BEYOND REPLACEMENT: LEVENSHTEIN EDITING FOR DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models (DLMs) offer a flexible alternative to autoregressive models, enabling any-order infilling without specialized prompting. While most existing decoding methods treat decoded tokens as fixed, some recent models allow previously decoded tokens to be revised only by replacing them with alternative tokens, a paradigm we refer to as *replacement-only editing* (ROE). Although ROE can correct individual tokens, it remains inefficient for sequence shifts caused by missing or extra tokens and struggles when errors occur across many positions. To address these limitations, we propose LED (**L**evenshtein **E**diting for **D**iffusion Language Models), a post-training framework that extends ROE with insertion and deletion, enabling Levenshtein-style refinement of generated sequences. It uses two key training designs. First, it reconstructs edit supervision on the fly during supervised fine-tuning using longest-common-subsequence alignment, enabling the model to learn insertion and deletion operations without manually annotated labels. Second, it adopts multi-stage training, where the state from one stage becomes the input to the next, exposing the model to its own earlier errors and enabling it to learn how to correct them. When applied to the 16B-parameter LLaDA2.1-Mini Base, LED raises the average score across twelve benchmarks from 54.86 to 57.10 under confidence-guided decoding while increasing tokens per forward from 6.14 to 7.05, whereas under confidence-free decoding it raises the average score from 27.52 to 47.19. We further show that, in this setting, LED achieves higher pass@k at large k than thresholded decoding without editing, demonstrating improved generation diversity and revealing the diversity potential of DLMs without confidence guidance. LED also improves robustness to large block sizes, raising the average score from 36.97 to 51.34 at block size 128. Together, these results suggest that LED supports more robust self-correction in DLMs.

1 INTRODUCTION

Masked diffusion language models (DLMs) generate text over successive denoising rounds by repeatedly filling masked positions with predicted tokens, allowing multiple positions within a block to be decoded in parallel (Sahoo et al., 2024; Shi et al., 2024; Nie et al., 2025b; Ye et al., 2025; Nie et al., 2025a; Gong et al., 2025). We call this progressive mask-filling procedure mask-to-token (M2T) generation. However, standard M2T generation cannot revisit an earlier prediction: once a token is decoded, later rounds use it as context but do not revise it. Consequently, an error made early in decoding can affect subsequent predictions and remain in the final output.

Recent methods address this limitation by remasking or editing previously decoded tokens (Wang et al., 2025; Huang et al., 2026; Zhai et al., 2026; Wu et al., 2026b; Hong et al., 2026; Frković et al., 2026; Jeon et al., 2026). Among these efforts, LLaDA2.1 is a prominent large-scale realization, with 16B Mini and 100B Flash variants (Bie et al., 2026). Beyond M2T generation, it introduces *token-to-token* (T2T) editing, which allows the model to replace an erroneous decoded token with a corrected token in a denoising round after its initial decoding. We refer to this operation as *replacement-only editing* (ROE). By enabling such corrections, ROE supports more aggressive parallel drafting, improving the trade-off between generation quality and decoding speed.

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.

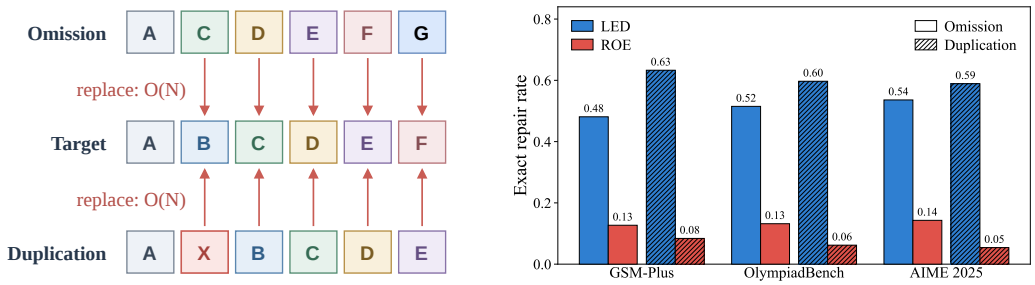


Figure 1: Sequence misalignment and repair. *Left:* Schematic illustration of sequence misalignment. The center row is the target sequence, and the outer rows show fixed-length corrupted blocks after one token is omitted or duplicated, shifting the suffix and displacing a token at the block boundary. *Right:* Exact repair rate under single-token sequence misalignment. The ROE baseline achieves low repair rates, whereas LED yields substantially higher rates. Experimental details are provided in Appendix A.2.

Despite these benefits, ROE exhibits two failure modes that limit its editing capability. First, ROE cannot efficiently correct sequence misalignment, which arises when token omission or duplication shifts the remaining suffix relative to the target sequence. This inefficiency stems from the fact that ROE can only replace tokens at their current positions, so realigning a suffix of length N may require replacing all N tokens, as illustrated in the left panel of Figure 1. This $\mathcal{O}(N)$ replacement cost makes reliable correction difficult and can consume additional denoising rounds, limiting both generation quality and decoding speed. The right panel of Figure 1 empirically illustrates this difficulty. Second, ROE remains ineffective when errors occupy a large portion of the sequence. To expose this failure mode, we decode every masked position in a block in a single forward pass without confidence filtering, after which the decoder must rely solely on ROE to repair the resulting draft. The resulting performance drops substantially (Section 4.3), indicating that ROE cannot reliably recover from widespread errors.

To address these limitations, we propose **L**evenshtein **E**diting for **D**iffusion Language Models (LED), a post-training framework that extends ROE with insertion and deletion, enabling Levenshtein-style refinement of generated sequences. LED models insertion and deletion by introducing a special output token for each operation. The training procedure in LED has two key designs. First, LED reconstructs edit supervision online via longest-common-subsequence (LCS) alignment, enabling the new edit actions to be learned from standard supervised fine-tuning (SFT) data. Second, LED constructs multi-stage training trajectories in which later stages receive states partly formed by model predictions from earlier stages. This exposes the model to its own intermediate errors and makes the training states more similar to those encountered during inference.

We post-train LLaDA2.1-Mini Base with LED for 3.7k H200 GPU-hours to obtain LLaDA-LED. Under confidence-guided decoding, which decodes only masked positions whose confidence exceeds a predefined threshold, LLaDA-LED raises the average score from 54.86 to 57.10 across twelve generation benchmarks while improving decoding parallelism (Section 4.2). Under confidence-free decoding, every masked position is decoded in a single forward pass without confidence filtering, so the model must rely solely on its own editing operations in later rounds to correct errors in the resulting draft. In this setting, LLaDA-LED substantially raises the average score from 27.52 to 47.19 (Section 4.3). This substantial gain indicates that LED markedly strengthens DLM self-correction when drafts contain widespread errors.

Beyond strengthening self-correction, combining LED with confidence-free decoding also addresses a limitation of confidence-guided decoding. As the most commonly used decoding strategy for DLMs, confidence-guided decoding commits the most confident positions first and defers uncertain ones. Prior work observes that this preference postpones high-entropy positions that may be crucial for reaching a correct answer (Ni et al., 2026; Olausson et al., 2026). We call this tendency to defer high-entropy positions the *low-entropy bias*. Simply removing confidence filtering is insufficient, because errors introduced at uncertain positions can propagate through subsequent rounds and sharply degrade output quality. By repairing these errors, LED mitigates the quality loss associated with confidence-free decoding. Consistent with this account, LLaDA-LED with confidence-free de-

coding achieves higher pass@k at large k than thresholded decoding without editing (Section 4.4). This result shows that pairing LED with confidence-free decoding can turn higher-entropy exploration into broader solution coverage, suggesting a viable path toward confidence-free decoding. Beyond these results, we evaluate the robustness of LED to larger block sizes (Section 4.6) and offer an entropy-based interpretation of its editing behavior. The main contributions of this work are three-fold.

- We identify two limitations of replacement-only editing in DLMs: difficulty correcting sequence misalignments and limited ability to recover from widespread errors in intermediate states.
- We propose LED, a post-training framework that equips DLMs with insertion and deletion capabilities through online reconstruction of LCS-based edit supervision and inference-aligned multi-stage training.
- We validate LED across twelve benchmarks and identify three benefits. With confidence guidance, it raises the average score while increasing tokens per forward. Under confidence-free decoding, it improves pass@k at large sampling budgets, indicating broader solution coverage. It also improves robustness to larger block sizes.

