CoDR: Training-Free Confidence-Drift Remasking for Diffusion Language Models

Yue Wu[1,†], **Qinghe Zheng**[2,†], **Yu Zhang**[2,*], **Jian Huang**[1,*]
[1]The Hong Kong Polytechnic University, Hong Kong, China
[2]Southern University of Science and Technology, Shenzhen, China
yue0301.wu@connect.polyu.hk, 12532603@mail.sustech.edu.cn
yu.zhang.ust@gmail.com, jianhuang@polyu.edu.hk
[†]Equal contribution. [*]Corresponding authors.

Abstract

Masked diffusion language models (MDLMs) decode by repeatedly committing tokens to masked positions, but these commitments are usually irreversible. A token chosen under sparse, partial context is kept fixed, even when later context no longer supports it. Existing samplers mainly decide when to commit a token, but rarely check whether an already committed token should still be kept, allowing early mistakes to propagate. We trace this issue to confidence drift, where the model's confidence in a committed token drops from its sparse commit-time context to the denser context available later. Based on this signal, we propose **CoDR** (**Co**nfidence-**D**rift **R**emasking), a training-free and sampler-agnostic refinement pass. CoDR estimates drift for all committed positions in only k forward passes via k-partition probing, then remasks and regenerates only the tokens the model no longer endorses. Across two backbones, four reasoning and coding tasks, and three base samplers, CoDR improves average accuracy across all evaluated model–sampler configurations and improves most individual task settings with modest overhead. Controlled experiments show that the gains come from targeted confidence-drift remasking rather than extra compute alone, and that CoDR uses far fewer forward passes than prior remasking methods. Code is available at https://github.com/YueWu0301/CoDR.

1 Introduction

Masked diffusion language models (MDLMs) (Austin et al., 2021a) provide a parallel alternative to autoregressive generation (Sahoo et al., 2024; Nie et al., 2025; Ye et al., 2025). Starting from a fully masked sequence, they iteratively commit tokens until generation is complete, enabling flexible any-order decoding (Ou et al., 2024; Kim et al., 2025a) and potentially lower inference latency. With model parameters fixed at inference time, recent work has therefore focused on samplers that improve commitment quality and efficiency (Wu et al., 2025; Ben-Hamu et al., 2026). However, these methods rarely revisit tokens after they are committed.

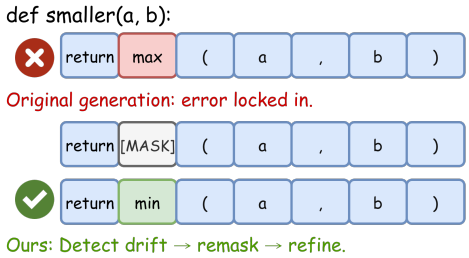


Figure 1: Standard generation fixes tokens once committed, leaving early mistakes unchanged. Drift-guided refinement detects unstable committed tokens, remasks them, and refills them under the updated context.

This creates irreversible commitment: a token selected under incomplete context remains fixed even if later context no longer supports it. A locally plausible token at commitment time may become incompatible after surrounding positions are filled. The problem is amplified when several positions are committed in parallel, because co-committed tokens do not condition on one another and may introduce local inconsistencies (Gu et al., 2018; Wu et al., 2025; Wu and Huang, 2026). These early commitments then condition later predictions, allowing errors to propagate through subsequent decoding. A natural remedy is to revisit unstable tokens once richer context becomes available rather than treating every commitment as final.

Existing samplers use confidence, entropy, or attention signals to decide which masked positions to commit (Wu et al., 2025; Ben-Hamu et al., 2026; Zhou et al., 2026), making the initial commitment more cautious but usually keeping committed tokens fixed. Recent remasking methods relax this constraint: ReMDM iteratively revisits commit-

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 26028–26044
October 24-29, 2026 ©2026 Association for Computational Linguistics

ments (Wang et al., 2026a), while CORE probes robustness to perturbed contexts (Zhai et al., 2026). These approaches establish the value of reversible generation, but do not directly measure how support for a token changes as decoding progresses. We instead ask whether the model's confidence in the same token decreases from its early commit-time context to a later, denser context. Such confidence drift directly identifies commitments that lose model support during decoding.

We propose **CoDR** (**Co**nfidence-**D**rift **R**emasking), a training-free refinement pass (Figure 1). CoDR compares each token's commit-time confidence with its confidence under later context and treats a confidence drop as evidence that the updated context provides less support. To avoid one forward pass per position, k-partition probing estimates drift for all commitments in only k passes. CoDR then remasks the selected positions and invokes the original sampler to refill them under denser bidirectional context, while leaving all other tokens fixed. It therefore acts as a sampler-agnostic plug-in for decoders that expose commit-time probabilities and support masked refill. Across LLaDA-8B (Nie et al., 2025) and Dream-7B (Ye et al., 2025), multiple samplers, and reasoning and coding tasks, CoDR improves the accuracy–compute trade-off in most settings. Controlled comparisons attribute these gains to targeted confidence-drift revision rather than random remasking or extra decoding steps alone.

Our contributions are summarized as follows:

- We identify confidence drift as a temporal symptom of decoding instability, where committed tokens lose model support as context grows.
- We propose CoDR, a training-free plug-in that efficiently detects and revises drifted commitments without modifying the base model or sampler.
- We show across models, tasks, and samplers that targeted revision improves the accuracy–compute trade-off over unguided extra computation.

2 Related Work

Masked Diffusion Language Models. Masked diffusion language modeling grew out of discrete diffusion. D3PM (Austin et al., 2021a) introduced discrete-state diffusion and showed that an absorbing "mask" state is particularly natural for language; subsequent work such as SEDD (Lou et al., 2024), MDLM (Sahoo et al., 2024), and RADD (Ou et al., 2024) developed increasingly effective objectives and parameterizations for discrete and masked diffusion language modeling. Scaling these ideas produced billion-parameter models competitive with autoregressive LLMs. LLaDA (Nie et al., 2025) demonstrated large-scale pretraining from scratch, while DiffuLLaMA (Gong et al., 2025a), Dream (Ye et al., 2025), DiffuCoder (Gong et al., 2025b), and LLaDA 2.0 (Bie et al., 2025) explored adaptation or initialization from pretrained autoregressive models. The paradigm has further expanded to block-wise generation (Arriola et al., 2025) and multimodal modeling (You et al., 2025; Yang et al., 2025). Recent work has also explored supervised, preference-based, parameter-efficient, and reinforcement-learning-based post-training of diffusion language models (Zhao et al., 2025; Zhu et al., 2025; Wang et al., 2026b), whereas CoDR improves generation without updating model parameters.

Decoding Samplers and Remasking. How a trained MDLM is decoded is a second, largely separate question. Existing samplers use model-internal signals such as entropy or margin (Koh et al., 2025; Kim et al., 2025a), confidence thresholds (Wu et al., 2025), entropy bounds (Ben-Hamu et al., 2026), denoising history (Wu and Huang, 2026; Wang et al., 2025), KL criteria (Kim et al., 2025b), or attention patterns (Zhou et al., 2026; Kim et al., 2026) to decide which positions to commit and when. These methods improve the commitment rule, but committed tokens are typically kept fixed. Recent remasking methods relax this constraint by allowing committed tokens to be revisited. ReMDM (Wang et al., 2026a) introduces an iterative remasking process, while CORE (Zhai et al., 2026) selects revision targets by testing token robustness under perturbed contexts. Our method also belongs to this reversible-generation family, but it uses a different signal: confidence drift, the confidence drop of the same committed token from its commit-time context to a later decoding context. This directly targets tokens whose support decreases as more surrounding context becomes available.

3 Method

We introduce **CoDR** (**Co**nfidence-**D**rift **R**emasking), a training-free refinement method that detects drifted commitments by comparing

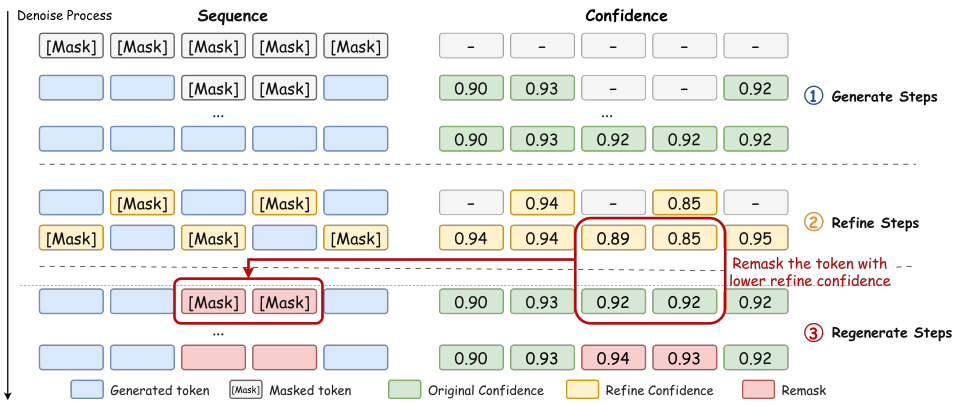


Figure 2: Overview of our method CoDR. The base sampler first fills a block, then the refine pass re-scores committed tokens under richer context, remasks positions whose confidence drifts downward, and refills only those positions.

commit-time and later-context confidence. CoDR estimates this signal with k-partition probing, then selectively remasks and refills drifted tokens with the original base sampler (Figure 2).

