WHEN TO REVISE: REMASKING OVERLAYS FOR MASKED DIFFUSION LANGUAGE MODELS ACROSS INFERENCE BUDGETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training-free remasking overlays let masked diffusion language models (MDMs) return committed tokens to the mask and decode them again. Each overlay is evaluated in its own harness and usually at one operating point, so it is unclear whether a reported gain reflects the method or the step budget, the compute spent, and the sample of benchmark problems. We study this question for four overlays (CoRe, ReMDM-conf, STaRR and T2M) and vanilla decoding on LLaDA-8B-Base with a common-harness protocol. It runs all methods in one decoder at three step budgets, counts each method's forward passes from its implementation, adds equal-cost vanilla controls, and estimates paired uncertainty for the equal-weight benchmark mean. CoRe, the only overlay that spends extra forward passes to verify tokens, gains about two to three points over vanilla at every budget, and the gain does not shrink under equal-cost controls. Its lead over the best of the three compute-free overlays, however, is within benchmark sampling uncertainty at two of three budgets. The overlays differ far more in how many answers they change than in net accuracy, and on a second backbone CoRe's effect on one benchmark reverses sign. Code is available at https://anonymous.4open.science/r/dLMM-benchmark-iclr27-81D2/.

1 INTRODUCTION

Masked diffusion language models (MDMs) generate text by unmasking a fully masked sequence over a fixed number of steps. They build on discrete diffusion over token sequences, developed through structured corruption processes, ratio estimation and simplified masked objectives (Austin et al., 2021a; Lou et al., 2024; Sahoo et al., 2024; Shi et al., 2024), and at the 7 to 8B scale LLaDA-8B-Base and Dream-7B have set new standards for open non-autoregressive generation (Nie et al., 2025; Ye et al., 2025). Because an MDM can revisit any position until decoding ends, it inherits the remask-and-refine pattern of earlier non-autoregressive decoders (Ghazvininejad et al., 2019; Gu et al., 2019), and a family of remasking overlays, most of them training-free, now returns committed tokens to the mask and decodes them again (Zhai et al., 2026; Wang et al., 2025; Yao, 2026; Sun et al., 2026; Huang et al., 2026). These overlays spend compute to raise accuracy, whereas block, caching and parallel decoding schemes mainly make MDMs faster (Arriola et al., 2025; Wu et al., 2026); some fast parallel decoders also revoke committed tokens and gain speed and accuracy together (Dong et al., 2026; Hong et al., 2026). As overlays multiply, the question of how to compare them becomes more important. More concretely, a practitioner who adds an overlay to a decoder wants to know what accuracy it buys on new problems from the same tasks, and at what cost in model calls. The current practice answers a narrower question: each overlay reports benchmark accuracy against vanilla decoding at one operating point, in its own harness, with one number per benchmark (Figure 1, left).

This practice leaves open what a reported gain is meant to represent. One reading is the accuracy difference, on the benchmark at hand, between the overlay and vanilla run for the same number of diffusion steps. Another reading, closer to how the overlay is used, is the expected difference on new problems when the overlay and its alternatives are given the same compute. The two readings coincide only when the overlay adds no model calls and the benchmark is large enough for its sampling noise to be negligible. Neither holds here: a verifying overlay such as CoRe spends extra

