CLOSING PERFORMANCE GAP FOR DIFFUSION LANGUAGE MODELS UNDER LOW INFERENCE BUDGETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models offer an architectural route to low-budget generation by updating multiple tokens in parallel, which can reduce the number of sequential decoding steps relative to token-by-token autoregressive decoding. However, realizing this advantage at low budgets is challenging, as generation quality often declines when the number of denoising and refinement steps is reduced. Editable diffusion models address this quality loss through post-draft refinement, but useful intermediate results may still be lost before final submission. We introduce CaRR (Commitment-aware Refinement and Recovery), a training-free framework that couples token remasking with trajectory-based output recovery. Token Commitment Remasking (TCR) remasks selected risky pure replacements and refills them on the next scheduled edit forward using the updated context. Submission Commitment Recovery (SCR) uses the resulting refinement trajectory to recover supported candidates when the final response does not satisfy task-specific output requirements. Both modules operate within the native mask/edit schedule without additional backbone forwards. Experiments on code generation and mathematical reasoning show consistent gains in average accuracy over native ME-DLM across four inference budgets, with the largest gains under tight budgets. In particular, at one-eighth of the full scheduled forward budget, CaRR improves the average accuracy across six metrics over native ME-DLM by 7.44 percentage points, including gains of 11.37 and 10.05 points on MBPP and MBPP+.

1 INTRODUCTION

Diffusion language models (DLMs) update multiple tokens in parallel, offering an architectural route to low-budget generation relative to token-by-token autoregressive decoding (Nie et al., 2026; Gong et al., 2025; Ye et al., 2025). However, parallel updates alone do not guarantee lower inference cost or latency. Studies of standard DLM decoding report that repeated full-sequence processing can make inference slower than autoregressive decoding at comparable output quality (Wu et al., 2026; Israel et al., 2026; Hu et al., 2026). Reducing the number of denoising steps lowers sampling cost, but can also degrade task accuracy (Wu et al., 2026; Hu et al., 2026). For DLMs, this motivates improving task accuracy under a limited budget of model forwards.

Iterative refinement offers a way to recover generation quality by revisiting uncertain predictions. Remasking methods revisit uncertain token decisions in later steps; most closely related, T2M delays commitment by remasking low-confidence committed tokens for subsequent reconstruction (Yao, 2026b). Edit-based refinement instead revises a complete draft, as in ME-DLM (Ren et al., 2026), which applies parallel replacement, insertion, and deletion operations to the current response. Yet useful information can still be underused: because parallel edits are predicted from the same pre-edit response, an immediate replacement cannot exploit simultaneous changes to its surrounding context, while terminal-only submission can discard a usable candidate that appeared earlier in the refinement trajectory. This leaves a common challenge: how to make better use of the existing refinement trajectory to improve both local editing decisions and final output selection under a fixed forward budget.

Our insight is to view editable refinement as involving two connected commitment decisions: when to commit a local token replacement during refinement, and which response to submit after refinement. Recognizing that a current token is unreliable does not imply that its immediate replace-

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.

ment is reliable, and completing refinement does not make the terminal response the only useful source of submission evidence. To improve accuracy under limited inference budgets, we introduce CaRR (Commitment-aware Refinement and Recovery), a training-free framework that couples token remasking with trajectory-based output recovery. The two stages are sequentially connected: token-level interventions change the response trajectory that submission-time recovery subsequently examines.

Token Commitment Remasking (TCR) uses the model's support for the existing token to identify risky pure replacements proposed by the native ME-DLM decoder. Instead of immediately committing the replacement tokens, it remasks the selected positions and refills them on the next scheduled edit forward, allowing the predictions to incorporate changes to the surrounding context. This adapts the detect–then–remask principle of T2M (Yao, 2026b) to editable refinement. Submission Commitment Recovery (SCR) uses the resulting refinement trajectory to recover supported candidates when the final response does not satisfy task-specific output requirements. It follows a final-first policy, using task-specific output contracts to determine recovery from the terminal response or earlier states. Code recovery uses AST parsing and public-interface constraints, while Math recovery uses answer normalization, equivalence, and trajectory consensus. Both modules operate within the native mask/edit schedule without additional backbone forwards.

Experiments on code generation and mathematical reasoning show that CaRR improves average accuracy over native ME-DLM across four inference budgets, with the largest gains under tight budgets. In particular, at one-eighth of the full scheduled forward budget, CaRR improves the average accuracy across six metrics by 7.44 percentage points, including gains of 11.37 and 10.05 points on MBPP and MBPP+. Component ablations show that both modules are effective individually and provide stronger gains when combined. Mechanism analyses further support the roles of updated-context remasking and trajectory-based recovery. Our contributions are summarized as follows:

- We formulate low-budget editable refinement as two connected commitment decisions: when to commit a local replacement and which response to submit after refinement.
- We propose CaRR, which adapts commitment deferral to native pure replacements in editable refinement and couples it with final-first trajectory-based recovery, while preserving the native inference schedule.
- We demonstrate higher average accuracy over native ME-DLM across four inference budgets on code generation and mathematical reasoning, with the largest gains under tight budgets. Component and mechanism analyses support the complementary contributions of the two modules.

2 Related Work

Diffusion Language Models and Editable Refinement. Discrete diffusion models provide a non-autoregressive approach to text generation through corruption and iterative denoising (Austin et al., 2021a; Lou et al., 2023; Sahoo et al., 2024), and subsequent work has scaled this paradigm by adapting autoregressive checkpoints or training large masked diffusion models directly (Gong et al., 2025; Nie et al., 2026; Ye et al., 2025). Iterative non-autoregressive refinement also predates modern DLMs. Mask-Predict repeatedly regenerates low-confidence positions, while Levenshtein Transformer and EDITOR refine hypotheses through explicit edit operations (Ghazvininejad et al., 2019; Gu et al., 2019; Xu & Carpuat, 2021). Recent DLMs further increase revisability by preserving intermediate predictive information. Soft-Masked DLMs retain information from previous predictions, while EvoToken-DLM replaces hard mask-to-token transitions with progressively evolving token representations (Hersche et al., 2026; Zhong et al., 2026). ME-DLM instead generates a complete draft and refines it through replacement, insertion, and deletion operations (Ren et al., 2026). Our work adopts this editable post-draft setting and studies how token-level refinement and the resulting response trajectory can be better used under a fixed inference budget.

Low-Budget Diffusion Inference. The efficiency of DLMs depends on maintaining generation quality under tight inference budgets. Theoretical analysis shows that the number of sampling steps required by masked diffusion can depend strongly on the evaluation objective, with sequence-level correctness exhibiting a less favorable efficiency trade-off than perplexity (Feng et al., 2026). Empirically, Fast-dLLM reports quality degradation under aggressive parallel decoding due to disrupted

token dependencies (Wu et al., 2026), while FlashDLM shows that reducing the number of denoising steps can degrade generation quality under standard sampling strategies (Hu et al., 2026). Adaptive Parallel Decoding, entropy-bounded unmasking, and KLASS improve the quality–efficiency trade-off by adapting token updates across denoising steps (Israel et al., 2026; Ben-Hamu et al., 2026; Kim et al., 2026), while dLLM-Cache reduces redundant computation through adaptive caching (Liu et al., 2025). Similar quality–budget trade-offs have also motivated efficient sampling and few-step generation in image diffusion (Song et al., 2020; Lu et al., 2022; Salimans & Ho, 2022; Song et al., 2023). CaRR complements these efficiency-oriented approaches by improving the reliability of post-draft refinement and submission under a fixed native inference budget.

