DELA: LEARNING CONTROLLABLE EDITS FOR DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models (DLMs) support bidirectional denoising and iterative refinement, but generation remains largely controlled by token-level masking schedules. This leaves draft-level refinement implicit and limits explicit control over when to insert missing content, delete redundant spans, revisit uncertain regions, or stop editing. We introduce **Diffusion Editing Language Adapter (DELA)**, a plug-in framework that enables controllable draft refinement with a frozen DLM. DELA operates on clean, variable-length drafts using four explicit editing actions: INSERT, DELETE, REMASK, and STOP. The controller determines when and where to edit the draft, while the frozen DLM fills masked regions through denoising, separating high-level edit control from token-level generation. A naive single-phase editor would be computationally expensive because arbitrary interior edits can disrupt prefix stability and limit reliable KV-cache reuse. To address this, DELA uses a two-phase decoding procedure: a fast, append-only drafting phase that supports KV-cache reuse, followed by a global refinement phase that enables targeted non-monotonic edits to the full draft. We train the controller using a staged curriculum that begins with learning local edit priors from synthetic recovery tasks and mined actions that improve draft quality. Subsequent stages refine edit selection through rollout-based outcome preferences and alignment with downstream task feedback. Experiments on mathematical reasoning and code generation show that DELA achieves gains of up to 5.5 percentage points over direct diffusion decoding while keeping the backbone frozen. DELA thus provides a modular approach to explicit draft-level control in diffusion language models.

1 INTRODUCTION

Diffusion language models (DLMs) offer a compelling alternative to autoregressive generation by enabling bidirectional context modeling and iterative refinement (Li et al., 2022; Austin et al., 2021; Campbell et al., 2022; Lou et al., 2023; Gat et al., 2024). Building on this paradigm, masked DLMs have improved generation quality, scalability, and theoretical understanding (Shi et al., 2024; Sahoo et al., 2024; Ou et al., 2025; Zheng et al., 2025; Nie et al., 2025; Ye et al., 2025). Complementary work has explored reasoning-oriented post-training (Zhao et al., 2025; Tang et al., 2026) and adaptive inference-time decoding (Kim et al., 2025; Wang et al., 2025; Ben-Hamu et al., 2025). Despite this progress, scalable masked DLMs still largely control generation through token-level denoising schedules, leaving draft-level refinement implicit.

This limitation is important because many generation errors are structural rather than purely lexical. A draft may omit a reasoning step, contain redundant content, include an inconsistent span, or already be correct. Effective refinement therefore requires deciding where to intervene, which edit to apply, and when to stop, rather than only choosing which masked positions to reveal. Prior non-autoregressive and edit-based methods show that insertion, deletion, and iterative refinement are useful abstractions (Lee et al., 2018; Ghazvininejad et al., 2019; Stern et al., 2019; Gu et al., 2019). However, masked denoising alone does not provide a controllable policy for refining clean variable-length drafts.

Recent variable-length diffusion and flow models address these limitations by modifying the generative process through edit operations, insertion expectations, expansion/deletion transitions, or deletion-insertion dynamics (Havasi et al., 2025; Kim et al., 2026; Wu et al., 2026b; Ding et al., 2026).

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.

These approaches are powerful, but they typically make structural editing part of the learned generative dynamics. Adopting them therefore often requires changing the model architecture, training objective, or backbone parameters, which can be costly at scale and limits the reuse of pretrained DLMs. We therefore ask: *Can we equip an existing diffusion language model with controllable draft-level refinement without modifying or retraining its denoising backbone?*

We introduce **Diffusion Editing Language Adapter (DELA)**, a plug-in framework that equips a frozen DLM with controllable draft-level refinement. DELA keeps the diffusion backbone fixed during both training and inference, and trains only a lightweight edit controller over clean drafts. The controller issues four actions—INSERT, DELETE, REMASK, and STOP—while the frozen backbone realizes masked content through conditional denoising. This design decouples high-level edit control from token-level generation.

The design of DELA follows a broader principle in modern model adaptation: reuse a strong pretrained generator and adapt its behavior through a lightweight control or alignment layer, rather than retraining the entire model. In DELA, the frozen DLM provides token-level denoising ability, while the edit controller learns how to use this ability to improve a clean draft. Directly learning such a policy from downstream rewards is inefficient because the action space is heterogeneous, the value of an edit is observed only after denoising, and useful feedback may require multi-step refinement. DELA therefore uses a staged curriculum. It first learns executable edit primitives from synthetic recovery tasks, then strengthens the warm start through mined improving-action distillation. Later stages use rollout-labeled candidate preferences and downstream task feedback to train outcome-aware edit selection. At inference time, DELA selects among a small set of executable edit candidates without task labels or rollout evaluation.

DELA also addresses the computational cost of structural editing. A naive single-phase editor would allow INSERT, DELETE, and REMASK throughout generation, but arbitrary interior edits make the sequence non-monotonic, break prefix stability, prevent reliable KV-cache reuse, and require repeated processing of changing full drafts. DELA instead uses a two-phase generation process. A *fast drafting phase* first constructs an initial draft through append-only masked-block insertion, preserving cache-friendly monotonic growth while allowing local remasking within each newly generated block. A *global refinement phase* then applies the full action space to the completed draft, enabling targeted insertion, deletion, remasking, and stopping. Thus, DELA is not a new diffusion backbone, but a lightweight draft-conditioned editing layer for controlling an already trained denoising model.

Our main contributions are as follows. (1) We formulate draft-level refinement for DLMs as a plug-in control problem and propose DELA, an edit-controller framework that refines clean variable-length drafts via INSERT, DELETE, REMASK, and STOP actions while keeping the denoising backbone fixed. (2) We introduce a two-phase decoding procedure that combines cache-friendly append-only drafting with targeted non-monotonic refinement, avoiding global editing throughout generation. (3) We develop a staged controller-learning curriculum that starts with synthetic edit recovery, strengthens the warm start with mined improving-action distillation, and uses rollout-labeled preferences and downstream feedback for outcome-aware edit selection. (4) We show that DELA improves mathematical reasoning and code generation by up to 5.5 percentage points while keeping the diffusion backbone fixed, demonstrating the effectiveness of explicit draft-level edit control.

2 RELATED WORK

Diffusion language models. Diffusion models for text generation have been explored in both continuous and discrete spaces. Diffusion-LM (Li et al., 2022) applies Gaussian diffusion to continuous word embeddings and decodes denoised embeddings back into tokens. Discrete diffusion models instead operate directly on token spaces. D3PM (Austin et al., 2021) formulates discrete corruption through Markov transition matrices, including absorbing-mask transitions. This line of work has led to masked DLMs, where the forward process corrupts tokens into `[MASK]` and the reverse model predicts clean tokens at masked positions. Subsequent work improves this formulation through ratio estimation (Lou et al., 2023), simplified masked diffusion objectives (Shi et al., 2024; Sahoo et al., 2024), and theoretical connections between absorbing diffusion and clean-token conditional distributions (Ou et al., 2025). DELA adopts this masked diffusion formulation and uses the pretrained model as a frozen token-level denoiser.

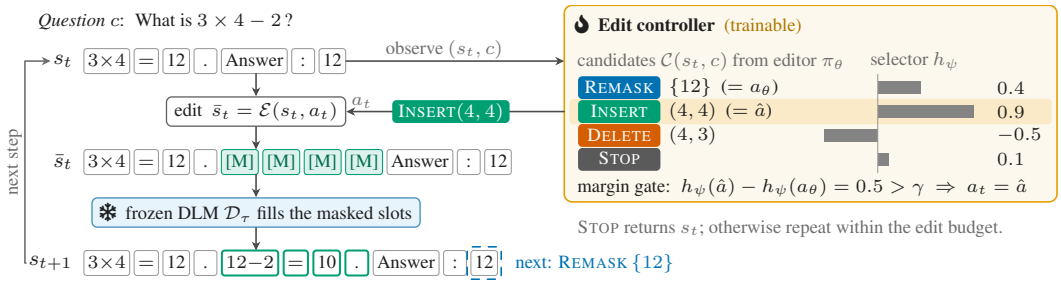

Figure 1: Overview of DELA. A learned edit controller operates on clean variable-length drafts and selects structural actions, while the frozen DLM realizes masked content through denoising. This design decouples high-level edit control from token-level lexical generation.