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

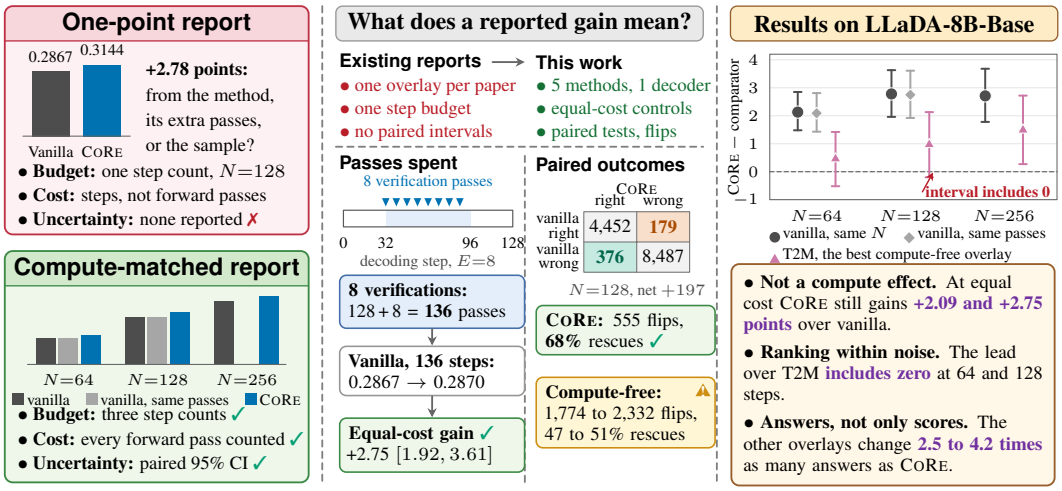


Figure 1: A gain reported at one operating point mixes the method with its step budget, its compute and the sample of problems (left). Our **common-harness protocol** counts every forward pass, adds equal-cost controls and pairs outcomes per problem (middle). On LLaDA-8B-Base, CoRe's gain over vanilla survives equal cost, while its lead over the best compute-free overlay includes zero at 64 and 128 steps (right).

forward passes, the differences between overlays are often about one point of mean accuracy, and a benchmark accuracy nets out the problems an overlay fixes against those it breaks. We refer to the gap between a gain measured at one setting and the gain that survives matched compute and benchmark sampling uncertainty as the *operating-point gap* of a revision overlay. This distinction leads us to treat the comparison itself as the object of evaluation: for each pair of methods and each compute budget, the paired accuracy difference over the problem distribution, together with the rates at which each method fixes and breaks the other's answers.

The papers that introduce these overlays make parts of this problem concrete, but each on its own terms. CoRe (Zhai et al., 2026) stress-tests candidate tokens under a perturbed context; it reports its headline accuracy at $N{=}128$ steps with a verification pass every $E{=}8$ steps, ablates E and the candidate count, compares against vanilla given the same 136 forward passes, and reports random- and margin-selection controls at matched compute. ReMDM (Wang et al., 2025) remasks with a schedule-derived probability and shows sample quality improving as the number of steps grows from 128 to 4,096 on text, with further budget sweeps on images and molecules. T2M (Yao, 2026) resets low-probability tokens to the mask rather than replacing them, reports accuracy–compute frontiers from a 53-configuration sweep, and transfers one setting to several backbones, including LLaDA-8B-Base and Dream-7B. STaRR (Sun et al., 2026) thresholds a spatial-temporal confidence signal and targets decoding speed at fixed accuracy, and Self-Reflective Remasking (Huang et al., 2026) learns a confidence head. These evaluations are careful, but each covers one overlay in its own harness and with its own implementation choices, and none compares the overlays with each other at several budgets with uncertainty on the paired differences.

How to make such comparisons has been studied outside MDMs. Audits of GANs, deep reinforcement learning and metric learning found reported gains smaller or less stable once methods were compared under matched conditions (Lucic et al., 2018; Henderson et al., 2018; Musgrave et al., 2020), and Dodge et al. (2019) proposed reporting expected performance as a function of the tuning budget. For systems scored on a shared test set, paired tests such as McNemar's are the standard tool (Dietterich, 1998; Dror et al., 2018), and recent work quantifies the variance of machine learning benchmarks and adds error bars to language model evaluations (Bouthillier et al., 2021; Madaan et al., 2024; Miller, 2024). Work on scaling inference-time compute for autoregressive models makes the related point that a method has to be charged for the compute it spends (Snell et al., 2025; Brown et al., 2024). Revision overlays need these tools together: their headline metric is an equal-weight mean over benchmarks of 164 to 6,511 problems, the differences at stake are about one point, and the overlays

differ in the forward passes they spend. We ask a different question. Specifically, when the overlays run in one decoder and are charged the forward passes they spend, which of their reported differences survive matched compute, a change of step budget, and the sampling uncertainty of the benchmarks? In particular, CoRe's equal-cost control covers one budget and one pair of methods, whereas we run it at two budgets and compare every overlay with every other.