3.1 Preliminaries: MDLM Decoding

Let $\mathcal{V}$ be a vocabulary and $[\mathtt{M}] \notin \mathcal{V}$ a special mask token. We decode a target sequence of length L, denoted as $x = (x^1, \ldots, x^L)$ where each $x^i \in \mathcal{V} \cup \{[\mathtt{M}]\}$, conditioned on a prompt (omitted in notation for brevity). The MDLM is parameterized by a denoising network f_θ that maps a partially masked sequence to a logit tensor $f_\theta(x) \in \mathbb{R}^{L \times |\mathcal{V}|}$ in a single forward pass. Its per-position predictive distribution over the vocabulary is given by:

$$p_\theta^i(\cdot \mid x) = \mathrm{Softmax}\big([f_\theta(x)]_i\big) \in \Delta(\mathcal{V}). \quad (1)$$

Decoding proceeds over discrete steps $t = 1, \ldots, T$. Let $C_t \subseteq \{1, \ldots, L\}$ denote the set of committed indices and $M_t = \{1, \ldots, L\} \setminus C_t$ the set of masked indices after step t. Initially, $C_0 = \emptyset$ and $M_0 = \{1, \ldots, L\}$, with all positions in M_0 holding $[\mathtt{M}]$.

At step t, a sampler selects a subset of masked positions $S_t \subseteq M_{t-1}$ to commit. Each token is drawn from its per-position marginal distribution, conditioned on the current partial sequence x_{t-1}:

$$\hat{x}_t^i \sim p_\theta^i(\cdot \mid x_{t-1}), \quad \forall i \in S_t. \quad (2)$$

The sequence is then updated position-wise as

$$x_t^i = \begin{cases} \hat{x}_t^i, & i \in S_t \quad \text{(unmask)}, \\ [\mathtt{M}], & i \in M_{t-1} \setminus S_t \quad \text{(stay masked)}, \\ x_{t-1}^i, & i \notin M_{t-1} \quad \text{(already committed)}. \end{cases} \quad (3)$$

The sets are then updated as $C_t = C_{t-1} \cup S_t$ and $M_t = M_{t-1} \setminus S_t$. Committing a group in this manner treats the positions in S_t as conditionally independent given x_{t-1}.

Crucially, when position i is committed at step t (i.e., $i \in S_t$), we record its commit-time confidence:

$$\psi_i^{\text{old}} = p_\theta^i\big(\hat{x}_t^i \mid x_{t-1}\big). \quad (4)$$

This scalar evaluates the marginal probability the network assigned to the chosen token $\hat{x}_t^i$ under the sparse, localized context x_{t-1} available at that exact step. This single recorded scalar per position is the only interface our refinement procedure requires from the underlying base sampler.

3.2 Confidence-Drift Detection via k-Partition Probing

Because each position i is committed at some step t while many positions in M_{t-1} remain masked, x^i is decided under a sparser context than it will eventually reside in. Furthermore, within a co-committed group S_t, positions are sampled independently of one another. The optimal token at i can therefore change once more context is filled in at a later time $t' > t$. We define this discrepancy as *confidence drift*, and detect it by re-scoring the model's confidence in the initially chosen token under the denser, later context.

Let $\tilde{x}^{[i \to [\mathtt{M}]]}$ denote the current sequence at time t' with position i artificially re-masked. The re-scored confidence is the predictive probability of the same originally committed token x^i under this richer context, and the drift is its confidence drop

from the moment of commitment:

$$\psi_i^{\text{new}} = p_\theta^i\big(x^i \mid \tilde{x}^{[i\to\texttt{[M]}]}\big), \qquad \delta_i = \psi_i^{\text{old}} - \psi_i^{\text{new}}. \tag{5}$$

Both confidences evaluate the probability of the same token x^i via Equation 1. The only difference is the conditioning context: ψ_i^{old} is computed under the sparse context available at commitment time, whereas ψ_i^{new} is computed under the denser context available at the current step. A large positive δ_i indicates that x^i was initially committed with high confidence, but is no longer strongly supported once its surrounding tokens become visible. We treat this confidence drop as the primary signal of a drifted commitment.

Because the logarithm is strictly increasing, the condition $\delta_i > 0$ is equivalent to $\log \psi_i^{\text{old}} - \log \psi_i^{\text{new}} > 0$. CoDR therefore revisits a commitment precisely when it loses model support under the later, denser context. This does not imply that the token is necessarily incorrect or that confidence drift is a universally optimal revision rule; it is a temporal instability signal whose practical informativeness is supported by the trigger comparisons in Table 2 and the revision diagnostics in Section 4.4.

Evaluating ψ_i^{new} exactly requires one forward pass per committed position, resulting in $|C|$ passes per invocation. This overhead is prohibitively expensive for generation. Instead, we approximate ψ_i^{new} for all committed positions simultaneously in just k passes. We partition the committed set C into k residue classes by stride k, defining $C^{(r)} = \{i \in C : i \equiv r \pmod k\}$ for $r = 0, \ldots, k-1$. For each class, we mask the positions as a whole:

$$\tilde{x}^{j,(r)} = \begin{cases} \texttt{[M]} & j \in C^{(r)}, \\ x^j & \text{otherwise.} \end{cases} \tag{6}$$

A single forward pass on $\tilde{x}^{(r)}$ yields $\psi_i^{\text{new}} \leftarrow p_\theta^i(x^i \mid \tilde{x}^{(r)})$ for every $i \in C^{(r)}$. Thus, k passes assign a δ_i to all committed positions. This probe is exact when $k = |C|$ (where each class is a singleton); for smaller k, each i is re-scored while its same-class neighbors are also masked. The stride-k spacing distributes masked positions evenly across the sequence, helping preserve regular local context around each probed position. We use $k = 2$ by default, separating even and odd indices. We discuss the importance of confidence-drift and k-partition probing in Experiment 4.3 and its ablation study in Experiment 4.6.

Algorithm 1 CoDR decoding with confidence-drift remasking

Require: MDLM f_θ, sequence length L, block size B, drift threshold τ_{drift}, partition count k

$x \leftarrow [\texttt{M}]^L$; $\quad C \leftarrow \emptyset$; $\quad \mathcal{R} \leftarrow \emptyset$
for $b = 1$ **to** $\lceil L/B \rceil$ **do**
  $\mathcal{B}_b \leftarrow \{i : (b-1)B < i \le \min(bB, L)\}$
  $(x, C, \psi^{\text{old}}) \leftarrow \text{Sampler}(\mathcal{B}_b)$ ▷ decode block $\mathcal{B}_b$
  for $r = 0$ **to** $k-1$ **do**
    $C^{(r)} \leftarrow \{i \in C : i \equiv r \pmod k\}$
    $\tilde{x} \leftarrow x$; $\quad \tilde{x}^i \leftarrow [\texttt{M}]\ \forall i \in C^{(r)}$
    $\psi_i^{\text{new}} \leftarrow p_\theta^i(x^i \mid \tilde{x}) \quad \forall i \in C^{(r)}$
  end for
  $\mathcal{R} \leftarrow \{i \in C : \psi_i^{\text{old}} - \psi_i^{\text{new}} > \tau_{\text{drift}}\}$
  if $\mathcal{R} \neq \emptyset$ **then**
    $x^i \leftarrow [\texttt{M}] \quad \forall i \in \mathcal{R}$; $\quad C \leftarrow C \setminus \mathcal{R}$
    $(x, C, \psi^{\text{old}}) \leftarrow \text{Sampler}(\mathcal{R})$ ▷ fill the re-masked positions
  end if
end for
return x

3.3 Confidence-Drift Remasking and Refill

Given the confidence-drift scores $\{\delta_i\}_{i\in C}$, CoDR intervenes only on commitments whose support has decreased beyond a tolerance threshold τ_{drift}. The remask set is defined as

$$\mathcal{R} = \{\, i \in C \mid \delta_i > \tau_{\text{drift}} \,\}. \tag{7}$$

If no drifted position is detected, i.e., $\mathcal{R} = \emptyset$, CoDR keeps the current sequence unchanged. Otherwise, it applies a remask–refill step. Each position in $\mathcal{R}$ is reset to $[\texttt{M}]$ and removed from the committed set, after which the base sampler is invoked again to fill these re-masked positions. Since later tokens have already been generated, this refill step conditions on a denser bidirectional context than the original commitment. Once the positions are regenerated, the base sampler also refreshes their commit-time confidences ψ_i^{old}, so that future refinement steps compare against the latest commitment.

