizations described below, so the prior requires no additional transformer forward pass on the masked state. The token predictor is the only component that outputs vocabulary distributions and, together with the prior, is retained at inference time.

Variational Posterior q_ϕ. The variational posterior q_ϕ is conditioned on the clean sequence prefix $\boldsymbol{x}^{\le b}$, comprising block b and all preceding blocks. A single forward pass over the clean sequence $\boldsymbol{x}$, using bidirectional intra-block attention and causal inter-block attention, produces a scalar order score $f_{\phi,u}(\cdot) \in \mathbb{R}$ for every position $u \in \{1, \ldots, L\}$. For block b, the corresponding scores parameterize the conditional posterior as a Plackett–Luce distribution:

$$q_\phi(z_i^b = u \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b}) = \mathrm{PL}\left(u \mid R_i^b, \boldsymbol{f}_\phi^b(\boldsymbol{x}^{\le b})\right) = \frac{e^{f_{\phi,u}^b(\boldsymbol{x}^{\le b})}}{\sum_{u' \in R_i^b} e^{f_{\phi,u'}^b(\boldsymbol{x}^{\le b})}}, \quad u \in R_i^b. \tag{47}$$

Consequently, $q_\phi(z_i^b \mid \boldsymbol{z}_{<i}^b, \boldsymbol{x}^{\le b})$ depends on the sampled prefix $\boldsymbol{z}_{<i}^b$ only through the remaining candidate set R_i^b. Under sampling without replacement, R_i^b is obtained by excluding the positions already selected in $\boldsymbol{z}_{<i}^b$, and no additional posterior forward pass is required. We consider two parameterizations of the score function f_ϕ.

Backbone-shared posterior. The backbone-shared posterior reuses the transformer backbone f_θ of the token predictor and introduces an additional SwiGLU-based scoring head (Shazeer, 2020). Specifically, before evaluating f_θ on a masked state, we perform a separate forward pass over the clean sequence $\boldsymbol{x}$ using block-wise causal attention. Let

$$\boldsymbol{h}_\phi = f_\theta(\boldsymbol{x}) \tag{48}$$

denote the hidden states produced by this forward pass. The position-wise posterior scores are then computed using a bias-free SwiGLU head:

$$\overline{\boldsymbol{h}}^b_{\phi,u} = \mathrm{RMSNorm}\left(\rho\left(\boldsymbol{h}^b_{\phi,u}\right)\right),$$
$$f^b_{\phi,u}\left(\boldsymbol{x}^{\leq b}\right) = W_{\phi,\mathrm{down}}\left[\mathrm{SiLU}\left(W_{\phi,\mathrm{gate}}\overline{\boldsymbol{h}}^b_{\phi,u}\right) \odot W_{\phi,\mathrm{up}}\overline{\boldsymbol{h}}^b_{\phi,u}\right]. \tag{49}$$

The matrices $W_{\phi,\mathrm{gate}}, W_{\phi,\mathrm{up}} \in \mathbb{R}^{d_\phi \times d_\theta}$ and $W_{\phi,\mathrm{down}} \in \mathbb{R}^{1\times d_\phi}$ are bias-free learnable projections, and $\odot$ denotes element-wise multiplication. Here, d_θ denotes the hidden dimension of the MDM backbone f_θ, and d_ϕ denotes the intermediate dimension of the posterior head. To keep the scalar scoring head lightweight, we use a bottleneck dimension of $d_\phi = \lfloor d_\theta/2 \rfloor$. We set $\rho(\boldsymbol{h}) = \boldsymbol{h}$ for the non-detached variant and $\rho(\boldsymbol{h}) = \mathrm{sg}(\boldsymbol{h})$ for the detached variant, where $\mathrm{sg}(\cdot)$ denotes the stop-gradient operator. Detachment leaves the forward computation unchanged. In the non-detached variant, gradients for posterior optimization in Eq. 41, including the score-function term, also propagate through the shared transformer backbone f_θ. The RMSNorm configuration (Zhang & Sennrich, 2019) follows that of the corresponding backbone. This parameterization therefore introduces only the posterior scoring head as additional model parameters, although it still requires a separate transformer forward pass over the clean sequence.

Independent posterior. The independent posterior instead uses a separately parameterized transformer g_ϕ, including its own token embeddings and transformer layers:

$$\boldsymbol{h}_\phi = g_\phi^{(L_q)}(\boldsymbol{x}), \tag{50}$$

where L_q denotes the configurable posterior depth. In our experiments, each posterior layer uses the same architectural configuration as the corresponding MDM backbone, while L_q is configured independently of the backbone depth. This is an implementation choice rather than a requirement of the formulation: g_ϕ has its own parameters, and its architecture and depth can in principle be specified independently of f_θ. The resulting hidden state $\boldsymbol{h}^b_{\phi,u}$ is mapped to a scalar order score using a bias-free linear head:

$$f^b_{\phi,u}\left(\boldsymbol{x}^{\leq b}\right) = W_{\phi,\mathrm{out}}\,\mathrm{RMSNorm}\left(\boldsymbol{h}^b_{\phi,u}\right). \tag{51}$$

Here, $\boldsymbol{h}^b_{\phi,u} \in \mathbb{R}^{d_q}$ is the posterior hidden state at position u in block b; $W_{\phi,\mathrm{out}} \in \mathbb{R}^{1\times d_q}$ is a bias-free learnable projection; and d_q denotes the hidden dimension of the independent posterior, which is set to the hidden dimension of the corresponding MDM backbone in our implementation.

For both posterior parameterizations, the scores for all positions are computed in a single clean-sequence forward pass. A complete posterior permutation can then be sampled using the Gumbel-top-k trick (Kool et al., 2019b). The two posterior paths required by the two-sample RLOO estimator are obtained by applying two independent sets of Gumbel perturbations to the same position-wise scores, without repeating the posterior-network forward pass. The posterior is used only during training and is discarded at inference time.

Prior p_η. The prior is conditioned on the same partially observed state $\widetilde{\boldsymbol{x}}^b_i$ as the token predictor. Because the mask pattern in $\widetilde{\boldsymbol{x}}^b_i$ uniquely identifies the set of previously revealed positions, the prefix $\boldsymbol{z}^b_{<i}$ need not be encoded separately. Given $\widetilde{\boldsymbol{x}}^b_i$, the prior assigns a scalar order score $f^b_{\eta,u}(\cdot) \in \mathbb{R}$ to each position $u \in \{1,\ldots,n_b\}$. These scores parameterize the conditional prior as a Plackett–Luce distribution:

$$p_\eta(z^b_i = u \mid \boldsymbol{z}^b_{<i}, \widetilde{\boldsymbol{x}}^b_i) = \mathrm{PL}\left(u \mid R^b_i, \boldsymbol{f}^b_\eta(\widetilde{\boldsymbol{x}}^b_i)\right) = \frac{e^{f^b_{\eta,u}(\widetilde{\boldsymbol{x}}^b_i)}}{\sum_{u'\in R^b_i} e^{f^b_{\eta,u'}(\widetilde{\boldsymbol{x}}^b_i)}}, \quad u \in R^b_i. \tag{52}$$

As with the posterior, the scores for all positions are computed once for each conditioning state, and the prefix affects the conditional distribution only through the remaining candidate set R^b_i. We consider two parameterizations of the prior scores.

Confidence-based prior. The confidence-based prior derives its order scores directly from the token distributions produced by p_θ, without introducing an additional neural scoring head. For each candidate position, we first compute its top-1 token confidence:

$$c_u\left(\widetilde{\boldsymbol{x}}^b_i\right) = \max_{y\in\mathcal{V}} p_\theta\left(y \mid \widetilde{\boldsymbol{x}}^b_i, u\right), \tag{53}$$

and define the corresponding order score as

$$f^b_{\eta,u}\left(\widetilde{\boldsymbol{x}}^b_i\right) = \alpha \log c_u\left(\widetilde{\boldsymbol{x}}^b_i\right), \tag{54}$$

where $\alpha \in \mathbb{R}$ is a learnable scalar that controls the direction and strength of the confidence-based preference. A positive α assigns higher prior probability to positions with greater top-1 confidence, a negative α reverses this preference, and $\alpha = 0$ yields a uniform distribution over R^b_i. In our experiments, we do not detach the confidence values, so gradients from the order-learning objectives propagate through the token distributions to the token-predictor parameters.