2 Background

We only consider the block diffusion setting in this paper, in which a DLM generates text autoregressively in fixed-size blocks while predicting the tokens within each block in parallel over iterative denoising rounds (Sahoo et al., 2024; Shi et al., 2024; Arriola et al., 2025). Let $x = (x_1, \ldots, x_B) \in \mathcal{V}^B$ denote a clean token block of length B, where $\mathcal{V}$ is the model's token vocabulary and $[\mathrm{Mask}] \notin \mathcal{V}$ is a special mask token. During training, the forward process progressively masks the clean block by replacing selected tokens with $[\mathrm{Mask}]$. Given a masking rate $t \in [0, 1]$, we independently sample $m_i^t \sim \mathrm{Bernoulli}(t)$ and construct the partially masked block $\tilde{x}^t$ as

$$\tilde{x}_i^t = \begin{cases} [\mathrm{Mask}], & m_i^t = 1, \\ x_i, & m_i^t = 0, \end{cases} \qquad i = 1, \ldots, B. \tag{1}$$

The model is then trained to recover the original tokens at the masked positions conditioned on the partially masked block $\tilde{x}^t$. At decoding round r, let $s^r = (s_1^r, \ldots, s_B^r) \in (\mathcal{V} \cup \{[\mathrm{Mask}]\})^B$ denote the current block state. We define the sets of masked and non-masked positions as

$$\mathcal{M}_r = \{i \in \{1, \ldots, B\} : s_i^r = [\mathrm{Mask}]\}, \qquad \mathcal{P}_r = \{i \in \{1, \ldots, B\} : s_i^r \in \mathcal{V}\}. \tag{2}$$

When discussing decoding strategies, a common choice is confidence-based decoding, which commits masked positions whose predicted tokens are sufficiently reliable after each forward pass. Let $\hat{v}_i^r$ denote the token predicted at masked position i in round r, with confidence $p_\theta(\hat{v}_i^r \mid c, s^r, i)$. Given a confidence threshold τ, the decoder commits the masked positions whose confidence exceeds τ, i.e., $\{i \in \mathcal{M}_r : p_\theta(\hat{v}_i^r \mid c, s^r, i) \geq \tau\}$. If no position satisfies the threshold, the highest-confidence masked position can be selected to ensure progress. When $\tau = 0$, all masked positions are decoded in one forward pass without confidence-based selection, which we refer to as confidence-free decoding in this paper.

3 Methodology

This section presents LED's framework for Levenshtein-style editing in DLMs. It combines token replacement, insertion, and deletion, the three basic operations of Levenshtein editing. We first describe how the model represents these operations as position-aligned actions (Section 3.1). We then describe how the model learns to predict these actions within a standard SFT pipeline, with LCS alignment providing the edit supervision (Section 3.2). Finally, we introduce multi-stage training to narrow the training–inference gap in learning insertion and deletion (Section 3.4).

3.1 Modeling Insertion and Deletion

Insertion and deletion are enabled by extending the output vocabulary with two special tokens, $[\mathrm{Del}]$ and $[\mathrm{Ins}]$, so that $\mathcal{V}_{\mathrm{out}} = \mathcal{V} \cup \{[\mathrm{Del}], [\mathrm{Ins}]\}$. Both tokens are excluded from the input vocabulary.

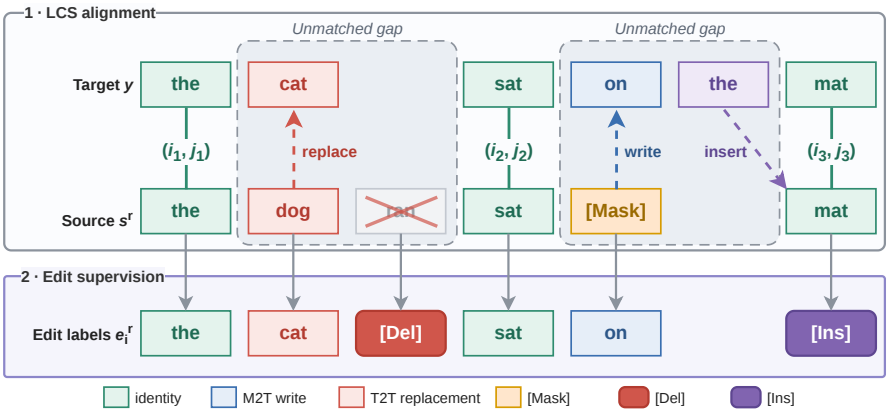


Figure 2: LCS-based edit-target construction. Green links denote matched anchor pairs (i_k, j_k). Within each unmatched gap, surplus source positions receive [Del], and at most one surplus target token is handled by assigning [Ins] to the matched source anchor immediately to its right.

Given the input s^r at denoising round r, the model predicts one token from $\mathcal{V}_{\text{out}}$ at each position. At each selected position i, a prediction of [Del] removes s_i^r, while a prediction of [Ins] inserts a fresh [Mask] immediately before it. As these edits can change the sequence length, the resulting sequence is truncated to B tokens or right-padded with [Mask] tokens before serving as the input for round $r+1$.

3.2 LCS-Based Edit Supervision

Standard SFT data does not provide direct supervision signals for insertion and deletion. To train the special tokens [Del] and [Ins], we reconstruct position-wise edit targets online during the SFT stage. Given a current source state s^r and its corresponding clean target $y = (y_1, \ldots, y_B) \in \mathcal{V}^B$, the reconstruction assigns each selected source position i an edit target $e_i^r \in \mathcal{V} \cup \{[\text{Del}], [\text{Ins}]\}$. The reconstruction therefore enables insertion and deletion to be learned within the standard SFT pipeline without manually annotated edit trajectories.

We construct the edit targets by aligning s^r with y through longest-common-subsequence (LCS) matching. Let $\pi^\star = \{(i_k, j_k)\}_{k=1}^K$ be a maximum-cardinality monotone matching of source and target indices, where $i_k, j_k \in \{1, \ldots, B\}$ and $s_{i_k}^r = y_{j_k} \in \mathcal{V}$ for each matched pair. We leave the supervision signals at these matched positions unchanged, thereby preserving as much correct information in the source state as possible and concentrating edit supervision on the unmatched regions. Figure 2 illustrates the matched anchors and the resulting edit targets.

The matched anchors partition the unmatched source and target positions into $K+1$ ordered gaps. For convenience, let $i_0 = j_0 = 0$ and $i_{K+1} = j_{K+1} = B+1$, and define the source and target positions in gap k as $I_k = \{i_k + 1, \ldots, i_{k+1} - 1\}$ and $J_k = \{j_k + 1, \ldots, j_{k+1} - 1\}$, respectively, for $k = 0, \ldots, K$. As described above, for each matched pair (i_k, j_k), we keep its supervision unchanged by setting $e_{i_k}^r = y_{j_k}$, unless the source anchor is used to supervise an insertion as described below. Within each gap, let $\ell_k = \min\{|I_k|, |J_k|\}$. We first pair the first ℓ_k source and target positions in order and assign $e_{I_k[p]}^r = y_{J_k[p]}$ for $p = 1, \ldots, \ell_k$, where $I_k[p]$ and $J_k[p]$ denote the p-th positions in the corresponding ordered gaps. If $|I_k| > \ell_k$, the remaining source positions correspond to surplus tokens and are assigned deletion targets, i.e., $e_{I_k[p]}^r = [\text{Del}]$ for $p = \ell_k + 1, \ldots, |I_k|$. If $|J_k| > \ell_k$, the remaining target positions require insertion; we therefore assign $e_{i_{k+1}}^r = [\text{Ins}]$ to the matched source anchor immediately to the right, provided that $k < K$. Each gap produces at most one [Ins] target per round.

The reconstruction is performed independently for every block. Exact LCS alignment costs $\mathcal{O}(B^2)$ time for a block of fixed length B, giving a total cost of $\mathcal{O}(N_{\text{blk}} B^2)$ where N_{blk} is the number of blocks. The online cost therefore grows linearly with the number of blocks when B is fixed. This

makes it practical to integrate insertion and deletion supervision into standard SFT. Appendix D provides the complete gap-level rules and Algorithm 2.

3.3 Edit-Augmented Inference

Our inference procedure follows the confidence-based decoding strategy in Section 2. At round r, the decoder selects masked positions according to threshold τ, using the highest-confidence masked position as a fallback when no position meets the threshold. We denote this set by $\mathcal{C}_r(\tau)$. However, $\mathcal{C}_r(\tau)$ only determines which masked positions are decoded and does not allow already generated tokens to be revised. We therefore introduce an edit threshold ν for selecting non-masked positions. For each $i \in \mathcal{P}_r$, let $\hat{v}_i^r$ denote the model's predicted output, with confidence $\eta_i^r = p_\theta(\hat{v}_i^r \mid c, s^r, i)$. We select a non-masked position only when the model proposes an output different from its current token with confidence at least ν, yielding $\mathcal{E}_r(\nu) = \{i \in \mathcal{P}_r : \hat{v}_i^r \neq s_i^r,\ \eta_i^r \geq \nu\}$. The positions updated in round r are therefore $\mathcal{Q}_r = \mathcal{C}_r(\tau) \cup \mathcal{E}_r(\nu)$.

The decoder updates only the positions in $\mathcal{Q}_r$ and retains all remaining positions. Predictions of [Del] and [Ins] are processed as defined in Section 3.1.

3.4 Inference-Aligned Multi-Stage Training

During inference, the input to each round depends on the model's predictions in preceding rounds. In contrast, standard training constructs intermediate states through random noising rather than the model's own decoding process, leaving the model unexposed to the states it actually encounters at inference time. To narrow this training–inference gap, we construct multi-stage training trajectories that mimic the inference process.

Specifically, we perform multiple forward passes on each training example. In the input to the first stage, all response tokens are masked, whereas in the input to the final stage, all response tokens have been decoded. The output of each stage influences the input to the subsequent stage. Figure 9 illustrates this multi-stage construction, and Appendix D provides the complete procedure.