Algorithm 1 instantiates this procedure as a post-block method with refinement after every decoded block. At each refinement point, CoDR probes all currently committed positions, including those from previous blocks, and selectively refills the positions in $\mathcal{R}$ whose confidence has drifted. The refinement frequency can be reduced without changing the probe–remask–refill logic; we compare

Table 1: Comprehensive benchmark results over four tasks. **Acc. (%)** is accuracy, and **NFE** is the average number of forward evaluations. Parenthesized annotations indicate absolute accuracy-point changes brought by refinement. **Average** reports the unweighted mean Acc. and NFE across the four tasks.

Model	Sampler	HumanEval		MBPP		MATH-500		GSM8K		**Average**	
		Acc. (%)	NFE	Acc. (%)	NFE	Acc. (%)	NFE	Acc. (%)	NFE	**Acc. (%)**	NFE
LLaDA	top1	41.46	256	39.20	256	38.60	256	72.10	256	47.84	256
	+CoDR	42.68 (↑1.22)	304	**40.60** (↑1.40)	312	38.40	314	73.09 (↑0.99)	304	48.69 (↑0.85)	309
	fd (τ=0.9)	42.68	79	39.00	84	36.80	105	73.01	74	47.87	86
	+CoDR	**43.90** (↑1.22)	113	40.40 (↑1.40)	125	37.20 (↑0.40)	145	**74.22** (↑1.21)	110	**48.93** (↑1.06)	123
	eb (γ=0.2)	40.24	80	38.80	82	37.80	101	71.27	73	47.03	84
	+CoDR	41.46 (↑1.22)	114	40.20 (↑1.40)	125	**39.80** (↑2.00)	143	72.40 (↑1.13)	109	48.47 (↑1.44)	123
	ReMDM	41.46	512	40.00	512	39.00	512	73.31	512	48.44	512
	CORE	38.41	272	39.40	272	37.40	272	72.25	272	46.87	272
Dream	top1	57.32	256	54.40	256	46.60	256	80.89	256	59.80	256
	+CoDR	**59.15** (↑1.83)	317	54.60 (↑0.20)	339	47.60 (↑1.00)	320	81.12 (↑0.23)	321	**60.62** (↑0.82)	324
	fd (τ=0.9)	54.88	78	54.20	105	47.20	116	80.06	64	59.09	91
	+CoDR	57.32 (↑2.44)	121	54.40 (↑0.20)	158	**48.20** (↑1.00)	158	80.89 (↑0.83)	101	60.20 (↑1.11)	135
	eb (γ=0.2)	56.71	77	**54.80**	100	47.80	111	79.98	66	59.82	89
	+CoDR	**59.15** (↑2.44)	124	**54.80** (+0.00)	159	47.20	155	81.27 (↑1.29)	109	60.61 (↑0.79)	137
	ReMDM	57.93	512	54.40	512	47.00	512	**81.73**	512	60.27	512
	CORE	57.32	272	53.80	272	45.00	272	79.61	272	58.93	272

every-block, periodic, final-only, and mid-block schedules in Table 4.

4 Experiments

Our experiments are designed to answer five research questions: **RQ1** Does CoDR improve performance across models, tasks, and base samplers? **RQ2** Do the gains come from CoDR's core mechanisms, namely drift-guided triggering and structured partition probing? **RQ3** What does confidence drift capture during decoding, and how is its magnitude related to token revision and successful correction? **RQ4** How effectively and practically does CoDR use additional inference compute? **RQ5** How robust is CoDR across hyperparameters and generation lengths?

4.1 Experimental Setup

We evaluate two open-weight MDLMs, LLaDA-8B-Instruct and Dream-v0-Instruct-7B, on four zero-shot benchmarks: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021b), MATH-500 (Hendrycks et al., 2021) , and GSM8K (Cobbe et al., 2021), use Pass@1 metric. For ablation study, we select a subset GSM8K with 500 samples. All runs use bf16 precision, generation length $L_{\text{gen}} = 256$, block length 32, and 0-shot prompting. Unless otherwise stated, the main accuracy experiments do not use a KV cache; Table 5 separately evaluates latency and memory with and without KV caching for the initial Fast-dLLM draft. Decoding uses argmax at temperature 0 and is deterministic; reruns with different seeds produce identical predictions. All experiments are performed on NVIDIA A100 GPUs.

We evaluate Fast-dLLM (`fd`, $\tau = 0.9$), EB-Sampler (`eb`, $\gamma = 0.2$), and `top1` as standalone generation baselines. Unless noted, "+CoDR" uses Algorithm 1 with $k = 2$, $\tau_{\text{drift}} = 0$, parity split. In the refill stage, `fd+CoDR` uses Fast-dLLM with $\tau = 0.9$, `eb+CoDR` uses EB-Sampler with $\gamma = 0.1$, and `top1+CoDR` uses EB-Sampler with $\gamma = 0.01$(a detailed analysis in Appendix C.3). We additionally compare with ReMDM, following the setting used with LLaDA in the original paper, each block run 32 ReMDM-loop steps, and CORE (detailed setting for CORE is shown in Appendix C.4). NFE (number of function evaluations) counts all decoding, probe, remasking, and refill forward passes.

4.2 Main Results (RQ1)

Table 1 shows that CoDR improves commitment-based decoding across both models and all three base samplers. On LLaDA-8B, adding CoDR improves the average accuracy of `top1`, `fd`, and `eb` by +0.85, +1.06, and +1.44 percentage points, respectively. On Dream-7B, the corresponding gains are +0.82, +1.11, and +0.79 percentage points. These consistent improvements indicate that CoDR is not specific to a particular decoding rule, but can be applied as a general refinement layer for

Table 2: Trigger ablations using `fd_refine` on LLaDA-8B (0-shot) for HumanEval and GSM8K. The `bottom-`m baselines remask the m committed tokens with the lowest current confidence ψ^{new}, without comparing against their commit-time confidence ψ^{old}.

	HumanEval				GSM8K			
Trigger	Acc. (%)	NFE	Rem.	Edit (%)	Acc. (%)	NFE	Rem.	Edit (%)
`fd` baseline	42.68	79	0.0	—	76.00	74	0.0	—
random $p = 0.045$	42.68 (+0.00)	104	50.9	0.21	76.00 (+0.00)	99	51.3	0.28
random $p = 0.20$	42.68 (+0.00)	108	227.6	0.27	75.80 (↓0.20)	103	229.7	0.27
bottom $m = 6$	42.68 (+0.00)	120.5	48.0	4.83	76.80 (↑0.80)	120.2	48.0	5.43
bottom $m = 8$	43.29 (↑0.61)	123.8	64.0	3.88	77.00 (↑1.00)	123.5	64.0	4.31
`top1_change`	43.29 (↑0.61)	99	3.9	36.02	76.80 (↑0.80)	94	3.8	32.05
drift (`conf_drift`)	**43.90** (↑1.22)	113	51.6	3.72	**77.40** (↑1.40)	110	57.2	3.20

Table 3: Structured stride vs. random partition using `fd_refine` on LLaDA-8B (0-shot) across probe strides k. Random partition results are reported as mean±std over 5 seeds.

		HumanEval				GSM8K			
k	Partition	Acc. (%)	NFE	Rem.	Edit (%)	Acc. (%)	NFE	Rem.	Edit (%)
2	Stride	**43.90**	113	51.6	3.72	**77.40**	110	57.2	3.18
	Random	43.53±0.6 (↓0.37)	119	83.4	3.30	76.40±0.9 (↓1.00)	115	99.4	2.74
3	Stride	43.29	120	37.6	4.39	76.80	116	38.5	4.60
	Random	43.29±0.7 (+0.00)	122	53.7	3.54	76.80±0.6 (+0.00)	119	59.6	3.54
4	Stride	43.29	127	32.1	4.95	77.20	123	36.1	4.54
	Random	43.17±0.2 (↓0.12)	129	45.0	4.02	76.60±0.6 (↓0.60)	125	48.0	3.90

different commitment-based samplers.