Remasking and Commitment Control. Another line of work restores revisability after token predictions begin to solidify. ReMDM allows generated tokens to be remasked and updated again (Wang et al., 2026a), while RemeDi learns confidence signals for identifying low-quality tokens and resampling them in later steps (Huang et al., 2026). CoRe detects context-brittle predictions through masked-context perturbations (Zhai et al., 2026). Most closely related, T2M identifies low-confidence committed tokens using the model's probability of the current token, remasks them, and reconstructs them in later denoising steps; controlled comparisons further separate the token detector from the remasking action (Yao, 2026b). Related approaches defer uncertain commitments, learn commitment policies from decoding traces, or coordinate multiple token commitments (Shu et al., 2026; Sun et al., 2026a; Yao, 2026a). RDD further introduces backtracking and selective remasking to revise earlier block-level commitments (Wang et al., 2026b), while recent analysis shows that commitment order can directly affect reasoning quality (Yeom et al., 2026). These studies establish the importance of revisability during diffusion generation. TCR adopts this commitment-deferral principle in a different decoding regime: it intervenes on pure replacements selected by the native ME-DLM edit decoder, preserves native INSERT/DELETE actions, and refills selected positions on the next already-scheduled edit forward without increasing the scheduled NFE. The resulting editable-refinement trajectory is subsequently available to SCR for submission-time recovery.

Trajectory-Aware Diffusion Reasoning and Recovery. Recent studies increasingly treat intermediate diffusion states as useful signals beyond their role in producing a terminal sample. DCoLT treats reverse-diffusion states as latent reasoning actions and optimizes the full trajectory with outcome-based reinforcement learning (Huang et al., 2025). Denoising process rewards provide process-level supervision based on the contribution of intermediate denoising intervals (Xie et al., 2026), while SDPO decomposes trajectory alignment into stepwise preference objectives (Han et al., 2026). TIE tracks confidence dynamics across intermediate states and transfers promising partially denoised trajectories between models (Yun et al., 2026), while FReDA uses model-generated drafts as implicit intermediate states for recursive refinement (Sun et al., 2026b). Prophet provides complementary evidence that useful answer information can emerge before terminal decoding and exploits this property for early commitment (Li et al., 2026). Together, these studies show that intermediate states can retain useful information before the terminal output is formed. SCR uses the completed refinement trajectory as submission-time evidence, recovering supported candidates when the native final does not satisfy task-specific output requirements.

3 METHOD: CaRR

3.1 PROBLEM DEFINITION

We study how to improve the final submission of a frozen editable diffusion language model under a fixed forward budget. Native ME-DLM first generates a complete draft and then revises it through parallel replacement, insertion, and deletion operations (Ren et al., 2026). The editing process can improve the draft, but a useful answer need not remain available in the terminal response.

Following ME-DLM's edit-iteration notation, let $x^{(0)}$ be the initial draft and $x^{(t)}$ the response state after t edit updates. With T scheduled edit forwards, the response evolves along

$$x^{(0)} \longrightarrow x^{(1)} \longrightarrow \cdots \longrightarrow x^{(T)}. \tag{1}$$

Native decoding extracts its submission from $x^{(T)}$. Our objective is to improve that submission by using both the editing process and the completed trajectory. We write the input prompt as q;

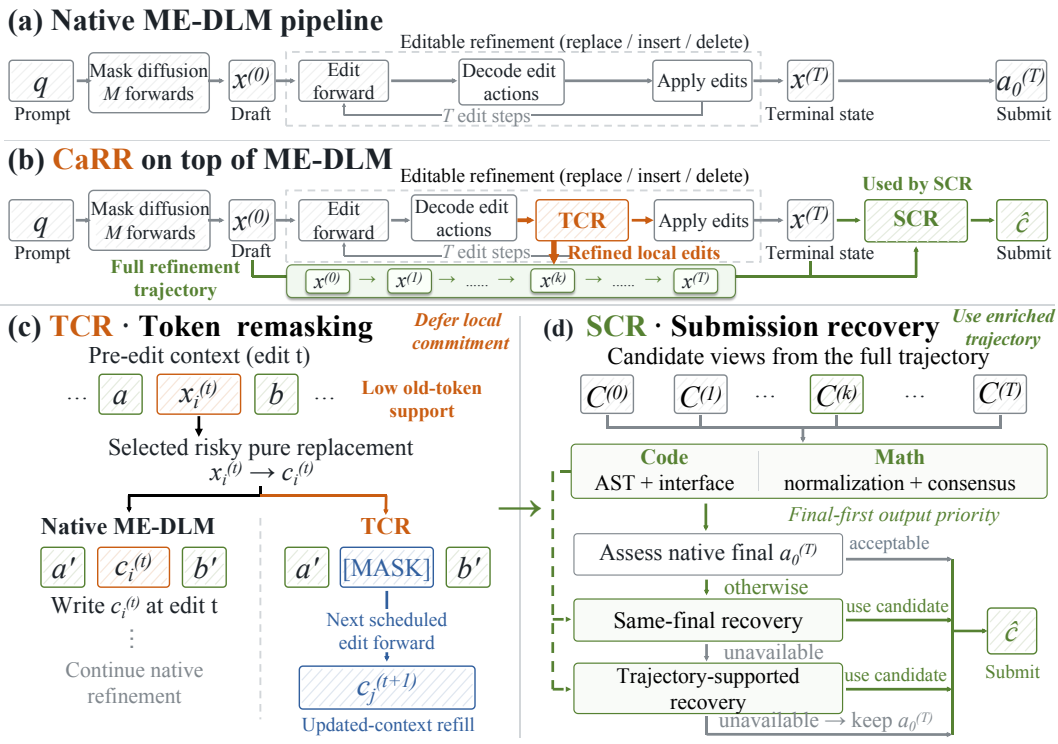


Figure 1: Overview of CaRR on top of ME-DLM. (a) Native decoding generates a draft $x^{(0)}$, applies T scheduled edit updates, and submits the native final view $a_0^{(T)}$. (b) CaRR adds TCR during refinement and SCR at submission. (c) TCR remasks selected native replacements and refills them on the next scheduled forward using updated context. (d) SCR uses candidate views $C^{(t)}$ from the completed trajectory for final-first output recovery.

throughout, t denotes an edit iteration rather than a diffusion noise level. The initial mask-diffusion phase uses M forwards, giving a total scheduled budget of $M + T$.

3.2 Framework Overview

Figure 1 shows where CaRR intervenes. Token Commitment Remasking (TCR) acts before a selected replacement is applied: it writes a mask instead of the proposed token and lets the next scheduled forward refill that position. The refill can therefore use surrounding changes that were unavailable to the original proposal. Submission Commitment Recovery (SCR) acts after the last edit: it extracts candidate views from the saved response states and uses task-specific output rules to determine the submission. The stages are connected through the trajectory: TCR changes the states that SCR subsequently examines. Throughout, $x^{(t)}$ denotes the response state of the run under discussion.

Algorithm 1 summarizes the execution order of CaRR. TCR operates inside the native edit loop, whereas SCR is invoked once after the completed trajectory has been collected.

3.3 Token Commitment Remasking

An immediate replacement answers two questions at once: should the current token change, and what should replace it? Low support for the current token motivates changing it, but does not establish the reliability of the proposed substitute. Moreover, parallel replacements are predicted before neighboring edits are applied. TCR separates these decisions by adapting the detect–then–remask principle of T2M (Yao, 2026b) to native ME-DLM replacements.