In practice, using the model output from one stage directly as the input to the next makes training challenging, especially early in training when its predictions are still unreliable. We therefore use mixed teacher forcing to construct stable trajectories while still exposing the model to its own predictions. At stage r, LCS provides labels $e_i^r \in \mathcal{V}_{\text{out}}$ for $\mathcal{Q}_r = \mathcal{U}_r \cup \mathcal{P}_r$. For each selected masked position $i \in \mathcal{U}_r$, we randomly mix its LCS-derived label and the model prediction when forming the next-stage input. For non-masked positions, we set $\tilde{v}_i^r = e_i^r$. We apply the selected predictions to form s^{r+1}, which serves as the input to the next stage. Using LCS-derived labels stabilizes the early stages, whereas using model predictions exposes later stages to model-generated inputs.

At each stage, we train the model to predict the LCS-derived target e_i^r at every selected position $i \in \mathcal{Q}_r$, conditioned on the context c and current response state s^r. We use the average negative log-likelihood over these selected positions:

$$\mathcal{L}_r = -\frac{1}{|\mathcal{Q}_r|} \sum_{i \in \mathcal{Q}_r} \log p_\theta(v = e_i^r \mid c, s^r, i). \tag{3}$$

Thus, each stage encourages the model to recover the edit targets specified by LCS from its current intermediate state. Normalizing by $|\mathcal{Q}_r|$ gives each stage equal weight regardless of its number of selected positions.

4 Experiments and Analysis

4.1 Setup

We post-train LLaDA2.1-Mini Base (Bie et al., 2026) on 745k SFT examples. The post-training run for LLaDA-LED consumes 3.7k H200 GPU-hours. The replacement-only baseline (w/o LED) and LLaDA-LED (w/ LED) use the same base model and data, differing only in whether LED is included in post-training. We set $R = 4$ and teacher forcing ratio to 0.5. Unless stated otherwise, evaluation uses a block size of 32 and a maximum generation length of 32,768. Appendix A.1 provides the

Table 1: Benchmark results under confidence-guided decoding with the balanced threshold $\tau = 0.5$. We compare the baseline (w/o LED) with our method (w/ LED). Higher score and TPF are better. Bold indicates the higher value within each metric pair.

Category	Benchmark	Score ↑		TPF ↑	
		w/o LED	w/ LED	w/o LED	w/ LED
Knowledge	MMLU-Pro	53.70	**55.33**	4.53	**5.60**
	GPQA-D	39.65	**40.91**	4.14	**5.74**
Reasoning	KOR-Bench	32.88	**44.00**	6.80	**8.94**
Math	GSM+	**83.57**	83.06	4.13	**4.67**
	Olympiad	**55.85**	54.11	5.19	**5.32**
	AIME 25	**30.00**	24.01	4.52	**5.02**
Coding	HumanEval+	76.22	**79.19**	**8.01**	7.49
	MBPP+	65.08	**70.90**	7.75	**8.78**
	LCB v6	13.88	**24.34**	8.32	**10.55**
Function Calling	BFCL v3	**65.39**	64.85	**9.15**	6.80
Inst. Following	IFEval	**69.69**	68.39	4.24	**6.03**
Chinese	C-Eval	72.36	**76.15**	6.97	**9.69**
Avg		54.86	**57.10**	6.14	**7.05**

remaining hyperparameters. We evaluate twelve benchmarks spanning seven capability categories and report per-benchmark scores, their unweighted average, and tokens per forward (TPF), where higher TPF indicates greater decoding parallelism. Appendix A.4 provides benchmark and scoring details.

4.2 Main Results under Confidence-Guided Decoding

Table 1 reports the main results under confidence-guided decoding with the default balanced threshold $\tau = 0.5$. The baseline and LLaDA-LED share the same base model and training data. They differ only in whether LED is applied during post-training.

At this setting, LED raises the unweighted average score across the twelve benchmarks from 54.86 to 57.10, a gain of 2.24 points, while increasing TPF from 6.14 to 7.05. The score improvement is task-dependent, with gains on seven of the twelve benchmarks, including the largest increases on KOR-Bench and LiveCodeBench v6. Together, these results indicate that the average score gain does not come at the expense of decoding parallelism. The joint improvement is consistent with the design of LED, in which insertion and deletion can repair local sequence misalignments without replacing an entire shifted suffix.

4.3 Confidence-Free Decoding

At $\tau = 0$, every masked position in a block is decoded in a single forward pass without confidence filtering, increasing within-block parallelism. However, decoding uncertain positions simultaneously can yield intermediate states with widespread errors, requiring subsequent rounds to correct them through editing operations.

As shown in Figure 3, the replacement-only baseline attains an average score of 27.52, revealing the difficulty of recovering from widespread errors with replacement-only editing. Applying LED improves the score on all twelve benchmarks. The average score increases from 27.52 to 47.19, a gain of 19.67 points. Per-benchmark gains range from $+0.98$ on BFCL v3 to $+34.07$ on HumanEval+. The largest improvements occur on HumanEval+, MBPP+, GSM+, and MMLU-Pro. Improvements across all twelve benchmarks span knowledge, reasoning, math, coding, function-

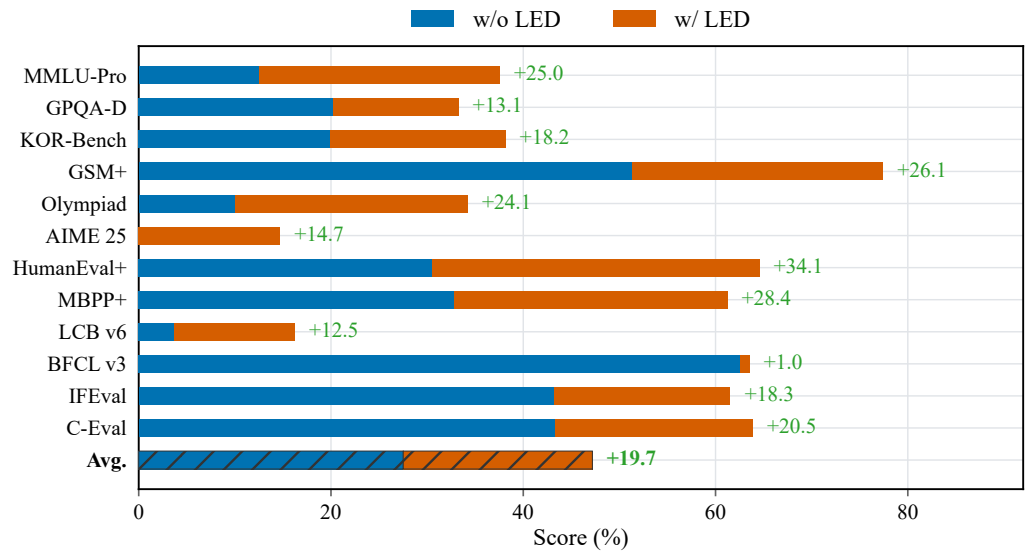


Figure 3: Per-benchmark scores under confidence-free decoding with threshold $\tau = 0$. Diagonal hatching highlights the average row.

calling, instruction-following, and Chinese evaluations, suggesting that LED helps recover from drafts containing widespread errors.

This result is consistent with the roles of insertion and deletion in repairing length and alignment errors, together with inference-aligned training that exposes the model to model-induced intermediate states. Moreover, these results support the claim that LED makes confidence-free decoding a more viable regime and motivate the solution-coverage analysis in Section 4.4.

4.4 Confidence-Free Decoding Broadens Solution Coverage

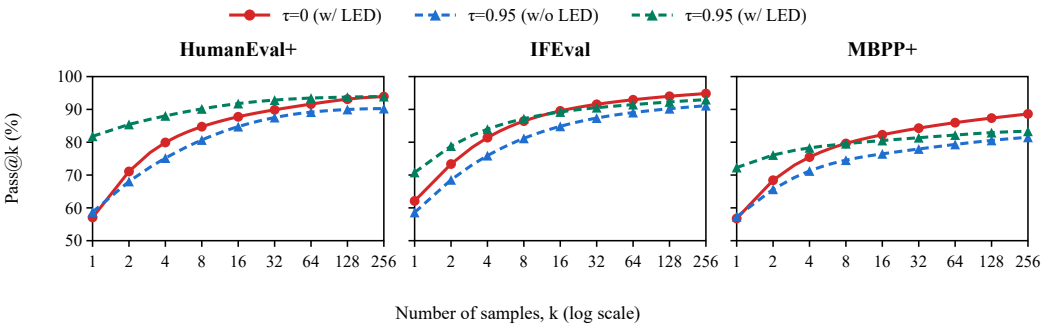


Figure 4: Pass@k under confidence-free and confidence-guided decoding on HumanEval+, IFEval, and MBPP+, using 256 samples per problem. Red denotes confidence-free decoding with LED ($\tau = 0$); blue and green denote confidence-guided decoding without editing and with LED, respectively ($\tau = 0.95$).