CoDR further improves the accuracy–compute trade-off. On LLaDA-8B, `fd+CoDR` achieves the best average accuracy of 48.93% with 123 NFE, outperforming ReMDM's 48.44% average accuracy while using far fewer forward evaluations than its 512 NFE setting. On Dream-7B, `top1+CoDR` obtains the best average accuracy of 60.62% with 324 NFE, and `fd+CoDR` reaches 60.20% with only 135 NFE. This suggests that CoDR uses additional computation in a targeted way by revisiting potentially unstable commitments, rather than uniformly increasing denoising iterations.

Because decoding is deterministic, uncertainty arises from the finite test sets rather than sampling variation. A 10,000-resample paired bootstrap over LLaDA predictions finds significant four-task average gains for `top1`, Fast-dLLM, and EB-Sampler; all 95% confidence intervals exclude zero. CoDR also improves both samplers on two additional benchmarks, BBH (Suzgun et al., 2023) and GPQA (Rein et al., 2024). Full tests and results are reported in Appendix B.

4.3 Effectiveness of Core Mechanisms (RQ2)

We examine whether CoDR's gains come from (i) confidence drift and (ii) structured dense-context probing.

Baselines. For triggering, we compare random remasking, `bottom-`m selection using only the lowest current confidence ψ^{new}, `top1_change`, and our temporal `conf_drift` signal. The bottom-m settings use $m = 6$, $m = 8$ and the random remasking use $p = 0.045$, $p = 0.20$ to approximately match CoDR's remasking budget. For probing, we compare deterministic stride and random partitions at the same k. We report accuracy, NFE, mean remasked positions (**Rem.**), and the percentage of remasked tokens that change after refill (**Edit**).

(i) Confidence Drift. Table 2 shows that random remasking adds compute without improving accuracy, while `bottom-`m selection remains less accurate and more expensive than CoDR at comparable remasking budgets. The `top1_change` trigger is precise but overly conservative, producing very few remasks. In contrast, `conf_drift` achieves the best accuracy on both datasets, showing that temporal comparison provides information beyond current confidence alone.

(ii) Structured dense-context probing. Table 3 shows that structured stride probing preserves more regular local context than random partitions. At

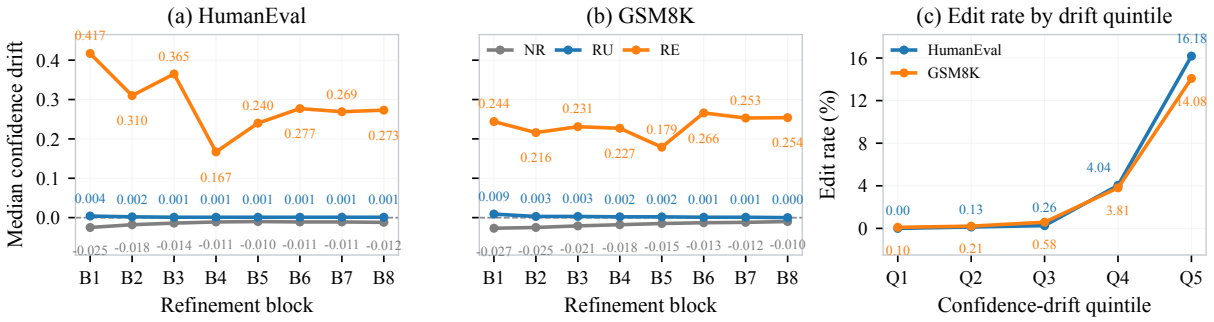


Figure 3: Confidence-drift diagnostics. Panels (a)–(b) show median drift at each refinement boundary for tokens that are not remasked (NR), remasked but unchanged (RU), or remasked and edited (RE). Panel (c) reports the edit rate among positive-drift events grouped into equal-frequency drift quintiles.

k=2, random partitioning increases remasking from 51.6 to 83.4 on HumanEval and from 57.2 to 99.4 on GSM8K, while reducing accuracy from 43.90% to 43.53% and from 77.40% to 76.40%. We therefore use deterministic stride probing by default.

4.4 What Confidence Drift Captures (RQ3)

We analyze default CoDR with LLaDA-8B/Fast-dLLM on HumanEval and GSM8K (L_{gen}=256, block size 32, k=2, and τ_{drift}=0). A token-event is one committed position observed at one refinement boundary.

Dynamic evolution. At the end of each block B1–B8, CoDR probes every position committed at that boundary and computes $\delta_i = \psi_i^{\text{old}} - \psi_i^{\text{new}}$. We then classify each token-event by its outcome in that refinement round: **NR** was probed but not remasked (under $\tau_{\text{drift}} = 0$, typically $\delta_i \leq 0$); **RU** had $\delta_i > 0$ and was remasked, but refill reproduced the same token; and **RE** had $\delta_i > 0$ and refill produced a different token. Figure 3(a–b) reports the median δ of each group separately at every block. RE maintains substantially larger drift than the other groups from B1 to B8 on both tasks, showing that drift tracks evolving contextual instability rather than a calibrated error probability.

Drift magnitude and token revision. For each dataset, we collect all token-events with positive drift ($\delta > 0$), which are selected for remasking under the default $\tau_{\text{drift}} = 0$. We sort these events by δ and divide them into five equal-frequency bins, from the lowest-drift quintile Q1 to the highest-drift quintile Q5. Within each bin, the edit rate is the fraction of events whose refill produces a token different from the original commitment. Figure 3(c) shows that this rate rises sharply with drift, reaching 16.18% on HumanEval and 14.08% on GSM8K in Q5. Thus, drift magnitude identifies commitments that the model is more likely to overturn during refill. Full numerical results are reported in Appendix C.

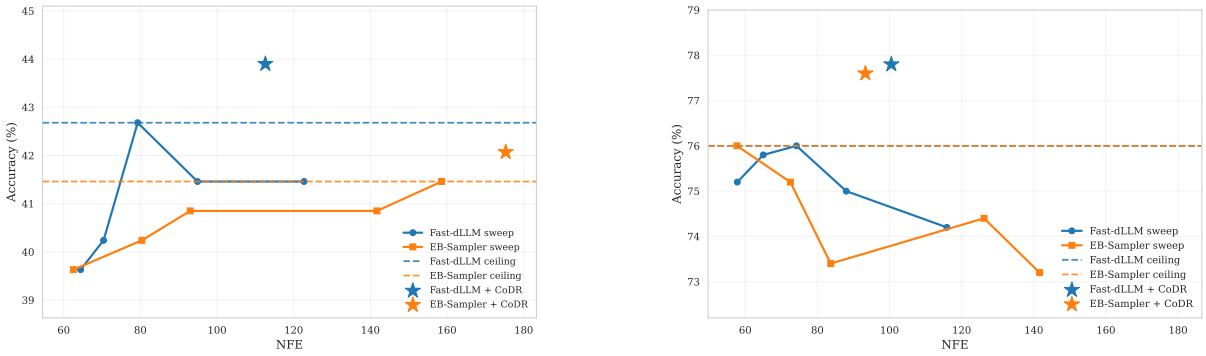
Table 4: Refinement schedules on LLaDA-8B/FD.

	HumanEval		GSM8K	
Schedule	Acc.	NFE	Acc.	NFE
None	42.68	79.0	76.00	74.0
Every 1	43.90	112.3	77.40	110.0
Every 2	**43.90**	**97.1**	**77.60**	**93.2**
Every 4	43.29	89.2	77.00	84.4
Final	43.29	85.3	76.20	79.7
Mid-block	42.68	142.3	76.40	141.8

Relation to successful correction. At the output level, Wrong→Correct examples have the largest maximum and edited-token drift on GSM8K (0.734/0.432); HumanEval shows the same direction (0.639/0.544), although only two examples make this transition. Full correctness-transition statistics are reported in Appendix C.

4.5 Practical Compute Efficiency (RQ4)

Compute allocation. Figure 4 tests whether the base sampler can recover CoDR's gains by spending a similar NFE budget. Sweeping the Fast-dLLM threshold τ and EB-Sampler bound γ reveals a clear scaling ceiling: additional base-sampler evaluations do not improve the best HumanEval or GSM8K accuracy. CoDR instead raises Fast-dLLM from 42.68% to 43.90% on HumanEval and from 76.00% to 77.80% on GSM8K, with the same trend for EB-Sampler. Thus, its gains come from targeted revision rather than additional NFE alone.



(a) HumanEval (b) GSM8K

Figure 4: Compute allocation on LLaDA-8B (0-shot). Solid curves sweep the base sampler's compute knob, using τ for Fast-dLLM and γ for EB-Sampler. Dashed lines mark the best swept baseline accuracy, and stars mark the best CoDR result. CoDR achieves higher accuracy than the swept baselines on both HumanEval and GSM8K, suggesting more effective use of additional NFE.

Table 5: HumanEval latency and memory on LLaDA-8B/FD (L_{gen}=256, batch size 1). En: refine every n blocks; F: final only.