Learned state-dependent prior. Alternatively, the learned prior applies either the linear or SwiGLU scoring head introduced above to the hidden states $\boldsymbol{h}_\theta$ produced by the MDM backbone from the current masked state. Specifically, the prior score is

$$f^b_{\eta,u}(\widetilde{\boldsymbol{x}}^b_i) = s_\eta(\boldsymbol{h}^b_{\theta,u}), \quad s_\eta \in \{s_{\text{Linear}}, s_{\text{SwiGLU}}\}. \tag{55}$$

Here, s_{Linear} and s_{SwiGLU} denote the linear and SwiGLU scoring heads, respectively, using the same configurations as their posterior counterparts described above. Because the token predictor already computes these hidden states, the learned prior requires no additional backbone forward pass. In the detached variant, we apply the stop-gradient operator to the backbone hidden states before passing them to the prior head; otherwise, gradients from the order-learning objectives also update the MDM backbone. The prior is retained together with p_θ at inference time to determine the order in which the remaining masked positions are decoded.

A.3 Supervised Fine-tuning

For an instruction-following example consisting of a prompt $\boldsymbol{c}$ and a response $\boldsymbol{y}$, we apply the same training framework to model the conditional response distribution $p(\boldsymbol{y} \mid \boldsymbol{c})$. The prompt tokens remain fully visible throughout training and are excluded from the latent permutations, receiving neither token-prediction nor order-learning supervision. The response positions are partitioned according to the same block structure used in pretraining. The conditional likelihood and variational posterior then factorize as

$$p_{\theta,\eta}(\boldsymbol{y} \mid \boldsymbol{c}) = \prod_{b=1}^{B} p_{\theta,\eta}\left(\boldsymbol{y}^b \mid \boldsymbol{c}, \boldsymbol{y}^{<b}\right), \quad q_\phi(\boldsymbol{z} \mid \boldsymbol{c}, \boldsymbol{y}) = \prod_{b=1}^{B} q_\phi(\boldsymbol{z}^b \mid \boldsymbol{c}, \boldsymbol{y}^{\le b}). \tag{56}$$

Accordingly, every masked state includes the fully visible prompt $\boldsymbol{c}$, while all terms in $\mathcal{L}_{\text{LO}}$, $\mathcal{L}_{\text{rec}}$, and $\mathcal{L}_{\text{ord}}$ are evaluated only over response positions. The unconditional pretraining formulation is recovered by setting $\boldsymbol{c} = \varnothing$. Evaluation results for the instruction-tuned models are reported in Appendix D.

B Experimental Details

B.1 Implementation Details

Unless otherwise specified, we use a block size of 32, the two-sample RLOO estimator, an independent transformer-based posterior, and a SwiGLU-parameterized learned prior head without backbone detachment. For SOLDIFF, we set the rank-decay temperature to $\tau = 8$ and fix $\lambda_{\text{rec}} = \lambda_{\text{ord}} = 1$. The token predictor, posterior, and prior are jointly optimized during training. For models initialized from Qwen3-1.7B-Base, we use a posterior depth of $L_q = 14$, corresponding to approximately 1B posterior parameters. For Qwen3-8B-Base and Qwen3-14B-Base, we use $L_q = 12$, corresponding to approximately 2.9B and 4.7B posterior parameters, respectively. We implement block-wise causal attention using FlexAttention (Dong et al., 2024) and train with FSDP2 full sharding (Zhao et al., 2023; Liang et al., 2025), bfloat16 mixed precision (Kalamkar et al., 2019), and gradient checkpointing (Chen et al., 2016). Detailed parameterizations of the token predictor, posterior, and prior are provided in Appendix A.2.

Continued Pretraining. We initialize the token predictor from existing pretrained language models, including Qwen3-1.7B-Base, Qwen3-8B-Base, and Qwen3-14B-Base, and continue pretraining on a mixture of web, knowledge, code, mathematical, and reasoning data. The independent posterior and

Table 4: Analytically estimated training FLOPs per original token at each model scale. The standard MDM uses a block size of 32; SOLDIFF uses a 14-layer posterior at 1.7B and a 12-layer posterior at 8B and 14B.

Scale	Method	Training GFLOPs / Original Token
1.7B	ARM	15.02
	Standard MDM	35.72
	SOLDIFF	79.00
8B	ARM	56.43
	Standard MDM	127.45
	SOLDIFF	275.15
14B	ARM	98.73
	Standard MDM	217.72
	SOLDIFF	467.17

learned prior head are initialized from scratch. We use a maximum sequence length of 4,096 and a global batch size of 1,024. We optimize all trainable parameters with AdamW (Loshchilov & Hutter, 2017), a learning rate of 2e-5, weight decay of 0.1, and gradient clipping at 0.1. The learning rate is held constant throughout training, without warmup. Continued pretraining with our method requires approximately 634, 1,801, and 2,780 NVIDIA H200 GPU-hours per 10B training tokens for the 1.7B, 8B, and 14B models, respectively. All baselines are trained using their official hyperparameter settings, with comparisons matched to our method in training FLOPs. We analytically estimate training FLOPs following Hoffmann et al. (2022), using the actual model dimensions and sequence lengths, together with the number of forward and backward passes required by each configuration. Detailed FLOP accounting and the resulting estimates are provided in Appendix B.3.

Supervised Fine-tuning. For supervised fine-tuning, we initialize the token predictor, posterior, and prior from the corresponding continued-pretrained checkpoint. We use a mixture of 9.79 million instruction-tuning examples spanning general instruction following, mathematical and logical reasoning, coding, tool use, multilingual dialogue, and long-form instruction following. We fine-tune for four epochs with a maximum sequence length of 8,192 and a global batch size of 1,024. We optimize with AdamW using a learning rate of 1e-5, weight decay of 0.01, and gradient clipping at 1.0. The learning rate is linearly increased to its peak value over the first 3% of training steps and subsequently decayed according to a cosine schedule (Loshchilov & Hutter, 2016). Supervised fine-tuning with our method requires approximately 887 and 3,043 H200 GPU-hours per epoch for the 1.7B and 8B models, respectively.

B.2 INFERENCE DETAILS

For all base models, we set the maximum generation length to 1,024 tokens, while for all instruction-tuned models, we use a maximum generation length of 8,192 tokens. For SOLDIFF, unless otherwise specified, we use deterministic prior ranking, sorting the remaining masked positions by the order scores produced by p_η. For all ARMs, we use standard left-to-right autoregressive decoding. For LLaDA and Dream, we follow their official decoding configurations. For Standard MDM, Bell-TS, and PosMask, we use the low-confidence remasking strategy with a block size of 32. For PUMA-1S, we additionally train a prior head on the confidence-induced masked states using a listwise ranking objective to reproduce the corresponding confidence ranking, thereby maintaining consistency between training and inference. During inference, the prior head ranks the remaining masked positions to determine the unmasking order. For R2LMask, we mask a suffix of the sequence during training, with the suffix length determined by the sampled mask ratio. At inference, we use a fixed left-to-right decoding order.

Table 5: Full results for the controlled comparison using Qwen3-1.7B-Base initialization. † denotes models continued-pretrained under the same setting as SOLDIFF.

Type	AR-Base Model		MDM-Base Model					SOLDIFF-Base (Ours)	
Model	Qwen3-1.7B	Qwen3-1.7B†	Standard MDM†	Bell-TS†	PosMask†	PUMA-1S†	R2LMask†	SOLDIFF$_{\text{LO}}$-1.7B	SOLDIFF-1.7B
Knowledge & Reasoning									
ARC-C	81.02	84.41	78.98	83.39	81.02	79.32	80.34	78.98	83.05
ARC-E	89.77	91.36	85.89	90.30	89.07	91.18	88.01	90.48	90.48
MMLU	61.53	61.36	58.12	59.00	57.82	57.69	58.73	60.56	60.94
MMLU-Pro	37.44	32.43	32.65	32.84	31.18	33.29	35.49	35.16	35.11
Mathematics									
GSM8K	77.03	75.28	70.51	68.54	70.28	74.83	73.54	75.66	74.07
MATH	42.50	44.38	34.02	34.82	36.10	38.14	41.16	43.78	43.64
OlympiadBench	15.10	11.20	10.10	12.30	12.20	13.10	14.10	14.20	14.10
Code Generation									
HumanEval	61.59	71.34	54.27	47.56	53.05	59.76	59.15	62.20	59.15
HumanEval+	53.05	64.63	50.00	42.68	50.00	52.44	53.66	56.71	53.05
MBPP	53.20	55.00	38.00	39.60	39.20	45.20	45.00	51.20	51.00
MBPP+	59.26	60.58	49.47	48.94	47.35	50.79	53.97	56.61	55.29
LiveCodeBench v6	7.05	8.81	7.27	7.05	7.05	7.27	7.93	7.49	8.59
Avg.	53.21	55.07	47.44	47.25	47.86	50.25	50.92	52.75	52.37

B.3 TRAINING FLOPS ESTIMATION

We analytically estimate training FLOPs following Hoffmann et al. (2022), using the actual model dimensions and Qwen3 architecture, including SwiGLU and grouped-query attention (GQA). We count each multiply–accumulate as two FLOPs and approximate the backward pass as twice the corresponding forward-pass cost. For consistency across attention patterns, we use the dense L^2 attention cost irrespective of the attention mask. All reported FLOPs are normalized by the number of original training tokens.