Confidence-guided decoding updates only masked positions whose predicted-token confidence exceeds the threshold τ. At a high threshold such as $\tau = 0.95$, it favors locally concentrated predictions and defers uncertain positions. This preference can cause independently sampled decoding trajectories to concentrate on similar regions of the solution space (Ni et al., 2026; Olausson et al., 2026). Confidence-free decoding removes this selection rule and resolves all masked positions in each round, thereby admitting decisions that would otherwise be postponed. However, simultaneously committing uncertain predictions can create noisy and locally misaligned drafts. LED addresses this

difficulty by enabling later rounds to revise token content and repair local sequence errors, allowing the decoder to retain potentially useful alternatives while correcting the errors introduced during parallel decoding. The remaining question is whether this repair capability converts the additional choices made under confidence-free decoding into useful solution coverage.

We evaluate this question using Pass@k, the probability that at least one of k sampled outputs is successful. Figure 4 compares confidence-free decoding with LED at $\tau = 0$ against confidence-guided decoding at $\tau = 0.95$, both without editing and with LED. These three configurations evaluate the contribution of editing under confidence-guided decoding and the effect of removing confidence-based position selection when editing is available. Each configuration generates 256 samples per problem.

At $\tau = 0.95$, adding LED improves Pass@256 over confidence-guided decoding without editing on all three benchmarks. Holding LED fixed, confidence-guided decoding achieves higher Pass@1, whereas confidence-free decoding catches up as the sampling budget increases. At $k = 256$, confidence-free decoding matches confidence-guided decoding with LED on HumanEval+ and exceeds it by 1.85 points on IFEval and 5.29 points on MBPP+, for an average gain of 2.38 points. These results show that editing supports both decoding regimes and that confidence-free decoding can broaden solution coverage when multiple samples are available.

4.5 Entropy-Based Analysis of LED's Editing Behavior

We examine whether LED later edits tokens that were uncertain when first decoded by M2T. Across 3,024 MBPP+ predictions, we compute each token's initial entropy over $\mathcal{V}_{\text{out}}$ as $H_i^r = -\sum_{v \in \mathcal{V}_{\text{out}}} p_\theta(v \mid c, s^r, i) \log p_\theta(v \mid c, s^r, i)$.

We trace each decoded token through subsequent operations and mark it as edited when it is modified in a later round. This definition accommodates insertions and deletions that change token positions.

Figure 5 compares the empirical cumulative distribution functions (CDFs) of initial predictive entropy for later-edited and never-edited tokens. The distribution for later-edited tokens is shifted toward higher entropy values, indicating that tokens selected for subsequent editing tend to have more uncertain initial output distributions. This comparison describes an association between initial predictive uncertainty and later editing.

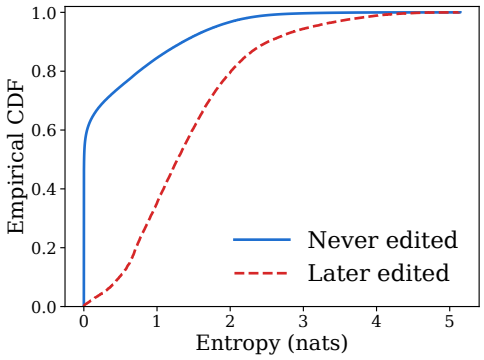


Figure 5: Empirical cumulative distribution functions (CDFs) of initial predictive entropy for MBPP+ tokens that are later edited and never edited.

4.6 Robustness to Large Block Sizes

Table 2: Comparison between LLaDA-LED and the baseline across different block sizes.

Metric	Method	Block size			
		32	64	128	256
Score	w/ LED	57.10	56.28	51.34	13.19
	w/o LED	54.86	53.86	36.97	6.45
	diff	**+2.24**	**+2.42**	**+14.37**	**+6.74**
TPF	w/ LED	7.05	7.85	8.63	8.09
	w/o LED	6.14	6.17	9.57	7.65
	diff	**+0.91**	**+1.68**	-0.94	**+0.44**

Larger blocks create longer within-block denoising chains, over which early local errors can affect subsequent predictions. We therefore evaluate robustness to test-time block-size expansion: both

compared models are trained with a block size of 32 and evaluated at larger block sizes without retraining. Table 2 shows modest gains from LED at block sizes 32 and 64, but a pronounced divergence at 128: LLaDA-LED attains an average score of 51.34, compared with 36.97 for the baseline. Although the baseline achieves a slightly higher TPF at this block size (9.57 versus 8.63), this additional parallelism coincides with a 14.37-point score deficit. At block size 256, both methods degrade substantially, yet LLaDA-LED retains an average score of 13.19 versus 6.45 for the baseline. The widening score gap under test-time block-size expansion is consistent with insertion and deletion alleviating the accumulation of local errors.

4.7 Grid Search over the Mask and Edit Confidence Thresholds

We sweep the mask confidence threshold $\tau \in \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ and the edit confidence threshold $\nu \in \{0, 0.2, 0.4\}$, with the block size fixed at 32. As defined in Sections 2 and 3.3, τ determines which masked positions are decoded in each round, whereas ν determines whether a non-masked position whose predicted output differs from its current token is selected for editing. Figure 7 in Appendix C shows a clear score–parallelism trade-off along the τ axis. Lower values of τ admit more masked positions per forward pass and yield higher TPF, but produce lower average scores. Increasing τ improves the average score while reducing TPF. In contrast, varying ν has a smaller effect on the aggregate metrics, with the largest score variation occurring at $\tau = 0$, where all masked positions are decoded without confidence filtering. For the default evaluation configuration, we use $\tau = 0.5$ as a balanced setting and set $\nu = 0.0$, which accepts every proposed non-identity edit.

5 Related Work

Diffusion language models and decoding. Masked DLMs have been scaled to large language models (Sahoo et al., 2024; Shi et al., 2024; Nie et al., 2025b; Ye et al., 2025). Recent work studies confidence-guided, learned, and block-wise decoding policies to balance generation quality and parallelism (Wu et al., 2026a; Jazbec et al., 2026; Song et al., 2025; Hu et al., 2026b).

Editing and refinement for sequence generation. Existing DLM revision methods remask or replace earlier predictions, whereas edit-based and flexible-length models use insertion and deletion (Wang et al., 2025; Huang et al., 2026; Zhai et al., 2026; Bie et al., 2026; Gu et al., 2019; Havasi et al., 2025; Wu et al., 2026b; Kim et al., 2025; Ding et al., 2026). Most closely related, ME-DLM applies sequence edits after completing a diffusion draft, with an additional next-token head and a three-stage training pipeline (Ren et al., 2026). LED instead learns token prediction, deletion, and insertion in one output vocabulary along the denoising trajectory, allowing alignment correction before block generation is complete while retaining the fixed-block interface of the base DLM. Detailed comparisons appear in Appendix B.

6 Conclusion and Future Work

We proposed LED, a post-training framework that augments replacement-only editing with deletion and insertion, using online LCS alignment and inference-aligned trajectories to teach a DLM to repair length and alignment errors. Across confidence-free decoding and large-block settings, LED substantially improves the average score, and controlled corruption probes support the role of length-changing edits in recovering from misaligned intermediate states. These results also suggest that confidence-free decoding merits further study: with LED, it achieves higher Pass@k than thresholded decoding without editing at large sampling budgets on the three evaluated benchmarks, indicating broader solution coverage. The current framework nevertheless constructs edit supervision locally at each denoising stage, so it does not explicitly optimize the quality or efficiency of the complete edit trajectory. Future work could therefore redesign insertion and deletion construction around path-level objectives and use trajectory-level learning, including reinforcement learning (Zhao et al., 2025; Hu et al., 2026a) or on-policy distillation (OPD) (Agarwal et al., 2024; Zhu et al., 2026), to learn when and how to coordinate these edits across denoising rounds. Such approaches may further improve the robustness and efficiency of confidence-free DLM decoding.

AI USE STATEMENT

We used a generative AI tool for language editing, literature search, and assistance with writing and running code, including code used to generate figures and tables. All cited works identified with AI assistance were verified against their original sources before citation. The authors reviewed all AI-assisted edits, verified the code and its outputs, and determined the research methodology, experimental design, analyses, scientific claims, and conclusions. We take full responsibility for the final content of this work.

ETHICS STATEMENT

This work post-trains a language model on an internally sourced supervised fine-tuning corpus. The corpus was processed with measures to protect personal information, and all required approvals for its use and the associated research procedures were obtained. We used the evaluation benchmarks in accordance with their respective terms of use and did not submit benchmark content to external services. We plan to release the training and inference code. Improvements in general text generation, code generation, and function calling may introduce downstream risks through misuse or unreliable outputs. This work does not include a dedicated evaluation of safety, fairness, or deployment impacts. Downstream applications should therefore undergo context-specific safety, privacy, fairness, and reliability assessments and include appropriate human oversight. The authors declare no competing interests.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024.

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

Tiwei Bie et al. LLaDA2.1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.

Fangyu Ding, Ding Ding, Sijin Chen, Kaibo Wang, Peng Xu, Zijin Feng, Haoli Bai, Kai Han, Youliang Yan, Binhang Yuan, and Jiacheng Sun. Beyond masks: Efficient, flexible diffusion language models via deletion-insertion processes. In *International Conference on Learning Representations*, 2026.

Stipe Frković, Metod Jazbec, Dan Zhang, Christian A. Naesseth, Ilija Bogunovic, and Eric Nalisnick. Re-evaluating confidence remasking in masked diffusion language models. In *International Conference on Machine Learning*, 2026.