Choosing which replacements to defer. At editable position i, let $x_i^{(t)}$ be the existing token and $c_i^{(t)}$ the native decoder's proposed token. TCR considers only *pure replacements*: native actions that replace an existing response token without inserting or deleting a token at that position. Among these

Algorithm 1 CaRR inference

Require: Prompt q; frozen ME-DLM; budgets (M, T); domain d
Ensure: Final submission $\hat{c}$
$x^{(0)} \leftarrow \textsc{MaskDiffusion}(q, M)$
for $t = 0, \ldots, T-1$ **do**
  $\mathcal{E}^{(t)} \leftarrow \textsc{EditForward}(q, x^{(t)})$
  $\mathcal{E}^{(t)} \leftarrow \mathrm{TCR}(x^{(t)}, \mathcal{E}^{(t)})$ ▷ refill and defer replacements
  $x^{(t+1)} \leftarrow \textsc{ApplyEdits}(x^{(t)}, \mathcal{E}^{(t)})$
end for
$C^{(0:T)} \leftarrow \textsc{ExtractViews}(q, x^{(0:T)}, d)$
$\hat{c} \leftarrow \mathrm{SCR}(C^{(0:T)}, d)$ ▷ final-first recovery
return $\hat{c}$

actions, it identifies positions where the scheduled forward assigns low probability to the existing token:

$$p_\theta^{\mathrm{cur}}\left(x_i^{(t)} \mid q, x^{(t)}, i\right) < \tau_{\mathrm{lp}}, \qquad \tau_{\mathrm{lp}} = 0.30. \tag{2}$$

Here p_θ^{cur} is the current-token distribution of the frozen backbone. The probability in Equation 2 measures support for the *old token*, not confidence in $c_i^{(t)}$. Rather than replacing one uncertain commitment with another immediately, TCR uses low old-token support to decide where to defer the replacement.

Keeping remasking local and sparse. INSERT and DELETE actions can change token positions, and a remask must remain associated with the token being edited. TCR therefore excludes positions within distance one of a native structural edit and allows each surviving token to be remasked at most once, carrying its remask history through position shifts. Positions involving control tokens are excluded, and new remasks are disabled on the last edit step, when no later forward remains to fill them. Among the remaining candidates, it selects the lowest-support positions under a per-step cap based on 25% of all editable response positions, with a minimum cap of one when candidates exist. We denote the selected positions by $\mathcal{R}^{(t)}$. The numerical threshold, count limit, and ratio follow the frozen configuration adopted from T2M v2 (Yao, 2026c, Section 5.4); Appendix A.1 gives the exact eligibility conditions, rounding, and position-tracking rules.

Refilling after the context changes. For each $i \in \mathcal{R}^{(t)}$, TCR writes [MASK] instead of $c_i^{(t)}$ and applies the other native actions for that step. The resulting state $x^{(t+1)}$ contains the pending mask together with the surrounding edits. Let j be the mask's position in this updated state; INSERT/DELETE actions elsewhere may have shifted it from i. On the next scheduled forward, the refill is the highest-probability token under the current-token distribution, excluding [PAD], [DEL], and [MASK] from the maximization:

$$c_j^{(t+1)} \leftarrow \arg\max_w p_\theta^{\mathrm{cur}}\left(w \mid q, x^{(t+1)}, j\right). \tag{3}$$

The refill is written to both aligned proposal slots so that native edit application treats it as a replacement rather than an insertion. It is then materialized in the subsequent edited response. The difference from immediate replacement is the conditioning context: $c_i^{(t)}$ was proposed from $x^{(t)}$, whereas the refill uses the surrounding edits already present in $x^{(t+1)}$. This deferral reuses the next scheduled forward instead of adding a separate model call. The exact proposal alignment and position mapping are specified in Appendix A.1.

3.4 Submission Commitment Recovery

Even after token refinement, a usable submission can be absent from the terminal response but present in an earlier state. SCR uses the completed trajectory to recover such candidates. Its *final-first* policy keeps the native final unless the task's output contract identifies an actionable defect or consistency violation and supports a recovery. Thus, the trajectory provides evidence for a submission decision, rather than requiring another round of generation.

Extracting submission evidence. A response can contain several candidate views: for example, a Python block embedded in prose, or a mathematical answer stated in both a box and a concluding equation. SCR extracts these views from each saved response using fixed domain-specific rules. We write the ordered views from $x^{(t)}$ as $C^{(t)}$, and denote the native final view by $a_0^{(T)}$. Candidate views are not new model samples; they are alternative readings or permitted normalizations of text already generated during refinement.

Recovering in final-first order. SCR first assesses the native final using the domain contract. When that assessment calls for intervention, it tries recovery from the same final response before applying the domain's trajectory-supported recovery rules. The first authorized result becomes the submission $\hat{c}$; if neither stage resolves a candidate, the native final is retained. Both assessment and recovery may consult trajectory evidence. The concrete contracts below determine what counts as an actionable problem and which alternative can be returned; the formal decision rule appears in Appendix A.2.

Code recovery. For Code, the contract checks whether a Python candidate parses under the AST and satisfies the public-interface requirement. A structurally valid native final is preserved. Otherwise, SCR first attempts a valid extraction from that same response, such as recovering a complete Python block from surrounding text. If this fails, it scans earlier states from nearest to farthest and returns the first candidate satisfying the same structural contract. This order distinguishes recovery within the terminal response from recovery of a program that survived only in an earlier state. These checks establish structural validity, not functional correctness.

Math recovery. For Math, surface differences can hide equivalent answers, while different parts of a response can support conflicting answers. SCR extracts boxed answers, explicit conclusions, terminal equations, and safe malformed-box views, then groups equivalent answers under frozen normalization rules. Agreement across distinct views within one state supplies support; repeated equivalent answers across consecutive states supply persistence. The recovery rules compare this support and persistence across competing answer groups to recover from an invalid final box or authorize narrowly specified terminal-consensus corrections. A separate canonicalization rule normalizes an explicit closed rational expression already present in the final box. A trajectory-supported candidate need not come from an earlier state: the terminal response can supply the candidate while earlier states supply evidence for it. Appendix A.2 specifies these branches, their thresholds, and their precedence.

The backbone parameters, response-canvas limit, and native (M, T) allocation remain fixed. TCR reuses scheduled forwards, while SCR performs deterministic CPU-side processing after generation. The scheduled NFE therefore remains $M + T$, although the edited response lengths can differ from native decoding.

4 Experiments

4.1 Experimental Setup

Datasets and Metrics. We evaluate CaRR on four benchmarks spanning code generation and mathematical reasoning. For code generation, we use HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b), and report both the original Base accuracy and standard EvalPlus+ accuracy (Liu et al., 2023), where a completion must pass both the Base and additional EvalPlus tests. For mathematical reasoning, we use GSM8K (Cobbe et al., 2021) and MATH-500 (Hendrycks et al., 2021) with benchmark-standard answer equivalence. Unless stated otherwise, "average" denotes the unweighted mean over these six metrics.

Baselines. We use ME-DLM (Ren et al., 2026) as the native decoder and compare against two locally evaluated methods under the same scheduled NFE: CoRe (Zhai et al., 2026), which reserves part of the forward budget for perturbation probes, and a fixed-quota port of KLASS (Kim et al., 2026), which retains the native per-step commitment quota. Budget accounting for the same-NFE comparison is detailed in Appendix B.1. For context, we also include the published per-benchmark scores of Soft Mask (Hersche et al., 2026), EvoToken (Zhong et al., 2026), and LLaDA-Instruct (Nie

Table 1: Fixed-budget accuracy (%) on code generation and mathematical reasoning. † denotes published results and ‡ local fixed-NFE ports. Average is the unweighted mean over six metrics. Gray bold text reports the absolute gain of CaRR over ME-DLM at the same budget. Cell shading indicates relative performance among the locally evaluated methods within each budget; darker shading indicates stronger performance.

