BLOCK REFLECTION: TRAINING-FREE REASONING REPAIR FOR DISCRETE DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Discrete diffusion large language models (dLLMs) generate text by iteratively denoising masked sequences under *bidirectional* context, which gives them a revision mechanism unavailable to autoregressive decoders. Existing block-wise decoding schemes, however, treat each revealed block as a fixed commitment. Once later blocks are drafted, earlier reasoning steps may become inconsistent with the new future context, yet the decoder cannot identify and repair the flawed block without expanding external candidates. We propose *Block Reflection*, a training-free inference procedure that turns this unused revision capability into online self-correction. To decide which committed block to regenerate, we design a dLLM-native instability signal: the drift of a block's conditional token distribution before and after new future context is drafted. *Block Reflection* remasks and regenerates only the most unstable block and keeps the rest of the trajectory fixed, with no external verifier, fine-tuning, or full-trajectory regeneration. Across four math and code benchmarks on 4 different dLLMs, *Block Reflection* consistently improves accuracy, with large gains of 6.2% on MATH-500 and 10.2% on MBPP.

1 INTRODUCTION

Unlike autoregressive (AR) models, which factorize text left to right, discrete diffusion language models (dLLMs) iteratively denoise masked sequences under bidirectional context (Nie et al., 2025; Ye et al., 2025). In principle this interface supports self-reflection: a span that has already been filled in can be masked again and regenerated with both left and right context, without restarting the sequence (Li et al., 2025; Yu et al., 2026). In practice, however, this editability is hard to exploit when full-sequence diffusion is applied directly to long-form generation, where variable-length responses, efficient KV caching, and coherent token dependencies during parallel decoding are all difficult to obtain (Li et al., 2025).

Recent work therefore moves dLLM inference toward block-wise generation. Block Diffusion interpolates between AR and diffusion models by generating autoregressively across blocks while denoising tokens in parallel within each block (Arriola et al., 2025). Block-wise decoding can approximate KV caching, and confidence-aware parallel decoding improves throughput while preserving generation quality (Wu et al., 2025). Blocks have thus become the practical interface through which dLLMs generate long outputs.

For reasoning, however, block-wise generation creates a new tension. A block is large enough to hold a reasoning fragment, such as an intermediate equation or a code statement, yet once a block is committed, standard block-wise decoding moves on without repairing it (Shu et al., 2026). Local reasoning errors can therefore accumulate across committed blocks (Huang et al., 2026): an incorrect intermediate block keeps conditioning later generations, much like an erroneous prefix in an AR model, even though the dLLM could remask and regenerate it.

Existing self-correction and test-time scaling methods, whether trained or training-free, do not directly address this repair problem. Training-based methods teach dLLMs to insert corrective refinement steps or to align any-order generation, which requires additional correction-oriented training or remask-aware objectives (Schiff et al., 2026; Jeung et al., 2026; Bie et al., 2026). Among training-free methods, dVoting regenerates uncertain tokens through cross-sample voting (Feng et al., 2026), HEX ensembles complete denoising trajectories by treating different block sizes as different experts (Lee et al., 2026), and Prism allocates compute via hierarchical search guided by prompted

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.

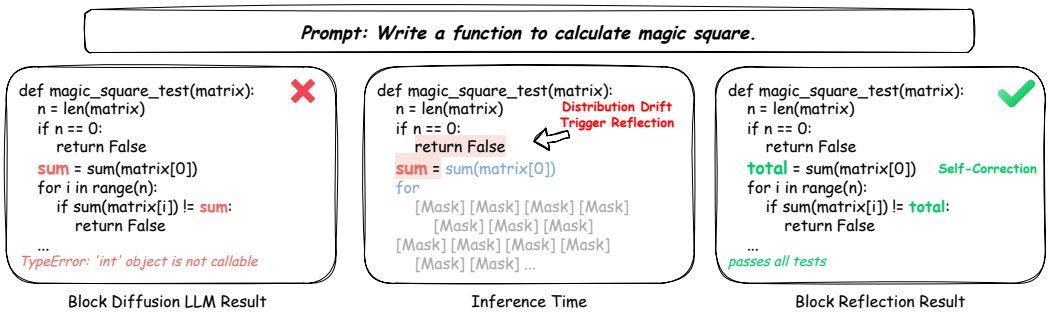

Figure 1: Block Reflection on an MBPP problem. Left: the plain decoder names a variable `sum`, hiding the built-in, so the program crashes. Middle: after the next block (blue) calls `sum(matrix[0])`, drift on the earlier block (highlighted) triggers reflection. Right: regenerating only that block renames the variable to `total`, and all tests pass.

self-verification (Bai et al., 2026). These methods are effective but inherit an AR-style view of correction: they resample, ensemble, or search over alternative outputs. Expanding the generation space this way adds test-time overhead and still treats correction as selection among competing forward trajectories. dLLMs offer a more intrinsic repair mechanism: previously committed spans can be remasked and re-estimated under newly available bidirectional context, so earlier reasoning blocks can be revised within the same trajectory. This raises a dLLM-specific question:

How to perform training-free self-correction for dLLMs via block-level reflection?

We propose *Block Reflection*, a training-free inference procedure that turns block-wise dLLM generation into online self-correction. After each new block is drafted, it scores every previously committed block by the per-position drift between its pre-draft and post-draft conditional token distributions and repairs the most-shifted block only if this drift exceeds a single fixed threshold; otherwise decoding proceeds unchanged. The selected block is repaired either by *refill*, a one-pass forward fill that commits high-confidence tokens at the masked positions, or by *rediffusion*, a short local denoising run on the same block budget. As an illustrative example in Figure 1, once the next block calls the built-in `sum`, drift flags the earlier block that named a variable `sum`; regenerating only that block renames the variable to `total`, the rest of the program follows the new name, and all tests pass.

Our contributions are as follows:

- **Block-level formulation.** We cast dLLM self-correction as block-level backward refinement of committed reasoning spans under later context. We show by construction that joint remasking enables coordinated revisions unreachable by greedy updates on smaller subsets.
- **dLLM-native signal.** We identify conditional drift in past blocks as a dLLM-native reflection signal that localizes potentially flawed reasoning spans from the model's own denoising distributions, and relate this drift quantitatively to the value of updating local token decisions.
- **Training-free repair.** We develop *Block Reflection*, a wrapper around block-diffusion decoders that repairs selected spans by refill or rediffusion, without a verifier, reward model, or fine-tuning.
- **Consistent gains.** We evaluate Block Reflection on mathematical reasoning and code-generation benchmarks with LLaDA-8B-Instruct and obtain consistent gains over the plain decoder, including absolute improvements of 6.2% on MATH-500 and 10.2% on MBPP. The gains carry over to three other dLLMs, with improvements of 2.6% to 6.0% on MBPP and 0.6% to 3.7% on HumanEval.

2 Related Work

Block-wise decoding Diffusion language models (dLLMs) generate text by iteratively unmasking tokens under bidirectional attention (Nie et al., 2025; Ye et al., 2025), and recent work scales them up and post-trains them for reasoning (Zhu et al., 2026; Song et al., 2025; Zhao et al., 2025; Tang et al., 2026; Wang et al., 2026c; Bie et al., 2025). Long outputs are usually decoded block by block, left to right across blocks and in parallel within each block (Arriola et al., 2025; Wu et al., 2025; Ma et al., 2025; Liu et al., 2026), an order that also suits reasoning, which benefits from near-left-to-right generation (Ni et al., 2026; Chen et al., 2025). Work on this decoder refines the forward pass by adapting the output length (Li et al., 2026a) or the block length (Liu et al., 2025), or by deferring

uncertain tokens until more context arrives (Shu et al., 2026). In all of these decoders a finished block stays fixed and conditions every later block, so an early error propagates as it would in an autoregressive (AR) prefix. Block Reflection reopens a committed block after later blocks are drafted and predicts it again with context on both sides, which AR decoding cannot do.

Resampling versus revising in place. Standard masked diffusion never updates a token once it is unmasked (Wang et al., 2025; Schiff et al., 2026). Test-time scaling methods therefore resample: they vote over block schedules (Lee et al., 2026) or over tokens on which samples disagree (Feng et al., 2026), or search over partially remasked trajectories (Bai et al., 2026) and generation orders (Shen et al., 2026). Their cost grows with the number of samples or branches, and they remove an error by outvoting or pruning it. In-place methods revise the current output, but most require training, adding corrective steps between unmasking steps (Schiff et al., 2026), learned remasking (Huang et al., 2026), token-to-token editing (Bie et al., 2026), or a global refinement pass (Ma et al., 2026). Training-free variants remask tokens on a stochastic schedule (Wang et al., 2025) or backtrack to earlier blocks when decoding stalls (Wang et al., 2026b), but they select tokens one by one, so a fix that needs several dependent tokens to change together can be blocked by those left in place (Proposition 4.2). Block Reflection also revises in place, but it needs no training, reopens a whole block at once, and spends extra computation only when its gate fires.