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. Scaling diffusion language models via adaptation from autoregressive models. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/0fa81c3f0d57f95b8776de3a248ef0ed-Abstract-Conference.html.

Jiatao Gu, Changhan Wang, and Jake Zhao. Levenshtein transformer. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Marton Havasi, Brian Karrer, Itai Gat, and Ricky T. Q. Chen. Edit flows: Variable length discrete flow matching with sequence-level edit operations. In *Advances in Neural Information Processing Systems 38*, 2025. URL https://openreview.net/forum?id=FXWwYz1p8a.

Chaoqun He et al. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective DLLMs. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=435cce71b4007699041dfffa4f034079.

Yanzhe Hu, Yijie Jin, Pengfei Liu, Kai Yu, and Zhijie Deng. LightningRL: Breaking the accuracy-parallelism trade-off of block-wise dLLMs via reinforcement learning. *arXiv preprint arXiv:2603.13319*, 2026a.

Yuezhou Hu, Harman Singh, Monishwaran Maheswaran, Haocheng Xi, Coleman Hooper, Jintao Zhang, Aditya Tomar, Michael W. Mahoney, Sewon Min, Mehrdad Farajtabar, Kurt Keutzer, Amir Gholami, and Chenfeng Xu. Residual context diffusion language models. In *International Conference on Machine Learning*, 2026b.

Yuzhen Huang et al. C-Eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Zemin Huang, Yuhang Wang, Zhiyang Chen, and Guo-Jun Qi. Don't settle too early: Self-reflective remasking for diffusion language models. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5d8bc24ef15b7a808b6dc403c01583e2-Abstract-Conference.html.

Naman Jain et al. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Metod Jazbec, Theo X. Olausson, Louis Béthune, Pierre Ablin, Michael Kirchhof, João Monteiro, Victor Turrisi, Jason Ramapuram, and Marco Cuturi. Learning unmasking policies for diffusion language models. In *Proceedings of the 2nd DeLTa Workshop at ICLR 2026*, 2026. URL https://openreview.net/pdf/ed1a53031b4d32871acd6bf9a4707a7592c8e148.pdf.

Kijung Jeon, Thuy-Duong Vuong, and Molei Tao. VGB for masked diffusion model: Efficient test-time scaling for reward satisfaction and sample editing. *arXiv preprint arXiv:2606.28301*, 2026.

Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngotiaoco, Sitan Chen, and Michael Albergo. Any-order flexible length masked diffusion. *arXiv preprint arXiv:2509.01025*, 2025.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. GSM-Plus: A comprehensive benchmark for evaluating the robustness of LLMs as mathematical problem solvers. *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Kaijing Ma et al. KOR-Bench: Benchmarking language models on knowledge-orthogonal reasoning tasks. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=SVRRQ8goQo.

Mathematical Association of America. MAA invitational competitions: American invitational mathematics examination (AIME). Web page, 2025. URL https://maa.org/maa-invitational-competitions/.

Zanlin Ni, Shenzhi Wang, Yang Yue, Tianyu Yu, Weilin Zhao, Yeguo Hua, Tianyi Chen, Jun Song, Cheng Yu, Bo Zheng, and Gao Huang. The flexibility trap: Rethinking the value of arbitrary order in diffusion language models. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. In *International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=WNvvwK0tut.

Shen Nie et al. Large language diffusion models. In *Advances in Neural Information Processing Systems*, 2025b. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/48b383b24230e0e6e649d9c98dae4d8c-Abstract-Conference.html.

Theo X. Olausson, Metod Jazbec, Xi Wang, Armando Solar-Lezama, Christian A. Naesseth, Stephan Mandt, and Eric Nalisnick. A tale of two temperatures: Simple, efficient, and diverse sampling from diffusion language models. In *Conference on Language Modeling*, 2026.

Shishir G. Patil et al. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning (ICML)*, 2025.

David Rein et al. GPQA: A graduate-level google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Houxing Ren, Mingjie Zhan, Zimu Lu, Ke Wang, Yunqiao Yang, Haotian Hou, Junting Pan, and Hongsheng Li. Edit-based refinement for parallel masked diffusion language models. In *Forty-third International Conference on Machine Learning*, 2026.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=xcqSOfHt4g.

Yuerong Song et al. Sparse-dLLM: Accelerating diffusion LLMs with dynamic cache eviction. *arXiv preprint arXiv:2508.02558*, 2025.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/pdf/0c977a863006164f57b33a6e638f6d83d52bf588.pdf.

Yubo Wang et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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. In *International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=3Z3Is6hnOT.

Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Shansan Gong, Yansong Feng, Zhenguo Li, Wei Bi, Guorui Zhou, and Lingpeng Kong. DreamOn: Diffusion language models for code infilling beyond fixed-size canvas. In *International Conference on Learning Representations*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/7c020dd308d7413b5078686723b2f857-Abstract-Conference.html.

Yixin Ye et al. Dream: On the power of diffusion language models. *arXiv preprint arXiv:2508.15487*, 2025.

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

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. In *Advances in Neural Information Processing Systems*, 2025. doi: 10.52202/085713-1898. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/52190a0362148d179f1cbd9080956872-Abstract-Conference.html.

Jeffrey Zhou et al. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Huaisheng Zhu, MingYu Liu, Junze Liu, Zhen Ge, Tian Wang, Jiri Gesi, Dakuo Wang, Weiqi Zhang, Houyu Zhang, Yufan Guo, Xian Li, Bing Yin, and Sujay Sanghavi. On-policy self-distillation for efficient diffusion language models with early-stage calibration. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26954–26965, 2026.

A Implementation Details

This section provides supplementary experimental details and pseudocode for the procedures described in Section 3.

A.1 Main Hyperparameters

Table 3 summarizes the configuration of our main experiments. These settings are shared by all compared systems unless a specific study states otherwise. All systems are post-trained from the same LLaDA2.1-Mini Base on an identical SFT corpus and use the same optimization schedule. Note that an editing threshold of 0.0 means that the token selected by self-correction is always accepted.

Table 3: Main training and evaluation hyperparameters.

Group	Hyperparameter	Value
Data	Number of training samples	$745k$
	Total tokens per epoch	1.6B
Training	Training epochs	10
	Maximum sequence length	16,384
	Batch size	256
	Learning rate	1.0×10^{-4}
	Learning-rate schedule	Cosine decay
	Minimum learning rate	1.0×10^{-6}
	Maximum denoising stages	4
	M2T confidence threshold	0.5
	Teacher-forcing ratio	0.5
Evaluation	Maximum new tokens	32,768
	Temperature	0.0
	M2T threshold	0.5
	T2T threshold	0.0
	Maximum post-mask editing rounds	16
	Block size	32

A.2 Misalignment-Repair Configuration

The controlled probe in Figure 1 compares LED with an ROE baseline trained under the same one-forward schedule. We evaluate on three math reasoning datasets—GSM-Plus, OlympiadBench, and AIME 2025—using clean blocks that the baseline previously generated correctly. We use each

clean block as the target sequence, which biases the comparison toward the baseline because the repair targets lie within its output distribution. At a seeded position, we corrupt each target block by omitting one token or inserting a random token. Either operation shifts the remaining suffix. In the fixed-length block view shown in Figure 1, a boundary token may also enter or leave the block, so the resulting block can have a Levenshtein distance greater than one from the target. Corruption positions and inserted tokens are seeded independently of model predictions, so both models receive identical corrupted inputs. A block is considered repaired only if the generation obtained after editing exactly matches the target sequence.

A.3 PASS@k DECODING CONFIGURATION

For the pass@k comparison in Section 4.4, both regimes use the same checkpoint, temperature 0.6, block size 32, and a maximum generation length of $32{,}768$. We draw 256 samples per problem. Confidence-free decoding uses $\tau = 0$ with LED and permits at most 16 post-mask editing rounds; thresholded decoding uses $\tau = 0.95$ without editing, matching the previous standard dLLM setup. The number of decoding iterations is dynamic: every iteration decodes at least one masked token, rather than using a fixed iteration count.

A.4 BENCHMARK SUITE

We evaluate on 12 benchmarks spanning seven capability categories, including knowledge, reasoning, math, coding, function calling, instruction following, and Chinese, as summarized in Table 4. All benchmarks are evaluated in a zero-shot setting. We use temperature 0.0 for the reported pass@1 scores; the separate pass@k comparison uses temperature 0.6 (Appendix A.3). We report pass@1 for every benchmark. Answer extraction and scoring follow each benchmark's native format. For math tasks, boxed expressions (`\boxed{}`) are verified either symbolically using MATH-Verify or by an external judge model with xVerify. For multiple-choice tasks, we use regex-matched letter answers. For code tasks, we rely on unit-test execution, including EvalPlus tests and sandboxed runs for LiveCodeBench. For KOR-Bench, IFEval, and BFCL v3, we use their task-specific rule-based scorers.

Table 4: Evaluation benchmarks, grouped by capability category.