Budget	Model	Evaluation Dataset						Average
		HumanEval	HumanEval+	MBPP	MBPP+	GSM8K	MATH-500	
	LLaDA-Instruct†	12.20	9.80	17.50	15.30	50.30	20.20	20.88
1/8	ME-DLM	23.17	20.73	26.46	23.28	63.99	33.80	31.90
	CoRe‡	25.61	23.78	26.98	23.28	63.31	32.40	32.56
	KLASS‡	28.66	28.05	30.95	25.93	63.53	33.00	35.02
	CaRR	31.10	28.05	37.83	33.33	69.75	36.00	39.34
	vs. ME-DLM	**+7.93**	**+7.32**	**+11.37**	**+10.05**	**+5.76**	**+2.20**	**+7.44**
	Soft Mask†	24.80	23.00	32.30	–	62.30	19.80	–
	EvoToken†	–	–	–	–	72.30	31.20	–
	LLaDA-Instruct†	29.30	25.60	26.70	23.00	66.00	23.40	32.33
1/4	ME-DLM	40.85	38.41	43.12	37.30	77.41	42.40	46.58
	CoRe‡	40.24	37.80	43.92	38.36	77.18	45.20	47.12
	KLASS‡	41.46	39.63	39.95	34.66	77.33	42.00	45.84
	CaRR	46.34	43.29	51.06	44.71	79.91	44.00	51.55
	vs. ME-DLM	**+5.49**	**+4.88**	**+7.94**	**+7.41**	**+2.50**	**+1.60**	**+4.97**
	Soft Mask†	38.30	33.80	48.40	–	79.40	38.80	–
	EvoToken†	–	–	–	–	81.80	37.40	–
	LLaDA-Instruct†	43.30	38.40	41.50	34.70	71.00	26.20	42.52
1/2	ME-DLM	48.17	44.51	53.70	45.24	84.08	48.00	53.95
	CoRe‡	48.17	44.51	54.50	47.35	82.18	48.60	54.22
	KLASS‡	55.49	51.83	52.38	45.77	83.62	47.00	56.01
	CaRR	50.61	47.56	56.61	48.15	84.91	48.60	56.07
	vs. ME-DLM	**+2.44**	**+3.05**	**+2.91**	**+2.91**	**+0.83**	**+0.60**	**+2.12**
	Soft Mask†	57.80	50.00	56.40	–	84.00	41.40	–
	EvoToken†	–	–	–	–	84.50	41.00	–
	LLaDA-Instruct†	44.50	37.80	49.50	40.70	71.90	27.60	45.33
1/1	ME-DLM	56.10	52.44	58.47	49.47	84.91	50.00	58.56
	CoRe‡	54.27	51.83	58.20	50.26	85.75	52.20	58.75
	KLASS‡	57.32	53.66	60.32	50.79	84.99	50.40	59.58
	CaRR	56.10	52.44	58.99	50.00	85.82	50.20	58.93
	vs. ME-DLM	**+0.00**	**+0.00**	**+0.52**	**+0.53**	**+0.91**	**+0.20**	**+0.36**

et al., 2026), as reported in Table 2 of Ren et al. (2026); these rows use their reported inference settings and are not included in paired analyses. The TCR-only and SCR-only variants serve as component ablations, and Replace-Veto, which applies the same selection rule as TCR but keeps the old token instead of remasking it, serves as the mechanism control.

Implementation Details. We evaluate the released ME-DLM Stage-3 model (Ren et al., 2026) at scheduled NFE budgets of 64, 128, 256, and 512 while preserving its native mask/edit allocation; exact schedules and baseline accounting are provided in Appendix B.1. CaRR uses the frozen configurations of Sections 3.3–3.4. The Math SCR contract was developed using $1/8$ evidence and frozen thereafter; replacing its $1/8$ Math results with the corresponding TCR-only values still yields a 7.25-point average gain over ME-DLM (Appendix B.3).

4.2 Main Results

Table 1 reports the four-budget comparison, and Figure 2 summarizes it. We make three observations.

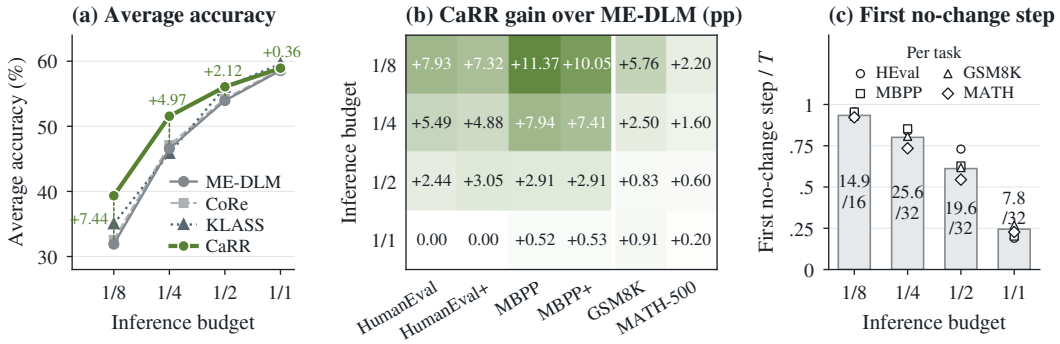


Figure 2: Performance across inference budgets. (a) Six-metric average accuracy of the locally evaluated methods. (b) Per-metric accuracy gain of CaRR over ME-DLM. (c) First no-change prediction step of native ME-DLM as a fraction of the scheduled T (bars: pooled; markers: per task).

Table 2: Module ablation using the same six-metric average accuracy (%) as Table 1. Bold indicates the best result at each budget.

Method	TCR	SCR	1/8	1/4	1/2	1/1
ME-DLM			31.90	46.58	53.95	58.56
TCR-only	✓		35.84	49.19	55.17	58.70
SCR-only		✓	34.95	48.03	54.94	58.70
CaRR	✓	✓	**39.34**	**51.55**	**56.07**	**58.93**

(1) CaRR **consistently improves the native decoder.** Across the four budgets, CaRR improves 22 of the 24 metric–budget settings over ME-DLM and matches the remaining two. Figure 2(b) shows gains across both code generation and mathematical reasoning, with the largest improvements on code. At $1/8$, the gains reach 11.37 and 10.05 points on MBPP and MBPP+, respectively.

(2) The benefit decreases as the inference budget increases. The average gain shrinks monotonically with the budget, from 7.44 points at $1/8$ to 0.36 points at $1/1$ (Figure 2(a)). Figure 2(c) shows that the first no-change prediction occurs progressively earlier relative to the scheduled edit budget as the inference budget increases, reaching an average step of 7.8 out of 32 at $1/1$. This pattern is consistent with reduced late-stage editing activity and fewer opportunities for refinement and recovery. All scheduled edit forwards are still executed, and we use this statistic as a local-stationarity diagnostic. Per-task values are reported in Appendix E.

(3) The advantage is broad at small budgets and less uniform at large budgets. CaRR achieves or ties the best locally evaluated result on all six metrics at $1/8$ and achieves the best result on five of six at $1/4$. As the budget increases, the same-NFE baselines become more competitive: KLASS attains the highest average at $1/1$ and the best HumanEval results at $1/2$, while CoRe is best on MATH-500 at $1/4$ and $1/1$ and ties CaRR at $1/2$. Overall, CaRR provides its largest improvements under tight budgets and remains competitive at larger ones.

4.3 Ablation Study

To isolate the contribution of each module, we evaluate TCR-only, which applies token remasking while retaining native final submission, and SCR-only which keeps native generation and applies submission recovery, under the same budgets. Table 2 reports the six-metric averages. We make three observations.

(1) Both modules are effective on their own. TCR-only and SCR-only improve the average accuracy over ME-DLM at all four budgets, showing that each stage provides useful gains on its own.

(2) TCR contributes more under tighter budgets. TCR-only exceeds SCR-only at $1/8$, $1/4$, and $1/2$, while the two become nearly identical at $1/1$. This trend coincides with later first-no-change points under tighter budgets in Figure 2(c).