Scaling and efficient generation with DLMs. Masked DLMs have also been scaled to competitive language models. LLaDA (Nie et al., 2025) trains an 8B masked diffusion model from scratch and demonstrates strong instruction-following and zero-shot performance. Dream (Ye et al., 2025) initializes diffusion models from pretrained autoregressive models and improves reasoning performance with context-adaptive noise scheduling. Beyond scaling, several works study how decoding affects efficiency and generation quality. Block Diffusion (Arriola et al., 2025) enables blockwise generation and KV-cache reuse, while Fast-dLLM (Wu et al., 2026a) introduces approximate KV caching and confidence-aware parallel decoding for DLMs. Other work studies adaptive token ordering and early stopping (Kim et al., 2025; Li et al., 2026), while ReMDM (Wang et al., 2025) and Tolerator (Tian et al., 2025) revisit generated tokens through inference-time remasking without retraining the backbone. DELA instead learns an external controller over clean drafts that can insert, delete, remask, or stop while keeping the denoising backbone frozen.

Variable-length and edit-based generation. Another approach to the length and structural limitations of masked DLMs is to introduce variable-length or edit-based dynamics. DiffusER (Reid et al., 2023) models generation as iterative editing from a prototype or noisy sequence. Edit Flows (Havasi et al., 2025) formulates sequence generation as a continuous-time Markov chain over edit operations, including INSERT, DELETE, and SUBSTITUTE, and trains the model with discrete flow matching over auxiliary alignments. FlexMDM (Kim et al., 2026) extends masked diffusion with a joint interpolant that learns both an unmasking posterior and an insertion expectation, enabling principled variable-length masked diffusion. DreamOn (Wu et al., 2026b) introduces length-changing transitions such as EXPAND and DELETE for code infilling beyond a fixed-size canvas. DID (Ding et al., 2026) replaces masking with a deletion-insertion diffusion process, where the forward process deletes tokens and the reverse process inserts tokens. OneFlow (Nguyen et al., 2025) extends insertion-based Edit Flows (Havasi et al., 2025) to multimodal non-autoregressive generation. Together, these methods enable flexible-length generation by making length-changing edits part of the diffusion or flow process. This couples structural control with token-level denoising, making it difficult to reuse frozen DLMs as controllable editors. DELA instead learns an external edit controller over a frozen denoiser, separating *when and where to edit* from *how masked content is realized*.

3 DELA: LEARNING CONTROLLABLE EDITS

We introduce **Diffusion Editing Language Adapter (DELA)**, a plug-in framework that refines variable-length clean drafts through four editing actions: INSERT, DELETE, REMASK, and STOP. DELA decouples *structural control* from *semantic generation*: a learned editor decides where and how to modify the sequence, while a frozen masked DLM realizes masked content as clean text. This turns the frozen denoiser into a controllable non-monotonic editor without modifying the diffusion backbone. Figure 1 gives an overview of the refinement loop.

3.1 EDIT-CONDITIONED REFINEMENT

DELA refines a variable-length clean draft $s_t \in V^*$ under context c. At each time step, the editor selects an action a_t, applies a deterministic edit operator $\mathcal{E}$, and obtains an intermediate sequence

$$\bar{s}_t = \mathcal{E}(s_t, a_t),$$

which may contain [MASK] tokens. A frozen DLM then denoises $\bar{s}_t$ back to a clean state:

$$s_{t+1} = T_\tau(a_t; s_t, c) = \mathcal{D}_\tau(\bar{s}_t, c), \tag{1}$$

where $\mathcal{D}_\tau$ denotes threshold-based denoising with the frozen DLM backbone. Thus, the editor controls structural changes, while lexical realization is delegated to the denoiser. For STOP, the refinement process terminates and no denoising transition is applied.

Action space. The editor chooses among four operations: INSERT, DELETE, REMASK, and STOP. For INSERT and DELETE, we write

$$a_t = (\mathrm{op}_t, \mathrm{pos}_t, \mathrm{len}_t).$$

Here $\mathrm{op}_t \in \{\textsc{Insert}, \textsc{Delete}\}$ is the selected operation, pos_t denotes the start position of the edit, and len_t denotes the span length. For $\mathrm{op}_t = \textsc{Insert}$, the edit creates len_t consecutive mask slots at pos_t. For $\mathrm{op}_t = \textsc{Delete}$, the edit removes the span $[\mathrm{pos}_t, \mathrm{pos}_t + \mathrm{len}_t)$. The proposal editor factorizes structural actions as

$$\pi_\theta(a_t \mid s_t, c) = \pi_\theta^{\mathrm{op}}(\mathrm{op}_t \mid s_t, c)\pi_\theta^{\mathrm{pos}}(\mathrm{pos}_t \mid \mathrm{op}_t, s_t, c)\pi_\theta^{\mathrm{len}}(\mathrm{len}_t \mid \mathrm{op}_t, \mathrm{pos}_t, s_t, c). \tag{2}$$

For $\mathrm{op}_t = \textsc{Remask}$, the editor predicts token-level logits u_i and samples a remask set

$$M_t = \{i : \ m_i \sim \mathrm{Bernoulli}(\sigma(u_i))\}. \tag{3}$$

Tokens in M_t are replaced by [MASK] and then regenerated by the frozen denoiser. This operation allows DELA to revisit uncertain or inconsistent content while preserving the surrounding structure. For $\mathrm{op}_t = \textsc{Stop}$, no edit is applied and the current sequence is returned as the final output.

Two-phase editing for efficient controllable refinement. Applying the full action space throughout generation is computationally inefficient because non-monotonic edits prevent reliable prefix reuse and may require repeatedly processing a changing full sequence. Moreover, early generation primarily requires constructing content rather than global restructuring. DELA therefore separates efficient draft construction from targeted non-monotonic refinement.

DELA therefore uses a two-phase procedure: a *fast drafting* phase followed by a *global refinement* phase. During *fast drafting*, the editor is restricted to append-only INSERT actions with $\mathrm{pos}_t = |s_t|$. The model constructs an initial draft by appending masked blocks and denoising them with the frozen DLM. Before each newly appended block is finalized, DELA applies a local REMASK step to reopen uncertain tokens and denoise them again. Thus, fast drafting remains monotonic at the structural level while still allowing local correction within each generated block. Once a block is finalized, its prefix KV states are kept fixed as later blocks are appended. The new block can use these cached states without recomputing the prefix. This matches an uncached implementation that also freezes finalized prefix states, but differs from full bidirectional recomputation, which would update them using later blocks.

During *global refinement*, DELA enables the full action space. Starting from the initial draft, the editor and selector can insert missing content, delete redundant spans, remask uncertain tokens, or stop, while the frozen denoiser realizes masked content. By restricting non-monotonic editing to this phase, DELA reserves expensive structural changes for targeted refinement. Figure 2 illustrates the resulting procedure.

3.2 LEARNING THE EDIT CONTROLLER

The frozen DLM provides strong token-level denoising but does not solve the control problem of deciding whether and how a clean draft should be edited. Learning this control directly from downstream rewards is difficult because actions combine operation selection, localization, remasking, and stopping, while their effects are observed only after denoising and may unfold over multiple refinement steps.

We therefore train the controller in three stages. Stage 1 learns **feasible** edit proposals, Stage 2 learns which proposals are **useful** under the frozen denoiser, and Stage 3 **aligns** editing with downstream task success. The editor proposes executable actions, while a selector learns from rollout-derived preferences to predict which proposals are likely to improve the draft, replacing expensive rollout evaluation with one-step scoring at inference time. Throughout training, the diffusion backbone remains frozen.