Let $F_\theta(L)$ denote the forward FLOPs of the Qwen3-based token predictor for a sequence of length L, including the transformer backbone and vocabulary projection; $F_q(L)$ the forward FLOPs of the independent posterior; and $F_p(L)$ the forward FLOPs of the learned prior head over L positions. We compute each term from the actual matrix dimensions at the corresponding model scale using the same component-wise accounting.

In our implementation, ARM training processes sequences of length $L = 4{,}096$, while block-diffusion training concatenates the noisy and clean sequences, resulting in a length of $2L$. SOLDIFF performs two token-predictor forward passes over length $2L$, corresponding to the two posterior samples in the two-sample RLOO estimator. The two permutations are sampled from a single posterior forward pass, and the prior head is evaluated once for each token-predictor path. The resulting training FLOPs per sequence are

$$C_{\text{ARM}} = 3F_\theta(L), \tag{57}$$

$$C_{\text{MDM}} = 3F_\theta(2L), \tag{58}$$

$$C_{\text{SOLDIFF}} = 3\left[2F_\theta(2L) + F_q(L) + 2F_p(L)\right]. \tag{59}$$

The parallel-aware auxiliary objectives reuse these forward passes and therefore incur no additional transformer computation.

Table 4 reports the resulting FLOPs per original training token. The default posterior uses 14 layers for the 1.7B model and 12 layers for the 8B and 14B models. The standard block-diffusion baseline uses a block size of 32.

C EXPERIMENTAL RESULTS

C.1 FULL RESULTS FOR THE CONTROLLED COMPARISON

Table 5 reports the full results across 12 benchmarks for the controlled comparison in §5.3. All methods use the same Qwen3-1.7B-Base initialization and training data, with training budgets matched by total FLOPs. Under this controlled setting, SOLDIFF outperforms all MDM baselines on average while remaining competitive with the ARM baseline, indicating that the gains arise from learned-order training rather than advantages in data, initialization, or compute budget.

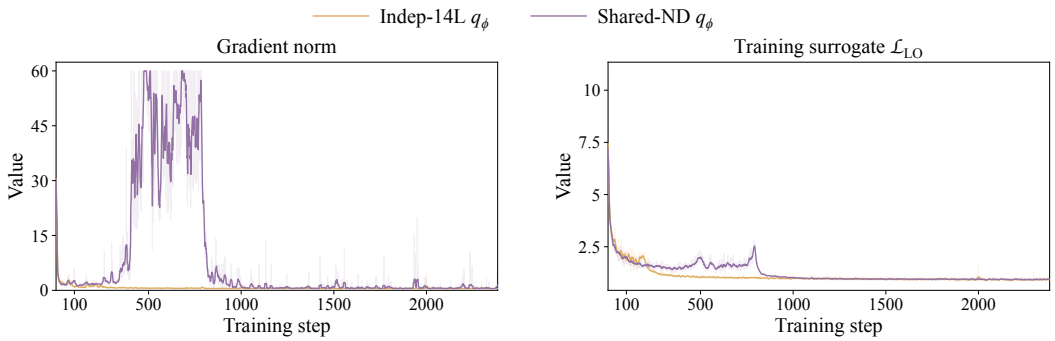

Figure 6: Gradient norm (left) and learned-order training surrogate $\mathcal{L}_{\text{LO}}$ (right) for the backbone-shared posterior without detachment (Shared-ND q_ϕ) and the independent posterior (Indep-14L q_ϕ). Darker curves show smoothed trends, with raw values shown faintly in the background.

C.2 Optimization Dynamics of the Backbone-shared Posterior

The results in §5.4 show that both the backbone-shared posterior without detachment (Shared-ND q_ϕ) and the independently parameterized posterior (Indep-14L q_ϕ) achieve strong downstream performance. To further characterize the trade-off between these parameterizations, Figure 6 compares their training dynamics. The shared variant allows the posterior to leverage the full backbone capacity, but backpropagating posterior gradients through f_θ also couples posterior optimization to the token predictor and learned prior through the shared backbone. Shared-ND consequently exhibits substantially larger fluctuations in both gradient norm and the learned-order training surrogate, including several sharp spikes, while Indep-14L shows considerably smoother training dynamics. Together with their strong downstream performance, these results support the independently parameterized posterior as a more stable default for large-scale training.

C.3 Effect of Order Scope on Parallel Decoding

The choice of order scope involves a trade-off between optimization difficulty and parallel decoding capability. As shown in §5.4, smaller blocks achieve stronger performance under serial decoding, consistent with reduced optimization difficulty, but also limit the degree of parallelism available at inference. To further characterize this trade-off, Figure 7 compares SolDiff-1.7B with block sizes $k = 8$ and $k = 32$ on GSM8K and MBPP across different levels of decoding parallelism, measured by average tokens per forward pass (TPF). At TPF $= 1$, the two settings achieve comparable performance. As TPF increases, the $k = 8$ model degrades sharply, while the $k = 32$ model maintains substantially better performance on both tasks. Smaller blocks restrict order learning and parallel decoding to fewer positions, limiting the flexibility to decode larger sets of tokens in parallel. Together with the greater optimization difficulty of full-sequence order learning observed in §5.4, these results motivate $k = 32$ as our default block size, providing a practical balance between optimization difficulty and parallel decoding flexibility.

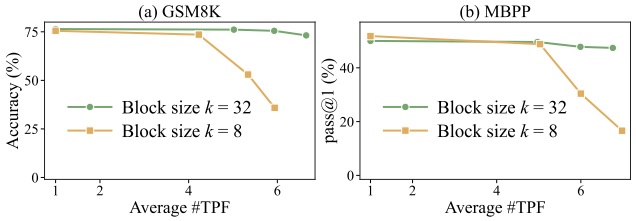

Figure 7: Effect of order scope on parallel decoding performance on GSM8K and MBPP.

C.4 Learned Order Distributions in Natural Language

Learned vs. Fixed LTR Training. The analysis in §5.7 shows that the learned order distribution on natural language develops a dominant left-to-right (LTR) preference while retaining local variation. To further assess the benefit of learning a distribution over orders, we compare SolDiff with a fixed

Table 6: Comparison of SOLDIFF and its fixed LTR variant after 20B, 40B, and 60B tokens of continued pretraining. Both variants use Qwen3-1.7B-Base initialization, identical data and optimization settings, and two token-predictor evaluations per example.

Method	SOLDIFF (Fixed LTR)			SOLDIFF		
Training Tokens	20B	40B	60B	20B	40B	60B
HumanEval	60.37	57.93	66.46	59.15	62.80	73.78
MBPP	47.60	48.00	49.40	51.00	50.60	50.00
GSM8K	75.21	76.57	78.24	74.07	75.97	76.42
MATH	42.80	46.02	44.64	43.64	44.88	45.54
Avg. (4)	56.50	57.13	59.69	56.97	58.56	61.44

LTR variant that replaces the learned posterior with a deterministic LTR posterior while retaining the token-prediction, prior-alignment, and parallel-aware objectives.