Category	Benchmark
Knowledge	MMLU-Pro (Wang et al., 2024) GPQA-Diamond (Rein et al., 2024)
Reasoning	KOR-Bench (Ma et al., 2025)
Math	GSM-Plus (Li et al., 2024) OlympiadBench (He et al., 2024) AIME 2025 (Mathematical Association of America, 2025)
Coding	HumanEval+ (Liu et al., 2023) MBPP+ (Liu et al., 2023) LiveCodeBench v6 (Jain et al., 2025)
Function calling	BFCL v3 (Patil et al., 2025)
Instruction following	IFEval (prompt-strict) (Zhou et al., 2023)
Chinese	C-Eval (Huang et al., 2023)

B ADDITIONAL RELATED-WORK COMPARISONS

LLaDA2.1 token editing. LLaDA2.1 allows previously generated tokens to be replaced during masked denoising, but each update remains tied to the original block positions (Bie et al., 2026). LED is post-trained from the same 16B LLaDA2.1-Mini Base and extends this action space with deletion and insertion. A deletion removes a committed token, whereas an insertion creates a new

mask whose content is resolved by a later mask-to-token update; the resulting state is then projected back to the original block size.

ME-DLM. ME-DLM is the closest edit-based refinement method: it first produces a complete masked-diffusion draft and subsequently applies replacement, deletion, and insertion to the resulting variable-length sequence (Ren et al., 2026). It adds a next-token prediction head alongside current-token prediction, and its training pipeline comprises three sequential stages. In contrast, LED represents token prediction, deletion, and insertion in a single per-position output vocabulary and uses one post-training objective along an inference-like denoising trajectory. Insertion creates a mask rather than directly predicting its content, allowing LED to reuse the base DLM's token predictor and to correct alignment before block generation is complete, without a separate post-draft refinement phase.

DreamOn and FlexMDM. DreamOn introduces expansion and deletion states for length-adaptive code infilling, with both states predicted at masked positions and the sequence length changing during inference (Wu et al., 2026b). FlexMDM instead alternates mask insertion and unmasking over a variable-length state and predicts an additional insertion expectation for each gap (Kim et al., 2025). These methods primarily use length-changing transitions to model or control output length. LED uses them to correct model-induced intermediate states: it can edit already visible tokens, requires no separate insertion-expectation head, and keeps each model call compatible with the fixed-block interface of its base DLM.

Edit-based generative processes. Edit Flows defines insertion, deletion, and substitution as transition rates of a continuous-time generative process over variable-length sequences (Havasi et al., 2025). Unlike this approach, LED does not replace the underlying generative process; it adds discrete correction actions through post-training of an existing masked DLM. The experimental scales also differ: ME-DLM and the scaled FlexMDM use LLaDA-8B Base, DreamOn uses several 7B DLMs, and Edit Flows primarily reports 1.3B language models trained from scratch, whereas our experiments use the later 16B LLaDA2.1-Mini Base. Because these backbones and training settings are not matched, we treat their scale as implementation context rather than as evidence of cross-paper performance superiority.

C Detailed Results

Figure 6 presents the detailed per-benchmark performance across different block sizes for the 12 benchmarks introduced in Section 4.6. As shown in the figure, LLaDA-LED outperforms the baseline on the majority of benchmarks, demonstrating that LED can greatly mitigate error accumulation within extended denoising chains by containing local alignment errors before they propagate into block-level corruption.

Figure 7 reports the aggregate grid-search results used to select the default thresholds in Section 4.7. Figure 8 provides the corresponding per-benchmark breakdown.

D Method Details and Algorithms

This section complements Section 3 with the formal details needed to reproduce each component. For each procedure, we first state its contract and boundary behavior in prose and then give pseudocode that makes the execution order explicit.

Forming the next input block (Algorithm 1). At round r, the model predicts one token from $\mathcal{V}_{\text{out}}$ at every position. The procedure uses the predictions at the selected positions $\mathcal{Q}_r$. It retains s_i^r outside $\mathcal{Q}_r$. At a selected position, [Del] removes s_i^r, [Ins] places a fresh [Mask] immediately before s_i^r, and a token in $\mathcal{V}$ replaces s_i^r. The temporary sequence may have a length different from B. Keeping its first B tokens and padding a shorter sequence on the right with [Mask] gives the input s^{r+1} for the next round.

LCS-based edit-target construction (Algorithm 2). Given the input block s^r and target block $y = (y_1, \ldots, y_B)$, the procedure constructs an edit target $e_i^r \in \mathcal{V}_{\text{out}}$ for every position i. It first obtains the maximum-cardinality monotone matching $\pi^\star$ defined in Section 3.2. The matched pairs

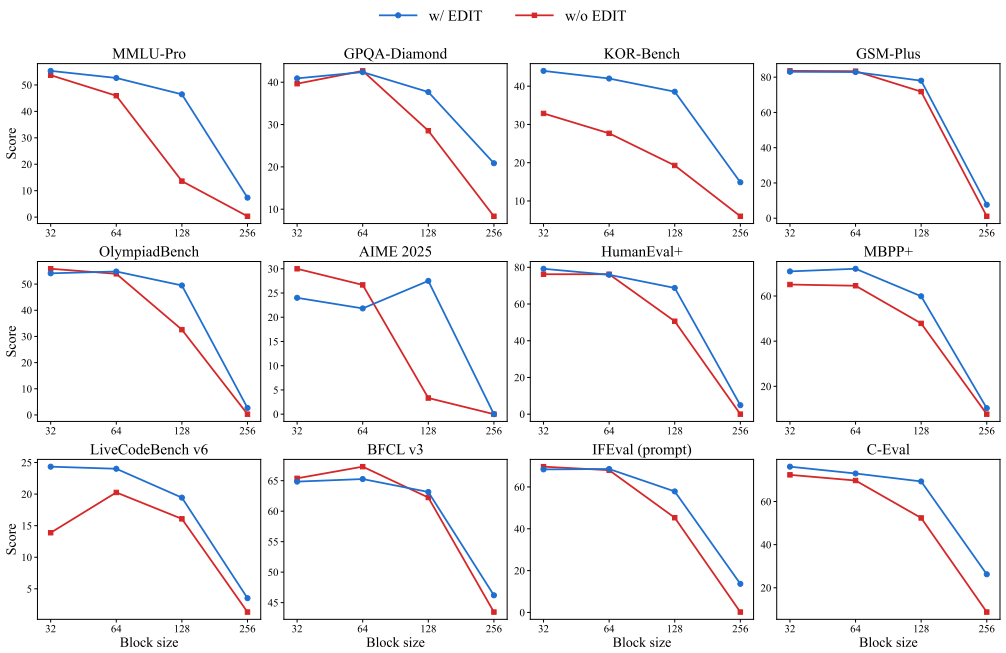
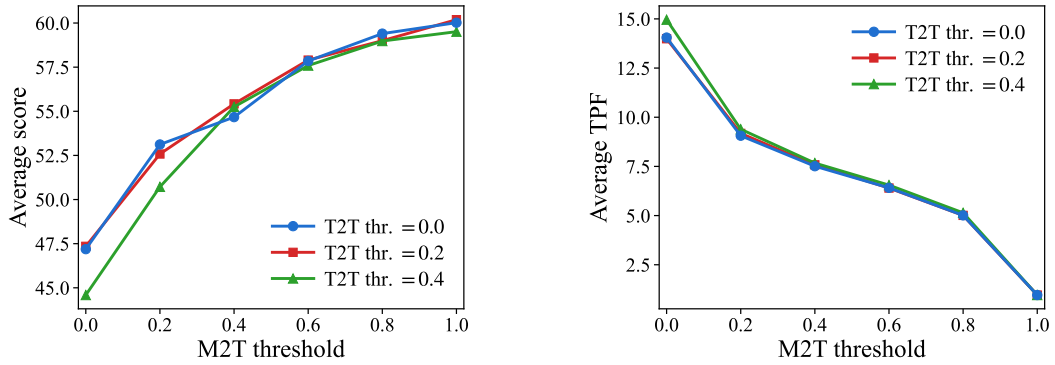


Figure 6: Detailed performance across different block sizes for the baseline and LLaDA-LED.



(a) Score across configurations.

(b) TPF across configurations.

Figure 7: Aggregate grid-search results with block size fixed at 32.

divide the remaining positions into the ordered gap pairs (I_k, J_k). Matched or paired positions receive their token from y, extra positions in s^r receive $[\mathrm{Del}]$, and a nonfinal gap with extra positions in y assigns one $[\mathrm{Ins}]$ target to its right matched anchor. Extra target positions in the final gap are handled in a later round.

Edit-augmented inference (Algorithm 3). At round r, the decoder updates the positions in $\mathcal{Q}_r = \mathcal{C}_r(\tau) \cup \mathcal{E}_r(\nu)$, using the sets defined in Section 3.3. It samples an output token at each selected position and forms the next block as described in Algorithm 1. Positions outside $\mathcal{Q}_r$ retain their current tokens.

After all masks are resolved, the decoder permits at most $K_{\max} = 16$ additional rounds and stops early when $\mathcal{E}_r(\nu) = \emptyset$. The decoder records previously visited block states in $\mathcal{H}$ and resamples a selected output token if an update would repeat a state. Algorithm 3 gives the complete procedure.