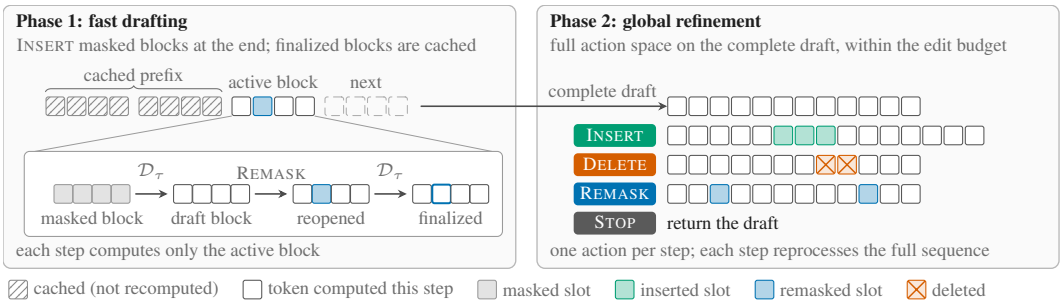

Figure 2: Two-phase editing in DELA. The fast drafting phase constructs an initial draft with append-only masked-block insertion and local remasking, enabling cache-friendly generation. The global refinement phase then unlocks the full action space to insert missing content, delete redundant spans, remask uncertain regions, or stop.

Stage 1: learning executable edit proposals. Stage 1 initializes the proposal editor through supervised imitation using two sources of data: synthetic recovery examples and mined improving actions. Synthetic examples teach the action structure, while mined actions bias the proposal distribution toward edits that are useful under the frozen denoiser.

First, synthetic recovery tasks provide action labels for the DELA action space. Given a clean sequence x and context c, we construct a set $\mathcal{D}_{\text{syn}}$ of labeled examples $(s, c, a^\star)$. For structural corruptions, the corrupted state $s = \xi(x)$ has an inverse edit that is either an INSERT or a DELETE action. Deleting a span from x induces an inverse INSERT action, while inserting a distractor span induces an inverse DELETE action. Each structural example provides a target action

$$a^\star = (\text{op}^\star, \text{pos}^\star, \text{len}^\star), \qquad \text{op}^\star \in \{\text{INSERT}, \text{DELETE}\}.$$

The proposal editor imitates the full structural action with the factorized loss

$$\begin{aligned}\mathcal{L}_{\text{struct}}(s, c, a^\star) = &- \log \pi_\theta^{\text{op}}(\text{op}^\star \mid s, c) - \log \pi_\theta^{\text{pos}}(\text{pos}^\star \mid \text{op}^\star, s, c) \\ &- \log \pi_\theta^{\text{len}}(\text{len}^\star \mid \text{op}^\star, \text{pos}^\star, s, c).\end{aligned} \tag{4}$$

In practice, scalar weights may be applied to the operation, position, and length terms.

For REMASK, each synthetic example consists of a corrupted sequence $\tilde{x}$ and a token-level corruption mask $m^\star$. The editor is trained to select REMASK and localize the corrupted tokens:

$$\begin{aligned}\mathcal{L}_{\text{remask}}(\tilde{x}, c, m^\star) = &- \log \pi_\theta^{\text{op}}(\text{REMASK} \mid \tilde{x}, c) \\ &- \frac{1}{|\tilde{x}|} \sum_i \left[m_i^\star \log \sigma(u_i(\tilde{x}, c)) + (1 - m_i^\star) \log(1 - \sigma(u_i(\tilde{x}, c))) \right].\end{aligned} \tag{5}$$

For uncorrupted target sequences, the target action is STOP, giving

$$\mathcal{L}_{\text{stop}}(x, c) = - \log \pi_\theta^{\text{op}}(\text{STOP} \mid x, c). \tag{6}$$

We summarize these cases with an action-conditioned imitation loss:

$$\mathcal{L}_{\text{act}}(s, c, a) = \begin{cases} \mathcal{L}_{\text{struct}}(s, c, a), & \text{op}(a) \in \{\text{INSERT}, \text{DELETE}\}, \\ \mathcal{L}_{\text{remask}}(s, c, M(a)), & \text{op}(a) = \text{REMASK}, \\ \mathcal{L}_{\text{stop}}(s, c), & \text{op}(a) = \text{STOP}, \end{cases} \tag{7}$$

where $M(a)$ denotes the target remask set for a REMASK example. Thus, each training example contributes only the loss corresponding to its labeled action. We omit per-head scalar weights in Eq. equation 7 for clarity.

Second, synthetic recovery teaches feasible local edits, but not all feasible edits are useful under the frozen denoiser. We therefore mine improving actions from task-agnostic general text states. For each sampled state, we enumerate candidate edits and execute each candidate, using the frozen denoiser whenever lexical tokens must be generated or remasked. We keep actions that improve the resulting draft under target-aware rollout evaluation. The target text is used only during offline label

construction and is never available at inference time. These mined actions provide outcome-informed imitation labels, but the update remains purely supervised. The selector is not trained, and the denoiser remains frozen. Let $\mathcal{D}_{\text{mine}}$ denote the resulting set of mined state-action pairs (s, c, a^+).

Stage 1 minimizes the mixture objective

$$\mathcal{L}_{\text{stage1}} = \mathbb{E}_{(s,c,a^\star)\sim\mathcal{D}_{\text{syn}}}\left[\mathcal{L}_{\text{act}}(s, c, a^\star)\right] + \lambda_{\text{mine}}\mathbb{E}_{(s,c,a^+)\sim\mathcal{D}_{\text{mine}}}\left[\mathcal{L}_{\text{act}}(s, c, a^+)\right]. \tag{8}$$

For INSERT and DELETE, $\mathcal{L}_{\text{act}}$ is the factorized action likelihood over operation, position, and length. For REMASK, it combines operation cross-entropy with multi-label binary cross-entropy over the target remask set. For STOP, it reduces to operation cross-entropy. We apply the objective as a curriculum, training first on synthetic recovery and then on mined improving actions.

Stage 2: learning which edits improve a draft. Executable edits are not necessarily useful. An insertion, deletion, or remasking action may be executable but still degrade the realized draft after denoising. Stage 2 therefore performs outcome-guided policy improvement on task-agnostic general text $\mathcal{D}_{\text{gen}}$, such as OpenWebText or WikiText. Editing states are sampled from synthetic corruptions or short rollouts of the Stage 1 editor. For each state, DELA builds executable candidate edits, evaluates their realized short-horizon outcomes during training, and uses the resulting preferences to improve both proposal and selection. The clean target is used only to construct training preferences and is never available at inference time. The rollout preferences supervise two coupled objectives: a selector-ranking objective and an editor-improvement objective.

Let π_0 denote a frozen reference editor initialized from Stage 1 or from its mined-imitation continuation, and let π_θ be the trainable editor initialized from π_0. At state (s, c), the current editor proposes a default executable action $a_\theta(s, c)$ by decoding its operation, position–length, or remask-set outputs. We construct a small executable candidate set

$$\mathcal{C}(s, c) = \{a_\theta(s, c), a_{\text{STOP}}\} \cup \mathcal{C}_{\text{pol}}(s, c) \cup \mathcal{C}_{\text{heur}}(s, c), \tag{9}$$

where $\mathcal{C}_{\text{pol}}$ contains high-probability actions decoded from π_θ, and $\mathcal{C}_{\text{heur}}$ contains legal boundary or span proposals. Remask candidates are obtained from top-ranked token or block scores produced by the remask head. Candidate construction uses only information available at inference time. The clean target x is used only to assign rollout-derived training preferences. Heuristic candidates serve as legal proposal augmentations and diagnostics for estimating the value of available edits.

For each non-STOP candidate action a, DELA executes $s_1^a = T_\tau(a; s, c)$ and estimates its short-horizon consequence by rolling out K additional steps with a fixed continuation policy ρ, yielding $(\tilde{s}_K^a, C_K^a) = \text{Roll}_K^\rho(s_1^a, c)$. For STOP, no edit or rollout is performed, so $s_1^a = \tilde{s}_K^a = s$ and $\text{cost}(a) = C_K^a = 0$. In our default implementation, ρ is a frozen reference editor initialized from Stage 1 or its mined-imitation continuation, and is not updated during Stage 2. Each candidate is scored by a truncated improvement value:

$$Q_K^\rho(s, a) = F_{\text{gen}}(\tilde{s}_K^a; x, c) - F_{\text{gen}}(s; x, c) - \lambda_{\text{edit}}\left(\text{cost}(a) + C_K^a\right), \tag{10}$$

where F_{gen} is a task-agnostic outcome score on general text. In practice, we instantiate it with structural recovery and denoising-consistency metrics.