Specifically, for block b, let $\boldsymbol{z}^b_{\mathrm{LTR}} = (1, \ldots, n_b)$ and define

$$q_{\mathrm{LTR}}(\boldsymbol{z}^b \mid \boldsymbol{x}^{\leq b}) = \begin{cases} 1, & \boldsymbol{z}^b = \boldsymbol{z}^b_{\mathrm{LTR}}, \\ 0, & \text{otherwise.} \end{cases} \tag{60}$$

We replace q_ϕ with q_{LTR} throughout training, while keeping the token-prediction and prior-alignment terms, both parallel-aware objectives, and $\lambda_{\mathrm{rec}}, \lambda_{\mathrm{ord}}$ unchanged. The token predictor and prior remain trainable, whereas the posterior is no longer optimized. Under this deterministic posterior, the one-step ELBO contribution becomes

$$F^b_{i,\mathrm{LTR}} = \log p_\theta\big(x^b_i \mid \widetilde{\boldsymbol{x}}^b_{i,\mathrm{LTR}}\big) + \log p_\eta\big(z^b_i = i \mid \boldsymbol{z}^b_{<i,\mathrm{LTR}}, \widetilde{\boldsymbol{x}}^b_{i,\mathrm{LTR}}\big), \tag{61}$$

where $\widetilde{\boldsymbol{x}}^b_{i,\mathrm{LTR}}$ is induced by the fixed LTR prefix. The posterior entropy and score-function terms therefore vanish, while the prior is trained to reproduce the prescribed order. We evaluate $\mathcal{L}_{\mathrm{rec}}$ and $\mathcal{L}_{\mathrm{ord}}$ on the fixed LTR suffix using the same rank weighting. We also retain the same step sampling and two token-predictor evaluations per example, with data, optimization settings, and decoding otherwise matched.

Table 6 compares the two variants after 20B, 40B, and 60B tokens of continued pretraining. SOLDIFF achieves higher average performance at all three training budgets, indicating that the learned order distribution captures useful ordering structure beyond its dominant LTR pattern. Importantly, the learned posterior determines not only the preferred ordering but also the masked contexts sampled during training, the weighting of candidate next-position predictions, and the ordering targets used to train the prior. In the fixed LTR variant, these training signals are instead induced by a single prescribed order.

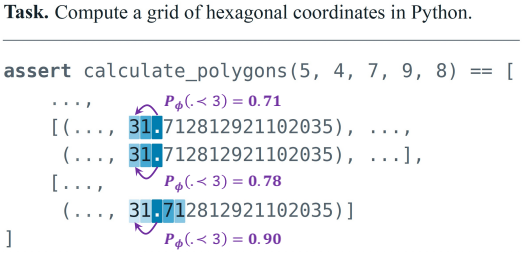

Figure 8: Local posterior structure in an MBPP example containing three occurrences of the same y-coordinate. Colors indicate relative ranks induced by the posterior scores among highlighted tokens, with darker blue denoting earlier ranks. Purple annotations report the pairwise posterior probabilities $P_\phi(\,. \prec 3)$. Ellipses denote omitted content.

Local Structure in the Learned Posterior. We further examine the local ordering structure captured by the learned posterior beyond its dominant LTR pattern. For a clean sequence, q_ϕ assigns a scalar score $f^b_{\phi,u}(\boldsymbol{x}^{\leq b})$ to each position u in block b. We visualize the relative ordering of selected tokens by directly ranking these scores, without Gumbel perturbations or permutation sampling. To quantify the strength of a local preference, we use the pairwise ordering probability induced by the Plackett–Luce posterior. For two positions u and v, let $u \prec v$ denote that u appears before v in the posterior permutation. Then

$$P_\phi(u \prec v \mid \boldsymbol{x}^{\leq b}) = \Pr_{\boldsymbol{z}^b \sim q_\phi(\cdot \mid \boldsymbol{x}^{\leq b})} \left[\mathrm{rank}_{\boldsymbol{z}^b}(u) < \mathrm{rank}_{\boldsymbol{z}^b}(v)\right] = \frac{\exp\Big(f^b_{\phi,u}(\boldsymbol{x}^{\leq b})\Big)}{\exp\Big(f^b_{\phi,u}(\boldsymbol{x}^{\leq b})\Big) + \exp\Big(f^b_{\phi,v}(\boldsymbol{x}^{\leq b})\Big)}. \tag{62}$$

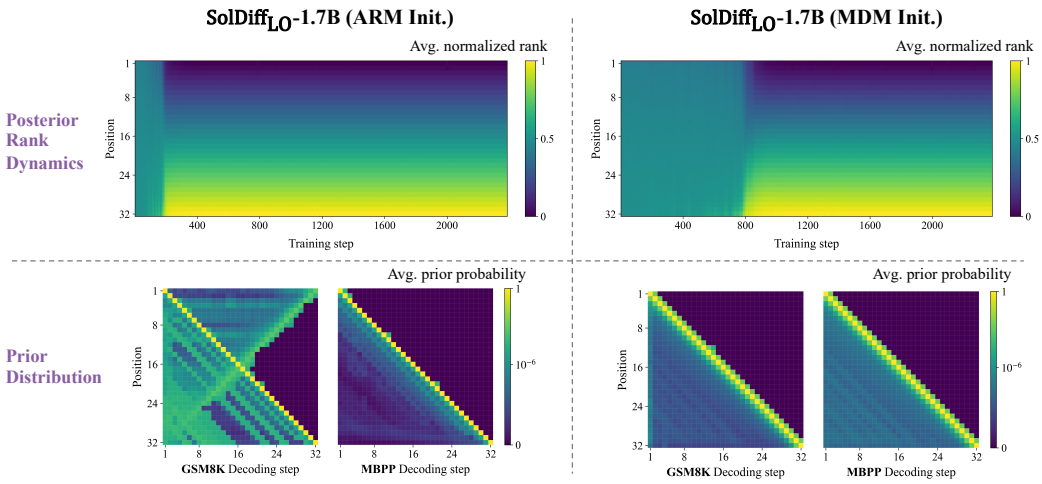

Figure 9: Effect of ARM and MDM initialization on learned ordering behavior. **Top:** Posterior rank dynamics across token positions during learned-order training. **Bottom:** Average prior distributions across decoding steps on GSM8K and MBPP after 20B tokens of learned-order training. The left and right columns correspond to ARM and MDM initialization, respectively.

Figure 8 shows an MBPP example involving the construction of a hexagonal coordinate grid, where the same y-coordinate appears three times in the target assertion. Although the decimal point follows the leading digit 3 in the text, the posterior consistently prefers to reveal the decimal point first, with $P_\phi(\,\texttt{.} \prec 3)$ equal to 0.71, 0.78, and 0.90 across the three occurrences. The repeated preference shows that these local departures from LTR reflect structure in the learned posterior rather than a particular sampled permutation. Moreover, the different preference strengths across occurrences indicate that the learned ordering remains context dependent even for the same local token pattern. This example illustrates how the learned distribution can capture structured local ordering preferences beyond its overall LTR tendency.

C.5 Order Learning under Different Initializations

Given the near LTR preference observed on natural-language data, we next ask whether this structure is tied to ARM initialization or also emerges when learned-order training starts from an MDM checkpoint. We compare SolDiff$_{LO}$-1.7B under two token-predictor initializations. The first initializes p_θ from Qwen3-1.7B-Base. The second initializes p_θ from a block-diffusion checkpoint obtained by continuing the same base architecture for 200B tokens with the standard block-diffusion objective. Starting from each checkpoint, we apply the same learned-order objective in Eq. 5 and track the posterior ordering preferences throughout training.

The top row of Figure 9 shows the posterior rank dynamics under the two initializations. With ARM initialization, the posterior rapidly develops a near LTR structure, with earlier within-block positions receiving lower normalized ranks. With MDM initialization, this structure emerges later: the posterior initially exhibits no clear positional preference, but subsequently develops the same overall near LTR pattern. By the later stages of training, the two initializations yield closely aligned posterior rank structures. These dynamics indicate that the dominant LTR preference is not simply inherited from direct ARM initialization, but is also recovered by learned-order training when starting from an MDM checkpoint.

We next compare the learned priors after 20B tokens of learned-order training. The bottom row of Figure 9 shows their average prior distributions across decoding steps on GSM8K and MBPP. Under both initializations, p_η exhibits a similar LTR-dominant structure while retaining local variation. Thus, although initialization affects how quickly the ordering preference emerges, both the posterior and prior ultimately recover similar ordering structure.

Table 7: Comparison of standard MDM and SOLDIFF$_{\mathrm{LO}}$ continuation from the same block-diffusion checkpoint under matched training FLOPs.