Cache	Method	Acc.	NFE	Sec.	MiB
No KV	FD	42.68	79.09	4.40	17,512
	CoDR-E1	43.90	112.27	6.36	17,512
	CoDR-E2	**43.90**	97.02	**5.55**	17,512
	CoDR-E4	43.29	89.23	5.10	17,512
	CoDR-F	43.29	85.32	4.88	17,512
KV draft	FD	43.90	82.82	2.77	18,322
	CoDR-E1	45.12	119.02	4.79	18,222
	CoDR-E2	**45.12**	102.20	**3.90**	18,222
	CoDR-E4	43.29	94.41	3.94	18,182
	CoDR-F	43.90	89.01	3.18	18,182

Refinement schedule. Table 4 shows that every-block refinement is not required. Refining every two blocks preserves HumanEval accuracy, slightly improves GSM8K, and reduces NFE by 14–15%. Less frequent schedules trade accuracy for further savings, whereas mid-block refinement exceeds 140 NFE without a meaningful gain. We retain every block as the simple default and recommend every two blocks when efficiency matters.

Latency, memory, and KV caching. Table 5 confirms that NFE reflects wall-clock cost. Every-two-block CoDR improves accuracy by 1.22 points while increasing latency from 4.40 to 5.55 seconds without KV caching, and from 2.77 to 3.90 seconds with KV drafting. This schedule is faster than every-block refinement in both settings and does not increase peak memory. KV drafting currently accelerates only initial sampling because remasking invalidates cached states for probing and refill.

4.6 Ablation Study (RQ5)

Effect of probe stride k. The stride k controls the number of partition groups used for probing. Larger k preserves denser context in each probe but requires more forward evaluations. Figure 5(a), with detailed results in Table 14, shows that $k = 2$ gives the best accuracy–NFE trade-off, while larger values add compute without clear accuracy gains. We therefore use $k = 2$ as the default.

Effect of drift threshold. The drift threshold τ_{drift} controls how conservative the remasking trigger is. The default $\tau_{\text{drift}} = 0$ remasks any token whose confidence drops under the probe context, while positive thresholds filter out small-drift tokens. Figure 5(b), with detailed results in Table 15, shows that $\tau_{\text{drift}} = 0$ achieves the best accuracy on both HumanEval and GSM8K. Positive thresholds reduce NFE but do not improve accuracy, so we keep $\tau_{\text{drift}} = 0$ as the default.

Effect of refinement passes. We ablate the number of remask–refill passes at each block boundary. Figure 5(c), with detailed results in Table 16, shows that one pass is generally sufficient. Additional passes substantially increase NFE; they leave HumanEval accuracy unchanged and bring only a small gain on GSM8K with two passes. This suggests that the first pass captures most useful drift signals, so we use one refinement pass by default.

Effect of block length. The block length b determines how often refinement is invoked. Smaller blocks create more refinement opportunities, while larger blocks reduce intervention frequency and NFE. Figure 5(d), with detailed results in Table 17,

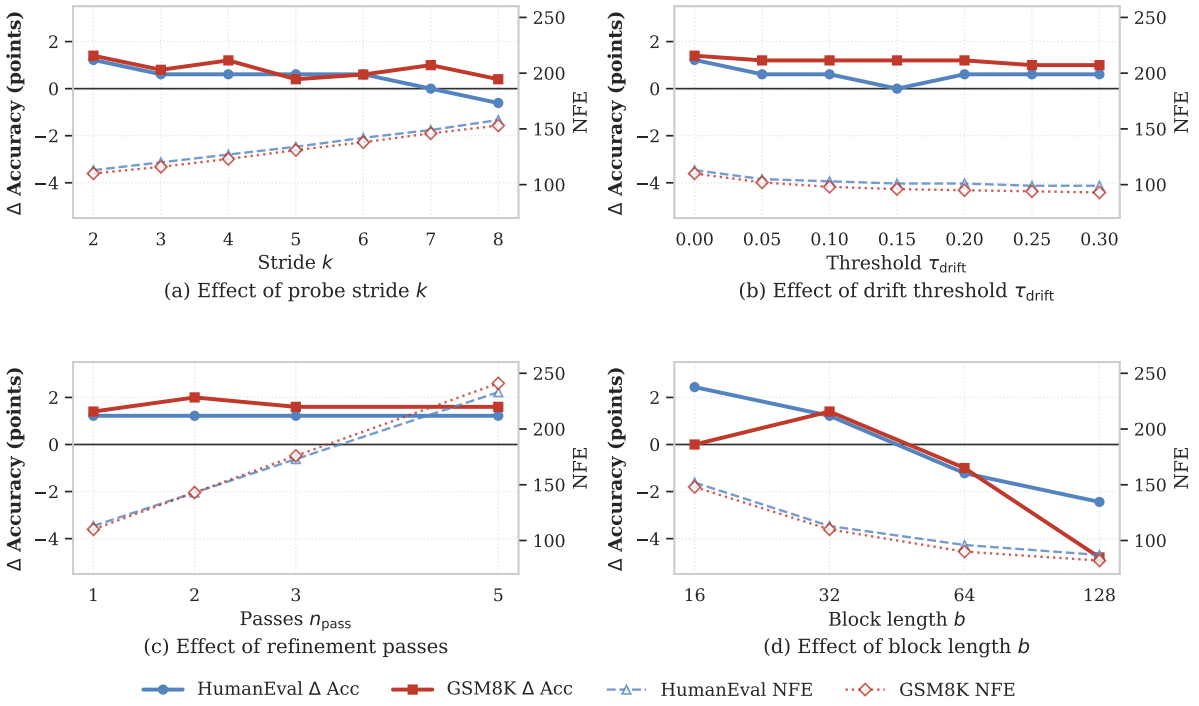


Figure 5: ablation study for our method.

shows that larger blocks degrade the base sampler, suggesting that sparse refinement boundaries allow early errors to accumulate. CoDR improves over the corresponding baseline across most block lengths. We use $b = 32$ as the default because it gives the best GSM8K result and a strong HumanEval result at lower NFE than $b = 16$, consistent with recent work (Wu et al., 2025).

Robustness across generation lengths. Figure 6 compares lengths 128, 256, and 512 across all four tasks. CoDR improves average accuracy for both samplers at every length, with gains of 1.90 points for Fast-dLLM and 0.90 for EB-Sampler at length 512. At length 128, gains are smaller and MATH-500 regresses, showing robust average improvements but not uniform per-task gains. NFE overhead increases with sequence length. Appendix Table 13 provides full results.

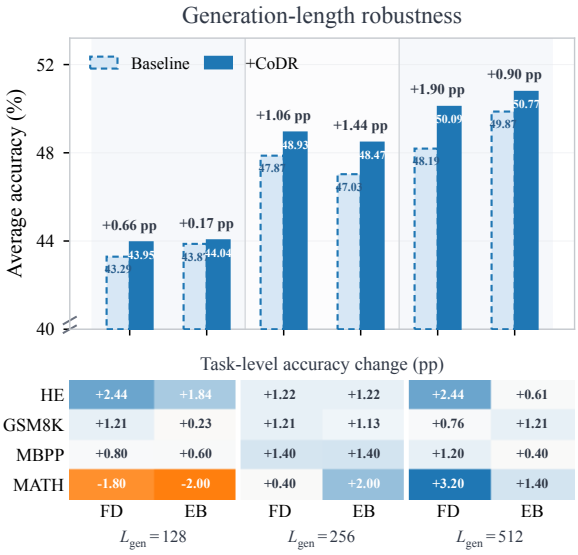


Figure 6: **Generation-length robustness on LLaDA-8B.** Four-task average accuracy of the baseline and CoDR under Fast-dLLM (FD) and EB sampling across generation lengths. The heatmap reports task-level accuracy gains (pp) from adding CoDR.

5 Conclusion

We introduce CoDR, a training-free method that revisits MDLM tokens whose confidence decreases as context grows. Its k-partition probe checks all commitments in a few forward passes, then selectively refills unstable tokens under richer context. Across two backbones, three samplers, and math and code tasks, CoDR improves the accuracy–compute trade-off. Diagnostics associate larger drift with token edits and successful corrections, while controlled comparisons distinguish targeted revision from extra decoding alone. These results show that reversible decoding can correct early commitments without retraining or replacing the base sampler. Future work may explore adaptive probing and cache-aware refinement.

Limitations

CoDR is an inference-time refinement method whose effectiveness depends on the underlying base sampler used for refill. The confidence trigger identifies remasked positions whose support decreases under denser context, but the quality of revised tokens remains bounded by the same MDLM and sampler used for generation. As a result, occasional regressions may occur when refill introduces fresh local errors, especially in arithmetic-heavy cases. In addition, confidence drift is designed as a targeted revision signal rather than a complete error detector. CoDR intentionally revises only tokens whose support decreases under denser context, leaving other tokens unchanged to avoid excessive editing. Future work could explore adaptive refinement schedules, stronger refill constraints, and broader evaluation on larger models, multilingual tasks, and KV-cache optimized decoding.