Inference-aligned multi-stage training (Algorithm 4). This loop implements the compressed training trajectory described in Section 3.4. Starting from $s^0 = [\mathrm{Mask}]^B$, each stage selects masks

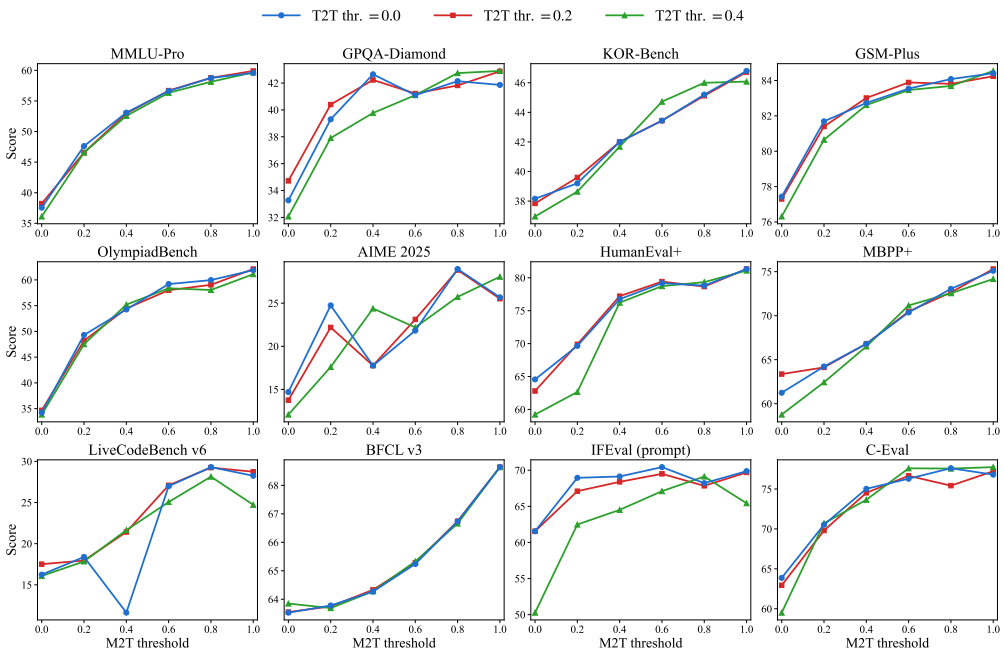


Figure 8: Per-benchmark scores across the mask and edit threshold grid at temperature 0.0.

Algorithm 1: Form the input block for round $r+1$

Input: input block s^r; selected positions $\mathcal{Q}_r \subseteq \{1, \ldots, B\}$; one predicted token in $\mathcal{V}_{\text{out}}$ for each $i \in \mathcal{Q}_r$; block length B
Output: input block s^{r+1} for round $r+1$

```
seq ← [];
for i = 1 ... B do
    if i ∉ Q_r then
        append s_i^r to seq;
    else if the prediction at i is [Del] then
        do not append s_i^r;
    else if the prediction at i is [Ins] then
        append [Mask], then s_i^r, to seq;
    else
        append the predicted token in V to seq;
keep the first B tokens of seq; if it has fewer than B tokens, append [Mask] tokens on the right
  until it has length B;
s^{r+1} ← seq;
return s^{r+1};
```

that satisfy the confidence threshold and supplements them with the highest-confidence remaining masks to meet $m_r = \min\{|\mathcal{M}_r|, \lceil B/R \rceil\}$. It derives LCS labels for the selected masks and visible positions, then compiles the resulting actions into the next state.

Selected masks use mixed teacher forcing with ratio ρ, while visible positions advance with their edit labels. The final editing-only stage supervises visible tokens produced by the last mask-update stage, and the returned objective averages the $R+1$ normalized stage losses. Figure 9 illustrates the trajectory, and Algorithm 4 specifies the stage loop and the final editing-only update.

Algorithm 2: Construct edit targets by LCS alignment

Input: input block s^r and target block $y = (y_1, \ldots, y_B)$
Output: edit targets $e_i^r \in \mathcal{V}_{\text{out}}$ for $i = 1, \ldots, B$
$\pi^\star \leftarrow$ a maximum-cardinality monotone matching of positions in s^r and y where $s_i^r = y_j \in \mathcal{V}$ for every matched pair (i, j);
If several matchings have the same cardinality, break ties by the indices in s^r first and then by the indices in y.;
$K \leftarrow |\pi^\star|$;
Write $\pi^\star = \{(i_k, j_k)\}_{k=1}^{K}$ and let (I_k, J_k) for $k = 0, \ldots, K$ be the ordered gap pairs.;
for $k = 1, \ldots, K$ **do**
  $e_{i_k}^r \leftarrow y_{j_k}$;
for $k = 0, \ldots, K$ **do**
  $\ell_k \leftarrow \min\{|I_k|, |J_k|\}$;
  for $p = 1, \ldots, \ell_k$ **do**
    $e_{I_k[p]}^r \leftarrow y_{J_k[p]}$;
  for $p = \ell_k + 1, \ldots, |I_k|$ **do**
    $e_{I_k[p]}^r \leftarrow [\text{Del}]$;
  if $|J_k| > \ell_k \wedge k < K$ **then**
    $e_{i_{k+1}}^r \leftarrow [\text{Ins}]$;
return $(e_i^r)_{i=1}^B$;

Algorithm 3: Decode a block with edit-augmented transitions

Input: context c, initial state $s^0 = [\text{Mask}]^B$, mask threshold τ, edit threshold ν, temperature T, maximum post-mask rounds $K_{\max} = 16$
Output: decoded block state s^r
$\mathcal{H} \leftarrow \{s^0\}$ $r \leftarrow 0$ $k \leftarrow 0$;
while *true* **do**
  $p_r \leftarrow p_\theta(\cdot \mid c, s^r)$;
  $\mathcal{M}_r \leftarrow \{i \in \{1, \ldots, B\} : s_i^r = [\text{Mask}]\}$ $\mathcal{P}_r \leftarrow \{i \in \{1, \ldots, B\} : s_i^r \in \mathcal{V}\}$;
  for $i = 1, \ldots, B$ **do**
    $\hat{v}_i^r \leftarrow \arg\max_{v \in \mathcal{V}_{\text{out}}} p_\theta(v \mid c, s^r, i)$;
  for $i \in \mathcal{P}_r$ **do**
    $\eta_i^r \leftarrow p_\theta(\hat{v}_i^r \mid c, s^r, i)$;
  $\mathcal{C}_r(\tau) \leftarrow$ mask-update set defined in Section 3.3;
  $\mathcal{E}_r(\nu) \leftarrow \{i \in \mathcal{P}_r : \hat{v}_i^r \neq s_i^r \ \wedge \ \eta_i^r \geq \nu\}$;
  if $\mathcal{M}_r = \emptyset \ \wedge \ \big(\mathcal{E}_r(\nu) = \emptyset \ \vee \ k = K_{\max}\big)$ **then**
    break;
  $\mathcal{Q}_r \leftarrow \mathcal{C}_r(\tau) \cup \mathcal{E}_r(\nu)$;
  for $i \in \mathcal{Q}_r$ **do**
    sample an output token $\tilde{v}_i^r$ from $p_\theta(\cdot \mid c, s^r, i)$ at temperature T;
  $\tilde{s} \leftarrow$ apply the sampled outputs to s^r as in Algorithm 1;
  while $\tilde{s} \in \mathcal{H}$ **do**
    choose j uniformly from $\mathcal{Q}_r$. Set the probability of $\tilde{v}_j^r$ to zero and resample $\tilde{v}_j^r$;
    $\tilde{s} \leftarrow$ apply the sampled outputs to s^r as in Algorithm 1;
  $s^{r+1} \leftarrow \tilde{s}$ $\mathcal{H} \leftarrow \mathcal{H} \cup \{s^{r+1}\}$;
  if $\mathcal{M}_r = \varnothing$ **then**
    $k \leftarrow k + 1$;
  $r \leftarrow r + 1$;
return s^r;

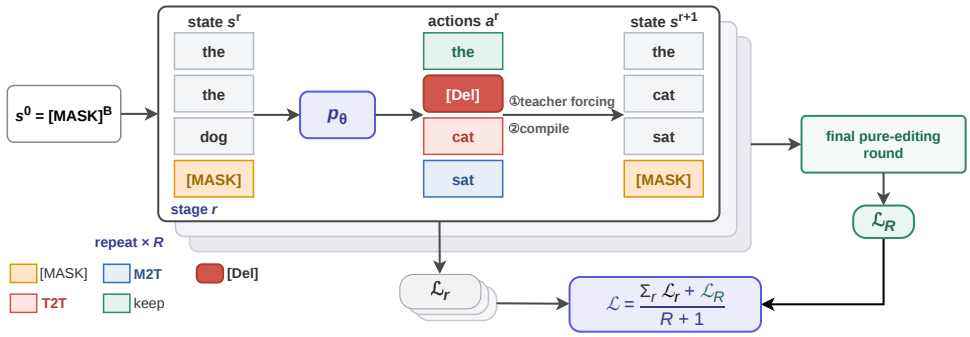


Figure 9: Inference-aligned multi-stage training. Each stage supervises selected masks and visible positions. A final editing-only stage covers the visible positions after the last mask-update stage.