Deciding what to revise. These methods choose what to revise from the model's confidence (Wang et al., 2026b; Ma et al., 2026), a learned remasking score (Huang et al., 2026), agreement across samples (Feng et al., 2026), or prompted self-evaluation (Bai et al., 2026), which has proven unreliable for correcting AR reasoning without external feedback (Huang et al., 2024). Other work tracks how predictions evolve over denoising steps and uses them to vote, stop early, or detect hallucinations (Wang et al., 2026a; Li et al., 2026b; Chang et al., 2026). None of these signals compares the model's belief about an earlier span with and without the context that follows it. A choice that looks plausible from its left context can therefore go undetected, such as naming a variable `sum` before the program calls the built-in `sum` (Figure 1). Block Reflection makes this comparison: its drift score, computed from the base model's own distributions, finds the past block whose predictions shift most once the newest block is visible, without a verifier, reward model, or extra samples.

3 PRELIMINARIES

Block Reflection relies on two properties of masked-diffusion language models: (i) every committed token carries a model-internal confidence that can be re-evaluated under new context, and (ii) masking a contiguous span and re-denoising it conditions on bidirectional context that was unavailable when the span was first generated. We summarize the formalism below; a consolidated notation table appears in Appendix A.

Masked diffusion language modeling. Let $\mathcal{V}$ be a discrete vocabulary with mask token $\mathtt{m} \notin \mathcal{V}$, and write $\mathcal{V}_{\mathtt{m}} = \mathcal{V} \cup \{\mathtt{m}\}$. A response is a token sequence $\mathbf{x} = (x_1, \ldots, x_L) \in \mathcal{V}_{\mathtt{m}}^L$; for $S \subseteq [L]$, $\mathrm{Mask}(\mathbf{x}, S)$ replaces every x_i with $i \in S$ by $\mathtt{m}$ and leaves the rest unchanged. Given a prompt $\mathbf{c}$, a dLLM generates a response by iteratively denoising a masked sequence (Nie et al., 2025; Ye et al., 2025). In the absorbing-mask formulation, a forward process independently masks each token of a clean sequence $\mathbf{x}_0$ with probability $1-\alpha_t$ (α_0=1, α_1=0), and a model p_θ is trained to recover the clean token at each masked position,

$$p_\theta(x_{0,i} \mid \mathbf{x}_t, \mathbf{c}), \qquad i \in M_t = \{j : x_{t,j} = \mathtt{m}\},$$

via a possibly reweighted masked cross-entropy loss (Nie et al., 2025; Ye et al., 2025).

Reverse denoising and token commitment. Starting from a fully masked sequence $\mathbf{x}^{(K)} = \mathtt{m}^L$, the decoder shrinks the mask set over K steps. At each step, the model proposes a candidate and a confidence at every masked position,

$$\hat{x}_i^{(k)} = \arg\max_{v \in \mathcal{V}} p_\theta(v \mid \mathbf{x}^{(k)}, \mathbf{c}), \qquad s_i^{(k)} = \max_{v \in \mathcal{V}} p_\theta(v \mid \mathbf{x}^{(k)}, \mathbf{c}), \tag{1}$$

and a decoding policy selects a subset $U^{(k)} \subseteq M^{(k)}$ to reveal (Nie et al., 2025; Huang et al., 2026). Once revealed, a token is *committed*: standard masked diffusion does not revisit it, even if later

context would favor a different choice (Lee et al., 2026). Block Reflection removes this restriction at the block level.

Block-wise decoding as a repair interface. Many practical dLLM decoders combine autoregressive progress across blocks with diffusion denoising inside each block (Arriola et al., 2025; Nie et al., 2025). Given an ordered partition $\mathcal{B} = (B_1, \ldots, B_N)$ of $[L]$ into contiguous blocks, a block-wise decoder freezes previously committed blocks and generates each successive block by K_j inner denoising steps conditioned on the prompt and all earlier blocks. When all blocks have the same length, we write $\ell = |B_j|$ for the block length, so that $N = L/\ell$. This factorization separates an *outer* schedule (which block is generated next) from an *inner* schedule (which tokens inside that block are revealed at each denoising step). The same view applies even without block-aware training: any contiguous span can be turned into a temporary block by masking it while keeping the surrounding context fixed. Re-running denoising on $\mathrm{Mask}(\mathbf{x}, B_j)$ therefore induces a block-level conditional

$$p_\theta\big(\mathbf{z}_{B_j} \mid \mathrm{Mask}(\mathbf{x}, B_j),\, \mathbf{c}\big),$$

which proposes alternatives for $\mathbf{x}_{B_j}$ conditioned on both preceding and succeeding blocks, including blocks drafted after B_j. Section 4 uses this conditional to decide which committed block to revisit and how to regenerate it.

4 Method

4.1 Motivation: Drift as an Audit Signal

Block-wise diffusion permits a check that left-to-right decoding does not: after drafting the current block B_k, any past block B_j ($j < k$) can be remasked and re-evaluated both *before* and *after* the new draft is appended. We call the resulting shift in the model's per-position distributions over B_j, measured by JS divergence, the block drift r_j (Equation (4)). To test whether this signal reflects errors, we average r_j over all probed blocks of each answer and compare correct and wrong answers in Figure 2. We observe a consistent gap: the median drift of wrong answers exceeds that of correct answers by about 20% on both GSM8K (0.204 vs. 0.168) and MATH-500 (0.223 vs. 0.189). Later context thus shifts the model's beliefs about earlier blocks more strongly when the answer is wrong. The two distributions still overlap, so drift is informative about errors without certifying them. This observation motivates Block Reflection, which uses drift, computed from the base model's own denoising distributions without a verifier, reward model, or additional training, to select a past block, and then regenerates that block under the updated context so that the model itself decides whether the original tokens are retained. Propositions 4.1 and 4.2 support the two components: a large gain from updating a block's decisions requires large drift, and joint remasking reaches revisions that updates of smaller subsets cannot.

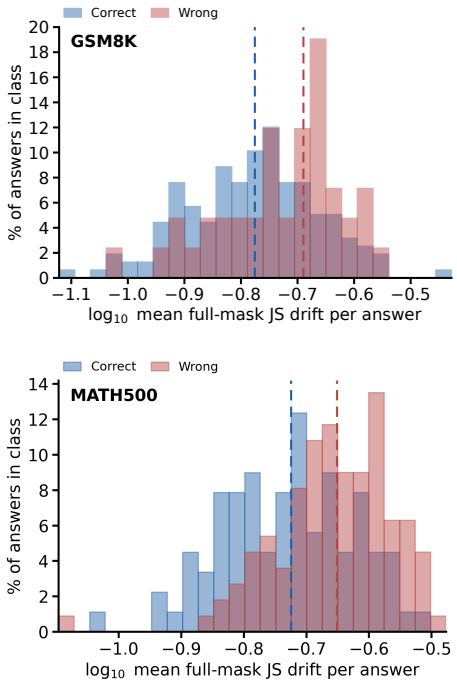

Figure 2: Per-answer block drift of correct and wrong answers on GSM8K (top) and MATH-500 (bottom), normalized per class; dashed lines denote class medians.

4.2 Block Reflection

Overview. Block Reflection turns the drift signal of Section 4.1 into an online repair loop around a block-wise diffusion decoder (Figure 3). After the decoder drafts block B_k at outer step k, three stages follow. (1) *Probe*: each past block B_j ($j < k$) is re-evaluated with the draft hidden and visible. (2) *Detect drift*: the past block with the largest drift r_j is selected if r_j exceeds a threshold τ. (3) *Repair*: the selected block is jointly remasked and regenerated under the updated context, by a few-step *refill* or a full-budget *rediffusion*, while all other blocks, including the draft, remain fixed.