Acknowledgments

The authors acknowledge support from The Hong Kong Polytechnic University (Research Grant P0046811), the National Natural Science Foundation of China under Grant No. 62136005, and the Shenzhen Fundamental Research Program under Grant No. JCYJ20250604144724032.

References

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021a. Structured denoising diffusion models in discrete state-spaces. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA. Curran Associates Inc.

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

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. 2026. Accelerated sampling from masked diffusion models via entropy bounded unmasking. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, and 12 others. 2025. Llada2.0: Scaling up diffusion language models to 100b. *Preprint*, arXiv:2512.15745.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, Hao Peng, and Lingpeng Kong. 2025a. Scaling diffusion language models via adaptation from autoregressive models. In *The Thirteenth International Conference on Learning Representations*.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. 2025b. Diffucoder: Understanding and improving masked diffusion models for code generation. *Preprint*, arXiv:2506.20639.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. *Preprint*, arXiv:1711.02281.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Bumjun Kim, Dongjae Jeon, Moongyu Jeon, and Albert No. 2026. Dependency-aware parallel decoding via attention for diffusion llms. *Preprint*, arXiv:2603.12996.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham M. Kakade, and Sitan Chen. 2025a. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net.

Seo Hyun Kim, Sunwoo Hong, Hojung Jung, Youngrok Park, and Se-Young Yun. 2025b. Klass: Kl-guided fast inference in masked diffusion models. *Preprint*, arXiv:2511.05664.

Hyukhun Koh, Minha Jhang, Dohyung Kim, Sangmook Lee, and Kyomin Jung. 2025. Conditional [mask] discrete diffusion language model. *Preprint*, arXiv:2411.06438.

Aaron Lou, Chenlin Meng, and Stefano Ermon. 2024. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. Large language diffusion models. *Preprint*, arXiv:2502.09992.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. 2024. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *Preprint*, arXiv:2406.03736.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. Simple and effective masked diffusion language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada. Association for Computational Linguistics.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2026a. Remasking discrete diffusion models with inference-time scaling. *Preprint*, arXiv:2503.00307.

Kangyu Wang, Zhiyun Jiang, Haibo Feng, Weijia Zhao, Lin Liu, Jianguo Li, Zhenzhong Lan, and Weiyao Lin. 2025. Creditdecoding: Accelerating parallel decoding in diffusion large language models with trace credits. *Preprint*, arXiv:2510.06133.

Shuaidi Wang, Zhan Zhuang, HUANG Ruping, and Yu Zhang. 2026b. NaRA: Noise-aware loRA for parameter-efficient fine-tuning of diffusion LLMs. In *Forty-third International Conference on Machine Learning*.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. 2025. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *Preprint*, arXiv:2505.22618.

Yue Wu and Jian Huang. 2026. Stability-weighted decoding for diffusion language models. *Preprint*, arXiv:2604.17068.

Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. 2025. Mmada: Multimodal large diffusion language models. *Preprint*, arXiv:2505.15809.

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

Zebin You, Shen Nie, Xiaolu Zhang, Jun Hu, Jun Zhou, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. 2025. Llada-v: Large language diffusion models with visual instruction tuning. *Preprint*, arXiv:2505.16933.

Kevin Zhai, Sabbir Mollah, Zhenyi Wang, and Mubarak Shah. 2026. Core: Context-robust remasking for diffusion language models. *Preprint*, arXiv:2602.04096.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. 2025. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *Preprint*, arXiv:2504.12216.

Xueyu Zhou, Yangrong Hu, and Jian Huang. 2026. Dos: Dependency-oriented sampler for masked diffusion language models. *Preprint*, arXiv:2603.15340.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *Preprint*, arXiv:2505.19223.

A Detailed Experiment Setting

Unless otherwise noted, all experiments follow the model, decoding, precision, prompting, and hardware settings described in Section 4.1. Sampler-specific implementation details are provided in the later appendix sections.

A.1 Model and Dataset Licenses

All pre-trained models (LLaDA and Dream) and datasets (HumanEval, MBPP, GSM8K, and MATH-500) used in this work are publicly available under open-source licenses, such as MIT or Apache 2.0, and are used strictly for academic research and methodological evaluation, consistent with their intended use.

B Statistical Reliability and Additional Benchmarks

B.1 Paired Bootstrap Analysis

Our argmax decoding at temperature 0 is deterministic: rerunning each setting with different seeds produces identical predictions and accuracies. We therefore quantify finite-test-set uncertainty using paired bootstrap resampling. For each LLaDA sampler, we resample paired baseline/CoDR predictions within each task 10,000 times, compute the accuracy gain per task, and average the four gains equally to match Table 1. Two-sided results are reported in percentage points in Table 6. All confidence intervals exclude zero, indicating a significant average gain for each sampler.

Table 6: Paired bootstrap tests for the LLaDA four-task average (10,000 resamples).

Sampler	Base	CoDR	Δ pp	95% CI	p
`top1`	47.84	48.69	$+0.85$	$[0.12, 1.65]$	0.026
Fast-dLLM	47.87	48.93	$+1.06$	$[0.33, 1.84]$	0.006
EB-Sampler	47.03	48.47	$+1.44$	$[0.63, 2.33]$	< 0.001

B.2 Additional Benchmarks

We further evaluate LLaDA-8B on BBH and GPQA. As shown in Table 7, CoDR improves accuracy over both Fast-dLLM and EB-Sampler on both benchmarks, extending the main results beyond the original four tasks.

Table 7: Additional benchmark results. Each cell reports accuracy (%)/NFE.

Method	BBH	GPQA
FD	50.94/64.48	21.72/138.45
CoDR-FD	**51.51**/87.48	**22.73**/172.90
EB	51.51/79.31	21.21/142.49
CoDR-EB	**51.73**/95.18	**22.22**/176.40

C Confidence Drift and Correctness Transitions

C.1 Token-Level Drift Diagnostics

Tables 8 and 9 provide the numerical results underlying Figure 3: per-block median drift by refinement outcome and edit rates across positive-drift quintiles, respectively.

C.2 Output-Level Correctness Transitions

We pair each initial Fast-dLLM output with its final CoDR output and group examples by correctness transition. For each example, Max δ is the largest token-event drift observed during refinement, and Edited-token δ is the median drift over events whose refill changes the token. Examples without edited events are excluded from the latter statistic. Table 10 reports the group median for each measure.

On GSM8K, Wrong→Correct examples exhibit higher maximum and edited-token drift than the other transition groups. HumanEval follows the same direction, although the Wrong→Correct group contains only two examples. Together with Table 9, these results associate larger drift with both actual token revision and successful output correction.

C.3 Top1 Setting and Analysis

Both variants of our Top1 refinement procedure use top-1 decoding for the initial generation pass; they differ only in how remasked positions are *refilled* during revision. We compare refilling with top-1 decoding against refilling with an entropy-bounded (EB) sampler, holding everything else fixed (top-1 phase-1 generation, identical probe and remask schedule). Top-1 refill commits exactly one token per forward pass, so every remasked span is regenerated one token at a time, adding one forward pass per remasked token and accumulating large number of function evaluations (NFE) across all blocks and revision rounds. The EB sampler can refill multiple tokens in one forward pass and therefore requires far fewer passes.

This change is free in terms of quality. With the phase-1 top-1 generation held fixed, top-1 and EB refill produce *identical* accuracy—pass@1 0.4268 on LLaDA-8B HumanEval and 0.5915 on Dream-7B HumanEval—while EB refill reduces NFE from 377 to 303 on LLaDA and from 393 to 317 on Dream, a $\sim 20\%$ saving in both cases (Table 11). We therefore use top-1 generation with EB refill ($\gamma = 0.1$) as our default: it keeps the high-precision, one-token-at-a-time ordering of top-1 where decision quality matters most—the main generation pass—while avoiding its redundant cost when re-filling remasked spans.

C.4 CoRE Setting and Analysis

We follow the CoRE protocol (Zhai et al., 2026): generation length 256, top-1 decoding (matching the baseline), revision every $E = 8$ steps within the mid-trajectory window $[0.25, 0.75]$, $k = 1$ token revised per step, and a replacement-confidence threshold of 0.3. Unlike the original paper, which decodes the entire response as a *single* block, we adopt *block-wise* decoding (block $= 32$) with candidate-set size $m = 4$ and protect $= 8$ for *both* LLaDA and Dream.