Motivated by the above questions, and rather than proposing another overlay, we build a common-harness evaluation protocol for revision overlays (Section 3; Figure 1, middle). It runs vanilla decoding, CoRe, and our reimplementations of ReMDM-conf, STaRR and T2M in a single decoder on LLaDA-8B-Base at $N \in \{64, 128, 256\}$ steps on five benchmarks. It counts each method's forward passes from its implementation, which for CoRe is $N + N/(2E)$ because verification runs only in the middle half of the schedule, and it runs vanilla at CoRe's pass counts as an equal-cost control. It reads every number from per-problem logs through one run manifest, estimates the uncertainty of the equal-weight five-benchmark mean with a bootstrap that resamples problems within benchmarks and keeps the methods paired, and splits each net difference into rescues and breakages. The protocol complements the overlays' own evaluations, which cover backbones, tasks and budgets that ours does not.

We apply the protocol to 13,494 problems per configuration. CoRe's gain over vanilla holds at every budget and does not shrink when vanilla receives CoRe's extra passes as decoding steps. Its lead over the best compute-free overlay, however, is within the sampling uncertainty at 64 and 128 steps, and the best method on individual benchmarks changes with the budget. At 128 steps CoRe changes three to four times fewer answers than the compute-free overlays, with a larger share of rescues, and on Dream-7B its effect on BBH is a significant loss. Figure 1 (right) summarizes these results.

Key contributions.

- We formulate the comparison of revision overlays as inference on paired, compute-indexed accuracy differences over a problem distribution, and decompose each difference into rescue and breakage probabilities.
- We build a common-harness protocol that counts forward passes from each implementation, adds equal-cost vanilla controls, regenerates every table from one run manifest, and estimates the uncertainty of the equal-weight mean with a within-benchmark paired bootstrap; the code is available at https://anonymous.4open.science/r/dLMM-benchmark-iclr27-81D2/.
- We show that CoRe's gain over vanilla on LLaDA-8B-Base holds at every budget and at equal cost ($+2.09$ and $+2.75$ points at 68 and 136 passes), while its lead over the best compute-free overlay excludes zero only at 256 steps.
- We show that the overlays differ far more in how many answers they change than in net accuracy, and that their per-benchmark effects depend on the budget and on the backbone.

2 Problem formulation

We first define the quantities the comparison targets. Let $b \in \{1, \dots, B\}$ index benchmarks, with $B{=}5$ here, and let P_b denote the distribution of problems of benchmark b, each problem carrying its scoring rule. A method j is a decoding rule (vanilla or an overlay) applied to a fixed backbone and run for N diffusion steps. For a problem $w \sim P_b$ and execution randomness ξ, let $Z_j(w; N, \xi) \in \{0, 1\}$ indicate that the output is scored correct, and define

$$\mu_j(w; N) = \mathbb{E}_\xi \big[Z_j(w; N, \xi) \big] \tag{1}$$

as the problem-level success probability. A deterministic method is the special case with no ξ; among the methods we study, only ReMDM-conf draws at random once tokens are selected greedily. Let $c_j(N)$ be the number of model forward passes that method j spends per generation at N steps. The benchmark-level target is $\theta_{j,b}(N) = \mathbb{E}_{W\sim P_b}[\mu_j(W;N)]$, and the headline target is the equal-weight mean

$$\bar{\theta}_j(N) = \frac{1}{B}\sum_{b=1}^{B}\theta_{j,b}(N). \tag{2}$$

For two configurations (j,N) and (k,N') we study the paired difference $\Delta = \bar{\theta}_j(N) - \bar{\theta}_k(N')$. We call the comparison *equal-step* when $N' = N$ and *equal-cost* when $c_k(N') = c_j(N)$. The two coincide for methods that add no forward passes and differ for a verifying overlay.