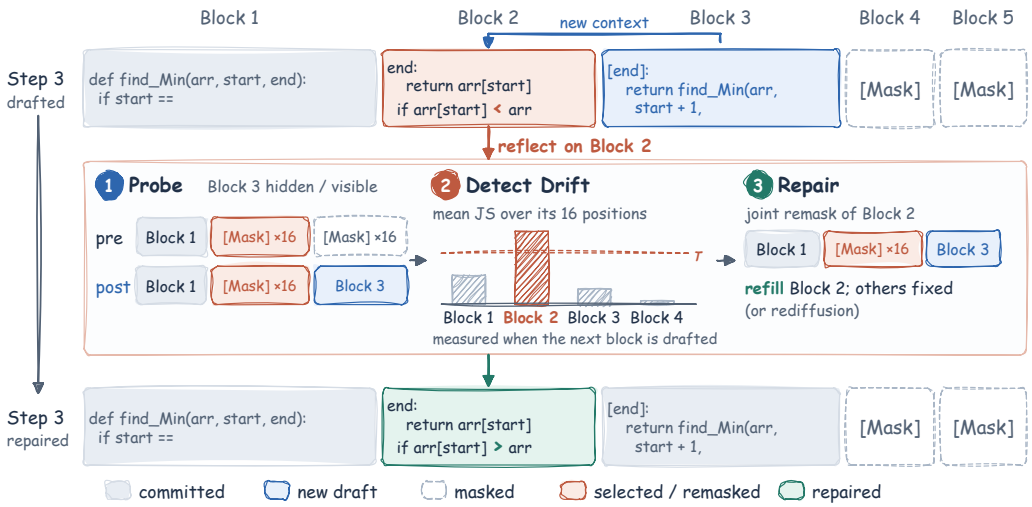

Figure 3: Block Reflection on an MBPP problem. The newly drafted Block 3 recurses with `start + 1`, discarding `arr[start]`, which is correct only if `arr[start] > arr[end]`. (1) Block 2 is probed with Block 3 hidden (pre) and visible (post); (2) its drift is the largest and exceeds τ; (3) refilling it with the other blocks fixed changes its comparison from < to >.

Online block state. Let $\mathcal{B} = (B_1, \ldots, B_N)$ be the block partition from Section 3. At outer step k, the decoder has committed $\mathbf{x}_{B_{<k}}$ and drafts the current block as

$$\tilde{\mathbf{x}}_{B_k} = \mathrm{Decode}_\theta\big(\mathbf{c}, \mathbf{x}_{B_{<k}}, B_k\big)\,, \tag{2}$$

where Decode_θ is the usual inner denoising schedule of the base dLLM. Block Reflection leaves this drafting step unchanged and treats $\tilde{\mathbf{x}}_{B_k}$ as a perturbation to the context used to judge earlier blocks: a past block that becomes distributionally unstable once the draft is visible is a candidate for repair.

Step 1: probe past blocks. For each past block B_j with $j < k$, we compare the model's per-position belief about B_j before and after it sees the current draft. Let $\mathbf{x}^{\mathrm{post}} = (\mathbf{x}_{B_{<k}}, \tilde{\mathbf{x}}_{B_k})$ be the answer with the draft and $\mathbf{x}^{\mathrm{pre}} = \mathrm{Mask}(\mathbf{x}^{\mathrm{post}}, B_k)$ the same answer with the draft hidden. Masking all of B_j in each gives the two probe inputs

$$\mathbf{y}_j^{\mathrm{pre}} = \mathrm{Mask}(\mathbf{x}^{\mathrm{pre}}, B_j), \qquad \mathbf{y}_j^{\mathrm{post}} = \mathrm{Mask}(\mathbf{x}^{\mathrm{post}}, B_j),$$

and for each $i \in B_j$ we define

$$p_{j,i}^{\mathrm{pre}}(v) = p_\theta(v \mid \mathbf{y}_j^{\mathrm{pre}}, \mathbf{c}), \qquad p_{j,i}^{\mathrm{post}}(v) = p_\theta(v \mid \mathbf{y}_j^{\mathrm{post}}, \mathbf{c}), \quad v \in \mathcal{V}. \tag{3}$$

Both inputs run in one batch, and $\mathbf{y}_j^{\mathrm{post}}$ is also the input that refill reads if B_j is later repaired (Step 3).

Step 2: detect drift and select a block. The drift score of past block B_j averages the per-position divergence between these distributions over the block:

$$r_j \;=\; \frac{1}{|B_j|} \sum_{i \in B_j} D_{\mathrm{JS}}\big(p_{j,i}^{\mathrm{pre}} \,\big\|\, p_{j,i}^{\mathrm{post}}\big)\,. \tag{4}$$

A large r_j means that the new draft shifts the model's per-position belief about B_j. Block Reflection repairs a block only when the largest drift among eligible past blocks exceeds τ:

$$j^\star = \arg\max_{j \in \{1,\ldots,k-1\} \setminus \mathcal{R}} r_j, \qquad R_k = \max_{j \in \{1,\ldots,k-1\} \setminus \mathcal{R}} r_j, \qquad \text{repair if } R_k > \tau, \tag{5}$$

where $\mathcal{R} \subseteq [k-1]$ is a cooldown set of blocks already repaired in this run. If $R_k \le \tau$ or $\mathcal{R}$ exhausts the past blocks, the decoder commits the draft and proceeds; otherwise $B_{j^\star}$ is repaired (Step 3) and added to $\mathcal{R}$. The cooldown repairs each block at most once per generation, which bounds the total number of repairs by $N-1$ and guarantees termination.

Why does drift identify blocks worth repairing? The following result connects drift to the value of updating local decisions, with all divergences in natural logarithms.

Proposition 4.1 (Drift and the value of updating decisions)**.** *Fix a nonempty past block B_j and a finite vocabulary with $|\mathcal{V}| \geq 2$, using a fixed deterministic tie-breaking rule. For each $i \in B_j$, write $p_i = p_{j,i}^{\mathrm{pre}}$, $q_i = p_{j,i}^{\mathrm{post}}$, $a_i = \arg\max_v p_i(v)$, and $b_i = \arg\max_v q_i(v)$. Let $\Delta_i = q_i(b_i) - q_i(a_i)$ be the advantage of the updated decision under q_i. Then*

$$0 \leq \overline{\Delta}_j := \frac{1}{|B_j|} \sum_{i \in B_j} \Delta_i \leq \sqrt{8 r_j}. \tag{6}$$

If $b_i \neq a_i$, let $m_i = p_i(a_i) - \max_{v \neq a_i} p_i(v)$ be the pre-context margin. Then

$$D_{\mathrm{JS}}(p_i \| q_i) \geq \frac{(m_i + \Delta_i)^2}{8}. \tag{7}$$

The proof is in Appendix C. Here $\overline{\Delta}_j$ is a reduction in expected token error measured under the model's post-context distributions. Equation (6) shows that a large decision advantage requires large drift, and Equation (7) quantifies the drift needed to overturn a confident earlier decision, so the gate screens for blocks in which the new draft can change local choices. Appendix C.2 extends the bound to the actual committed tokens, separating their pre-existing decision gap from the contribution of the new context.

Two drift proxies: JS divergence and entropy delta. We use the symmetric, bounded Jensen–Shannon divergence as the default instantiation of r_j, since its range $[0, \log 2]$ gives the score a common scale and supports the quantitative connection in Proposition 4.1. As a complementary proxy, we instantiate r_j with the per-position entropy delta

$$r_j^{\Delta H} \;=\; \frac{1}{|B_j|} \sum_{i \in B_j} \big|\, H(p_{j,i}^{\mathrm{post}}) - H(p_{j,i}^{\mathrm{pre}}) \,\big|, \tag{8}$$

which measures how much the new draft changes the model's uncertainty at each position. Unlike JS, entropy delta is invariant to permutations of token probabilities, so it can miss a shift between strongly preferred tokens; Appendix C gives an explicit example, and Section 5 compares the two signals empirically.

Step 3: repair the selected block. When the gate fires, Block Reflection regenerates the selected block $B_{j^\star}$, whose original inner denoising budget is $K_{j^\star}$. It forms

$$\mathbf{x}^{\mathrm{mask}} = \mathrm{Mask}((\mathbf{x}_{B_{<k}}, \tilde{\mathbf{x}}_{B_k}), B_{j^\star})$$

and regenerates only $\mathbf{x}_{B_{j^\star}}$ under the fixed surrounding context $\mathbf{x}_{[L]\setminus B_{j^\star}}$, which now includes the just-drafted $\tilde{\mathbf{x}}_{B_k}$. Two regeneration modes trade compute against thoroughness:

(i) Refill (few-step). The model performs a single forward pass on $\mathbf{x}^{\mathrm{mask}}$ and commits the argmax token at every position whose top-1 confidence is at least γ:

$$x_i^{\mathrm{new}} = \begin{cases} \arg\max_{v \in \mathcal{V}} p_\theta(v \mid \mathbf{x}^{\mathrm{mask}}, \mathbf{c}), & \max_{v \in \mathcal{V}} p_\theta(v \mid \mathbf{x}^{\mathrm{mask}}, \mathbf{c}) \geq \gamma, \\ \mathtt{m}, & \text{otherwise}, \end{cases} \qquad i \in B_{j^\star}. \tag{9}$$

All accepted positions are committed simultaneously without intra-pass logit recomputation; any position left masked is completed by a short residual schedule of length $K^{\mathrm{refill}} \ll K_{j^\star}$ conditioned on the just-committed positions and the unchanged surrounding blocks, so a refill repair costs far less than the block's original drafting.