Block-wise decoding outperforms single-block on LLaDA. While the paper's single-block configuration runs without collapse on LLaDA, it is overall inferior to block-wise decoding, particularly on multi-step reasoning. Table 12 compares the two at matched compute (NFE ≈ 272). Single-block is marginally stronger on HumanEval, where the larger global context aids code structure, but block-wise decoding is substantially better on MBPP, MATH-500, and GSM8K, raising the four-task average from 0.394 to 0.469. The gap is most pronounced on reasoning: single-block with $m = 32$ degrades MATH-500 to 0.248 and GSM8K to 0.536, whereas block-wise decoding with $m = 4$ recovers them to 0.374 and 0.723, near the no-revision baseline. We therefore use block-wise decoding for LLaDA as well.

Single-block decoding fails outright on Dream. On Dream the single-block configuration does not merely underperform—it collapses to near-empty generations (with accuracy only 0.189 in humaneval). This is not a CoRE-specific implementation artifact: Dream's own validated samplers (`fastdllm`, `top1`) also collapse under single-block decoding (~ 12 output tokens). The failure is a property of Dream under single-block decoding—given a single fully-masked block, the model places high confidence on an early end-of-sequence token before any context has been committed, terminating generation prematurely. Block-wise decoding (block $= 32$) removes this collapse.

Candidate-set size must scale with block length. Within block-wise decoding, the paper's $m = 32$ is far too large: masking 32 candidates inside a 32-token block erases almost the entire committed context at revision time. The verify pass then scores token stability under a near-empty context, mis-flags already-committed tokens, and derails generation (on Dream HumanEval, pass@1 $0.5732 \rightarrow 0.3598$; the same over-perturbation drives LLaDA's single-block reasoning collapse above). We therefore rescale m by *perturbation ratio* rather than absolute count: the paper's $m = 32$ perturbs roughly 25% of the ~ 128 tokens committed at mid-window revision time, which on a 32-token block (~ 16 tokens filled at that point) corresponds to $m \approx 4$. With block $= 32$ and $m = 4$, both models produce normal-length outputs and recover baseline-level accuracy.

Table 8: Per-block median confidence drift by refinement outcome. NR: not remasked; RU: remasked but unchanged; RE: remasked and edited.

HumanEval								
Group	B1	B2	B3	B4	B5	B6	B7	B8
NR	−0.025	−0.018	−0.014	−0.011	−0.010	−0.011	−0.011	−0.012
RU	0.004	0.002	0.001	0.001	0.001	0.001	0.001	0.001
RE	**0.417**	**0.310**	**0.365**	**0.167**	**0.240**	**0.277**	**0.269**	**0.273**
GSM8K								
Group	B1	B2	B3	B4	B5	B6	B7	B8
NR	−0.027	−0.025	−0.021	−0.018	−0.015	−0.013	−0.012	−0.010
RU	0.009	0.003	0.003	0.002	0.002	0.001	0.001	0.000
RE	**0.244**	**0.216**	**0.231**	**0.227**	**0.179**	**0.266**	**0.253**	**0.254**

Table 9: Edit rate (%) among positive-drift events grouped by equal-frequency drift quintile.

Data	Q1	Q2	Q3	Q4	Q5
HumanEval	0	0.13	0.26	4.04	**16.18**
GSM8K	0.10	0.21	0.58	3.81	**14.08**

Table 10: Confidence drift by output correctness transition.

Data	Transition	n	Max δ	Edited-token δ
GSM8K	Wrong→Correct	13	**0.734**	**0.432**
	Correct→Correct	374	0.404	0.282
	Wrong→Wrong	107	0.527	0.287
	Correct→Wrong	6	0.627	0.376
HumanEval	Wrong→Correct	2	**0.639**	**0.544**
	Correct→Correct	70	0.461	0.314
	Wrong→Wrong	92	0.494	0.311
	Correct→Wrong	0	—	—

Table 11: Refill sampler comparison with top-1 generation held fixed (HumanEval, 0-shot). Top-1 and EB refill give identical accuracy; EB refill cuts NFE by $\sim$20%.

Model	Refill	pass@1	NFE
LLaDA-8B	top-1	0.4268	377
	EB	0.4268	**303**
Dream-7B	top-1	0.5915	393
	EB	0.5915	**317**

Table 12: LLaDA-8B-Instruct CoRE: single-block (m=32) vs. block-wise (m=4), 0-shot, NFE $\approx$ 272. Block-wise decoding wins on three of four tasks and on average, with the largest gains on reasoning.

Setting	HE	MBPP	MATH	GSM8K	Avg.
Single block, m=32	**0.427**	0.364	0.248	0.536	0.394
Block-wise, m=4	0.384	**0.394**	**0.374**	**0.723**	**0.469**

D Comprehensive Ablation Results

This section provides the exact numerical data corresponding to the hyperparameter sensitivity analysis visualized in Section 4.6. While the main text focuses on the overall Pareto-optimal trends and performance trade-offs, the tables below detail the discrete Accuracy and NFE metrics across all evaluated configurations of the CoDR framework.

Table 13 reports robustness across generation lengths. Table 14 reports the isolated effects of probe stride k, drift threshold τ_{drift}(Table 15), and the number of refinement passes n_{pass} (Table 16). Table 17 details the impact of block length b, explicitly comparing the baseline generation against the refined output to illustrate the compounding effect of early errors in sparse commit boundaries. All results are evaluated using the Fast-dLLM base sampler (`fd`, τ=0.9) on LLaDA-8B under a 0-shot setting.

Table 13: Full numerical results for generation-length robustness on LLaDA-8B. Task columns report accuracy (%); averages are unweighted over HumanEval (HE), full GSM8K, MBPP, and MATH-500.

Method	HE	GSM8K	MBPP	MATH	Avg. Acc.	Avg. NFE
$L_{\text{gen}} = 128$						
FD	33.54	69.22	36.60	33.80	43.29	47.50
+CoDR	35.98	70.43	37.40	32.00	**43.95**	64.96
EB	34.14	69.75	38.00	33.60	43.87	55.78
+CoDR	35.98	69.98	38.60	31.60	**44.04**	65.88
$L_{\text{gen}} = 512$						
FD	42.68	75.89	38.00	36.20	48.19	144.29
+CoDR	45.12	76.65	39.20	39.40	**50.09**	227.48
EB	45.73	75.74	39.20	38.80	49.87	143.25
+CoDR	46.34	76.95	39.60	40.20	**50.77**	234.48

Table 14: Ablation results on probe stride k using `fd_refine` with LLaDA-8B. Accuracy is reported in percentage. Bold values indicate optimal configurations.

	HumanEval		GSM8K	
k	Acc (%)	NFE	Acc (%)	NFE
2	**43.90**	113	**77.40**	110
3	43.29	120	76.80	116
4	43.29	127	77.20	123
5	43.29	134	76.40	131
6	43.29	142	76.60	138
7	42.68	149	77.00	146
8	42.07	158	76.40	153

Table 15: Ablation results on drift threshold τ_{drift} using `fd_refine` with LLaDA-8B. Positive thresholds reduce NFE by making the trigger more selective.

	HumanEval		GSM8K	
τ_{drift}	Acc (%)	NFE	Acc (%)	NFE
0.00	**43.90**	113	**77.40**	110
0.05	43.29	105	77.20	102
0.10	43.29	103	77.20	98
0.15	42.68	101	77.20	96
0.20	43.29	101	77.20	95
0.25	43.29	99	77.00	94
0.30	43.29	99	77.00	93

Table 16: Ablation results on refinement passes n_{pass} using `fd_refine` with LLaDA-8B. Additional refinement passes beyond $n = 1$ add substantial compute for little or no gain.

	HumanEval		GSM8K	
n_{pass}	Acc (%)	NFE	Acc (%)	NFE
1	**43.90**	113	77.40	110
2	43.90	143	**78.00**	143
3	43.90	173	77.60	176
5	43.90	233	77.60	241

Table 17: Ablation on block length b using `fd_refine` with LLaDA-8B. Block length controls the intervention frequency and the accuracy–NFE trade-off. Extreme sparse boundaries ($b = 128$) allow early contextual errors to compound, degrading baseline performance and limiting refinement capabilities.

HumanEval				
b	Base A.	Base N.	CoDR A.	CoDR N.
16	**44.51**	87	**45.12**	152
32 **(def.)**	42.68	79	43.90	113
64	41.46	77	41.46	96
128	39.02	76	40.24	87
GSM8K				
b	Base A.	Base N.	CoDR A.	CoDR N.
16	75.20	81	76.00	148
32 **(def.)**	76.00	74	**77.40**	110
64	73.20	71	75.00	90
128	69.60	72	71.20	82

E Detailed Case Studies

To provide a comprehensive analysis of our method, this section presents detailed generation trajectories across mathematics and code generation tasks. We contrast the base Fast-dLLM sampler ($\tau = 0.9$) against our CoDR formulation on LLaDA-8B in setting.