A net difference on a benchmark hides how the two methods disagree. On benchmark b,

$$\theta_{j,b}(N) - \theta_{k,b}(N') = r_b - \beta_b, \qquad r_b = \Pr\big(Z_j{=}1, Z_k{=}0\big), \quad \beta_b = \Pr\big(Z_j{=}0, Z_k{=}1\big), \tag{3}$$

where the probabilities are over $W \sim P_b$ and the randomness of both methods. We call r_b the rescue probability (j solves a problem that k fails) and β_b the breakage probability, and $\rho_b = r_b/(r_b+\beta_b)$ the rescue fraction. Two pairs of methods can share the same Δ with very different $r_b + \beta_b$. In practice these quantities must be estimated from finite benchmarks of 164 to 6,511 problems each. Section 3 therefore defines the protocol used to estimate them and their uncertainty.

3 Common-harness protocol

The protocol runs each method once per configuration on every problem of every benchmark and derives all comparisons from the resulting per-problem correctness logs, so that every pair of methods is compared on the same problems, prompts and scoring. The middle column of Figure 1 shows its two accounting steps: the forward passes each method spends, and the paired outcome of every problem. Section 3.1 fixes the methods, Section 3.2 counts their cost, and Section 3.3 estimates the targets of Section 2 and their uncertainty.

3.1 Decoding loop and overlays

At each of the N steps the model predicts a distribution at every position and the base sampler commits the scheduled number of new positions by the standard low-confidence rule; generation length is $L{=}512$ in a single block. A revision overlay may, in addition, return committed tokens to the mask so that they are decoded again. CoRe (Zhai et al., 2026) verifies: at every E-th step inside $[\lfloor N/4\rfloor, \lfloor 3N/4\rfloor)$ it masks the $m{=}32$ committed tokens with the smallest top-2 margin, runs one extra forward pass on this perturbed sequence, and remasks at most one token whose perturbed-context prediction differs from the committed token (with replacement probability at least 0.3), choosing the one whose committed token has the lowest log-probability under the perturbed context. ReMDM-conf (Wang et al., 2025), STaRR (Sun et al., 2026) and T2M (Yao, 2026) decide what to remask from quantities of the ordinary decoding pass and add no forward passes; we call them compute-free. ReMDM-conf draws its remasking decisions at random, so its results depend on the seed even though token selection is greedy. Appendix A.2 gives each rule and its hyperparameters, which we hold fixed across budgets and benchmarks.

3.2 Compute accounting and equal-cost controls

We measure cost in model forward passes per generation. The verification pass has the same sequence length as a decoding pass, so passes are a fair unit for model compute; we do not report wall-clock time. Vanilla and the compute-free overlays spend $c_j(N) = N$ passes. CoRe adds one pass per verification call, so

$$c_{\text{CoRe}}(N) = N + \big|\{\, i : \lfloor N/4\rfloor \le i < \lfloor 3N/4\rfloor,\ i \bmod E = 0\,\}\big| = N + \frac{N}{2E}, \tag{4}$$

which is 68, 136 and 272 at $N{=}64$, 128 and 256 with $E{=}8$, a 6.25% overhead (Figure 1, middle left). An equal-step comparison therefore gives CoRe more compute than every other method. To separate the overlay from its extra compute we also run vanilla at 68 and 136 steps, CoRe's pass counts at $N{=}64$ and 128 (*equal-cost vanilla*); the second is the control Zhai et al. (2026) report.

3.3 Estimation and uncertainty

Every number is computed from per-problem `lm-evaluation-harness` logs through one loader and one run manifest (Appendix A.1), both released with our code at `https://anonymous.4open.science/r/dLMM-benchmark-iclr27-81D2/`. The loader reads every sub-task file, keys records on (sub-task, problem id), and selects each benchmark's headline filter, so $\hat{\theta}_{j,b}$ is the fraction of the n_b problems scored correct; for BBH and MATH this equals the harness group score. $\hat{r}_b$ and $\hat{\beta}_b$ are the fractions of problems on which the two methods disagree in each direction; Figure 1 (middle right) shows the four outcome counts for CoRe against vanilla at $N{=}128$. We treat each benchmark as a sample of problems from its task: greedy decoding removes run-to-run variance for the deterministic methods, but not this sampling uncertainty. For $\hat{\Delta}$ we resample problems with replacement within each benchmark, keeping the two methods paired, recompute the mean of the five per-benchmark differences, and report the 2.5 and 97.5 percentiles of 200,000 draws. For a single benchmark we use the exact McNemar test on the problems where the two methods disagree (Dietterich, 1998), and for rescue fractions exact (Clopper–Pearson) intervals. We report many comparisons without a multiplicity correction and read p values near 0.05 as weak evidence.