Setting	HumanEval	MBPP	GSM8K	MATH	Avg.
Initial MDM	60.98	42.40	76.72	41.14	55.31
MDM continuation	60.37	46.00	76.88	42.50	56.44
SOLDIFF$_{\mathrm{LO}}$ continuation	68.29	50.20	78.17	42.56	59.81

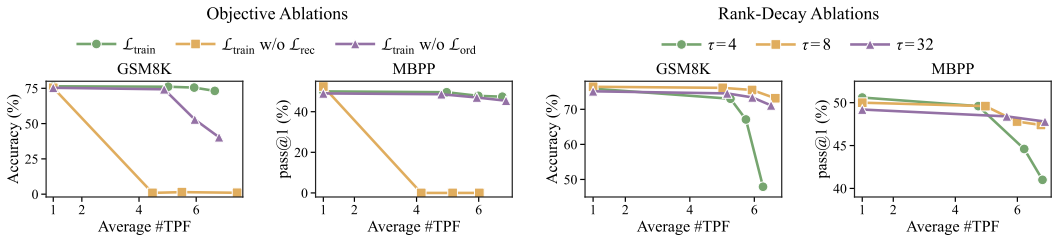

Figure 10: Parallel decoding ablations on GSM8K and MBPP. **Left:** objective ablations with $\tau = 8$. **Right:** rank-decay ablations with different τ.

Finally, we examine whether learned-order training remains beneficial when starting from an existing MDM checkpoint. From the same block-diffusion checkpoint, we train SOLDIFF$_{\mathrm{LO}}$ for 20B tokens using $\mathcal{L}_{\mathrm{LO}}$ and compare against standard block-diffusion continuation under matched training FLOPs. As shown in Table 7, SOLDIFF$_{\mathrm{LO}}$ outperforms the standard MDM continuation on all four benchmarks and achieves a higher average score (59.81 vs. 56.44). This result shows that the benefit of learned-order training is not limited to direct ARM-to-MDM conversion and extends to an already diffusion-adapted checkpoint.

C.6 Ablation of Parallel-aware Training

We ablate the two parallel-aware objectives in $\mathcal{L}_{\text{train}}$ and the rank-decay temperature τ to assess their effects on the performance–efficiency trade-off of SOLDIFF. All variants are initialized from Qwen3-1.7B-Base and trained for 60B tokens. Table 8 reports performance when decoding one token per step, while Figure 10 evaluates the performance–efficiency trade-off under parallel decoding.

We first ablate $\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{ord}}$ with $\tau = 8$. Removing either objective reduces performance under serial decoding, indicating that they provide complementary supervision. Their roles become more apparent under parallel decoding. Removing $\mathcal{L}_{\text{rec}}$ reduces supervision for predicting later suffix tokens from the same masked state, and performance drops sharply as decoding parallelism increases. Removing $\mathcal{L}_{\text{ord}}$ causes a smaller degradation because $\mathcal{L}_{\text{rec}}$ still trains parallel token prediction, but the prior is no longer explicitly supervised to rank multiple future reveal positions.

We next vary the rank-decay temperature τ. Increasing τ flattens the rank weights and assigns more weight to later suffix positions, which generally improves parallel decoding and can also improve performance under serial decoding. However, an overly large τ shifts too much auxiliary supervision to later suffix positions, weakening the overall emphasis on the next reveal and reducing serial-decoding performance. Overall, we observe a sweet spot at $\tau = 8$, which balances serial and parallel decoding performance, and use it as the default setting.

Table 8: Ablation of the parallel-aware objectives and rank-decay temperature τ under serial decoding.

Method	HumanEval	MBPP	GSM8K	MATH	Avg.
Default					
$\mathcal{L}_{\text{train}},\ \tau = 8$	73.78	50.00	76.42	45.54	61.44
Objective Ablations					
$\mathcal{L}_{\text{train}}$ w/o $\mathcal{L}_{\text{rec}},\ \tau = 8$	63.41	52.40	75.36	44.60	58.94
$\mathcal{L}_{\text{train}}$ w/o $\mathcal{L}_{\text{ord}},\ \tau = 8$	62.80	49.00	75.36	44.86	58.01
Rank-Decay Ablations					
$\mathcal{L}_{\text{train}},\ \tau = 2$	62.80	51.20	73.54	43.20	57.69
$\mathcal{L}_{\text{train}},\ \tau = 4$	68.29	50.60	75.89	43.46	59.56
$\mathcal{L}_{\text{train}},\ \tau = 6$	72.56	50.40	75.59	44.92	60.87
$\mathcal{L}_{\text{train}},\ \tau = 12$	65.85	49.40	75.89	45.26	59.10
$\mathcal{L}_{\text{train}},\ \tau = 16$	62.80	48.20	75.97	41.84	57.20
$\mathcal{L}_{\text{train}},\ \tau = 32$	64.02	49.20	75.13	42.74	57.77
$\mathcal{L}_{\text{train}}$, Uniform	62.20	48.60	76.80	43.52	57.78

D SUPERVISED FINE-TUNING

D.1 EXPERIMENTAL SETUP

We further evaluate SOLDIFF after supervised fine-tuning (SFT). Starting from the corresponding continued-pretraining checkpoints, we fine-tune SOLDIFF$_{\text{LO}}$-8B and SOLDIFF-8B using the SFT setup described in Appendix B.1.

Evaluation Benchmarks and Metrics. We evaluate the instruction-tuned models on a wide range of benchmarks: (1) general knowledge and reasoning: CEval (Huang et al., 2023), IFEval (Zhou et al., 2023), KorBench (Ma et al., 2025), MMLU-Pro (Wang et al., 2024), and GPQA-Diamond (Rein et al., 2023); (2) mathematics: GSM-Plus (Li et al., 2024), MATH-500 (Lightman et al., 2024), and OlympiadBench (He et al., 2024); and (3) code generation: HumanEval+ (Liu et al., 2023), MBPP+ (Liu et al., 2023), and LiveCodeBench v6 (Jain et al., 2025). We report pass@1 for code-generation benchmarks and accuracy for all others.

Baselines. We compare against instruction-tuned autoregressive models, including Llama-3.1-8B-Instruct and Qwen3-8B, and instruction-tuned MDMs, including LLaDA-8B-Instruct, Dream-v0-Instruct-7B, SDAR-8B-Chat (Cheng et al., 2026), and Nemotron-Labs-Diffusion-8B-Instruct. We follow the official evaluation configuration for each baseline. For SDAR-8B-Chat, we use the official checkpoint with a block size of 32.

D.2 MAIN RESULTS

Table 9 shows that the advantage of SOLDIFF persists after supervised fine-tuning. SOLDIFF$_{\text{LO}}$-8B and SOLDIFF-8B achieve average scores of 67.70% and 67.33%, respectively, outperforming all evaluated ARM and MDM baselines and establishing state-of-the-art performance at this scale. In particular, SOLDIFF$_{\text{LO}}$-8B improves the overall average by 2.13 points over the strongest ARM baseline and by 7.59 points over the strongest MDM baseline. Between them, the two SOLDIFF variants achieve the best performance on 8 of the 11 benchmarks, with gains across reasoning, mathematics, and code generation. These results show that the benefits of learned-order training persist after supervised fine-tuning and are not limited to base-model performance.

Table 9: Comparison of SOLDIFF-Instruct with different instruction-tuned models. Best and second-best results are shown in **bold** and underlined, respectively.