(3) Combining the two modules provides additional gains. Full CaRR achieves the highest average at every budget, exceeding the stronger single module by 3.50, 2.37, 0.90, and 0.22 points. At $1/8$ and $1/4$, the marginal gain of SCR after TCR is 3.50 and 2.37 points, compared with 3.04

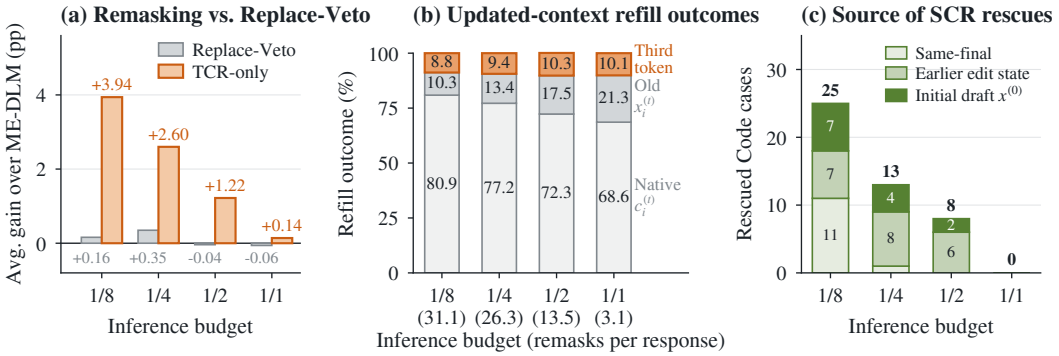


Figure 3: Mechanism analysis of CaRR. (a) Gain over ME-DLM of TCR-only and the matched Replace-Veto control. (b) Outcome composition of updated-context refill at remasked positions. (c) Rescued Code Base cases of SCR by source state.

and 1.45 points on the native trajectory. This interaction is not uniform at larger budgets. Task-level exceptions are reported in Appendix C.

4.4 Mechanism Analysis

We further analyze where the improvements come from. For TCR, we compare against Replace-Veto and classify every updated-context refill in the TCR-only runs by its outcome: the suppressed native replacement $c_i^{(t)}$, the old token $x_i^{(t)}$, or a third token. For SCR, we attribute each rescued Code Base case on pooled HumanEval and MBPP native ME-DLM trajectories to the state supplying the recovered candidate. Figure 3 shows the results, with remasks per response reported in parentheses in panel (b).

(1) Remasking provides clear benefits beyond replacement veto. Replace-Veto stays within 0.35 points of ME-DLM at every budget, whereas TCR-only gains 3.94 and 2.60 points at $1/8$ and $1/4$, exceeding the matched control by 3.77 and 2.25 points (Figure 3(a)). These gaps support the contribution of updated-context refill beyond replacement veto under a shared selection and gating policy.

(2) Updated-context refill changes the selected token outcome. Figure 3(b) shows that the refill differs from the suppressed native replacement in 19.1%, 22.8%, 27.7%, and 31.4% of remasked positions from $1/8$ to $1/1$. Across budgets, 10.3%–21.3% of remasks return to the old token, 8.8%–10.3% produce a third token, and remasking frequency decreases from 31.1 to 3.1 positions per response.

(3) Earlier trajectory states contribute substantially to SCR recovery. Figure 3(c) shows that earlier-state recovery provides 14, 12, and 8 of the 25, 13, and 8 rescued Code Base cases at $1/8$, $1/4$, and $1/2$, respectively; at $1/2$, same-final recovery adds no new case. Notably, 7, 4, and 2 rescues recover the initial mask-diffusion draft $x^{(0)}$. Because the Code resolver scans predecessor states from nearest to farthest, these cases indicate that no later edit state supplied an authorized structurally valid recovery. No paired broken cases were observed in this analysis. Full Base and EvalPlus+ attribution results are reported in Appendix D.2.

5 Conclusion

We presented CaRR, a training-free framework for improving low-budget editable diffusion inference by coordinating token-level and submission-level commitments. TCR defers selected risky pure replacements and refills them on the next scheduled edit forward using updated context, while SCR uses the resulting refinement trajectory for final-first output recovery. Across code generation and mathematical reasoning, CaRR improves average accuracy over native ME-DLM at all four tested budgets, with the largest gains under tight budgets and no additional backbone forwards. Ablations and mechanism analyses further show complementary benefits from updated-context refill and trajectory-based recovery. These results suggest that low-budget editable refinement can benefit from treating both local replacement and final submission as explicit commitment decisions.

AI USE STATEMENT

Generative AI tools were used to provide feedback on research methodology and experimental design, assist with code implementation and debugging, and support the analysis and interpretation of experimental results. They were also used for manuscript drafting and editing, literature-search queries, and identifying relevant prior work. The authors made the final decisions on the research questions, methodology, experimental design, and conclusions. All AI-assisted text and code were manually reviewed, and experimental artifacts and numerical claims were independently checked against the underlying records. The authors take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

The complete inference procedure is specified in Section 3 and Appendix A, including the frozen TCR and SCR rules. Appendix B reports the exact inference-budget schedules, baseline-port accounting, evaluation metrics, and development protocol; Appendices C–D provide the task-level ablations and mechanism analyses, and Appendices E–F document the first-no-change and compute accounting. The supplementary material provides the implementation, frozen configurations, and machine-readable evaluation artifacts used to produce the reported results.

REFERENCES

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021a.

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

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas S Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *Advances in Neural Information Processing Systems*, 38:55981–56007, 2026.

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

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

Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. Theoretical benefit and limitation of diffusion language model. *Advances in Neural Information Processing Systems*, 38: 24415–24459, 2026.

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.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. In *International Conference on Learning Representations*, volume 2025, pp. 5046–5073, 2025.

Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. *Advances in neural information processing systems*, 32, 2019.

Jiaqi Han, Austin Wang, Minkai Xu, Wenda Chu, Meihua Dang, Haotian Ye, Huayu Chen, Yisong Yue, and Stefano Ermon. Discrete diffusion trajectory alignment via stepwise decomposition. In *International Conference on Learning Representations*, volume 2026, pp. 5023–5047, 2026.

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

Michael Hersche, Samuel Moor-Smith, Thomas Hofmann, and Abbas Rahimi. Soft-masked diffusion language models. In *International Conference on Learning Representations*, volume 2026, pp. 105685–105716, 2026.

Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. Flashdlm: Accelerating diffusion language model inference via efficient kv caching and guided diffusion. In *International Conference on Learning Representations*, volume 2026, pp. 153529–153544, 2026.

Zemin Huang, Zhiyang Chen, Zijun Wang, Tiancheng Li, and Guo-Jun Qi. Reinforcing the diffusion chain of lateral thought with diffusion language models. *Advances in Neural Information Processing Systems*, 38:152677–152710, 2025.

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*, volume 2026, pp. 56998–57026, 2026.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. *Advances in neural information processing systems*, 38:52870–52888, 2026.

Seo Hyun Kim, Sunwoo Hong, Hojung Jung, Youngrok Park, and Se-Young Yun. Klass: Kl-guided fast inference in masked diffusion models. *Advances in Neural Information Processing Systems*, 38:92267–92301, 2026.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. Diffusion language model knows the answer before it decodes. In *The Fourteenth International Conference on Learning Representations*, 2026.

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

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyan Wei, Shaobo Wang, Yichen Zhu, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in neural information processing systems*, 35:5775–5787, 2022.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

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. *arXiv preprint arXiv:2605.09603*, 2026.

Subham S 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*, 37:130136–130184, 2024.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

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

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32211–32252. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/song23a.html.

Bohang Sun, Max Zhu, Francesco Caso, Jindong Gu, Junchi Yu, Philip Torr, Pietro Liò, and Jialin Yu. The path matters: Learning a token-commitment policy for diffusion language models. *arXiv preprint arXiv:2605.24697*, 2026a.

Haotian Sun, Rushi Qiang, Yuqian Zheng, and Bo Dai. Forward-free diffusion language models. *arXiv preprint arXiv:2606.08357*, 2026b.

Guanghan Wang, Yair Schiff, Subham Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. *Advances in Neural Information Processing Systems*, 38:147282–147339, 2026a.

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

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*, volume 2026, pp. 57027–57051, 2026.

Shaoan Xie, Lingjing Kong, Xiangchen Song, Xinshuai Dong, Guangyi Chen, Eric Xing, and Kun Zhang. Advancing reasoning in diffusion language models with denoising process rewards. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 42703–42720, 2026.

Weijia Xu and Marine Carpuat. Editor: An edit-based transformer with repositioning for neural machine translation with soft lexical constraints. *Transactions of the Association for Computational Linguistics*, 9:311–328, 2021.

Lin Yao. Don't commit alone: Joint token commitment in diffusion large language models. *arXiv preprint arXiv:2607.04469*, 2026a.