4 Experiments

Revision overlays for MDMs are usually compared at one operating point, and a gain measured there need not survive matched compute, another budget, or the sampling of benchmark problems. Our experiments test which of the reported differences do. We organize these studies around the following three Research Questions (RQs):

(**RQ1**) Does CoRe's gain over vanilla hold when both methods receive the same forward passes, at every budget, and on a second backbone?

(**RQ2**) Is the ranking among the overlays distinguishable from benchmark sampling uncertainty, and does it hold across budgets?

(**RQ3**) Does the net accuracy difference reflect how many answers an overlay changes?

We evaluate on LLaDA-8B-Base (Nie et al., 2025) with five benchmarks: HumanEval (Chen et al., 2021) (0-shot, 164 problems, pass@1), GSM8K (Cobbe et al., 2021) (4-shot, 1,319, strict-match), BBH (Suzgun et al., 2023) (3-shot chain of thought, 6,511 over 27 sub-tasks, exact-match), MATH (Hendrycks et al., 2021) (`minerva_math`, 4-shot, 5,000 over 7 sub-topics, exact-match) and MBPP (Austin et al., 2021b) (3-shot, 500, pass@1). Each method runs at $N \in \{64, 128, 256\}$, and vanilla also runs at 68 and 136 steps as the equal-cost control. All methods share the same decoder and scoring and differ in their remasking rule. Vanilla and CoRe also received the same prompt for every problem, but the logged GSM8K runs of the three compute-free overlays drew different four-shot demonstrations for 1313 of the 1,319 problems (Appendix A.4); we mark those cells with † and check each comparison with them on the four benchmarks where every prompt matches. Unless otherwise stated, tokens are selected greedily, CoRe uses $m{=}32$ and $E{=}8$, the seed is 42, and intervals use 200,000 bootstrap draws. We compare the following five methods:

- **Vanilla** decodes with the low-confidence rule and never remasks.
- **CoRe** (Zhai et al., 2026) verifies candidate tokens under a perturbed context and remasks at most one per call; we run the released code.
- **ReMDM-conf** (Wang et al., 2025) remasks each committed token at random with a probability that grows as its confidence falls.
- **STaRR** (Sun et al., 2026) remasks tokens whose confidence falls below a threshold built from their recent history and their neighbours.
- **T2M** (Yao, 2026) remasks tokens whose own probability falls below a fixed threshold, at most three times per position.

Appendix B provides supporting studies: the revision interval (Appendix B.1), prompt length (Appendix B.2), sub-task breakdowns (Appendix B.3), per-method churn (Appendix B.4), Dream-7B details (Appendix B.5), two further seeds for ReMDM-conf (Appendix B.6), a comparison with

Table 1: CoRe minus vanilla at equal steps and at equal forward passes, in points.

Backbone	N	Passes CoRe / vanilla	CoRe − vanilla per benchmark (points) GSM8K	MATH	BBH	HumanEval	MBPP	Five-benchmark mean Δ [95% CI]
LLaDA-8B	64	68 / 64	+0.76 ✗	+0.26 ✓	+1.97 ✓	+3.05 ✗	+4.60 ✓	+2.13 [+1.48, +2.85] ✓
	64	68 / 68	+0.83 ✓	+0.24 ✓	+1.90 ✓	+3.05 ✗	+4.40 ✓	+2.09 [+1.43, +2.81] ✓
	128	136 / 128	+0.83 ✗	+0.42 ✓	+1.80 ✓	+1.83 ✗	+9.00 ✓	+2.78 [+1.96, +3.63] ✓
	128	136 / 136	+0.83 ✗	+0.44 ✓	+1.83 ✓	+1.83 ✗	+8.80 ✓	+2.75 [+1.92, +3.61] ✓
	256	272 / 256	0.00 ✗	+0.32 ✗	+1.74 ✓	+4.88 ✓	+6.60 ✓	+2.71 [+1.78, +3.68] ✓
Dream-7B	128	136 / 128	+0.53 ✗	+1.18 ✓	−3.33 ✗	+16.46 ✓	+7.20 ✓	+4.41 [+3.18, +5.71] ✓