(ii) Rediffusion (original budget). The method reruns the full inner denoising schedule of length $K_{j^\star}$ on $B_{j^\star}$ from an all-mask state, conditioned on the unchanged surrounding blocks, so the regeneration uses exactly the budget of the original drafting.

The two variants differ only in this repair mode and share the same dLLM parameters; we use refill by default, since both modes reach the same accuracy in our ablation and refill is cheaper (Section 5).

Why jointly reopen a block? Step 3 remasks the whole selected block at once because tokens within a committed fragment can reinforce one another: keeping part of the fragment visible can preserve the current completion even when the surrounding context favors a different one. The following construction isolates how the scope of masking changes the reachable revision, assuming exact conditional predictions for the constructed distribution.

Proposition 4.2 (Joint remasking releases coupled commitments)**.** *For every block length $\ell \geq 2$ and refill threshold $\gamma \in [0, 1)$, there exists a strictly positive distribution Q on $\{0, 1\}^\ell$, under fixed external context, with the following properties. Starting from* **0***, any sequence of updates that remasks a nonempty proper subset of positions and fills it with the conditional marginal argmaxes leaves the block at* **0***. Remasking all ℓ positions and applying one parallel refill pass with threshold γ instead yields* **1***, the unique global maximizer of Q.*

Appendix C.3 gives the construction and proof. The separation locates a source of repair power in the masking unit itself: jointly removing mutually supporting commitments allows coordinated changes in a single pass. It predicts a larger advantage for block repair when a revision requires several dependent positions to change together, which motivates choosing spans that cover those dependencies. This complements Proposition 4.1: drift identifies where new context changes local decisions, and joint remasking determines which revisions the repair can reach.

Overall procedure. Algorithm 1 in Appendix B combines the three steps into the full online loop. After outer step k, the answer is either $(\mathbf{x}_{B_{<k}}, \tilde{\mathbf{x}}_{B_k})$, if the gate of Equation (5) does not fire, or the same answer with one past block revised in place. We do not redraft $\tilde{\mathbf{x}}_{B_k}$ after an intermediate-step repair: keeping the current draft fixed makes each outer step a single pass and lets any remaining inconsistency surface at step $k{+}1$, where it can trigger the gate on the now-most-recent block.

5 Experiments

5.1 Experiment Setup

Baselines. We apply Block Reflection to a representative diffusion language model, LLaDA-8B-Instruct (Nie et al., 2025), under block-wise decoding (Arriola et al., 2025) and compare against three test-time scaling methods: HEX (Lee et al., 2026), Prism (Bai et al., 2026), and dVoting (Feng et al., 2026). We also include LLaDA-1.5 (Zhu et al., 2026), which further post-trains LLaDA-8B-Instruct with reinforcement learning specifically for reasoning, as a strong reasoning-oriented baseline. Following prior work, we evaluate at generation lengths $L \in \{128, 256\}$ with block lengths $\ell = 8$ and 16, respectively; both settings yield $N = 16$ outer blocks per response, each the unit of drafting and repair in Algorithm 1. Section 5.4 extends the evaluation to LLaDA-1.5, Dream (Ye et al., 2025) and LLaDA-2.1-mini (Bie et al., 2026).

Benchmarks. Following recent studies on refinement and test-time scaling for dLLMs (Schiff et al., 2026), we evaluate on mathematical reasoning and code generation. The reasoning benchmarks are GSM8K (Cobbe et al., 2021) and MATH-500, a subset of MATH (Hendrycks et al., 2021) commonly used for process-verification evaluation (Lightman et al., 2024); the code benchmarks are HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). For a fair comparison, all evaluations use the *lm-evaluation-harness* package (Gao et al., 2024); Appendix D gives further details.

5.2 Main Result

Table 1 supports three observations. **(1) Consistent gains over the plain decoder.** Block Reflection improves Pass@1 over LLaDA-8B-Instruct on every (benchmark, generation length) cell, indicating that conditional drift is a usable reflection signal for both mathematical reasoning and code generation. **(2) Stronger benefits on more complex benchmarks.** Among training-free methods, Block Reflection is best on both code benchmarks at both generation lengths, and on MATH-500 it ties Prism for first at $L{=}128$ and is second to HEX at $L{=}256$, showing that it is especially effective on longer or more structured generations, where previous blocks give the gate richer evidence. It trails HEX and dVoting on GSM8K, which we attribute to GSM8K's shorter solutions leaving fewer past blocks for the gate to inspect. **(3) Favorable accuracy–compute tradeoff.** These gains come at a fraction of the compute used by HEX and Prism, and the HumanEval cell at $L{=}256$ even falls below

Table 1: Pass@1 accuracy (%) on mathematical reasoning and code generation benchmarks with LLaDA-8B-Instruct. Columns 128 and 256 denote the generation length L (total response tokens); the corresponding block lengths are $\ell = 8$ and 16. † indicates methods that rely on an additional instruction-tuned or RL-enhanced policy model. The number of decoding steps is reported after the slash. For each setting, the best results are in **bold**, the second best are underlined.

	MATH-500 (0-shot)		GSM8K (0-shot)		MBPP (3-shot)		HumanEval (0-shot)	
Method	L=128	L=256	L=128	L=256	L=128	L=256	L=128	L=256
LLaDA-8B-Instruct	31.4 / 128	35.0 / 256	70.6 / 128	76.7 / 256	31.8 / 128	37.6 / 256	26.2 / 128	31.7 / 256
LLaDA1.5†	34.4 / 128	41.8 / 256	76.0 / 128	77.6 / 256	34.4 / 128	36.4 / 256	32.9 / 128	38.4 / 256
HEX	32.8 / 1600	**42.0** / 3200	**80.1** / 1600	**82.9** / 3200	36.0 / 1600	39.0 / 3200	28.1 / 1600	28.7 / 3200
Prism	37.6 / 592	37.6 / 754	67.1 / 584	75.5 / 626	30.8 / 805	33.4 / 628	20.7 / 604	28.6 / 749
dVoting	34.8 / 292	40.2 / 473	78.1 / 173	82.9 / 237	39.0 / 292	39.6 / 393	26.8 / 148	35.4 / 190
Block Reflection (ours)	**37.6** / 136	40.8 / 268	76.5 / 244	79.5 / 272	**42.0** / 159	**43.0** / 289	**33.5** / 129	**39.6** / 223

the plain decoder's step count, showing that the overhead scales with how often the gate fires rather than with the worst-case repair budget. Block Reflection also exceeds RL-tuned LLaDA-1.5 on all four benchmarks at L=128 and on all but MATH-500 at L=256, suggesting that a sizable portion of the LLaDA-8B-Instruct → LLaDA-1.5 gap is recoverable purely at inference.

Where repairs occur. Figure 4 shows where Block Reflection repairs at L=256 and ℓ=16. The distribution depends strongly on the benchmark. HumanEval is front-loaded, with blocks 1–5 receiving about 60% of repairs, reflecting the importance of early structural decisions in code generation. MBPP instead concentrates on blocks 7–12, which receive about three-quarters of repairs, consistent with longer prompts delaying the transition into code generation. MATH-500 repairs peak in the middle of the response, with blocks 4–11 receiving about two-thirds, and fall to 3% per block over blocks 12–14. GSM8K is spread almost uniformly, with 5–8% of repairs per block. The last block is never repaired, since no later draft exists to probe it. Across benchmarks and generation lengths, the overall trigger frequency ranges from 12.6% to 94.5%.

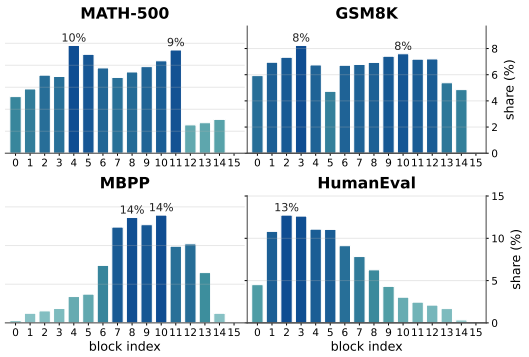

Figure 4: Share of repairs landing on each block with L=256 and ℓ=16 on the four benchmarks.