Lin Yao. Remask, don't replace: Token-to-mask refinement in diffusion language models, 2026b. URL https://arxiv.org/abs/2604.18738.

Lin Yao. Remask, don't replace: Token-to-mask refinement in diffusion large language models, 2026c. URL https://arxiv.org/abs/2604.18738v2. Version 2.

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.

Jewon Yeom, Jaewon Sok, Seonghyeon Park, Jeongjae Park, Hwiyeong Lee, and Taesup Kim. Answer first, reason later: Commitment order in diffusion llms. *arXiv preprint arXiv:2608.05687*, 2026.

Heecheol Yun, Joonhyung Park, Joowon Kim, and Eunho Yang. Who should lead decoding now? tracking reliable trajectories for ensembling masked diffusion language models. *arXiv preprint arXiv:2606.16281*, 2026.

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

Linhao Zhong, Linyu Wu, Bozhen Fang, Tianjian Feng, Chenchen Jing, Wen Wang, Jiaheng Zhang, Hao Chen, and Chunhua Shen. Beyond hard masks: Progressive token evolution for diffusion language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6936–6959, 2026.

Roadmap. Appendix A gives the exact inference protocols for TCR and SCR. Appendix B documents budget accounting, evaluation metrics, and development protocols. Appendix C reports the full task-level component results. Appendix D expands the mechanism analyses in Figure 3. Appendix E provides the native first-no-change statistics used in Figure 2(c), and Appendix F summarizes compute accounting.

A Detailed Inference Protocols

All inference-time decisions use public task inputs, including Code entry-point metadata, model outputs, and frozen deterministic rules. Benchmark labels, unit-test outcomes, and reference answers are used only for evaluation after a submission has been materialized.

A.1 Exact TCR Protocol

TCR operates on the native edit actions decoded for a scheduled edit forward. At edit step t, let $x_i^{(t)}$ be the existing token at editable response position i, and let $c_i^{(t)}$ and $n_i^{(t)}$ denote the native current- and next-token proposals aligned to that destination position. A *pure replacement* satisfies $c_i^{(t)} \neq x_i^{(t)}$ and $c_i^{(t)} = n_i^{(t)}$, with no length-changing action at that position. The next-token proposal is destination-aligned as in the runtime: the raw next-token head from the preceding position is shifted to the token position it predicts.

Following T2M's LowProb detector, the old-token gate is

$$d_i^{(t)} = \mathbf{1}\Big\{ p_\theta^{\mathrm{cur}}\Big(x_i^{(t)} \mid q, x^{(t)}, i \Big) < \tau_{\mathrm{lp}} \Big\}. \tag{4}$$

The probability in Equation 4 measures support for retaining the existing token; it is not confidence in the proposed replacement.

Let $m_i^{(t)}$ be the previous-remask count attached to the token lineage currently occupying position i, propagated through native INSERT/DELETE updates, and let $\delta_i^{(t)}$ be its distance to the nearest native structural edit. With $\mathcal{S} = \{[\texttt{PAD}], [\texttt{DEL}], [\texttt{MASK}], [\texttt{EOS}]\}$, the exact eligible and selected sets are

$$\mathcal{R}_0^{(t)} = \Big\{ i : \ i \text{ editable},\ d_i^{(t)} = 1,\ c_i^{(t)} \neq x_i^{(t)},\ c_i^{(t)} = n_i^{(t)},\ x_i^{(t)}, c_i^{(t)} \notin \mathcal{S},\ m_i^{(t)} < C_{\max},\ \delta_i^{(t)} > r,\ t < T-1 \Big\}, \tag{5}$$

$$k_t = \begin{cases} 0, & \mathcal{R}_0^{(t)} = \varnothing, \\ \min\Big\{ |\mathcal{R}_0^{(t)}|, \max(1, \lfloor \rho_{\max} n_t \rfloor) \Big\}, & \text{otherwise}, \end{cases} \tag{6}$$

$$\mathcal{R}^{(t)} = \mathrm{BottomK}_{i \in \mathcal{R}_0^{(t)}} \Big(p_\theta^{\mathrm{cur}}\Big(x_i^{(t)} \mid q, x^{(t)}, i \Big) ; k_t \Big). \tag{7}$$

Here n_t is the number of editable response positions. Equal support values retain native position order. Thus, TCR intervenes only on native pure replacements, remains isolated from length-changing edits, remasks each surviving lineage at most once, and disables new remasks on the terminal edit step.

Table 3: Frozen TCR configuration used in all reported benchmark runs.

Component	Symbol	Value	Role
Retention threshold	τ_{lp}	0.30	Select low-support old tokens
Per-step ratio cap	$\rho_{\max}$	0.25	Cap remasked positions per edit step
Structural radius	r	1	Isolate remasking from INSERT/DELETE actions
Lineage budget	$C_{\max}$	1	Remask each surviving lineage at most once
Terminal remasking	–	disabled	Require a later scheduled refill forward
Additional backbone forwards	–	0	Reuse the next scheduled edit forward

Updated-context refill. For each $i \in \mathcal{R}^{(t)}$, TCR writes [MASK] instead of the native concrete replacement and applies the other native actions for that step. After structural updates, let $j = \pi_t(i)$

be the position of the surviving masked lineage in $x^{(t+1)}$. On the next scheduled edit forward, the current-token logits provide the refill:

$$c_j^{(t+1)} \leftarrow \arg \max_{w \notin \{\texttt{[PAD]},\texttt{[DEL]},\texttt{[MASK]}\}} p_\theta^{\text{cur}}\left(w \mid q, x^{(t+1)}, j\right),$$
$$n_j^{(t+1)} \leftarrow c_j^{(t+1)}. \tag{8}$$

Writing the same refill to the aligned current- and next-token proposal slots preserves the native interpretation of the action as a replacement. The refill is then materialized by the native edit application. No additional backbone call is introduced.

Parameter selection. We use $(\tau_{\text{lp}}, C_{\max}, \rho_{\max}) = (0.30, 1, 0.25)$, following the LowProb operating point identified as most cost-efficient in T2M v2 (Yao, 2026c, Section 5.4 and Appendix D), where cost counts token modifications per sample. In that version, the LowProb detector sweeps $\tau_{\text{lp}} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, together with $C_{\max} \in \{1, 3, 5\}$ and $\rho_{\max} \in \{0.25, 0.50, 1.0\}$. Our fixed-budget ME-DLM adaptation keeps this numerical operating point and adds lineage tracking, structural isolation with $r = 1$, and the finite-schedule terminal constraint above. The configuration was fixed before the reported benchmark runs, and the scheduled backbone budget remains $M + T$.

Development protocol. The frozen TCR gate was validated on an auxiliary 256-request development set containing 64 Code and 192 Math requests, partitioned into 176 calibration requests and 80 held-out-source requests. Trajectory generation used the request inputs alone; clean responses and target token IDs were attached offline for analysis after generation. The detector and remasking policy were then fixed for the benchmark runs reported in the paper.

A.2 Exact SCR Contracts

For task domain d, the deterministic adapter produces an ordered candidate-view sequence

$$C^{(t)} = A_d\left(q, x^{(t)}\right) = \left(a_0^{(t)}, \ldots, a_{J_t}^{(t)}\right), \qquad V_d\left(q, a; C^{(0:T)}\right) \in \{\text{PASS}, \text{FAIL}, \text{ABSTAIN}\}. \tag{9}$$

Here $a_0^{(T)}$ is the native final view. The validator states are mechanical intervention statuses: PASS means no intervention is required, FAIL identifies an actionable mechanical defect or frozen consistency violation, and ABSTAIN indicates insufficient evidence for intervention.