published numbers (Appendix B.7) and qualitative examples (Appendix B.8). All runs use NVIDIA A100 80 GB GPUs.

4.1 RQ1: Gain over vanilla under matched compute

We first answer **RQ1** by comparing CoRe with vanilla at the same step budget and at the same number of forward passes. Each row of Table 1 is one comparison: CoRe at N steps against vanilla at N steps (equal-step) or at CoRe's pass count (equal-cost). The column "Passes" gives the forward passes per generation of CoRe and of vanilla. The benchmark columns give CoRe minus vanilla in points; ✓ marks a benchmark on which rescues significantly outnumber breakages (exact McNemar, $p < 0.05$) and ✗ one on which they do not. The last column gives the difference in the five-benchmark mean with its 95% paired bootstrap interval.

Table 1 and the right column of Figure 1 show that CoRe's gain over vanilla does not depend on the extra compute it spends. Giving vanilla CoRe's pass count moves its mean from 0.1573 to 0.1577 at N=64 and from 0.2867 to 0.2870 at N=128, with no benchmark changing by more than 0.2 points, so the equal-cost rows nearly repeat the equal-step rows (+2.09 against +2.13, and +2.75 against +2.78). The gain is significant at every budget on BBH and MBPP, on MATH at N=64 and 128 but not at 256, on HumanEval only at N=256 (few HumanEval problems change), and on GSM8K only in the equal-cost comparison at N=64. The equal-cost control agrees with CoRe's own control at 136 passes (Zhai et al., 2026), whose vanilla scores are within 1.2 points of ours (Appendix B.7), and it shows why the control has to be run rather than interpolated: a straight line between vanilla's GSM8K scores at 128 and 256 steps predicts 0.525 at 136 steps, whereas the measured value is 0.5178. We did not run the equal-cost control at N=256.

The last row repeats the equal-step comparison on Dream-7B (Ye et al., 2025) at N=128, with Dream's shift-by-one logits convention and without the compute-free overlays (Appendix B.5). CoRe raises Dream's mean by +4.41 points [+3.18, +5.71], but its per-benchmark profile differs from LLaDA-8B-Base's: HumanEval gains 16.46 points from a vanilla score of 0.0122, and BBH loses 3.33 points, with 193 rescues against 410 breakages ($p < 0.001$).

Takeaway 1

CoRe's gain over vanilla is a gain of the overlay, not of its extra compute: it holds at every budget and survives equal-cost controls, while its size, and even its sign on a single benchmark, depends on the backbone.

4.2 RQ2: Ranking among the overlays

We next answer **RQ2** by placing all five methods on the same budget grid. Table 2 reports each method's accuracy per benchmark and its five-benchmark mean. The column "Passes" counts forward passes per generation, so CoRe's rows cost 6.25% more than the other rows of the same block. The last column gives CoRe's mean minus the row's mean with a 95% paired bootstrap interval, and ✓ marks an interval that excludes zero. Bold marks the best entry of each column within a block.

Table 2: Accuracy of five methods at three step budgets on LLaDA-8B-Base.