5.3 Repair Lowers Block Entropy

A repair regenerates a past block with the later blocks already in its context, so the model can resolve positions that were ambiguous when the block was first drafted. For every block that the gate reopens, we compute the mean per-position predictive entropy of the original draft, which decoding without reflection would keep, and of the repaired block. As Figure 5 shows, repair lowers this entropy on all four benchmarks, by 14–15% on GSM8K, MATH-500, and HumanEval and by 33% on MBPP. Code blocks start at much lower entropy than math blocks, which fits the tighter syntax of code, and repair reduces it further. The drop fits the selection rule: the gate reopens a block only when the new draft has shifted the model's per-position beliefs about it (Proposition 4.1), which singles out blocks whose predictions depend on the added context. Combined with the accuracy gains in Table 1, the lower entropy suggests that repairs replace uncertain early choices with tokens that agree better with the rest of the response.

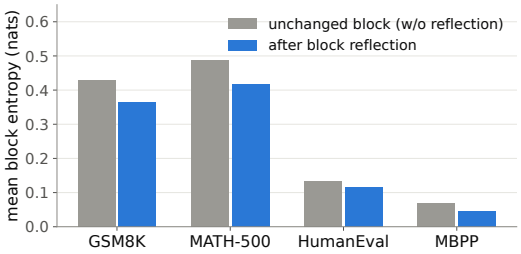

Figure 5: Mean per-position predictive entropy of reopened blocks before and after repair, for LLaDA-8B-Instruct with L=128 and ℓ=8 on the four benchmarks.

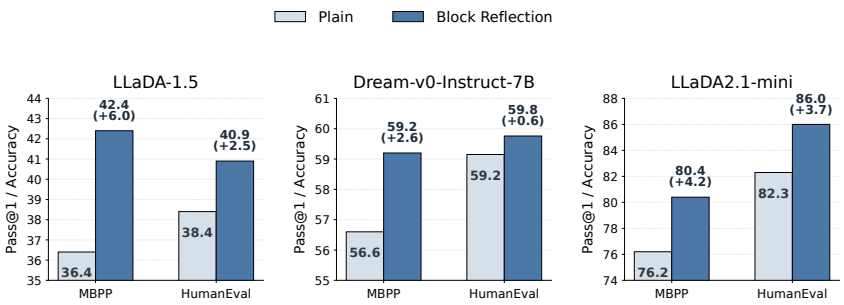

Figure 6: Pass@1 (%) on MBPP and HumanEval for three dLLMs with the plain block-wise decoder and Block Reflection. Numbers in parentheses are gains over the plain decoder.

Table 2: Ablation study of Block Reflection under different settings MATH-500 with length 256.

	Threshold τ for JS			Proxy		Refill Steps			Repair Mode		Block Length ℓ		
Method	0.05	0.2	0.4	JS	ΔH	1	2	4	Refill	Rediff	8	16	32
Block Reflection	40.2	40.8	40.4	40.8	40.6	40.6	40.8	40.8	40.8	40.8	40.2	40.8	40.4

5.4 Generalization Across dLLMs

We wrap the default block-wise decoder of three other dLLMs: LLaDA-1.5 (Zhu et al., 2026), the RL-tuned baseline in Table 1, Dream-v0-Instruct-7B (Ye et al., 2025), and LLaDA2.1-mini (Bie et al., 2026). Since LLaDA2.1-mini is trained with token-to-token (T2T) editing, which rewrites committed tokens without remasking them, Block Reflection repairs its selected block by triggering T2T editing instead of remasking. Block Reflection improves Pass@1 for every model on both benchmarks (Figure 6), by 2.6 to 6.0 points on MBPP and 0.6 to 3.7 points on HumanEval. LLaDA2.1-mini, the strongest model, still gains 4.2 and 3.7 points, showing that the block-reflection does not depend on remasking and can direct the model's native T2T editing; the 6.0-point MBPP gain on LLaDA-1.5 suggests that it also complements RL post-training.

5.5 Ablation Study

Table 2 measures the sensitivity of Block Reflection on MATH-500 with a generation length of 256; accuracy is stable across hyperparameter choices. The JS threshold $\tau = 0.2$ gives the best accuracy, 40.8%, and nearby thresholds lead to only minor changes. JS divergence slightly outperforms entropy delta, suggesting that distributional drift is a more reliable reflection signal than uncertainty change alone. Two refill steps improve on one, and four bring no further gain, so a small refill budget suffices. Refill and rediffusion obtain the same accuracy, so we use the cheaper refill by default. Block length $\ell = 16$ performs best, suggesting a better balance between local repair granularity and contextual completeness. The gains therefore come mainly from drift-guided block-level repair rather than careful hyperparameter tuning.

6 Conclusion

We presented *Block Reflection*, a training-free inference procedure that turns the bidirectional denoising interface of discrete diffusion language models into a mechanism for self-correction through local block repair. After each newly drafted block, Block Reflection re-evaluates earlier blocks and measures their pre/post-draft distributional drift, a native signal of spans whose model belief has become unstable under the new future context. When the drift exceeds a threshold, the method remasks and regenerates only the selected past block and keeps the rest of the solution fixed. Across mathematical reasoning and code generation benchmarks, Block Reflection consistently improves over the plain block-diffusion decoder. The results suggest that a sizable portion of test-time reasoning errors in dLLMs can be addressed by targeted local revision rather than by sampling more complete trajectories, and the strong gains on code generation indicate that block-level repair is especially useful when correctness depends on localized structural consistency.

AI USE STATEMENT

In this work, we used generative AI tools to assist with and polish the writing, as well as to support code development for our experiments. All AI-assisted text and code were reviewed, verified, and revised by the authors, and the experimental code was tested for correctness. We take full responsibility for the final content of this work, including all text, claims, results, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We will release our code during the rebuttal period to facilitate reproducibility. In addition, a detailed description of our method is provided in Section 4.2 and Algorithm 1. The experimental settings, including implementation details and evaluation protocols, are described in Section 5.1 and further documented in Appendix D.

REFERENCES

Marianne Arriola, Subham Sekhar Sahoo, Aaron Gokaslan, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Justin T Chiu, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

Jinbin Bai, Yixuan Li, Yuchen Zhu, Yi Xin, Qingyu Shi, Aosong Feng, Xiaohong Liu, Molei Tao, Jianru Xue, Xiangtai Li, and Ming-Hsuan Yang. PRISM: Efficient test-time scaling via hierarchical search and self-verification for discrete diffusion language models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=QrC8OgQyOI.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei, Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, and Yihong Zhuang. LLaDA2.1: Speeding up text diffusion via token editing, 2026. URL https://arxiv.org/abs/2602.08676.

Shenxu Chang, Junchi Yu, Weixing Wang, Yongqiang Chen, Jialin Yu, Philip Torr, and Jindong Gu. TRACEDET: HALLUCINATION DETECTION FROM THE DECODING TRACE OF DIFFUSION LARGE LANGUAGE MODELS. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=4puxTouUSV.

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, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Qiguang Chen, Hanjing Li, Libo Qin, Dengyun Peng, Jinhao Liu, Jiangyi Wang, Chengyue Wu, Xie Chen, Yantao Du, and Wanxiang Che. Beyond surface reasoning: Unveiling the true long chain-of-thought capacity of diffusion large language models. *arXiv preprint arXiv:2510.09544*, 2025.

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. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Sicheng Feng, Zigeng Chen, Xinyin Ma, Gongfan Fang, and Xinchao Wang. dVoting: Fast voting for dLLMs, 2026. URL https://arxiv.org/abs/2602.12153.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

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. In *NeurIPS Datasets and Benchmarks*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Zemin Huang, Yuhang Wang, Zhiyang Chen, and Guo-Jun Qi. Don't settle too early: Self-reflective remasking for diffusion language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=BsZeTuB5fD.

Wonje Jeung, Sangyeon Yoon, Yoonjun Cho, Dongjae Jeon, Sangwoo Shin, Hyesoo Hong, and Albert No. A2D: Any-order, any-step safety alignment for diffusion language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=URTnuyQJI1.

Jihoon Lee, Hoyeon Moon, Kevin Zhai, Arun Kumar Chithanar, Anit Kumar Sahu, Soummya Kar, Chul Lee, Souradip Chakraborty, and Amrit Singh Bedi. Test-time scaling in diffusion llms via hidden semi-autoregressive experts. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=L5y7in91vd.

Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. Beyond fixed: Training-free variable-length denoising for diffusion large language models. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=Ic2A2gCseC.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. Diffusion language model knows the answer before it decodes. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=g88nt4ieTG.

Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. A survey on diffusion language models. *arXiv preprint arXiv:2508.10875*, 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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Yangzhou Liu, Yue Cao, Hao Li, Gen Luo, Zhe Chen, Weiyun Wang, Xiaobo Liang, Biqing Qi, Lijun Wu, Changyao Tian, Yanting Zhang, Yuqiang Li, Tong Lu, Yu Qiao, Jifeng Dai, and Wenhai Wang. Sequential diffusion language models, 2025. URL https://arxiv.org/abs/2509.24007.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyan Wei, Shaobo Wang, Yichen Zhu, and Linfeng Zhang. dLLM-cache: Accelerating diffusion large language models with adaptive caching. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=guLrgDb2jB.

Linrui Ma, Yufei Cui, Kai Han, and Yunhe Wang. Diffusion in diffusion: Reclaiming global coherence in semi-autoregressive diffusion, 2026. URL https://arxiv.org/abs/2601.13599.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dKV-cache: The cache for diffusion language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Gppo2JImHs.

Zanlin Ni, Shenzhi Wang, Yang Yue, Tianyu Yu, Weilin Zhao, Yeguo Hua, Tianyi Chen, Jun Song, Cheng Yu, Bo Zheng, et al. The flexibility trap: Why arbitrary order limits reasoning potential in diffusion language models. *arXiv preprint arXiv:2601.15165*, 2026.

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. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=KnqiC0znVF.

Yair Schiff, Omer Belhasin, Roy Uziel, Guanghan Wang, Marianne Arriola, Gilad Turok, Michael Elad, and Volodymyr Kuleshov. Learn from your mistakes: Self-correcting masked diffusion models. *arXiv preprint arXiv:2602.11590*, 2026.

Yangyi Shen, Tianjian Feng, Jiaqi Han, Wen Wang, Tianlang Chen, Chunhua Shen, Jure Leskovec, and Stefano Ermon. Improving diffusion language model decoding through joint search in generation order and token space. *arXiv preprint arXiv:2601.20339*, 2026.

Yingte Shu, Yuchuan Tian, Chao Xu, Yunhe Wang, and Hanting Chen. Deferred commitment decoding for diffusion language models with confidence-aware sliding windows. *arXiv preprint arXiv:2601.02076*, 2026.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Xiaohang Tang, Rares Dolga, Sangwoong Yoon, and Ilija Bogunovic. wd1: Weighted policy optimization for reasoning in diffusion language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=L2rfd2Czbj.

Guanghan Wang, Yair Schiff, Subham Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 147282–147339. Curran Associates, Inc., 2025. doi: 10.52202/085713-4927. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/d877ea0dd78bdbe54830670618c1de09-Paper-Conference.pdf.

Wen Wang, Bozhen Fang, Chenchen Jing, Yongliang Shen, Yangyi Shen, Qiuyu Wang, Hao Ouyang, Hao Chen, and Chunhua Shen. Time is a feature: Exploiting temporal dynamics in diffusion language models. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=HsB6CtagP7.

Xinyun Wang, Min Zhang, Sen Cui, Zhikang Chen, Bo Jiang, Kun Kuang, and Mingbao Lin. Reversible diffusion decoding for diffusion language models. *arXiv preprint arXiv:2602.00150*, 2026b.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. In *The Fourteenth International Conference on Learning Representations*, 2026c. URL https://openreview.net/forum?id=KNAyc9DMe3.

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

Runpeng Yu, Qi Li, and Xinchao Wang. Discrete diffusion in large language and multimodal models: A survey. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=0DsqnkP8Cp.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=7ZVRlBFuEv.

Zhanhui Zhou, Lingjie Chen, Hanghang Tong, and Dawn Song. dllm: Simple diffusion language modeling. *arXiv preprint arXiv:2602.22661*, 2026.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. LLaDA 1.5: Variance-reduced preference optimization for large language diffusion models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11425–11460, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.524. URL https://aclanthology.org/2026.acl-long.524/.

A NOTATION

Table 3: Summary of frequently used notation.

Symbol	Meaning
$\mathcal{V}$, $\mathtt{m}$	Token vocabulary and special mask token.
$\mathbf{c}$, $\mathbf{x}$, L	Input prompt, generated response, and generation length (total response tokens).
p_θ	Conditional token distribution of the dLLM with parameters θ.
$\mathcal{B}$, B_j, N, ℓ	Ordered block partition, the j-th block (a set of positions), total number of blocks, and block length (tokens per block; $N = L/\ell$).
$\mathbf{x}_{B_{<k}}$, $\tilde{\mathbf{x}}_{B_k}$	Committed prefix and newly drafted block at outer step k.
$\mathbf{x}^{\mathrm{pre}}$, $\mathbf{x}^{\mathrm{post}}$	Answer at outer step k with the current draft hidden and visible.
$\mathbf{y}_j^{\mathrm{pre}}$, $\mathbf{y}_j^{\mathrm{post}}$	Probe inputs with all of B_j masked, before and after adding the current draft.
$p_{j,i}^{\mathrm{pre}}$, $p_{j,i}^{\mathrm{post}}$	Token distributions at position i in masked block B_j, before and after adding the current draft.
D_{JS}, H	Jensen–Shannon divergence and entropy used to measure distributional change.
r_j, $r_j^{\Delta H}$	Block-level risk scores based on JS divergence and absolute entropy change, respectively.
a_i, b_i, m_i	Pre- and post-context marginal argmaxes and the pre-context top-token margin (for a fixed block).
Δ_i, $\overline{\Delta}_j$	Post-context advantage of the updated token decision and its block average.
e_j, g_j	Pre-context decision gap and post-context revision value of the committed block.
$j^\star$, R_k, τ	Index of the highest-risk eligible block, its risk score, and the threshold for triggering repair.
$B_{j^\star}$, $\mathcal{R}$	Selected repair block and the set of already repaired block indices.
$\mathbf{x}^{\mathrm{mask}}$	Response state with the selected block remasked for local repair.
γ	Confidence threshold for accepting tokens during refill.
K_j, K^{refill}	Original denoising budget for block B_j and short residual budget for refill.
ϕ, ρ	Choice of risk proxy and repair mode, respectively.

B FULL PROCEDURE

Algorithm 1 gives the online inference loop of Section 4.2; Score_ϕ implements the probe and drift stages of Figure 3, and Repair_ρ the repair stage. Draft is the plain block-diffusion decoder of Equation (2). Score_ϕ returns the JS drift score of Equation (4) when $\phi = \mathrm{JS}$ and the entropy-delta score of Equation (8) when $\phi = \Delta H$. Repair_ρ is either refill (Equation (9), with residual budget K^{refill}) or rediffusion (with budget $K_{j^\star}$). Each implicitly conditions on $\mathbf{c}$ and the committed prefix; Score_ϕ and Repair_ρ also condition on the current draft $\tilde{\mathbf{x}}_{B_k}$.

Algorithm 1 BLOCKREFLECTION

Require: $\mathbf{c}$; $\mathcal{B} = (B_1, \ldots, B_N)$; p_θ; threshold τ; proxy $\phi \in \{\mathrm{JS}, \Delta H\}$;
mode $\rho \in \{\text{refill}, \text{rediff}\}$; γ; budgets $\{K_j\}$, K^{refill}.
Ensure: answer $\mathbf{x}$.

$\mathbf{x} \leftarrow \emptyset$; $\mathcal{R} \leftarrow \emptyset$ ▷ repaired-block set
for $k = 1, \ldots, N$ **do**
  $\tilde{\mathbf{x}}_{B_k} \leftarrow \mathrm{Draft}(B_k; K_k)$ ▷ Equation (2)
  if $k \geq 2$ **then**
    for $j \in \{1, \ldots, k-1\} \setminus \mathcal{R}$ **do**
      $r_j \leftarrow \mathrm{Score}_\phi(B_j)$ ▷ Equation (4) or Equation (8)
    end for
    if $\{1, \ldots, k-1\} \setminus \mathcal{R} \neq \emptyset$ **then**
      $j^\star \leftarrow \arg\max_j r_j$; $R_k \leftarrow r_{j^\star}$
      if $R_k > \tau$ **then**
        $\mathbf{x}_{B_{j^\star}} \leftarrow \mathrm{Repair}_\rho(B_{j^\star}; \gamma; K^{\text{refill}})$
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{j^\star\}$
      end if
    end if
  end if
  $\mathbf{x}_{B_k} \leftarrow \tilde{\mathbf{x}}_{B_k}$
end for
return $\mathbf{x}$

C DECISION VALUE AND THE SCOPE OF BLOCK REPAIR

We first prove the quantitative connection between drift and local decision value, then extend it to the committed text and establish the separation between joint and partial remasking. All logarithms are natural, and all argmax operations use a fixed deterministic tie-breaking rule. The first two results apply directly to the finite per-position distributions in Equation (3); they do not require these distributions to be conditionals of a common joint model. The block separation uses an explicit joint distribution to isolate the effect of the masking unit.