The rollout values define a soft preference target over the candidate set:

$$p^\star(a \mid s, c) = \frac{\exp(Q_K^\rho(s, a)/\tau_{\text{pref}})}{\sum_{b\in\mathcal{C}(s,c)} \exp(Q_K^\rho(s, b)/\tau_{\text{pref}})}. \tag{11}$$

This target supervises both edit selection and proposal improvement.

The selector defines a candidate-normalized distribution

$$q_\psi(a \mid s, c, \mathcal{C}) = \frac{\exp h_\psi(s, c, a)}{\sum_{b\in\mathcal{C}(s,c)} \exp h_\psi(s, c, b)}, \tag{12}$$

and is trained with

$$\mathcal{L}_{\text{sel}} = -\mathbb{E}_{(s,c)} \sum_{a\in\mathcal{C}(s,c)} p^\star(a \mid s, c) \log q_\psi(a \mid s, c, \mathcal{C}). \tag{13}$$

To improve the editor itself, we normalize the editor distribution over the same candidate set:

$$\bar{\pi}_\theta(a \mid s, c, \mathcal{C}) = \frac{\pi_\theta(a \mid s, c)}{\sum_{b\in\mathcal{C}(s,c)} \pi_\theta(b \mid s, c)}. \tag{14}$$

For structural candidates, $\pi_\theta(a \mid s, c)$ is given by the operation–position–length factorization in Eq. 2. For remask candidates, we combine the operation probability of REMASK with the corresponding Bernoulli mask-set score as a surrogate action log-probability. We distill the editor toward outcome-preferred actions while anchoring it to the frozen reference editor:

$$\mathcal{L}_{\text{edit}} = -\mathbb{E}_{(s,c)} \sum_{a \in \mathcal{C}(s,c)} p^\star(a \mid s, c) \log \bar{\pi}_\theta(a \mid s, c, \mathcal{C}) + \lambda_{\text{KL}} D_{\text{KL}} \left(\bar{\pi}_\theta(\cdot \mid s, c, \mathcal{C}) \| \bar{\pi}_0(\cdot \mid s, c, \mathcal{C}) \right), \quad (15)$$

where $\bar{\pi}_0$ is the candidate-normalized distribution induced by the frozen reference editor over the same candidate set. The Stage 2 objective is

$$\mathcal{L}_{\text{stage2}} = \mathcal{L}_{\text{sel}} + \eta_{\text{edit}} \mathcal{L}_{\text{edit}}. \quad (16)$$

Stage 2 therefore improves the edit controller from realized short-horizon outcomes. The selector learns conservative candidate selection, while the editor is distilled toward actions that lead to better general structural outcomes.

At inference time, DELA does not evaluate candidates by rolling them out against a target. Instead, the selector scores one-step candidate actions before execution, and a conservative margin gate decides whether to apply the selected candidate. Let $\hat{a} = \arg\max_{a \in \mathcal{C}(s,c)} h_\psi(s, c, a)$. DELA executes $\hat{a}$ only when its score exceeds that of the default editor action a_θ by an operation-dependent margin:

$$a_t = \begin{cases} \hat{a}, & h_\psi(s, c, \hat{a}) - h_\psi(s, c, a_\theta) > \gamma_{\text{op}(a_\theta)}, \\ a_\theta, & \text{otherwise.} \end{cases} \quad (17)$$

The gate limits deviations from the default editor action; by itself, it does not favor STOP over an edit.

Stage 3: aligning edits with task success. General structural improvement does not always coincide with downstream success. Stage 3 specializes DELA with task feedback while reusing the Stage 2 candidate construction, short-horizon rollout evaluation, selector-ranking objective, and editor-improvement objective. The task-agnostic score F_{gen} is replaced with a task reward F_{task}. Depending on the task, F_{task} can be an exact-match or verifier score for reasoning, execution success or pass rate for code, or a learned preference reward for open-ended generation. Task rewards are computed on candidate rollouts from $\mathcal{D}_{\text{task}}$ during training and converted into preferences using Eq. equation 11. The task-level preferences align the edit controller with downstream success while keeping the diffusion backbone frozen. At inference time, DELA uses the same conservative margin gate for task-aligned global refinement.

Inference-time execution. At inference time, DELA first constructs a draft through append-only masked-block insertion with local remasking and then performs global refinement. No target text or rollout-based candidate evaluation is used: the editor proposes one-step actions, the selector scores them, and the margin gate in Eq. equation 17 determines whether to override the default editor action. Refinement continues until STOP is selected or the edit budget is exhausted.

4 EXPERIMENTS

We primarily evaluate DELA as a plug-in controller for frozen LLaDA-1.5 and additionally test its applicability to Dream-7B. Because all backbone parameters remain fixed, any improvement must come from modifying edit trajectories over clean drafts rather than updating the denoising model. Targets and task evaluators are used only for offline training and analysis, never for inference-time edit selection.

4.1 EXPERIMENTAL SETUP

We evaluate DELA on mathematical reasoning and code generation. For mathematical reasoning, we use GSM8K and MATH and report exact-match accuracy after answer extraction. For code generation, we use HumanEval and MBPP and report pass@1 under the standard unit-test protocol. For all methods, the initial draft is generated by the same frozen backbone under the same decoding budget. Stage 3 uses task feedback to train the controller offline; the non-oracle benchmark variants decode with the trained controller fixed and without task-evaluator calls. The proposal editor and selector contain approximately 200M trainable parameters in total, corresponding to about 2.5% of the frozen 8B backbone.

Table 1: Main downstream performance with a frozen LLaDA-1.5 backbone. All entries are percentages; numbers in parentheses denote absolute gains over DIRECT.

Method	GSM8K	MATH	HumanEval	MBPP
DIRECT	80.5	36.7	37.2	39.6
EDITOR-ONLY	83.6 (+3.1)	37.9 (+1.2)	39.0 (+1.8)	41.4 (+1.8)
EDITOR+SELECTOR	85.7 (+5.2)	39.3 (+2.6)	42.7 (+5.5)	43.6 (+4.0)
ORACLE	86.3 (+5.8)	39.8 (+3.1)	43.9 (+6.7)	44.0 (+4.4)

Table 2: Inference-time comparison on HumanEval using the same frozen LLaDA-1.5 backbone.

Method	pass@1	NFE	Latency (s)
DIRECT	37.2	256	5.43
Fast-dLLM	38.4	87	1.72
Tolerator-style	38.4	275	5.46
ReMDM	38.4	512	10.61
DELA-REMASK	40.9	261	5.57
DELA	**42.7**	264	5.74

We compare four main variants. DIRECT returns the shared initial draft without global refinement. This initial draft includes the same per-block local remasking used by DELA. EDITOR-ONLY applies the top action from the proposal editor. EDITOR+SELECTOR uses the learned selector to choose among proposed edit candidates without test-time rollout. ORACLE selects the best candidate using the task evaluator and serves only as an upper bound on the candidate set. DELA-REMASK restricts global refinement to REMASK and STOP.

4.2 MAIN RESULTS

Table 1 reports the downstream results with the frozen LLaDA-1.5 backbone. Since the backbone is fixed for all methods, these comparisons isolate the effect of controlling edit trajectories.

DELA consistently improves over direct decoding without updating the denoising model. The gain from EDITOR-ONLY shows that the learned editor can propose useful interventions, while the further improvement from EDITOR+SELECTOR shows that deciding which proposed edits to execute is important. The remaining gap to ORACLE indicates that useful candidates are sometimes available but are not selected by the learned selector.

4.3 COMPARISON WITH INFERENCE-TIME REFINEMENT

We next compare DELA with alternative inference-time decoding and refinement strategies on HumanEval under the same frozen LLaDA-1.5 backbone and inference environment in Table 2. This comparison separates the effect of DELA's learned edit policy from differences in backbone training or model scale. NFE denotes the number of backbone forward evaluations. The NFE comparison includes the shared drafting phase; the additional evaluations for DELA come from global refinement.