N	Method	Passes	Accuracy per benchmark GSM8K	MATH	BBH	HumanEval	MBPP	Five-benchmark mean Mean	CoRe − row [95% CI]
64	Vanilla	64	0.2130	0.0474	0.4050	0.0549	0.0660	0.1573	+2.13 [+1.48, +2.85] ✓
	Vanilla (equal cost)	68	0.2123	0.0476	0.4056	0.0549	0.0680	0.1577	+2.09 [+1.43, +2.81] ✓
	ReMDM-conf	64	0.2077†	0.0398	0.4053	0.0732	0.1040	0.1660	+1.25 [+0.28, +2.24] ✓
	STaRR	64	0.1729†	0.0362	0.4075	0.0610	0.0960	0.1547	+2.38 [+1.44, +3.35] ✓
	T2M	64	0.2123†	0.0454	0.4127	**0.0915**	0.1080	0.1740	+0.46 [−0.52, +1.42] ✗
	CoRe	68	**0.2206**	**0.0500**	**0.4247**	0.0854	**0.1120**	**0.1785**	*reference*
128	Vanilla	128	0.5178	0.1680	0.4617	0.1220	0.1640	0.2867	+2.78 [+1.96, +3.63] ✓
	Vanilla (equal cost)	136	0.5178	0.1678	0.4614	0.1220	0.1660	0.2870	+2.75 [+1.92, +3.61] ✓
	ReMDM-conf	128	0.4822†	0.1522	0.4551	**0.1707**	0.2100	0.2940	+2.04 [+0.70, +3.36] ✓
	STaRR	128	0.5102†	0.1634	0.4608	0.1220	**0.2620**	0.3037	+1.08 [−0.17, +2.33] ✗
	T2M	128	0.5034†	0.1662	0.4698	0.1463	0.2380	0.3048	+0.97 [−0.20, +2.13] ✗
	CoRe	136	**0.5262**	**0.1722**	**0.4796**	0.1402	0.2540	**0.3144**	*reference*
256	Vanilla	256	0.6368	0.2694	0.4580	0.2500	0.2920	0.3812	+2.71 [+1.78, +3.68] ✓
	ReMDM-conf	256	0.5997†	0.2352	0.4698	0.2683	0.3320	0.3810	+2.73 [+1.35, +4.13] ✓
	STaRR	256	**0.6482**†	0.2552	0.4572	0.2622	0.3220	0.3890	+1.93 [+0.81, +3.08] ✓
	T2M	256	0.6399†	0.2568	**0.4824**	0.2500	0.3380	0.3934	+1.49 [+0.27, +2.72] ✓
	CoRe	272	0.6368	**0.2726**	0.4753	**0.2988**	**0.3580**	**0.4083**	*reference*

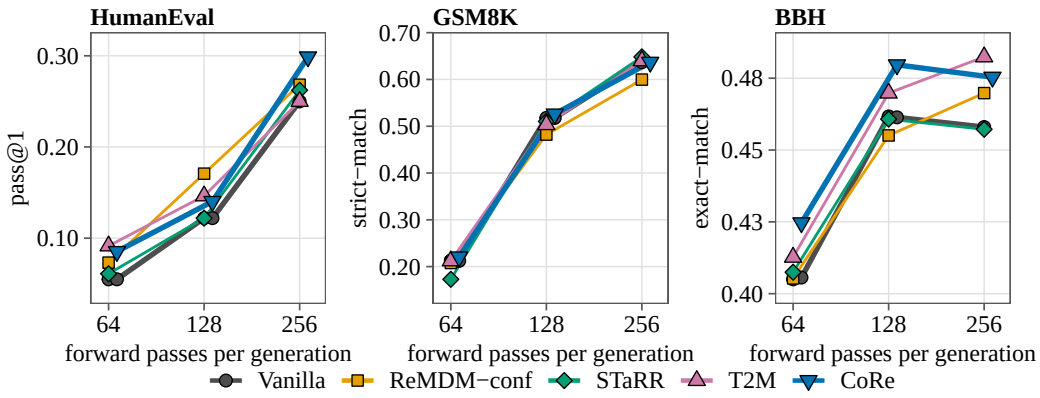


Figure 2: Accuracy against forward passes per generation on HumanEval, GSM8K and BBH. CoRe is plotted at the passes it spends; vanilla includes the equal-cost runs.