Type	AR-Instruct Model		MDM-Instruct Model				SOLDIFF-Instruct (Ours)	
Model	Llama-3.1-8B	Qwen3-8B	LLaDA-8B	Dream-7B	SDAR-8B	Nemotron-Labs-Diffusion-8B	SOLDIFF$_{\text{LO}}$-8B	SOLDIFF-8B
Knowledge & Reasoning								
CEval	54.67	81.66	60.32	56.38	76.75	57.48	**84.34**	84.06
IFEval	74.49	**82.62**	55.82	56.38	53.42	56.75	80.41	81.33
KorBench	44.24	**54.16**	33.52	36.00	37.84	46.96	51.68	47.92
MMLU-Pro	48.92	53.15	37.14	45.40	50.91	60.90	**64.89**	63.15
GPQA-Diamond	28.66	47.60	23.74	29.04	32.83	41.79	**49.24**	48.86
Mathematics								
GSM-Plus	75.43	86.03	71.30	73.71	80.07	85.77	86.94	**87.02**
MATH-500	51.00	83.40	40.80	50.60	69.60	84.60	86.60	**86.80**
OlympiadBench	17.48	53.04	11.85	16.74	33.93	50.81	55.70	**55.85**
Code Generation								
HumanEval+	61.59	81.71	37.20	56.10	45.73	75.61	**82.32**	**82.32**
MBPP+	49.74	71.43	44.97	63.49	57.14	71.69	75.93	**76.46**
LiveCodeBench v6	13.44	26.43	6.17	10.79	11.89	**28.85**	26.65	26.87
Avg.	47.24	65.57	38.44	44.97	50.01	60.11	**67.70**	67.33

E DETAILS OF THE CONTROLLED PARTIAL-ORDER TASKS

E.1 TASK CONSTRUCTION

The goal of *Parallel Chains* is to construct a controlled generation task with a known partial-order structure. Each example consists of a prompt $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_S)$ and an answer $\boldsymbol{b} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_S)$, where $\boldsymbol{b}_s = (b_{s,1}, \ldots, b_{s,K})$. Each prompt segment $\boldsymbol{a}_s$ specifies an independent task whose answer forms a length-K dependency chain. By construction, the target conditional distribution factorizes as $p(\boldsymbol{b} \mid \boldsymbol{a}) = \prod_{s=1}^{S} \prod_{k=1}^{K} p(b_{s,k} \mid \boldsymbol{a}_s, \boldsymbol{b}_{s,<k})$. We use $b_{s,1} \prec b_{s,2} \prec \cdots \prec b_{s,K}$ to denote the step-by-step recovery order: each state is obtained from the previously recovered state by one inverse operation, although a model can in principle skip intermediate states by composing multiple inverse operations. No ordering constraints are imposed across chains. Any interleaving of the S chains that preserves these relations therefore constitutes a valid generation order under this step-by-step dependency structure. This construction separates the sequential dependency depth, controlled by K, from the available parallelism, controlled by S, without specifying a unique target permutation.

We realize each chain as a backward state-recovery problem. The prompt specifies a sequence of deterministic operations together with the terminal state obtained after applying them, and the model is asked to recover the preceding intermediate states in reverse order. Starting from the given terminal state, the immediately preceding state can be recovered first; this recovered state then determines the next preceding state, and the process continues until the entire chain is reconstructed. This construction realizes the dependency chain defined above.

Concretely, we use a finite discrete state space and a fixed set of invertible operations. Each state takes a value x from

$$\mathcal{X} = \{0, 1, \ldots, Q-1\}. \tag{63}$$

Each operation $o \in \{1, \ldots, A\}$ defines an affine transformation $\pi_o : \mathcal{X} \to \mathcal{X}$ given by

$$\pi_o(x) = (\alpha_o x + \beta_o) \bmod Q, \qquad x \in \mathcal{X}. \tag{64}$$

Because the task requires recovering states in the reverse direction, each operation must be invertible. We choose Q to be prime and require $\alpha_o \neq 0$, ensuring that each π_o is a bijection on $\mathcal{X}$ and therefore admits a unique inverse.

For each chain s, we independently sample a sequence of K operations $(o_{s,1}, \ldots, o_{s,K})$ from the A available operations and an initial state $x_{s,0} \in \mathcal{X}$. Applying the sampled operations sequentially gives

$$x_{s,0} \xrightarrow{\pi_{o_{s,1}}} x_{s,1} \xrightarrow{\pi_{o_{s,2}}} \cdots \xrightarrow{\pi_{o_{s,K}}} x_{s,K}, \tag{65}$$

where

$$x_{s,k} = \pi_{o_{s,k}}(x_{s,k-1}), \qquad k = 1, \ldots, K. \tag{66}$$

Table 10: Serialization of a *Parallel Chains* example. The prompt specifies the operation sequence and terminal state for each chain, while the answer contains the preceding states in reverse order.

Prompt
<BOS>
$o_{1,1}; \cdots ; o_{1,K}; x_{1,K}; \backslash\mathtt{n}$
$\vdots$
$o_{S,1}; \cdots ; o_{S,K}; x_{S,K}; \backslash\mathtt{n}$
Answer
$x_{1,K-1}; \cdots ; x_{1,0};$
$\vdots$
$x_{S,K-1}; \cdots ; x_{S,0}$

The prompt for chain s contains the complete operation sequence together with the terminal state,

$$\boldsymbol{a}_s = (o_{s,1}, \ldots, o_{s,K}, x_{s,K}), \tag{67}$$

while the corresponding answer contains the preceding states in reverse order,

$$\boldsymbol{b}_s = (x_{s,K-1}, x_{s,K-2}, \ldots, x_{s,0}). \tag{68}$$

Because the operations are invertible, the first answer state is recovered directly from the terminal state:

$$b_{s,1} = \pi^{-1}_{o_{s,K}}(x_{s,K}). \tag{69}$$

Each subsequent answer state is then recovered recursively:

$$b_{s,k} = \pi^{-1}_{o_{s,K-k+1}}(b_{s,k-1}), \qquad k = 2, \ldots, K. \tag{70}$$

This defines the canonical within-chain recovery order

$$b_{s,1} \prec b_{s,2} \prec \cdots \prec b_{s,K}. \tag{71}$$

Since the S chains are constructed independently, no ordering constraints are imposed across chains. Therefore, any interleaving of the S chains that preserves the within-chain order constitutes a valid generation order.

We use the following fixed set of operation parameters:

$$\{(\alpha_o, \beta_o)\}_{o=1}^{A} = \{(1,1), (2,3), (3,5), (5,7), (7,11), (11,13), (13,17), (17,19)\}. \tag{72}$$

The same operation set is used throughout data generation and shared across all training and test examples.

Each example is serialized as shown in Table 10. Operation symbols and state symbols use disjoint token sets, and newline tokens are used only to delimit prompt segments. The prompt remains fully visible throughout training, while only the SK answer tokens participate in masking, token-prediction losses, and latent order permutations.

E.2 Training Details

All backbone models follow the Qwen3-0.6B-Base architecture but are initialized from scratch, without using pretrained model weights. The backbone consists of 28 transformer layers with a hidden size of 1,024, an intermediate size of 3,072, 16 attention heads, and 8 key–value heads. The ARM baseline uses causal attention and the standard next-token prediction objective. The standard MDM uses full bidirectional attention and is trained to predict randomly masked answer tokens at their original positions. SoLDiff$_{\mathrm{LO}}$ uses the same fully bidirectional backbone as the standard MDM and is trained with the learned-order objective in Eq. 5. Its posterior q_ϕ uses $L_q = 14$ transformer layers and contains approximately 376M parameters.

For data generation, we use $Q = 97$. Each prompt segment is uniquely determined by its terminal state and K operations, resulting in QA^K possible prompt segments and $|\Omega_{S,K}| = (QA^K)^S$ possible examples for a fixed (S, K) configuration. We generate examples without replacement using a seeded bijective mapping over the discrete sample space. The training and test sets are assigned disjoint index ranges, ensuring no overlap between the two splits. For each (S, K) configuration, we generate 20M training examples and 2,000 test examples.

All models are trained for one epoch with a global batch size of 8,192. We optimize with AdamW using a learning rate of 1e-4 and weight decay of 0.1. The learning rate is held constant throughout training without warmup, the gradient norm is clipped at 1.0, and training uses bfloat16 mixed precision.

E.3 Evaluation and Order Metrics

Evaluation Protocol. During evaluation, the prompt remains fully observed and the answer is generated one token per decoding step. The ARM follows a fixed left-to-right generation order, the standard MDM uses low-confidence remasking, and SolDiff$_{\mathrm{LO}}$ selects the next position according to its learned prior p_η. We report sequence-level exact-match accuracy over the complete answer.

Posterior Rank. Given N posterior permutations $\{\boldsymbol{z}^{(n)}\}_{n=1}^N$ collected during training, we compute the average normalized rank of each answer position u as

$$\bar{r}_u = \frac{1}{N(SK-1)} \sum_{n=1}^{N} \left(\mathrm{rank}_{\boldsymbol{z}^{(n)}}(u) - 1\right). \tag{73}$$

Here, $\mathrm{rank}_{\boldsymbol{z}^{(n)}}(u) \in \{1, \ldots, SK\}$ denotes the position of u in permutation $\boldsymbol{z}^{(n)}$. Smaller values of $\bar{r}_u$ indicate earlier unmasking, and a uniformly random permutation yields $\mathbb{E}[\bar{r}_u] = 0.5$ for every position. We use this statistic to visualize the posterior rank dynamics during training in Figure 5(a).