A single resolver R_d preserves the implemented recovery precedence: when the native final is marked FAIL, it first attempts same-final recovery and then applies the domain's trajectory-supported recovery rules. Let $a^\star = R_d(q; C^{(0:T)})$, with $a^\star = \perp$ when no recovery is authorized. The final-first submission rule is

$$\hat{c} = \begin{cases} a^\star, & V_d\left(q, a_0^{(T)}; C^{(0:T)}\right) = \text{FAIL} \ \wedge \ a^\star \neq \perp, \\ a_0^{(T)}, & \text{otherwise.} \end{cases} \tag{10}$$

Both recovery stages may use evidence from the complete trajectory. *Trajectory-supported* describes the evidence used by a resolver, whereas *earlier-state* describes the source of a returned candidate; these are not interchangeable.

Code candidate extraction. For the reported HumanEval and MBPP runs, the public entry point is taken from benchmark-supplied interface metadata after checking consistency with the public prompt. Prompt-based entry-point inference is used only in auxiliary workflows. The structural extractor considers complete Python fenced blocks, adjacent fence-marker segments that recover a complete block from malformed fencing, contiguous code-looking regions, and the raw response when no fence occurs. A candidate is structurally valid when it parses under the Python AST and contains the required public top-level definition.

When a state contains multiple structurally valid candidates, the resolver selects the latest valid structural candidate. The output precedence is fixed as follows:

1. preserve the native final byte-for-byte when it is structurally valid;

2. otherwise use a valid structural extraction from the same final state;
3. otherwise scan predecessor states from nearest to farthest and use the first structurally valid candidate;
4. if no candidate is authorized, return the native final unchanged.

Program execution is reserved for offline evaluation; the selection rule uses parsing and public-interface checks only.

Math candidate views. The Math adapter extracts four surface families: a well-formed boxed answer, an explicit conclusion, a terminal equation, and a conservatively extracted malformed trailing box. Box, conclusion, and terminal-equation views are grouped into equivalence classes using frozen normalization and equivalence rules. Within a state, a unique maximum-support class defines the observed answer; tied classes abstain. Across states, consecutive equivalent observations form an answer basin characterized by run length and independent-view support.

Math recovery from an invalid final box. When the final box is invalid, the same-final resolver searches malformed-box, conclusion, and terminal-equation views and returns a candidate when one equivalence class has unique maximum support from at least two views. The trajectory resolver next considers earlier runs of a well-formed boxed answer with run length at least two. It prioritizes longer runs and breaks equal-length ties by the latest ending state, followed by the latest starting state. A terminal multi-view basin is eligible when it reaches the terminal state, persists for at least two observed states, has a safe candidate surface, and strictly exceeds every incompatible basin in both run length and total independent-view support.

Canonicalization and terminal consensus. For a well-formed final box, the resolver first applies exact canonicalization when the box already contains an eligible explicit closed rational expression. The expression must contain no free symbols, simplify to a rational number, change the surface representation, and satisfy the safe-box predicate.

If canonicalization does not apply, the resolver can use an edit-local dominant terminal-consensus correction when the terminal answer basin is persistent and strictly dominant, the terminal state supplies at least two independent supporting views, the normalized final and candidate scalar surfaces have character-level Levenshtein distance one or two, the candidate is nonzero, and the candidate surface is safe. If neither branch applies, the inherited answer-linked consensus resolver remains available as a fallback; that separate branch requires the conflicting final answer to participate in an invalid terminal arithmetic relation and to lack valid executable support. These are alternative recovery branches in both the 17-state implementation and its higher-budget generalization; the arithmetic checks are not prerequisites for the edit-local branch. Evidence not resolved by the frozen precedence and authorization rules retains the native final.

Frozen cross-budget transfer. The $1/8$ trajectory contains 17 complete response states and was used during development of the Math contract. The $1/4$, $1/2$, and $1/1$ trajectories contain 33 states. For the higher-budget runs, the terminal-state index is generalized while the absolute predicates remain fixed: persistence requires at least two consecutive states, dominance is strict in both run length and independent support, and the edit-local condition remains a one-to-two-character difference. The same thresholds are used across all higher-budget trajectories.

B Evaluation Protocol and Accounting

B.1 Budget and Baseline-Port Accounting

All local comparisons are indexed by scheduled neural function evaluations (NFE). ME-DLM, TCR, SCR, Replace-Veto, fixed-quota KLASS, and full CaRR use the native mask/edit allocation. CoRe reserves part of the same total NFE budget for perturbation probes. Table 4 gives the exact schedules.

Table 4: Scheduled forward allocation for the locally evaluated methods. "Normal mask" excludes CoRe perturbation probes.

Budget	Native M	Native T	Native/CaRR NFE	CoRe normal mask + probes + edit	Total NFE
$1/8$	48	16	64	$45 + 3 + 16$	64
$1/4$	96	32	128	$90 + 6 + 32$	128
$1/2$	224	32	256	$211 + 13 + 32$	256
$1/1$	480	32	512	$452 + 28 + 32$	512

All local runs use the released ME-DLM Stage-3 model, response canvas $L = 512$, and a single RTX 5090. Our KLASS port keeps the native per-step commitment quota and applies the KLASS selection rule within that quota. Soft Mask, EvoToken, and LLaDA-Instruct are included only as published context in Table 1; local paired analyses use the reproduced ME-DLM trajectories and locally evaluated methods.

B.2 Benchmarks, Metrics, and Provenance

Table 5: Evaluation sets and metrics used in the paper.

Domain	Benchmark	Examples	Reported metrics
Code	HumanEval	164	Base, EvalPlus+
Code	MBPP (EvalPlus)	378	Base, EvalPlus+
Math	GSM8K	1,319	Answer accuracy
Math	MATH-500	500	Answer accuracy

For Code, we report Base accuracy and standard EvalPlus+ accuracy, where the same completion must pass both the Base and additional EvalPlus tests. The paired SCR analyses use this same EvalPlus+ criterion. The six-metric average in the main paper is the unweighted mean of HumanEval Base, HumanEval+, MBPP Base, MBPP+, GSM8K, and MATH-500 accuracy. Locally evaluated main-table values are derived from the unified cross-method evaluation outputs; the four-arm comparison and rescue attribution use per-example paired evaluation records. Execution traces are used for refill-outcome and first-no-change analyses.

B.3 Sensitivity to Math Development Evidence

The Math contract of SCR was developed using $1/8$ GSM8K and MATH-500 evidence and then frozen before transfer to $1/4$, $1/2$, and $1/1$. To quantify the contribution of this development contact to the headline $1/8$ result, we replace the $1/8$ GSM8K and MATH-500 entries of full CaRR with their TCR-only values from Table 6 (69.83 and 34.80), leaving the four Code metrics unchanged. The resulting six-metric average is 39.16, corresponding to a 7.25-point gain over native ME-DLM, compared with the reported 7.44-point gain. The higher-budget results use the same benchmark problem sets and the frozen contract without budget-specific retuning; they assess cross-budget transfer rather than generalization to an independent held-out dataset.

C Full Component Results

Table 6 expands the average ablation in Table 2 into task-level results. For Code, the "+" rows use the same standard EvalPlus+ criterion as the cross-method main table.

Table 6: Full task-level component results (%). "SCR" denotes native ME-DLM generation with SCR; "TCR" denotes TCR generation with native final-only submission. Code "+" denotes standard EvalPlus+ accuracy.