C.1 PROOF OF PROPOSITION 4.1

Notation and proof strategy. Fix a nonempty past block B_j and write $p_i = p^{\text{pre}}_{j,i}$, $q_i = p^{\text{post}}_{j,i}$, $a_i = \arg\max_v p_i(v)$, and $b_i = \arg\max_v q_i(v)$. Let $D_i = D_{\mathrm{JS}}(p_i \| q_i)$ and $\Delta_i = q_i(b_i) - q_i(a_i)$. We bound decision advantages by total variation, relate total variation to JS using Pinsker's inequality, and then average over the block.

Proof. Step 1: Relating JS to total variation. For any two distributions p, q on the finite vocabulary, let $M = (p+q)/2$ and

$$T = \mathrm{TV}(p, q) = \frac{1}{2} \sum_{v \in \mathcal{V}} |p(v) - q(v)|.$$

Then $\mathrm{TV}(p, M) = \mathrm{TV}(q, M) = T/2$. Pinsker's inequality with natural logarithms states that $D_{\mathrm{KL}}(u \| v) \geq 2\,\mathrm{TV}(u, v)^2$. Applying it to (p, M) and (q, M) gives

$$D_{\mathrm{JS}}(p \| q) = \tfrac{1}{2} D_{\mathrm{KL}}(p \| M) + \tfrac{1}{2} D_{\mathrm{KL}}(q \| M) \geq \tfrac{1}{2} T^2, \qquad T \leq \sqrt{2 D_{\mathrm{JS}}(p \| q)}. \tag{10}$$

Both KL divergences are finite because $M(v)$ is positive wherever either input has positive mass; zero-mass terms are interpreted by continuity.

Step 2: Bounding the value of an updated decision. For a fixed position, suppress i and write $a = \arg\max p$ and $b = \arg\max q$. Since $q(b) \geq q(a)$, we have $\Delta \geq 0$. Since $p(a) \geq p(b)$ and $|q(v) - p(v)| \leq T$ for each token v,

$$\begin{aligned} \Delta &= q(b) - p(b) + p(b) - p(a) + p(a) - q(a) \\ &\leq |q(b) - p(b)| + |p(a) - q(a)| \leq 2T \leq \sqrt{8 D_{\mathrm{JS}}(p \| q)}. \end{aligned} \tag{11}$$

Averaging and using concavity of the square root yields

$$0 \le \overline{\Delta}_j \le \frac{1}{|B_j|} \sum_{i \in B_j} \sqrt{8D_i} \le \sqrt{\frac{8}{|B_j|} \sum_{i \in B_j} D_i} = \sqrt{8r_j},$$

which proves Equation (6).

Step 3: Incorporating the original decision margin. If $b \neq a$, the margin $m = p(a) - \max_{v \neq a} p(v)$ is defined because $|\mathcal{V}| \ge 2$, and $p(a) - p(b) \ge m$. Consequently,

$$m + \Delta \le p(a) - p(b) + q(b) - q(a) = (p(a) - q(a)) + (q(b) - p(b)) \le 2T.$$

The two sides are nonnegative, so Equation (10) implies $D_{\mathrm{JS}}(p\|q) \ge T^2/2 \ge (m + \Delta)^2/8$, proving Equation (7). The inequalities also cover zero margins and zero decision advantages; if $p = q$, then $a = b$, $\Delta = 0$, and the first bound holds with equality. □

Decision interpretation. Under per-position target distributions q_i, the expected normalized Hamming loss of a deterministic block $\mathbf{z}$ is

$$\mathcal{L}_q(\mathbf{z}) = \frac{1}{|B_j|} \sum_{i \in B_j} [1 - q_i(z_i)]. \tag{12}$$

Thus $\overline{\Delta}_j = \mathcal{L}_q(\mathbf{a}) - \mathcal{L}_q(\mathbf{b})$. This additive loss depends only on the marginals: no independence assumption is needed. The result quantifies local value under the model's audit distributions; task-level correctness is evaluated by the reasoning and code benchmarks. In particular, $r_j \le \tau$ implies $\overline{\Delta}_j \le \sqrt{8\tau}$, and $\overline{\Delta}_j > \varepsilon$ implies $r_j > \varepsilon^2/8$. These are bounds on the opportunity created by a context update, rather than a claim that every large drift produces a large advantage.

Why uncertainty change can miss a valuable revision. Consider $p = (1-\eta, \eta)$ and $q = (\eta, 1-\eta)$ for $0 < \eta < 1/2$. Their entropies are identical, yet their preferred tokens differ and $\Delta = 1 - 2\eta$. Moreover, $D_{\mathrm{JS}}(p\|q) = \log 2 - H(p)$ approaches $\log 2$ as η tends to zero. Thus entropy delta can vanish even when the post-context advantage of changing the decision approaches one. Conversely, $p = (0.5, 0.3, 0.2)$ and $q = (0.5, 0.2, 0.3)$ have positive JS but the same preferred token, giving $\Delta = 0$. Together, these examples distinguish distributional movement, uncertainty change, and decision value.

C.2 Value of Revisiting the Actual Committed Tokens

The pre-context audit decision a_i may differ from the committed token x_i, because the context used for auditing a past block can already include blocks that were absent when it was first generated. Define the committed block's pre-context decision gap and its post-context revision value by

$$e_j = \frac{1}{|B_j|} \sum_{i \in B_j} [p_i(a_i) - p_i(x_i)], \qquad g_j = \frac{1}{|B_j|} \sum_{i \in B_j} [q_i(b_i) - q_i(x_i)]. \tag{13}$$

Both are nonnegative and can be computed from the audit distributions without additional model evaluations.

Corollary C.1 (Revision value of the committed text)**.** *Under the assumptions of Proposition 4.1, for any committed* $\mathbf{x}_{B_j} \in \mathcal{V}^{|B_j|}$,

$$0 \le g_j \le e_j + \sqrt{8r_j}. \tag{14}$$

Furthermore, $g_j = \mathcal{L}_q(\mathbf{x}_{B_j}) - \mathcal{L}_q(\mathbf{b})$ *is the largest decrease in the frozen post-context loss Equation* (12) *achievable by replacing the block with a deterministic token vector.*

Proof. Let $T_i = \mathrm{TV}(p_i, q_i)$. At each position, $p_i(b_i) \le p_i(a_i)$, so

$$\begin{aligned} q_i(b_i) - q_i(x_i) &= [q_i(b_i) - p_i(b_i)] + [p_i(b_i) - p_i(x_i)] + [p_i(x_i) - q_i(x_i)] \\ &\le p_i(a_i) - p_i(x_i) + 2T_i. \end{aligned}$$

Averaging this inequality and applying Equation (10) and concavity of the square root gives Equation (14). For any replacement $\mathbf{z}$, $q_i(z_i) \le q_i(b_i)$ at every position, hence $\mathcal{L}_q(\mathbf{z}) \ge \mathcal{L}_q(\mathbf{b})$. Substitution in Equation (12) proves the exact expression and optimality of g_j. □

Connection to refill. The refill input $\mathbf{x}^{\mathrm{mask}}$ equals the post-context probe input $\mathbf{y}_j^{\mathrm{post}}$ of the selected block, so the first refill pass reads exactly the distributions q_i. If all post-context marginal argmaxes pass the confidence threshold in Equation (9), that pass outputs $\mathbf{b}$ and realizes g_j under this frozen audit loss. If only a subset $A \subseteq B_j$ passes, the accepted positions contribute $|B_j|^{-1} \sum_{i \in A} [q_i(b_i) - q_i(x_i)]$ to that loss decrease; the remaining tokens are determined by the residual decoding schedule under its updated context. The decomposition separates an existing reason to revisit the committed block, e_j, from the additional opportunity associated with the new draft's drift. The diagnostic g_j also distinguishes large distributional changes that leave the committed choices optimal from changes that favor revising them; it is an analysis quantity and does not alter the gate in Equation (5).

C.3 Proof of Proposition 4.2

Setup and proof strategy. We condition throughout on a fixed external context and consider a block of $\ell \geq 2$ binary tokens. For a remasked subset S, a greedy update predicts the marginals $Q(X_i \mid \mathbf{X}_{\bar{S}} = \mathbf{x}_{\bar{S}})$ for $i \in S$ and commits their argmaxes simultaneously. All such predictions in this construction are exact conditionals of Q. We construct a distribution with two distinguished configurations: partial remasking retains a zero that excludes $\mathbf{1}$, whereas full remasking makes the probability mass on $\mathbf{1}$ available at every position.