Prior-induced Decoding Order. To characterize the inference-time order induced by the prior, we decode M test examples and record the position $u_r^{(m)}$ selected at decoding step r for example m. For each answer position u, we compute

$$\hat{f}_{r,u} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{1}\left[u_r^{(m)} = u\right], \tag{74}$$

where $M = 2{,}000$ in our experiments. Thus, $\hat{f}_{r,u}$ is the fraction of test examples in which position u is decoded at step r, as visualized in Figure 5(b).

We additionally measure the precedence violation rate over the ground-truth within-chain dependency edges. Let

$$\mathcal{E} = \{(b_{s,k}, b_{s,k+1}) : \ s = 1, \ldots, S, \ k = 1, \ldots, K-1\}.$$

A dependency edge is counted as violated when its predecessor is decoded after its successor, and the violation rate is the fraction of violated edges, averaged over test examples. A uniformly random generation order has an expected violation rate of 50%.

Additional Order Analysis. Figure 11 reports the posterior rank dynamics and prior-induced decoding order for the longer-chain setting $(S, K) = (3, 16)$, complementing the $(6, 8)$ setting shown in §5.7. The posterior develops the same monotonic within-chain ranking, while positions at comparable depths across independent chains receive similar ranks. At inference, the prior likewise preserves the within-chain precedence relations while interleaving the three chains, achieving a 0% precedence violation rate.

E.4 Performance and Scaling Behavior on Controlled Tasks

Model Comparison. Figure 12 compares sequence-level exact-match accuracy across representative (S, K) configurations. ARM provides a strong reference because its fixed left-to-right order is consistent with the within-chain dependencies and does not require latent-order optimization.

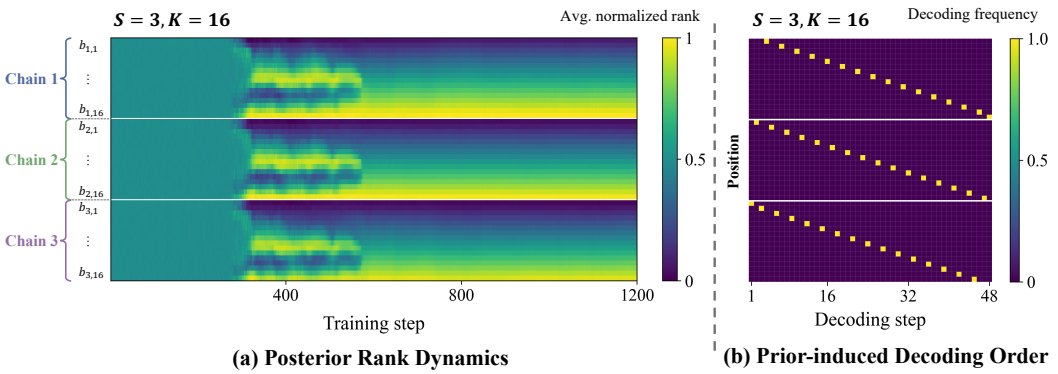

Figure 11: Additional order-learning results on *Parallel Chains* for $S = 3,\ K = 16$. (a) Posterior rank dynamics during training. (b) Prior-induced decoding order at inference.

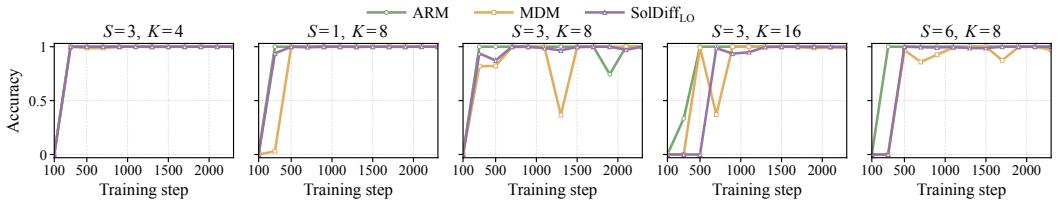

Figure 12: Test exact-match accuracy over training for ARM, standard MDM, and SOLDIFF$_{\mathrm{LO}}$ under representative (S, K) settings.

SOLDIFF$_{\mathrm{LO}}$ outperforms the standard MDM across most settings, consistent with the benefit of concentrating training on dependency-consistent orders rather than uniformly averaging over possible orders. The slower convergence for $(S, K) = (3, 16)$ suggests that increasing dependency depth makes order learning more difficult, which we examine systematically below.

Scaling with Dependency Structure and Model Capacity. We next examine how learned-order modeling scales with dependency depth, the number of parallel chains, and model capacity. We first vary the chain length K while fixing the number of chains at $S = 3$. As shown in Figure 13(left), increasing K makes the task substantially more difficult to learn. For $K = 4$ and $K = 8$, test accuracy rapidly approaches 100%. The $K = 16$ setting also converges, but requires more training to reach a comparable level. By contrast, with the default model capacity, test accuracy remains near zero for $K = 32$ and $K = 48$. These results show that learning becomes substantially more difficult as the sequential dependency depth increases.

To separate the effect of dependency depth from that of the number of generated tokens, we next fix $K = 8$ and vary the number of independent chains S. As shown in Figure 13(middle), the model successfully learns all configurations from $S = 1$ to $S = 18$, with test accuracy eventually approaching 100% across all settings. Increasing S therefore has a much smaller effect on learning difficulty than increasing K. Adding independent chains increases the output length and the number of valid global interleavings, but leaves the dependency depth within each chain unchanged. Together, these results indicate that the main source of difficulty in this controlled task is the depth of the sequential dependency rather than the number of parallel chains or the total number of answer tokens.

We finally examine whether the failure on long chains reflects an inherent limitation of learned-order modeling or insufficient model capacity. For the challenging $(S, K) = (3, 32)$ setting, the default configuration uses a 0.6B token predictor p_θ and a 376M posterior q_ϕ, and shows little learning progress. As shown in Figure 13(right), increasing the capacity of either q_ϕ or p_θ enables successful learning in the $(S, K) = (3, 32)$ setting. The 2.9B q_ϕ with the 0.6B p_θ converges earlier, while the 1.7B p_θ with the 376M q_ϕ also converges but requires more training. These results suggest that

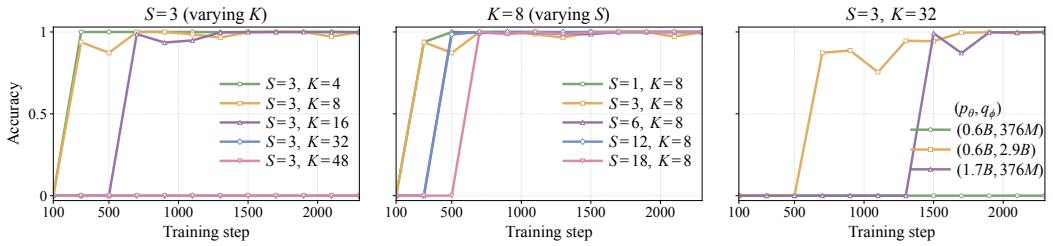

Figure 13: Scaling behavior of SOLDIFF$_{\mathrm{LO}}$ on *Parallel Chains*. **Left:** Test accuracy when varying the dependency depth K with $S = 3$. **Middle:** Test accuracy when varying the number of independent chains S with $K = 8$. **Right:** Capacity ablation for the challenging $(S, K) = (3, 32)$ setting. The 1.7B token predictor uses the Qwen3-1.7B-Base architecture and is trained from scratch. The 2.9B posterior uses $L_q = 12$, with each transformer layer following the architectural configuration of a Qwen3-8B-Base layer.

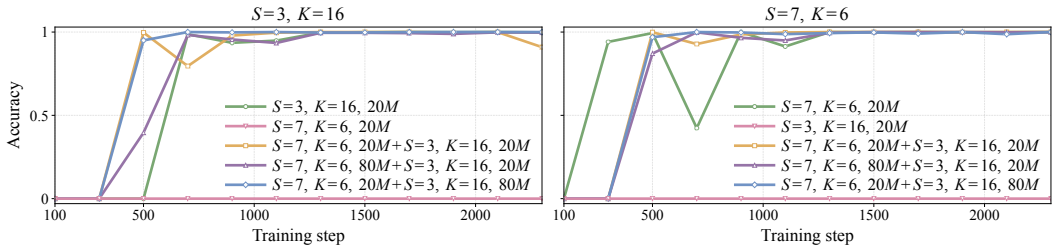

Figure 14: Test accuracy of SOLDIFF$_{\mathrm{LO}}$ on the $(S, K) = (3, 16)$ and $(S, K) = (7, 6)$ dependency configurations under single-pattern and mixed-pattern training.

the failure on long dependency chains is not inherent to the learned-order formulation. Increasing the capacity of either the posterior or the token predictor substantially improves learning on deeper dependency structures, indicating that both components can become capacity bottlenecks as the dependency depth increases.