The baselines expose different accuracy–cost trade-offs. Fast-dLLM substantially reduces the denoising budget while obtaining a modest improvement over DIRECT, whereas Tolerator-style refinement and ReMDM reach the same 38.4% pass@1 with additional denoising. In contrast, DELA-REMASK reaches 40.9% with only five additional NFEs over DIRECT, showing that learned targeted remasking already captures most of the refinement benefit. Enabling the full edit space further improves performance to 42.7% with only eight additional NFEs. Within the tested budgets, these results show that a small number of targeted interventions can improve generation quality. They do not isolate the effect of the learned edit policy from the extra compute relative to DIRECT.

We also test whether this behavior is specific to LLaDA-1.5. Using the same DELA formulation with a separately trained controller over frozen Dream-7B, DELA improves HumanEval from 41.5% to 43.9% and MBPP from 57.8% to 60.6%. The smaller but consistent gains on Dream provide initial evidence that the editing formulation is applicable across different diffusion backbones rather than

Table 3: Component analysis on HumanEval with the frozen LLaDA-1.5 backbone.

Configuration	# Correct	pass@1
Stage 1	18	11.0
Stage 1 + Stage 2	61	37.2
DELA-REMASK	67	40.9
DELA	70	42.7
ORACLE	72	43.9

relying on a particular LLaDA decoding implementation. Additional decoding comparisons and inference-time sensitivity analyses are provided in Appendix C.4 and Appendix C.5.

4.4 CONTROLLER AND EDITING ANALYSIS

We analyze which components drive DELA's improvement and how the learned controller behaves in practice. Table 3 summarizes the main system-level ablations on HumanEval.

Training stages. Stage 1 teaches the controller to construct executable edits, but not whether an intervention is beneficial on naturally generated drafts. When applied directly to HumanEval, the Stage 1 policy edits every example, breaking 43 previously correct drafts while fixing none. Stage 2 recovers the performance of DIRECT, suggesting that task-agnostic outcome supervision helps avoid the harmful edits made by the Stage 1 policy. Stage 3 then aligns the outcome-aware policy with downstream task success, yielding the full DELA model. Together with the gap between EDITOR-ONLY and EDITOR+SELECTOR in Table 1, these results show that useful proposals alone are insufficient; DELA must also learn which edits are worth executing.

Edit behavior. DELA-REMASK improves HumanEval from 37.2% to 40.9%, while the full action space further improves performance to 42.7%. The REMASK-only result suggests that targeted remasking accounts for most of the gain. This pattern is reflected in the executed actions: across HumanEval, DELA performs 24 REMASK, 2 INSERT, and 1 DELETE actions. Overall, DELA edits only 15 of 164 outputs (9.1%). Among edited drafts, the median intervention changes 12 tokens, approximately 6% of the draft. These results indicate that DELA primarily performs sparse, targeted repair rather than broad rewriting.

Two-phase efficiency. The two-phase design also limits the cost of refinement. In a separate batch-size-one profile, two-phase drafting and refinement requires 0.461 s per sample, compared with 0.823 s when global edit control is invoked after every generated block. Delaying non-monotonic refinement until after draft construction therefore avoids repeatedly invoking the global controller during generation. Additional diagnostics for proposal quality, edit selection, edit behavior, and two-phase efficiency are provided in Appendix C.

5 CONCLUSION

We introduced **DELA**, a plug-in framework for bringing explicit draft-level editing to frozen diffusion language models. DELA keeps the pretrained diffusion backbone fixed and learns an external editor that refines clean drafts through structural edit actions, thereby decoupling draft-level control from token-level denoising. To make the editing policy learnable, we design a staged curriculum that first teaches local edit recovery and then incorporates outcome-guided signals to favor edits that improve downstream task performance. Experiments on reasoning and code generation show that DELA can improve over direct diffusion decoding without retraining the underlying backbone. Our study is limited to two diffusion backbones and a focused task set; broader evaluation across scales and generation settings remains future work.

AI USE STATEMENT

We used generative AI tools primarily for language polishing, including improving grammar, clarity, and readability of the manuscript. The research ideas, methodology, experiments, analysis, and conclusions were developed by the authors. All AI-assisted text was reviewed and edited by the authors, who take full responsibility for the final content of the paper.

REFERENCES

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. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, 2021.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. In *Advances in Neural Information Processing Systems*, 2025.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. In *Advances in Neural Information Processing Systems*, 2022.

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

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. In *Advances in Neural Information Processing Systems*, 2024.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 6112–6121, 2019.

Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Marton Havasi, Brian Karrer, Itai Gat, and Ricky TQ Chen. Edit flows: Variable length discrete flow matching with sequence-level edit operations. In *Advances in Neural Information Processing Systems*, 2025.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. In *International Conference on Machine Learning*, 2025.

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. In *International Conference on Learning Representations*, 2026.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1173–1182, 2018.

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 *International Conference on Learning Representations*, 2026.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-lm improves controllable text generation. In *Advances in Neural Information Processing Systems*, 2022.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion language modeling by estimating the ratios of the data distribution. In *Advances in Neural Information Processing Systems*, 2023.

John Nguyen, Marton Havasi, Tariq Berrada, Luke Zettlemoyer, and Ricky TQ Chen. Oneflow: Concurrent mixed-modal and interleaved generation with edit flows. *arXiv preprint arXiv:2510.03506*, 2025.

Shen Nie, Jingyang Ou, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems*, 2025.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *International Conference on Learning Representations*, 2025.

Machel Reid, Vincent J Hellendoorn, and Graham Neubig. Diffuser: Discrete diffusion via edit-based reconstruction. In *International Conference on Learning Representations*, 2023.

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. In *Advances in Neural Information Processing Systems*, 2024.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K Titsias. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems*, 2024.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pp. 5976–5985. PMLR, 2019.

Xiaohang Tang, Rares Dolga, Sangwoong Yoon, and Ilija Bogunovic. wd1: Weighted policy optimization for reasoning in diffusion language models. In *International Conference on Learning Representations*, 2026.

Runchu Tian, Junxia Cui, Xueqiang Xu, Feng Yao, and Jingbo Shang. Finish first, perfect later: Test-time token-level cross-validation for diffusion large language models. *arXiv preprint arXiv:2510.05090*, 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.

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.

Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Yansong Feng, Zhenguo Li, Victoria W., Guorui Zhou, and Lingpeng Kong. Dreamon: Diffusion language models for code infilling beyond fixed-size canvas. In *International Conference on Learning Representations*, 2026b.

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. URL https://arxiv.org/abs/2508.15487.

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.

Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. In *International Conference on Learning Representations*, 2025.

A Theoretical Insights

A.1 Why Singleton-Remask Consistency Works

We now analyze the remask-based denoising-consistency score Q_{gen}, a component of the task-agnostic score F_{gen} used in the main text. The main practical score uses general multi-token or blockwise remask patterns, but for analysis we focus on the singleton-remask case, where exactly one position is remasked at a time. This setting admits the cleanest statistical interpretation: the corresponding population objective is exactly the conditional pseudo-log-likelihood of the data distribution. We then show why this objective is a principled surrogate for joint likelihood: for strictly positive discrete distributions, matching all full-conditionals is sufficient to recover the same joint distribution.

Setup. Fix a conditioning context c and a length-n sequence $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{V}^n$. Let $q^\star(\mathbf{x} \mid c)$ denote the data distribution, assumed strictly positive:

$$q^\star(\mathbf{x} \mid c) > 0 \qquad \text{for all } \mathbf{x} \in \mathcal{V}^n.$$

For each position $i \in \{1, \ldots, n\}$, let $\mathbf{x}_{\setminus i}$ denote the partially observed sequence obtained by masking out the i-th token.

Assume that the frozen absorbing diffusion backbone induces singleton-remask reconstruction distributions

$$p_\phi(x_i \mid \mathbf{x}_{\setminus i}, c),$$

which we interpret as model full-conditionals.