E.1 Success Cases

CoDR excels at identifying and correcting early irrational commitments induced by sparse context, cleanly overwriting them once the bidirectional global context stabilizes.

Code Generation Recovery. Figures 7 and 8 illustrate CoDR's capacity to repair localized syntactic and API-level fractures. In Figure 7 (HumanEval/89), the base sampler commits to a malformed character shift logic due to premature evaluation. CoDR perfectly isolates the confidence drift precisely at this arithmetic expression and recovers the exact `+4` rotation algorithm. Similarly, Figure 8 demonstrates the recovery of a missing dependency; CoDR detects the structural vulnerability of invoking `math.ceil` without an import and seamlessly prepends `import math` during the refinement pass.

Mathematical Reasoning Correction. Figures 9 and 10 highlight CoDR's precision in arithmetic and algebraic contexts. In Figure 9, a pure arithmetic slip ($1764/12 = 142$) derails the entire downstream revenue projection. CoDR flags the fractional setup, unmasks the flawed quotient, and enforces the mathematically sound answer. More impressively, Figure 10 shows CoDR untangling a distorted algebraic substitution (introducing a sign and term error) and re-evaluating the true algebraic path ($P = 50$).

E.2 Regression Cases

While CoDR demonstrates an overwhelmingly positive net performance vector, it is non-monotonic. Tracking its failure modes provides sharp boundaries regarding the limitations of training-free inference-time interventions.

Figures 11 and 12 illustrate the two primary modes of regression: dropped context and fresh hallucinations. In Figure 11, CoDR correctly isolates a cluster of low-confidence stylistic tokens at the block boundary but accidentally drops a vital preceding string operation (`split()`) during the unmask-refill process, reducing list evaluation to character-level iterations. In Figure 12, the base sampler actually derives the exact combinatorial solution. However, CoDR triggers a stylistic layout change, and during generation, the underlying MDLM introduces a new arithmetic corruption ($\binom{14}{5} \rightarrow 1002$).

Analysis Our diagnostic inspection reveals that regressions are fundamentally driven by *spurious variations during the refill pass* rather than systematic failures in drift detection. Because the targeted tokens are fed directly back into the underlying MDLM's standard conditional sampling loop, the refinement step remains statistically bounded by the base model's inherent local calculation failure rates. This explains why code generation tasks exhibit virtually zero regressions (e.g., 0 out of 164 on HumanEval), while arithmetic tasks are more vulnerable to fresh hallucinations.

F AI Ethics Statement

The authors acknowledge the use of LLM assistants only for context refinement and some code refinement during the preparation of this manuscript, and have checked carefully for any potential ethical concerns.

Base Sampler (HumanEval/89) – FAIL

```
def encrypt(s):
    encrypted_string = ''
    for char in s:
        if char.isalpha():
            # BUG: malformed shift under sparse context
            encrypted_string += chr((ord(char) - ord('a') * 2) %
                    26 + ord('a'))
        else:
            encrypted_string += char
    return encrypted_string
```

CoDR Refinement (HumanEval/89) – PASS

```
def encrypt(s):
    encrypted_string = ''
    for char in s:
        if char.islower():
            # FIXED: Correct +4 rotation logic refilled
            encrypted_string += chr((ord(char) - ord('a') + 4) %
                    26 + ord('a'))
        else:
            encrypted_string += char
    return encrypted_string
```

Figure 7: Code Success 1: The base sampler commits to operator-precedence garbage (`ord('a')*2`). CoDR isolates the drift at this precise expression and recovers the exact rotation algorithm under dense context.

Base Sampler (HumanEval/133) – FAIL

```
def sum_squares(lst):
    # BUG: math library invoked without being imported
    rounded_lst = [math.ceil(x) for x in lst]
    squared_lst = [x ** 2 for x in rounded_lst]
    return sum(squared_lst)
```

CoDR Refinement (HumanEval/133) – PASS

```
import math # FIXED: Prepended during refill
def sum_squares(lst):
    rounded_lst = [math.ceil(x) for x in lst]
    squared_lst = [x ** 2 for x in rounded_lst]
    return sum(squared_lst)
```

Figure 8: Code Success 2: Confidence drift captures a structural failure where the base model utilizes a library function but omits the mandatory import header, leading to a runtime `NameError`. CoDR repairs the header seamlessly.

Base Sampler (GSM8K/50) – WRONG

Lloyd's chickens produce 252 eggs per day. There are 7 days in a week, so: $252 \times 7 = 1764$ eggs/week. There are 12 eggs in a dozen, so:

$$\frac{1764 \text{ eggs/week}}{12 \text{ eggs/dozen}} = 142 \text{ dozens/week}$$

(BUG: Arithmetic Slip)

Lloyd sells each dozen for \$2, so:

$$142 \text{ dozens/week} \times 2 \text{ dollars/dozen} = 284 \text{ dollars/week}$$

Therefore, Lloyd makes $\boxed{284}$.

CoDR Refinement (GSM8K/50) – CORRECT

Lloyd's chickens produce 252 eggs per day. There are 7 days in a week, so: $252 \times 7 = 1764$ eggs/week. There are 12 eggs in a dozen, so:

$$\frac{1764 \text{ eggs/week}}{12 \text{ eggs/dozen}} = 147 \text{ dozens/week}$$

(FIXED: Long Division)

Lloyd sells each dozen for \$2, so:

$$147 \text{ dozens/week} \times 2 \text{ dollars/dozen} = 294 \text{ dollars/week}$$

Therefore, Lloyd makes $\boxed{294}$.

Figure 9: Math Success 1: The base model suffers an arithmetic calculation failure ($1764/12 = 142$). CoDR re-scores the fractional setup and enforces the mathematically sound quotient (147), altering the downstream revenue projection.

Base Sampler (GSM8K/931) – WRONG

Equations: (1) $P + P + J = 100$, (2) $P = J + 10$, (3) $P = P + J$. From the third equation, substitute P in the first: $2P + J = 100$.
Now, substitute P from the second equation:

$$2P + (P - 10) = 100$$

(BUG: Distorted Substitution)

$3P - 10 = 100 \implies P = 30$. So, Peter is 30 years old.

CoDR Refinement (GSM8K/931) – CORRECT

Relationships: (1) $P + P + J = 100$, (2) $P = J + 10$, (3) $P = P + J$. Using the third equation, substitute P in the first: $2P + J = 100$.
Now, using the second equation, substitute P in the above equation:

$$2P + 0 + 10 = 100$$

(FIXED: Algebraic Soundness)

$2P = 100 \implies P = 50$. So, Peter is 50 years old.

Figure 10: Math Success 2: The base model maps the constraints accurately but introduces a sign and term error during variable evaluation. CoDR tracks the logical fracture and re-evaluates the true algebraic path ($P = 50$).

Base Sampler (MBPP/117) – PASS

```
def long_words(n, words):
    words = words.split() # Crucial string-to-list setup
    long_words = [word for word in words if len(word) > n]
    return long_words
```

CoDR Refinement (MBPP/117) – FAIL

```
def long_words(n, words):
    # REGRESSION: split() operation dropped during refill
    return [word for word in words if len(word) > n]
```

Figure 11: Code Regression: CoDR correctly isolates a cluster of low-confidence stylistic tokens at the block boundary but accidentally drops a vital preceding token string operation (`split()`) during the unmask-refill process, reducing list evaluation to character-level iterations.

Base Sampler (MATH-500 Doc 330) – PASS

1. Calculate the total number of ways to choose 5 members out of 14:

$$\binom{14}{5} = \frac{14!}{5!(14-5)!} = 2002$$

(CORRECT Combinatorics)

2. Ways from Republicans: $\binom{8}{5} = 56$. 3. Ways from Democrats: $\binom{6}{5} = 6$.
4. Subtract exclusions: $2002 - 56 - 6 = 1940$.

CoDR Refinement (MATH-500 Doc 330) – FAIL

1. Calculate the total number of ways to choose 5 members out of 14:

$$\binom{14}{5} = \frac{14!}{5!(14-5)!} = 1002$$

(REGRESSION: Hallucinated Factorial)

2. Ways from Republicans: $\binom{8}{5} = 56$. 3. Ways from Democrats: $\binom{6}{5} = 6$.
4. Subtract exclusions: $1002 - 56 - 6 = 940$.

Figure 12: Math Regression: A classic instantiation of fresh hallucination during resampling. The base sampler derives the exact combinatorial solution. CoDR triggers a stylistic layout change, but the model introduces a new arithmetic corruption ($\binom{14}{5} \rightarrow 1002$) during generation.