E.5 Order Learning under Mixed Dependency Patterns

We further investigate whether one SOLDIFF$_{\mathrm{LO}}$ model can learn multiple dependency structures simultaneously, rather than collapsing to a fixed global generation order. We jointly train on two *Parallel Chains* configurations, $(S, K) = (3, 16)$ and $(S, K) = (7, 6)$. The former consists of three relatively long dependency chains, while the latter contains seven shorter chains and therefore allows substantially greater parallelism across chains.

We consider both single-pattern and mixed-pattern training. As single-pattern references, we train one model on 20M examples from $(S, K) = (3, 16)$ and another on 20M examples from $(S, K) = (7, 6)$. For mixed-pattern training, we construct three datasets containing 20M/20M, 20M/80M, and 80M/20M examples from the $(3, 16)$ and $(7, 6)$ configurations, respectively, corresponding to mixture ratios of 1:1, 1:4, and 4:1. For each mixture, examples from the two configurations are combined and randomly shuffled, and a single model is trained on the resulting dataset. Each mixed-pattern model is then evaluated separately on the test sets of both configurations.

Figure 14 shows test accuracy on the two dependency configurations throughout training. The single-pattern models provide references for learning each configuration in isolation and, as expected, achieve zero accuracy on the unseen configuration. Across all three mixed-pattern settings, a single model successfully learns both configurations, including the imbalanced 1:4 and 4:1 mixtures in which one configuration accounts for only a minority of the training data. Performance on each configuration remains comparable to that of the corresponding single-pattern reference. These results show that training on heterogeneous dependency structures does not force the model to specialize to

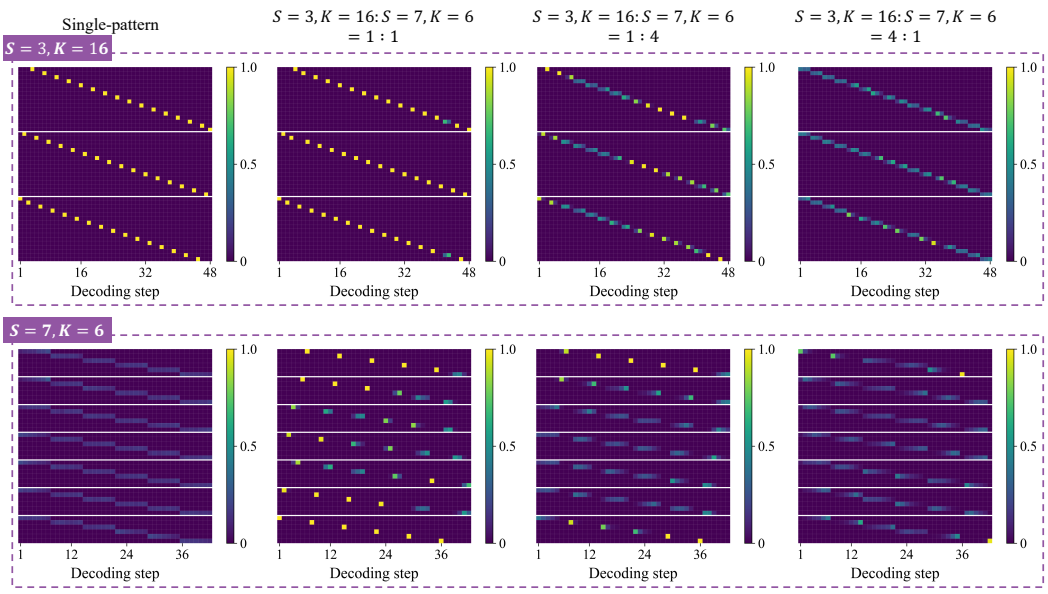

Figure 15: Prior-induced decoding orders under single-pattern and mixed-pattern training. The top and bottom rows show evaluation on $(S, K) = (3, 16)$ and $(S, K) = (7, 6)$, respectively. From left to right, the columns correspond to the single-pattern reference and mixed-pattern training with 1:1, 1:4, and 4:1 mixture ratios.

a single pattern. Instead, the same model can simultaneously learn tasks with substantially different dependency depths and degrees of parallelism.

We next examine whether the jointly trained model also learns the appropriate generation order for each dependency structure. For each training setting, we use the checkpoint saved at training step 2,400 for this analysis. We decode one token at each step, using the learned prior p_η to select the next position, and record the decoding step at which each answer position is selected. Figure 15 visualizes the resulting prior-induced decoding orders. For each answer position, the heatmap shows the distribution of its decoding step across test examples. Across all mixture ratios, the learned prior recovers the corresponding partial-order structure for both configurations. Within each chain, decoding follows the required reverse-chain order, starting from the state immediately preceding the observed endpoint and proceeding toward the initial state. By contrast, positions from different chains exhibit strongly overlapping decoding-step distributions and are interleaved throughout generation, consistent with the absence of dependencies across chains. For example, under $(S, K) = (7, 6)$, all seven chains exhibit the same monotonic six-step progression, while their decoding trajectories overlap rather than following a fixed chain-by-chain order.

These results show that learned-order training does not simply induce a single position-dependent decoding schedule. Instead, a single posterior–prior pair can represent multiple input-dependent partial-order structures and induce the appropriate generation order for each example. This qualitative ordering behavior remains stable even when the two dependency patterns occur at substantially different frequencies.

E.6 Parallel Decoding Analysis

The controlled partial-order tasks provide a direct test of whether the learned order exposes opportunities for parallel decoding. For a task with S independent chains of depth K, serial decoding requires SK steps, while the minimum parallel decoding depth is K, allowing up to S tokens to be decoded per step. The left panel of Figure 16 evaluates this setting by decoding S tokens per step for $(S, K) = (6, 8)$ and $(3, 16)$. Although SolDiff$_{\text{LO}}$ is trained without the parallel-aware auxiliary objectives, it achieves nearly 100% exact-match accuracy in both settings. This contrasts sharply with its weaker parallel decoding performance on natural-language tasks.

Interestingly, the right panel of Figure 16 shows that, using the checkpoint at training step 2,000 for $(S, K) = (6, 8)$, performance remains high even when the decoding parallelism slightly exceeds the available parallelism $S = 6$, e.g., at TPF $= 8$ or 10. Performance then collapses at larger TPF values. Under the ground-truth partial order, at most six positions can be decoded in parallel without violating the within-chain precedence constraints. Decoding more than six tokens therefore necessarily includes positions whose predecessors within the same chain have not yet been revealed. However, because the controlled task is defined by deterministic functions, a sufficiently capable model can sometimes predict such later tokens by implicitly composing multiple dependency steps. This allows modest decoding beyond the available parallelism, but the benefit disappears as decoding reaches positions farther along the dependency chains.

We further measure the pairwise precedence violation rate when decoding six tokens per step. For every within-chain pair $b_{s,i} \prec b_{s,j}$ with $i < j$, we count a violation if $b_{s,i}$ is decoded in the same step as or after $b_{s,j}$. The resulting violation rate is 1.19%. Together, these results show that learned-order modeling captures the underlying dependency structure and enables parallel generation when no precedence constraint exists.

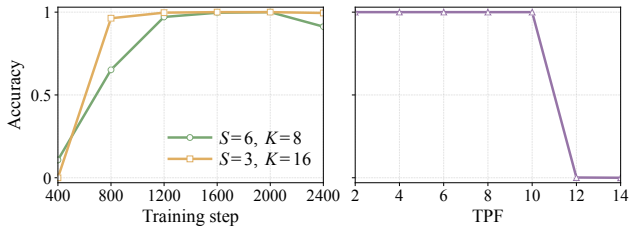

Figure 16: Parallel decoding on controlled partial-order tasks. **Left:** Exact-match accuracy when decoding S tokens per step for $(S, K) = (6, 8)$ and $(3, 16)$. **Right:** Exact-match accuracy of the $(S, K) = (6, 8)$ model at different TPF values using the 2,000-step checkpoint.