Define the singleton-remask score

$$Q^{(1)}_{\text{gen}}(\mathbf{x}, c) = \frac{1}{n} \sum_{i=1}^{n} \log p_\phi(x_i \mid \mathbf{x}_{\setminus i}, c), \tag{18}$$

and the corresponding population objective

$$\mathcal{J}(\phi) = \mathbb{E}_{(\mathbf{x},c)\sim q^\star} \left[Q^{(1)}_{\text{gen}}(\mathbf{x}, c) \right]. \tag{19}$$

Theorem 1 (Population consistency of singleton-remask pseudo-likelihood)**.** *For every ϕ,*

$$\mathcal{J}(\phi) = \mathcal{C} - \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{(\mathbf{x}_{\setminus i},c)\sim q^\star} \left[\mathrm{KL}\big(q^\star(\cdot \mid \mathbf{x}_{\setminus i}, c) \,\big\|\, p_\phi(\cdot \mid \mathbf{x}_{\setminus i}, c)\big) \right], \tag{20}$$

where

$$\mathcal{C} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{(\mathbf{x},c)\sim q^\star} \left[\log q^\star(x_i \mid \mathbf{x}_{\setminus i}, c) \right] \tag{21}$$

is independent of ϕ.

Hence $\mathcal{J}(\phi) \le \mathcal{C}$ for all ϕ, with equality if and only if

$$p_\phi(\cdot \mid \mathbf{x}_{\setminus i}, c) = q^\star(\cdot \mid \mathbf{x}_{\setminus i}, c) \quad \textit{for all } i \textit{ and } q^\star\textit{-a.e. } (\mathbf{x}_{\setminus i}, c).$$

In particular, maximizing the singleton-remask objective is exactly equivalent to maximizing the conditional pseudo-log-likelihood of the data distribution.

Proof. By definition,

$$\mathcal{J}(\phi) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{(\mathbf{x},c)\sim q^\star} \left[\log p_\phi(x_i \mid \mathbf{x}_{\setminus i}, c) \right]. \tag{22}$$

Fix any position i. Using conditional expectation,

$$\mathbb{E}_{(\mathbf{x},c)\sim q^\star} \left[\log p_\phi(x_i \mid \mathbf{x}_{\setminus i}, c) \right] = \mathbb{E}_{(\mathbf{x}_{\setminus i},c)\sim q^\star} \left[\sum_{v \in \mathcal{V}} q^\star(v \mid \mathbf{x}_{\setminus i}, c) \log p_\phi(v \mid \mathbf{x}_{\setminus i}, c) \right]. \tag{23}$$

Add and subtract the data conditional log-probability:

$$\sum_{v\in\mathcal{V}} q^\star(v \mid \mathbf{x}_{\setminus i}, c)\log p_\phi(v \mid \mathbf{x}_{\setminus i}, c)$$
$$= \sum_{v\in\mathcal{V}} q^\star(v \mid \mathbf{x}_{\setminus i}, c)\log q^\star(v \mid \mathbf{x}_{\setminus i}, c)$$
$$- \mathrm{KL}\big(q^\star(\cdot \mid \mathbf{x}_{\setminus i}, c) \,\big\|\, p_\phi(\cdot \mid \mathbf{x}_{\setminus i}, c)\big)\,. \tag{24}$$

Substituting Eq. equation 24 into Eq. equation 23 and then into Eq. equation 22 yields Eq. equation 20. The equality condition follows from the nonnegativity of KL divergence. □

Corollary 1 (Why pseudo-likelihood works as a surrogate for joint likelihood)**.** *Assume furthermore that the model family can represent the data full-conditionals and that, for each context c with $q^\star(c) > 0$, the family of model full-conditionals*

$$\big\{p_\phi(x_i \mid \mathbf{x}_{\setminus i}, c)\big\}_{i=1}^n$$

is compatible with a unique strictly positive joint distribution $p_\phi(\mathbf{x} \mid c)$. If $\phi^\star$ maximizes $\mathcal{J}(\phi)$, then

$$p_{\phi^\star}(\mathbf{x} \mid c) = q^\star(\mathbf{x} \mid c) \qquad \textit{for all } \mathbf{x} \in \mathcal{V}^n \textit{ and } c \textit{ with } q^\star(c) > 0.$$

Therefore, although singleton-remask training optimizes only local full-conditionals, it identifies the same target joint distribution as joint maximum likelihood.

Proof. By the realizability assumption, some parameter attains $\mathcal{J}(\phi) = \mathcal{C}$. Theorem 1 shows that $\mathcal{C}$ is an upper bound, so any maximizer $\phi^\star$ also attains it and matches the data full-conditionals:

$$p_{\phi^\star}(x_i \mid \mathbf{x}_{\setminus i}, c) = q^\star(x_i \mid \mathbf{x}_{\setminus i}, c)$$

for every i and $q^\star$-almost every $(\mathbf{x}, c)$.

It remains to show why equality of all full-conditionals determines the joint distribution. Fix a reference state

$$\mathbf{x}^{(0)} = (a_1, \ldots, a_n) \in \mathcal{V}^n.$$

For any target state $\mathbf{x} = (x_1, \ldots, x_n)$, define the path

$$\mathbf{x}^{(k)} = (x_1, \ldots, x_k, a_{k+1}, \ldots, a_n), \qquad k = 0, 1, \ldots, n.$$

Then $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(n)} = \mathbf{x}$ are connected by a sequence of single-coordinate changes. For any strictly positive joint distribution p, we have

$$\frac{p(\mathbf{x})}{p(\mathbf{x}^{(0)})} = \prod_{k=1}^n \frac{p(\mathbf{x}^{(k)})}{p(\mathbf{x}^{(k-1)})}$$
$$= \prod_{k=1}^n \frac{p(x_k \mid x_1, \ldots, x_{k-1}, a_{k+1}, \ldots, a_n)}{p(a_k \mid x_1, \ldots, x_{k-1}, a_{k+1}, \ldots, a_n)}. \tag{25}$$

Thus, all probability ratios relative to the reference state are completely determined by the full-conditionals.

Applying Eq. equation 25 to both $p_{\phi^\star}(\mathbf{x} \mid c)$ and $q^\star(\mathbf{x} \mid c)$, and using the fact that their full-conditionals are equal, yields

$$\frac{p_{\phi^\star}(\mathbf{x} \mid c)}{p_{\phi^\star}(\mathbf{x}^{(0)} \mid c)} = \frac{q^\star(\mathbf{x} \mid c)}{q^\star(\mathbf{x}^{(0)} \mid c)} \qquad \text{for all } \mathbf{x} \in \mathcal{V}^n.$$

Hence there exists a constant $Z(c) > 0$ such that

$$p_{\phi^\star}(\mathbf{x} \mid c) = Z(c)\, q^\star(\mathbf{x} \mid c) \qquad \text{for all } \mathbf{x} \in \mathcal{V}^n.$$

Since both distributions are normalized,

$$1 = \sum_{\mathbf{x}} p_{\phi^\star}(\mathbf{x} \mid c) = Z(c) \sum_{\mathbf{x}} q^\star(\mathbf{x} \mid c) = Z(c),$$

so $Z(c) = 1$. Therefore

$$p_{\phi^\star}(\mathbf{x} \mid c) = q^\star(\mathbf{x} \mid c) \qquad \text{for all } \mathbf{x} \in \mathcal{V}^n.$$

□

Remark. The theorem above explains why pseudo-likelihood works. It does not need to equal the normalized joint log-likelihood pointwise. Instead, its population optimum matches the data full-conditionals, and for strictly positive discrete distributions these full-conditionals determine the entire joint distribution. This is why pseudo-likelihood has long been used as a tractable surrogate when the exact joint likelihood is difficult to compute. This result concerns fixed-length singleton remasking; it does not establish that edits selected using the practical score improve downstream task outcomes.

In the main DELA reward, we use more general multi-token or blockwise remask patterns:

$$Q_{\mathrm{gen}}(s, c) = \mathbb{E}_{M \sim \mu(\cdot|s)} \left[\frac{1}{|M|} \sum_{i \in M} \log p_\phi(s_i \mid \mathrm{mask}(s, M), c) \right].$$

These practical variants are motivated by the singleton-remask theory above, but should be viewed as conditional-consistency surrogates rather than exact normalized joint log-likelihood estimators.