Budget	Domain	Task / metric	ME-DLM	SCR	TCR	CaRR
$1/8$	Math	GSM8K	63.99	64.90	**69.83**	69.75
	Math	MATH-500	33.80	35.00	34.80	**36.00**
	Code	HumanEval Base	23.17	26.22	25.61	**31.10**
	Code	HumanEval+	20.73	23.78	23.17	**28.05**
	Code	MBPP Base	26.46	31.75	33.07	**37.83**
	Code	MBPP+	23.28	28.04	28.57	**33.33**
$1/4$	Math	GSM8K	77.41	77.56	79.53	**79.91**
	Math	MATH-500	42.40	43.20	43.00	**44.00**
	Code	HumanEval Base	40.85	42.07	43.90	**46.34**
	Code	HumanEval+	38.41	39.63	40.85	**43.29**
	Code	MBPP Base	43.12	46.03	46.56	**51.06**
	Code	MBPP+	37.30	39.68	41.27	**44.71**
$1/2$	Math	GSM8K	84.08	84.31	84.61	**84.91**
	Math	MATH-500	48.00	**48.60**	48.00	**48.60**
	Code	HumanEval Base	48.17	49.39	**50.61**	**50.61**
	Code	HumanEval+	44.51	45.73	**47.56**	**47.56**
	Code	MBPP Base	53.70	55.29	54.23	**56.61**
	Code	MBPP+	45.24	46.30	46.03	**48.15**
$1/1$	Math	GSM8K	84.91	85.14	85.60	**85.82**
	Math	MATH-500	50.00	**50.60**	49.60	50.20
	Code	HumanEval Base	**56.10**	**56.10**	**56.10**	**56.10**
	Code	HumanEval+	**52.44**	**52.44**	**52.44**	**52.44**
	Code	MBPP Base	58.47	58.47	58.73	**58.99**
	Code	MBPP+	49.47	49.47	49.74	**50.00**

Full CaRR achieves the strongest six-metric average at every budget in Table 2, while Table 6 exposes localized task-level interactions between the two stages. At $1/8$, TCR-only is one GSM8K example higher than full CaRR (69.83 vs. 69.75); at $1/1$, SCR-only is two MATH-500 examples higher (50.60 vs. 50.20); and TCR-only reaches 49.60 on $1/1$ MATH-500 compared with 50.00 for native ME-DLM. These task-level results complement the aggregate module analysis in Section 4.3.

D Additional Mechanism Analysis

D.1 TCR Refill Outcomes

For every position remasked by TCR, we pair the refill written on the next scheduled edit forward with the old token $x_i^{(t)}$ and the suppressed native replacement $c_i^{(t)}$ recorded at the remasking step. The runtime stores selected positions and refill tokens in the same order, and refill counts match selection counts for every paired step in the benchmark logs. Table 7 reports the outcome composition by task, while Figure 3(b) shows the pooled composition.

Table 7: Updated-context refill outcomes (%) by task. "Native" denotes a refill equal to the suppressed native replacement $c_i^{(t)}$, "Old" a refill equal to $x_i^{(t)}$, and "Third" any other token. Remasks per response are pooled across all 2,361 responses at each budget.

	Remasks	HumanEval			MBPP			GSM8K			MATH-500		
Budget	/ resp.	Native	Old	Third	Native	Old	Third	Native	Old	Third	Native	Old	Third
$1/8$	31.1	79.4	9.6	10.9	80.2	9.6	10.3	81.3	10.5	8.2	81.1	10.6	8.3
$1/4$	26.3	77.3	12.2	10.6	77.2	11.8	11.0	77.1	14.0	8.9	77.5	13.6	8.9
$1/2$	13.5	74.4	14.6	11.0	74.4	15.7	9.9	72.0	18.0	10.0	71.0	18.1	10.9
$1/1$	3.1	71.1	16.1	12.8	69.9	20.5	9.5	69.6	20.9	9.5	66.0	22.9	11.1

Across tasks, the updated-context refill frequently reproduces the suppressed native replacement and also produces two systematic alternatives: returning to the old token and selecting a third token. Pooled across tasks, the refill differs from the suppressed native replacement in 19.1%, 22.8%, 27.7%, and 31.4% of remasked positions from $1/8$ to $1/1$. The remasking frequency decreases over the same budgets, from 31.1 to 3.1 positions per response.

D.2 SCR Paired Attribution

The three-arm Code analysis reuses identical saved native ME-DLM trajectories and compares the native final, same-final structural recovery, and full SCR with earlier-state recovery. Table 8 reports Base and EvalPlus+ outcomes pooled over HumanEval and MBPP ($164 + 378 = 542$ examples).

Table 8: Paired Code attribution for SCR. R/B denotes rescued/broken cases relative to the preceding arm.

Metric	Budget	Native final	Same-final	Full SCR	Native→same R/B	Same→full R/B
Base	$1/8$	138	149	163	11/0	14/0
	$1/4$	230	231	243	1/0	12/0
	$1/2$	282	282	290	0/0	8/0
	$1/1$	313	313	313	0/0	0/0
EvalPlus+	$1/8$	122	132	145	10/0	13/0
	$1/4$	204	205	215	1/0	10/0
	$1/2$	244	244	250	0/0	6/0
	$1/1$	273	273	273	0/0	0/0

The $1/2$ setting provides the clearest separation between same-final and earlier-state recovery: same-final extraction leaves both Base and EvalPlus+ counts unchanged, while earlier-state recovery adds eight Base and six EvalPlus+ passes. Across the paired Code analysis, no broken case is observed in the transitions reported in Table 8.

D.3 SCR Rescue Sources

Table 9 decomposes rescued cases by the state that supplies the returned candidate. These analyses reuse native ME-DLM trajectories; under the unified notation, the initial draft is $x^{(0)}$ and the intermediate edit states are $x^{(1)}, \ldots, x^{(T-1)}$.

Table 9: Source of rescued Code cases on pooled HumanEval and MBPP native ME-DLM trajectories.

Metric	Budget	Same-final	Earlier edit	Initial draft $x^{(0)}$	Total
Base	$1/8$	11	7	7	25
	$1/4$	1	8	4	13
	$1/2$	0	6	2	8
	$1/1$	0	0	0	0
EvalPlus+	$1/8$	10	7	6	23
	$1/4$	1	7	3	11
	$1/2$	0	4	2	6
	$1/1$	0	0	0	0

Earlier states provide most of the additional Code recovery at the tighter budgets. At $1/2$, all eight rescued Base cases come from earlier states, including two from the initial mask-diffusion draft. Selection of $x^{(0)}$ occurs after all later edit states have been evaluated without an authorized structurally valid candidate.

E NATIVE FIRST-NO-CHANGE ANALYSIS

To characterize when native refinement first becomes locally stationary, we use the logged `real_edit_step`, defined as the first edit step at which the predicted response no longer changes, capped at the scheduled edit budget T. All scheduled edit forwards are executed; this statistic records the first unchanged prediction within that fixed schedule. Table 10 reports task-level averages and the example-weighted pooled value.

Table 10: Average first no-change prediction step of native ME-DLM. "Pooled" is example-weighted across the four benchmarks; the denominator is the scheduled edit budget T.

Budget	HumanEval	MBPP	GSM8K	MATH-500	Pooled
$1/8$	15.29	15.27	14.87	14.75	14.94 / 16
$1/4$	26.85	27.28	25.81	23.50	25.63 / 32
$1/2$	23.35	20.07	19.76	17.43	19.57 / 32
$1/1$	6.07	6.48	8.64	7.27	7.83 / 32

Relative to the scheduled edit budget, the first unchanged prediction occurs progressively earlier as the inference budget increases: the pooled fraction is 0.933, 0.801, 0.611, and 0.245 from $1/8$ to $1/1$.

This trend accompanies lower remasking activity and smaller average gains of CaRR at larger budgets, consistent with reduced late-stage editing activity. The corresponding pooled first-no-change trend is summarized in Figure 2(c).

F COMPUTE ACCOUNTING

Local comparisons use scheduled NFE as the common inference-budget measure. For ME-DLM, TCR, SCR, Replace-Veto, fixed-quota KLASS, and full CaRR, the scheduled backbone budget is the native $M + T$ allocation. TCR reads logits and decoded actions from scheduled edit forwards and performs refill on the next scheduled forward; SCR performs deterministic CPU-side candidate extraction and resolution after generation. Full CaRR therefore adds no backbone forward to the native schedule.

Actual runtime cost also depends on the response lengths produced along the edit trajectory, which determine processed token positions and attention cost. CoRe allocates a portion of its matched total NFE to perturbation probes as detailed in Table 4. Scheduled NFE is therefore the controlled inference-budget axis throughout the paper. Wall-clock latency, processed token positions, and an attention-length-squared proxy provide complementary runtime measures for future system-level comparisons.