Algorithm 4: Inference-aligned multi-stage training

Input: context c, clean target y, block size B, stages R, threshold τ, teacher-forcing ratio ρ
$s^0 \leftarrow [\text{Mask}]^B \quad \mathcal{P}_0 \leftarrow \varnothing \quad \mathcal{L} \leftarrow 0;$
for $r = 0 \dots R-1$ **do**
  $p_r \leftarrow p_\theta(\cdot \mid c, s^r);$
  $\mathcal{C}_r(\tau) \leftarrow$ confidence-thresholded masked positions from $p_r;$
  $m_r \leftarrow \min\{|\mathcal{M}_r|, \lceil B/R \rceil\} \quad \mathcal{U}_r \leftarrow \mathcal{C}_r(\tau);$
  if $|\mathcal{U}_r| < m_r$ **then**
    $\mathcal{U}_r \leftarrow \mathcal{U}_r \cup \text{top}_{m_r - |\mathcal{U}_r|}(\mathcal{M}_r \setminus \mathcal{U}_r);$
  $\mathcal{Q}_r \leftarrow \mathcal{U}_r \cup \mathcal{P}_r \quad e^r \leftarrow \text{BUILDEDITLABELS}(s^r, y);$
  $\mathcal{L} \mathrel{+}= \mathcal{L}_r$ on $\mathcal{Q}_r$ via Eq. equation 3;
  form $\tilde{a}^r_{\mathcal{Q}_r}$ by mixed teacher forcing on $\mathcal{U}_r$ (ratio ρ) and edit labels e^r_i on $\mathcal{P}_r;$
  $s^{r+1} \leftarrow \text{COMPILE}(s^r, \tilde{a}^r_{\mathcal{Q}_r}, \mathcal{Q}_r, B) \quad \mathcal{P}_{r+1} \leftarrow \{i : s^{r+1}_i \in \mathcal{V}\};$
$\mathcal{U}_R \leftarrow \varnothing \quad e^R \leftarrow \text{BUILDEDITLABELS}(s^R, y) \quad p_R \leftarrow p_\theta(\cdot \mid c, s^R);$
$\mathcal{L} \mathrel{+}= \mathcal{L}_R$ on $\mathcal{P}_R$ via Eq. equation 3;
return $\mathcal{L}/(R{+}1);$

E Case Study: Edit Trajectories at $\tau = 0$

We trace six single-block decoding runs drawn from a panel of GSM-Plus problems (each a paraphrase of the same robe task) under the aggressive $\tau = 0$ regime, where the model commits every position at each round. Each run applies the same per-token corruption to both LED and a *baseline* (an identical model trained without the edit actions), and we group the runs into three *successful repairs* where LED converges exactly to the target sequence while the baseline fails, and three *failures* where LED does not. Each example states the fixed *prefix* (everything before the editable region) and then traces, over a window of token positions, the row labeled *Target* followed by the *input* and *output* of each step. We show as much of the editable block as fits and elide the rest with ···. Notation: [Del] deletes its position, [Ins] opens a mask before its position, [M] is an empty mask, ↩ is a newline, ··· marks elided positions; the `injected corruption` is red and a **`restored`** token is green. Because [Del]/[Ins] shift the suffix, only the columns before the first edit are strictly position-aligned across rows.

E.1 Successful Repairs

Example 1 (insertion) is a successful repair at edit distance 0: a spurious `rumah` is inserted after `blue`, which shifts the suffix and drops the final token `white`; the model removes it with [Del] and refills the reopened tail [M] with **`white`**.

Prefix (fixed): `Question: A robe takes 3 bolts of blue fiber and half that much white fiber. How many bolts in total does it take? ...`

within \boxed{}. Let's break it down step by step:↩↩1. The robe requires **3 bolts

position	17	18	19	20	21	22	23	···	29	30	31
Target	of	blue	fiber	**.	↩	↩	2	···	as	much	white
Step 0 in	of	blue	rumah	fiber	**.	↩	↩	···	half	as	much
Step 0 out	of	blue	[Del]	fiber	**.	↩	↩	···	half	as	much
Step 1 in	of	blue	fiber	**.	↩	↩	2	···	as	much	[M]
Step 1 out	of	blue	fiber	**.	↩	↩	2	···	as	much	**white**

Example 2 (deletion) is the harder case where a *deleted* token must be actively re-introduced. The opening ** before blue is deleted, so blue slides into the bold marker's slot. LED first rewrites the shifted blue back into ** (step 1), then emits [Ins] to open a slot (step 2) and refills it with **blue** (step 3); the run converges in five rounds (the leading whitespace-only round is omitted) at edit distance 0, while the baseline cannot reopen a slot and settles at edit distance 2 with fiber dropped.

Prefix (fixed): Question: A robe takes 200 bolts of blue fiber ... within \boxed{}. Let's break this down step by step:↩↩

position	11	12	13	14	15	16	17	18	19	20
Target	1	.	The	number	of	**	blue	fiber	**	bolts
Step 1 in	1	.	The	number	of	blue	fiber	**	bolts	needed
Step 1 out	1	.	The	number	of	**	fiber	**	bolts	needed
Step 2 in	1	.	The	number	of	**	fiber	**	bolts	needed
Step 2 out	1	.	The	number	of	**	[Ins]	fiber	**	bolts
Step 3 in	1	.	The	number	of	**	[M]	fiber	**	bolts
Step 3 out	1	.	The	number	of	**	**blue**	fiber	**	bolts

Example 3 (deletion) follows the same [Ins]-then-refill pattern more directly: the token It is deleted, pulling also into its slot, and LED emits [Ins] before also and refills the mask with **It**, reaching edit distance 0 in three steps where the baseline fails at edit distance 2.

Prefix (fixed): Question: A robe takes 2 bolts of blue fiber and 0.5 times ... within \boxed{}. Let's break this down step by step:↩↩- The robe requires **2 bolts

position	18	19	20	21	22	23	24	25	26	27
Target	of	blue	fiber	**.	↩	-	It	also	requires	**
Step 0 in	of	blue	fiber	**.	↩	-	also	requires	**	0
Step 0 out	of	blue	fiber	**.	↩	-	[Ins]	also	requires	**
Step 1 in	of	blue	fiber	**.	↩	-	[M]	also	requires	**
Step 1 out	of	blue	fiber	**.	↩	-	**It**	also	requires	**

E.2 FAILURES

The runs that do not converge cluster into three distinct failure modes. The first is *inaction on a locally fluent corruption*.

Example 4 (deletion, edit distance 1): the token `'s` is deleted, pulling `break` into position 31; because `Let break` is not flagged as anomalous by the local context, the model keeps `break` and converges in one step instead of rewriting it to **`'s`** or predicting [Ins] to re-insert it.

Prefix (fixed): `Question: A robe takes 3 bolts of blue fiber and half that much white fiber. How many bolts in total does it take? Please reason step by step, and put your final answer within \boxed{} <|role_end|>`

position	25	26	27	28	29	30	31
Target	<role>	ASS	IST	ANT	</role>	Let	's
Step 0 in	<role>	ASS	IST	ANT	</role>	Let	break
Step 0 out	<role>	ASS	IST	ANT	</role>	Let	break

The second mode is *commitment to the corruption*: rather than deleting the junk, the model treats it as a valid prefix and extends it.

Example 5 (insertion): an inserted `Ric` before `fiber` is completed into the word `Richet` by a token-to-token rewrite that overwrites `fiber`, cementing the error at edit distance 3.

Prefix (fixed): `Question: A robe takes 2 bolts of blue fiber and 0.5 times ... within \boxed{}. Let's break this down step by step:↩↩- The robe requires **`

position	16	17	18	19	20	21	22	23
Target	2	bolts	of	blue	fiber	**.	↩	-
Step 0 in	2	bolts	of	blue	Ric	fiber	**.	↩
Step 0 out	2	bolts	of	blue	Ric	het	**.	↩

The third mode is *over-deletion*.

Example 6 (insertion): an inserted `Flint` is removed, but LED fires a second [Del] on the adjacent valid `of`, then refills with a spurious `**`, so the genuine token `blue` is never restored.

Prefix (fixed): `Question: A robe takes 2 bolts of blue fiber ... Each fiber costs $12 ... within \boxed{}. Let's break this down step by step.↩↩1. **Blue fiber used**:↩The robe takes **`

position	21	22	23	24	25	26	27	28
Target	The	robe	takes	**	2	bolts	of	blue
Step 0 in	The	robe	takes	**	2	bolts	Flint	of
Step 0 out	The	robe	takes	**	2	bolts	[Del]	[Del]
Step 1 in	The	robe	takes	**	2	bolts	[M]	[M]
Step 1 out	The	robe	takes	**	2	bolts	**	of

E.3 Synthesis

Taken together, the six runs show that LED has genuinely learned to *use* all three edit actions and, unlike the baseline, can repair both spurious insertions (with [Del]) and outright deletions (with [Ins]), often reaching the target sequence exactly where the action-free baseline cannot. The failures are not random: they concentrate where the corrupted text is locally fluent so no anomaly signal fires (the `break` case), or where the model commits to or over-extends the corruption rather than excising it (the `Richet` and `Flint` cases). The common thread is that the model edits along a *locally* plausible path rather than the globally shortest one, which is exactly what the local LCS labels

supervise. This motivates stronger path-level supervision, such as offline edit-path construction that scores whole recovery trajectories, beyond the per-position LCS labels used here.