B Implementation Details

B.1 Experimental Configuration

Unless otherwise stated, all experiments use LLaDA-1.5 as the frozen denoising backbone. The backbone parameters are never updated. We train only the DELA edit-control modules, including the proposal editor and the selector. The DIRECT baseline uses the same initial-draft pipeline as DELA, including local per-block remasking, and returns the draft before global refinement. In our current implementation, this initial draft is generated with `gen_length` $= 256$, `steps` $= 256$, and `block_length` $= 16$. NFE counts calls to the frozen backbone across the full decoding procedure, including local remasking during drafting and any subsequent global refinement. Thus, the 256 NFEs reported for DIRECT cover the shared drafting phase, while the eight additional NFEs reported for DELA arise during global refinement. Table 4 summarizes the main implementation hyperparameters used by the training and decoding algorithms.

Table 4: Main implementation hyperparameters for DELA.

Hyperparameter	Value
Denoising backbone	LLaDA-1.5
Backbone update	Frozen
Denoising operator	$\mathcal{D}_\tau$
Initial draft length	`gen_length` $= 256$
Initial denoising iterations	`steps` $= 256$
Initial block length	`block_length` $= 16$
Confidence threshold	$\tau = 0.9$
Edit refinement budget	$T_{\max} = 2$ actions
Candidate rollout horizon	$K = 1$ edit step
Stage2 selector margin	$\gamma_{\mathrm{op}} \in \{0.10, 0.15\}$
Stage3 selector margin	$\gamma_{\mathrm{op}} = 0$ unless otherwise stated
Trainable modules	Proposal editor π_θ, selector h_ψ

For conservative selector diagnostics, we additionally report fixed-margin variants. These margins are specified in the corresponding result tables. During final task evaluation, the selector scores executable candidates using learned logits only. Rollout rewards, verifier feedback, target answers, and oracle candidate scores are used only for training or diagnostic upper bounds, never for test-time candidate selection. For Stage3 task evaluation, we disable test-time rollout reranking by setting `selector_rollout_rerank_topk` $= 0$.

B.2 Compute Resources

We report the compute resources used in our experiments to support reproducibility. All main experiments were conducted on a workstation with 8 NVIDIA RTX PRO 6000 GPUs. The LLaDA-

1.5 backbone was kept frozen throughout all experiments, and training only updated the DELA edit-control modules. Unless otherwise stated, training and evaluation used the same machine.

Table 5: Compute resources used for DELA experiments.

Experiment	GPUs	Trainable components
Stage 1: learning executable edit proposals	8 RTX PRO 6000	Proposal editor
Stage 2: learning which edits improve a draft	8 RTX PRO 6000	Proposal editor, selector
Stage 3: aligning edits with task success	8 RTX PRO 6000	Proposal editor, selector
Downstream evaluation	8 RTX PRO 6000	None

B.3 Training Algorithm

Algorithm 1 summarizes the DELA training curriculum. The curriculum first learns executable edit proposals through imitation, then uses mined improving actions to strengthen the warm start. Subsequent stages use rollout-derived preferences and downstream feedback to train outcome-aware edit selection.

Algorithm 1 DELA training curriculum

Require: General corpus $\mathcal{D}_{\text{gen}}$, task corpus $\mathcal{D}_{\text{task}}$, frozen denoiser $\mathcal{D}_\tau$, editor π_θ, selector h_ψ

Stage 1: synthetic edit imitation
Construct synthetic recovery dataset $\mathcal{D}_{\text{syn}}$ from $\mathcal{D}_{\text{gen}}$.
for each labeled example $(s, c, a^\star) \sim \mathcal{D}_{\text{syn}}$ **do**
  Update π_θ with the action imitation loss $\mathcal{L}_{\text{act}}(s, c, a^\star)$.
end for
Stage 1.5: mined improving-action distillation
Mine improving actions $\mathcal{D}_{\text{mine}}$ from task-agnostic text states using the frozen denoiser.
for each mined example $(s, c, a^+) \sim \mathcal{D}_{\text{mine}}$ **do**
  Update π_θ with $\mathcal{L}_{\text{act}}(s, c, a^+)$.
end for
Set frozen reference editor $\pi_0 \leftarrow \pi_\theta$.
Initialize the trainable editor from π_0.
Stage 2: general outcome-guided improvement
for each editing state (s, c, x) from $\mathcal{D}_{\text{gen}}$ **do**
  Build executable candidate set $\mathcal{C}(s, c)$ using Eq. 9.
  for each candidate $a \in \mathcal{C}(s, c)$ **do**
    if $a =$ STOP **then**
      Set $s_1^a = \tilde{s}_K^a = s$ and $C_K^a = 0$.
    else
      Execute $s_1^a = T_\tau(a; s, c)$ and roll out $(\tilde{s}_K^a, C_K^a) = \text{Roll}_K^\rho(s_1^a, c)$ with reference policy ρ.
    end if
    Compute $Q_K^\rho(s, a)$ using Eq. 10.
  end for
  Convert rollout values to $p^\star(a \mid s, c)$ using Eq. 11.
  Update h_ψ and π_θ with Eq. 16.
end for
Stage 3: task-specific alignment
for each editing state (s, c, x) from $\mathcal{D}_{\text{task}}$ **do**
  Repeat candidate construction and short-horizon rollout evaluation as in Stage 2.
  Replace F_{gen} with task reward F_{task}.
  Update the same outcome-guided objectives with task-level preferences.
end for

B.4 Decoding Algorithm

Algorithm 2 details DELA decoding. The procedure first constructs a draft through append-only insertion with local per-block remasking, and then performs global refinement with the full edit action

space. The fast-drafting steps, including local remask selection, are shared by DIRECT and DELA. Each edit-conditioned denoising call has a finite iteration budget and resolves any masks remaining at its final iteration by argmax prediction. During refinement, the selector can override the default action of the editor only when the margin gate is satisfied, keeping decoding conservative.

Algorithm 2 DELA decoding with fast drafting and global refinement

Require: Prompt/context c, frozen denoiser $\mathcal{D}_\tau$, editor π_θ, selector h_ψ, draft budget $B_{\max}$, refinement budget $T_{\max}$

Initialize $s \leftarrow \emptyset$
Fast drafting
for $b = 1, \ldots, B_{\max}$ **do**
  Select append-only action $a_b = \textsc{Insert}(\text{pos} = |s|, \text{len} = \ell_b)$.
  Apply edit–denoise transition $s \leftarrow T_\tau(a_b; s, c)$.
  Let $\mathcal{B}_b$ denote the newly generated block.
  Predict local remask scores on $\mathcal{B}_b$ and select $M_b \subseteq \mathcal{B}_b$.
  if $M_b \neq \emptyset$ **then**
    Apply local remask transition $s \leftarrow T_\tau(\textsc{Remask}(M_b); s, c)$.
  end if
  if draft stopping criterion is met **then**
    break
  end if
end for
Global refinement
for $t = 1, \ldots, T_{\max}$ **do**
  Compute default editor action $a_\theta = \arg\max_a \pi_\theta(a \mid s, c)$.
  Build candidate set $\mathcal{C}(s, c)$ using Eq. 9.
  Ensure $a_\theta \in \mathcal{C}(s, c)$.
  Select $\hat{a} = \arg\max_{a \in \mathcal{C}(s,c)} h_\psi(s, c, a)$.
  if $h_\psi(s, c, \hat{a}) - h_\psi(s, c, a_\theta) > \gamma_{\text{op}(a_\theta)}$ **then**
    $a_t \leftarrow \hat{a}$
  else
    $a_t \leftarrow a_\theta$
  end if
  if $a_t = \textsc{Stop}$ **then**
    break
  end if
  Apply edit–denoise transition $s \leftarrow T_\tau(a_t; s, c)$.
end for
return s

C ADDITIONAL RESULTS

C.1 STAGE 1: SYNTHETIC RECOVERY

Stage 1 provides a supervised initialization for the proposal editor through synthetic recovery. Given a clean sequence, we construct a corrupted draft using synthetic structural perturbations and train the editor to predict a recovery action that moves the corrupted draft back toward the clean target. For example, deleting a span from the clean sequence induces an INSERT recovery action, while inserting a distractor span induces a DELETE recovery action. This diagnostic evaluates whether the editor has learned the basic syntax and semantics of executable structural edits, including the operation type, edit position, and span length.