Proof. Step 1: A strictly positive family. For any prescribed $\ell \geq 2$ and $\gamma \in [0, 1)$, choose

$$\eta = \min\left\{\frac{1-\gamma}{5}, \frac{1}{10}\right\} > 0, \qquad h = \frac{\eta}{2^\ell - 2},$$

and define

$$Q(\mathbf{x}) = \begin{cases} 1 - 3\eta, & \mathbf{x} = \mathbf{1}, \\ 2\eta, & \mathbf{x} = \mathbf{0}, \\ h, & \mathbf{x} \notin \{\mathbf{0}, \mathbf{1}\}. \end{cases} \tag{15}$$

The probabilities sum to one and are all positive. Since $\eta \leq 1/10$, $Q(\mathbf{1}) \geq 7/10$, $Q(\mathbf{0}) \leq 1/5$, and $h \leq 1/20$; thus $\mathbf{1}$ is the unique global maximizer.

Step 2: Every proper-subset update stays at the old configuration. Suppose the current block is $\mathbf{0}$, and remask any subset S with $1 \leq s := |S| < \ell$. At least one zero remains visible, excluding $\mathbf{1}$ from the conditional support. The support consists of $\mathbf{0}$, of mass 2η, and $2^s - 1$ other configurations, each of mass h. For any $i \in S$, exactly 2^{s-1} of those configurations have $X_i = 1$, so

$$Q(X_i = 1 \mid \mathbf{X}_{\bar{S}} = \mathbf{0}) = \frac{2^{s-1}h}{2\eta + (2^s - 1)h} < \frac{1}{2}. \tag{16}$$

The strict inequality is equivalent to $h < 2\eta$, which holds because $2^\ell - 2 \geq 2$. Every conditional marginal argmax is therefore zero, and the block remains $\mathbf{0}$. Induction over the updates proves that any finite sequence of such operations, with arbitrary or adaptive proper-subset choices, remains at $\mathbf{0}$. In particular, repeated single-token greedy updates cannot reach $\mathbf{1}$.

Step 3: Full remasking enables a single confident refill. When all positions are masked, the marginal at any position receives mass $1 - 3\eta$ from $\mathbf{1}$ and mass $(2^{\ell-1} - 1)h = \eta/2$ from the remaining configurations with $X_i = 1$. Hence

$$Q(X_i = 1) = 1 - \frac{5}{2}\eta \geq \frac{3}{4} > \frac{1}{2}, \qquad Q(X_i = 1) \geq \frac{1+\gamma}{2} > \gamma. \tag{17}$$

All marginal argmaxes are one and all pass the prescribed refill threshold. One parallel pass therefore commits $\mathbf{1}$, completing the proof. □

Mechanism and prediction. The separation arises with the same distribution, the same external context, and exact predictions in both update procedures. Its source is the set of old commitments retained as conditioning variables. It establishes an expressivity advantage of joint remasking for coupled revisions, including under arbitrarily high refill thresholds strictly below one. For a selected repair span, the construction predicts that the ability to cover mutually dependent positions can

matter more than repeating greedy updates on smaller spans. This suggests comparing repair units at matched compute on examples grouped by whether the required changes are isolated or coordinated; the proposition itself is an analytical construction, not an empirical claim about the benchmark examples.

D Additional Experimental Details

D.1 Hardware Resources

All experiments were conducted on a server equipped with two Intel Xeon Platinum 8358P CPUs (2.6 GHz), two NVIDIA A800 GPUs (80 GB each), and 1 TB of RAM.

D.2 Baseline Settings

We follow the default settings reported in the original papers or official implementations for all test-time scaling baselines.

- **dVoting.** We use the default dVoting configuration with entropy-threshold parallel decoding. Following the original setup, we set $\alpha = 0.3$, sampling upper bound $n = 5$, and temperature 0.6. Early stopping is applied when the voted answer converges.
- **HEX.** We follow the default HEX ensemble setting. Specifically, we sample five trajectories for each block size in $\{8, 16, 32, 64, 128\}$, resulting in 25 trajectories in total. We use temperature 0.9 and follow the original tie-breaking rule, which selects the answer generated with the smallest block size when multiple answers have the same vote count.
- **Prism.** We report the $K = 4$ setting from Prism. The remaining hyperparameters follow the paper's default configuration, including pruning window $W = [0.1, 0.6]$, pruning interval $i = 3$, survivor count $S = 4$, and decay factor $d = 1.8$.

D.3 dLLM Plain Inference Settings

The plain block-wise decoders of the additional dLLMs in Section 5.4 use the following settings; all settings not listed follow each model's official recommended configuration.

- **LLaDA-1.5.** We use generation length $L = 256$ and block length $\ell = 16$ with temperature 0.
- **Dream-v0-Instruct-7B.** We use generation length $L = 256$ and block length $\ell = 16$ with temperature 0.
- **LLaDA2.1-mini.** We set the maximum generation length to 1024 tokens, the block length to 32, the confidence threshold to 0.5, and the temperature to 0, and we enable early stopping at the end-of-sequence token.

D.4 Block Reflection Setting

We implement Block Reflection based on the `dllm` package (Zhou et al., 2026) and use inference with temperature 0.0 for all reported results. For the main experiments, we search over a small set of inference hyperparameters. Specifically, we consider refill budgets $K^{\text{refill}} \in \{1, 2, 4\}$, JS-divergence thresholds $\tau_{\text{JS}} \in \{0.05, 0.1, 0.2, 0.4\}$, and entropy-delta thresholds $\tau_{\Delta H} \in \{0.05, 0.1, 0.2\}$. Unless otherwise specified, the remaining decoding settings follow the default configuration of the underlying block-wise dLLM decoder. Drift is computed with natural logarithms after clamping probabilities at 10^{-12}, and positions of B_j holding end-of-sequence or padding tokens are excluded from the average in Equation (4).

E Example

Here we present Block Reflection outputs for four benchmarks for LLaDA model.

Math-500 Result Example

Question:

Compute: $1−2+3−4+5− \dots +99−100$.

Answer:

First, observe the pattern in the sequence: \(1 − 2 + 3 − 4 + 5 − 6 + \dots + 99 − 100\).

We can group the terms in pairs:
\[(1 − 2) + (3 − 4) + (5 − 6) + \dots + (99 − 100)\]

Each pair sums to \(−1\):
\[(1 − 2) = −1\]
\[(3 − 4) = −1\]
\[(5 − 6) = −1\]
\[\vdots\]
\[(99 − 100) = −1\]

There are 50 pairs in the sequence. Therefore, the sum of all these pairs is:
\[50 \times (−1) = −50\]

Thus, the sum of the sequence \(1 − 2 + 3 − 4 + 5 − 6 + \dots + 99 − 100\) is
\[\boxed{−50}\]

GSM8K Result Example

Question:

Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?

Answer:

To determine how much Janet makes every day at the farmers' market, we need to follow these steps:

1. Calculate the total number of eggs laid by Janet's ducks per day.
2. Subtract the number of eggs Janet eats and bakes from the total number of eggs laid.
3. Determine the number of eggs she sells at the farmers' market.
4. Calculate the revenue from selling the eggs.

1. The total number of eggs laid by Janet's ducks per day is 16.
2. Janet eats 3 eggs for breakfast and bakes 4 eggs for her friends. Therefore, she eats and bakes a total of \(3 + 4 = 7\) eggs.
3. The number of eggs she sells at the farmers' market is the total number of eggs laid minus the number of eggs she eats and bakes, which is \(16 − 7 = 9\) eggs.
4. Janet sells each egg for $2. Therefore, the revenue from selling 9 eggs is \(9 \times 2 = 18\) dollars.

Thus, the amount Janet makes every day at the farmers' market is \boxed{18} dollars.

MBPP Result Example

Question:

Write a function to find the perimeter of a square.

Answer:

```python
def square_perimeter(side):
    return 4 * side

# Test cases
assert square_perimeter(10) == 40
assert square_perimeter(5) == 20
assert square_perimeter(4) == 16
```

HumanEval Result Example

Question:

```python
from typing import List


def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
```

Answer:

```python
from typing import List


def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
    for i in range(len(numbers)):
        for j in range(i + 1, len(numbers)):
            if abs(numbers[i] - numbers[j]) < threshold:
                return True
    return False
```
````

#### Canonical Solution

```python
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold:
                    return True

    return False
```
```