Table 6 reports the Stage 1 synthetic recovery results. The editor achieves high operation and position accuracy, indicating that it can identify the intended edit type and approximate edit location. Length prediction is more challenging, especially for insertion examples, because multiple nearby span lengths can often lead to similar repaired states. For this reason, we report tolerance-based length and joint metrics in addition to exact joint accuracy. These results should be interpreted as action-grammar diagnostics rather than evidence of downstream task improvement.

Table 6: Stage 1 synthetic recovery diagnostics. The editor is evaluated on synthetic corrupted drafts and predicts the recovery action toward the clean target. These metrics measure whether the editor learns executable edit syntax, not whether the resulting policy improves downstream generation.

Scope	Op acc.	Pos acc.	Op+Pos acc.	Len ±1	Len ±2	Joint ±1	Joint ±2	Exact joint
Overall	89.5	83.5	74.8	41.5	59.8	30.3	45.8	12.3
Insert	95.5	70.0	67.5	27.5	43.5	19.0	31.0	4.0
Delete	83.5	97.0	82.0	55.5	76.0	41.5	60.5	20.5

Table 7: Stage 1 proposal diagnostics, averaged over two seeds. Large gain denotes $\Delta > 0.02$. All entries except Avg. gain are percentages.

Policy	Mined-positive	Top-1 improves	Large gain	Avg. gain
Synthetic recovery only	38.3	50.0	14.8	0.0070
Final Stage 1 editor	84.4	87.5	57.8	0.0230

C.2 Stage 1: Proposal Quality

Synthetic recovery teaches the editor how to construct executable actions, but does not establish whether these actions are useful under the frozen denoiser. We therefore evaluate the effect of the mined improving-action distillation used in the second part of Stage 1.

Table 7 compares the editor trained only on synthetic recovery with the final Stage 1 editor after mined-positive distillation. Mined-positive measures the fraction of examples for which the candidate set contains a rollout-mined improving action, while Top-1 improves measures whether the highest-probability editor action improves the completed draft. Large gain denotes an improvement greater than 0.02 under the rollout score.

Mined-positive distillation substantially improves proposal quality. The fraction of examples containing an improving proposal increases from 38.3% to 84.4%, while the fraction for which the editor's top action improves the draft increases from 50.0% to 87.5%. Thus, synthetic recovery provides the action grammar, while mined positives shift the proposal distribution toward edits that are useful under the frozen denoiser.

C.3 Stage 2: Selection Diagnostics

Stage 2 trains the selector to distinguish useful executable edits from actions that are locally valid but harmful after denoising. We evaluate this behavior on 256 held-out editing states.

The Stage 2 selector substantially improves both average reward and recovery rate over directly executing the Stage 1 editor action. The remaining gap to the oracle indicates that useful candidates are often available but are not always identified by the learned selector. This complements the downstream results in Section 4.4, where Stage 2 primarily prevents harmful over-editing before task-specific alignment.

C.4 Additional Inference-Time Comparisons

We provide additional controlled decoding comparisons on HumanEval using the same frozen LLaDA-1.5 backbone. These experiments test whether the gains of DELA can be explained by alternative block sizes, token ordering, or additional token-level refinement.

Changing the block size alone does not consistently improve generation quality. A block size of 32 provides a modest improvement over the default configuration, whereas larger blocks eventually degrade substantially. Random ordering also performs poorly, indicating that the gain is not explained simply by changing the order in which positions are generated. Tolerator-style refinement improves over DIRECT, but requires additional denoising evaluations and remains below both DELA-REMASK and full DELA.

Table 8: Stage 2 selection diagnostics on 256 validation examples. Recovered is the percentage of examples recovered after the selected edit.

Policy	Avg. reward	Recovered (%)
Stage 1 editor-only	-0.1270	40.6
Stage 2 selector	-0.0250	63.5
Oracle over candidates	0.1002	79.7

Table 9: Additional decoding comparisons on HumanEval. All methods use the same frozen LLaDA-1.5 backbone.

Method	pass@1	NFE	Latency (s)
Block size $= 8$	37.8	256	5.09
Block size $= 16$ (DIRECT)	37.2	256	5.43
Block size $= 32$	39.0	256	4.99
Block size $= 64$	37.8	256	5.04
Block size $= 256$	14.0	256	5.26
Random ordering ($b = 32$)	22.6	256	5.13
Tolerator-style ($T = 2, k = 8$)	38.4	275	5.46
Tolerator-style ($T = 2, k = 16$)	38.4	291	6.19
DELA-REMASK	40.9	261	5.57
DELA	42.7	264	5.74

C.5 INFERENCE-TIME SENSITIVITY

We study the sensitivity of DELA to the selector margin and the maximum number of refinement rounds on HumanEval.

Performance is stable across $\gamma_{\text{op}} \in [0, 0.5]$, while disabling selector-guided overrides with $\gamma_{\text{op}} = \infty$ reduces performance to the EDITOR-ONLY result of 64 correct solutions. Increasing the refinement budget from one to two rounds improves the number of solved problems from 68 to 70, while a third round provides no further gain. These results indicate that DELA does not require aggressive selector overrides or long refinement trajectories in the current setting.

C.6 EDIT BEHAVIOR AND DIFFICULTY

We provide a more detailed characterization of the edits executed by DELA on HumanEval. We report statistics both over the full benchmark and over the subset of examples for which the initial DIRECT draft is incorrect.

The executed edits are predominantly sparse and local. Only 9.1% of all HumanEval drafts receive an edit, and the median edited example changes 12 tokens. REMASK dominates the executed action distribution, while INSERT and DELETE are used only occasionally for length-changing corrections. These results support the interpretation of DELA as a sparse, local editing mechanism rather than a general-purpose rewriting system.

C.7 TWO-PHASE DECODING AND CACHE PROFILING

We provide additional profiling results for the two-phase decoding procedure. Because the released LLaDA implementation does not expose prefix KV caching, we implement prefix reuse during the fast-drafting phase and compare cached and uncached execution under the same frozen-prefix condition. In both variants, finalized prefix states remain fixed as new blocks are generated. The comparison measures the cost of reusing those states, not the effect of the frozen-prefix approximation relative to full bidirectional recomputation.

Prefix reuse reduces the number of processed query positions by $7.4\times$ and the number of attention pairs by $8.5\times$. The corresponding wall-clock improvement is more modest, from 0.414 s to 0.364 s per sample, consistent with short-sequence batch-one inference being dominated by costs beyond

Table 10: Sensitivity to the selector margin γ_{op} and refinement budget $T_{\max}$ on HumanEval.

Setting	Value	# Correct
Selector margin γ_{op}	0	70
	0.1	70
	0.5	69
	∞	64
Refinement budget $T_{\max}$	1	68
	2	70
	3	70

Table 11: Edit statistics on HumanEval. Draft-length and changed-token statistics are computed only over examples receiving at least one edit.

Metric	All examples	DIRECT-failure subset
Examples receiving an edit	15 / 164 (9.1%)	13 / 103 (12.6%)
REMASK / INSERT / DELETE	24 / 2 / 1	18 / 1 / 1
Changed tokens per REMASK, median	9	9
Changed tokens per edited example, median	12	12
Changed-token fraction, median	6.3%	5.9%

attention computation. The wall-clock benefit increases with sequence length, from $1.14\times$ at 256 tokens to $1.25\times$ at 512 tokens and $1.62\times$ at 1,024 tokens.

We also compare the default two-phase procedure with invoking global edit control throughout drafting.

The $1.8\times$ configuration is a conservative measure of the overhead of repeatedly invoking global edit control, because the controller selects STOP at all block boundaries in this experiment. The forced-edit configuration illustrates the larger cost of performing non-monotonic refinement throughout draft construction. Together, these results show that the two-phase design avoids repeated global-controller overhead while making prefix reuse possible during the fast-drafting phase.

Table 12: Prefix-cache profiling for 256-token fast drafting with batch size one.

Configuration	Latency (s)	Query positions	Attention pairs
Without prefix cache	0.414	5,370	1.51M
With prefix cache	0.364	727	0.18M

Table 13: Cost of two-phase decoding compared with invoking global refinement throughout draft construction.

Procedure	Latency (s/sample)	Relative cost
Two-phase drafting and refinement	0.461	1.0×
Refinement after every block	0.823	1.8×
Forced edit after every block	2.466	5.4×