CoRe has the highest mean at every budget, but its lead over the best compute-free overlay, T2M at every budget, is +0.46 [−0.52, +1.42] points at N=64 and +0.97 [−0.20, +2.13] at N=128, both intervals including zero (Figure 1, right), and +1.49 [+0.27, +2.72] at N=256. On the four prompt-matched benchmarks the lead is +0.36 [−0.66, +1.37], +0.64 [−0.61, +1.88] and +1.94 [+0.58, +3.34], the same pattern. An exact McNemar test pooled over all 13,494 problems gives p=0.0067 for CoRe against T2M at N=128, but it weights problems rather than benchmarks, so MATH and BBH (11,511 of the problems) dominate it, and it does not test the equal-weight mean. The best method on a benchmark also changes with the budget: at N=128 ReMDM-conf leads HumanEval and STaRR leads MBPP, at N=256 STaRR leads GSM8K and T2M leads BBH, and no method is best on every benchmark at any budget. ReMDM-conf's HumanEval lead rests on one draw of its random remasking; with seeds 7 and 1337 its HumanEval score is 0.2012 rather than 0.1707 (Appendix B.6). The compute-free overlays gain mostly on MBPP (+4.6 to +9.8 points at N=128) and fall below vanilla on GSM8K and MATH at N=128.

Figure 2 plots the accuracies of Table 2 against forward passes on three benchmarks, with CoRe at the passes it spends. On HumanEval CoRe's margin over vanilla is largest at the highest budget; on GSM8K it shrinks to zero at N=256, where CoRe and vanilla reach the same accuracy through 20 rescues and 20 breakages; on BBH vanilla gains 5.7 points from 64 to 128 steps and then changes by

Table 3: Correctness flips relative to vanilla at the same step budget.

N	Method	Rescues / breakages per benchmark GSM8K	MATH	BBH	HumanEval	MBPP	All five benchmarks Flips	Rescue fraction [95% CI]
64	CoRe	16/6	21/8	287/159	5/0	25/2	529	0.669 [0.627, 0.709] ✓
	ReMDM-conf	152/159†	98/136	601/599	4/1	27/8	1,785	0.494 [0.471, 0.518] ✗
	STaRR	121/174†	78/134	443/427	1/0	19/4	1,401	0.473 [0.446, 0.499] ✗
	T2M	159/160†	103/113	681/631	6/0	23/2	1,878	0.518 [0.495, 0.540] ✗
128	CoRe	21/10	39/18	265/148	5/2	46/1	555	0.677 [0.637, 0.716] ✓
	ReMDM-conf	172/219†	321/400	546/589	10/2	48/25	2,332	0.470 [0.450, 0.491] ✗
	STaRR	199/209†	309/332	318/324	4/4	62/13	1,774	0.503 [0.479, 0.526] ✗
	T2M	203/222†	312/321	605/552	7/3	44/7	2,276	0.514 [0.494, 0.535] ✗
256	CoRe	20/20	62/46	258/145	10/2	33/0	596	0.643 [0.603, 0.681] ✓
	ReMDM-conf	146/195†	317/488	557/480	10/7	53/33	2,286	0.474 [0.453, 0.494] ✗
	STaRR	163/148†	294/365	228/233	5/3	42/27	1,508	0.485 [0.460, 0.511] ✗
	T2M	158/154†	302/365	573/414	7/7	37/14	2,031	0.530 [0.508, 0.552] ✓

-0.37 points (p=0.49), while CoRe keeps a margin of about 1.8 points. With three budgets these patterns describe the sweep and are not fitted trends.

Takeaway 2

The ranking among overlays is less settled than single numbers suggest: CoRe's lead over the best compute-free overlay is within benchmark sampling uncertainty at two of three budgets, and the best method on a benchmark changes with the budget.

4.3 RQ3: Correctness flips

To answer **RQ3**, we split each overlay's net change relative to vanilla at the same step budget into rescues (problems vanilla fails and the overlay solves) and breakages (the reverse), the sample analogues of r_b and β_b in (3). Table 3 reports them per benchmark as rescues/breakages, with the total number of changed problems over the five benchmarks and the rescue fraction with its exact 95% interval; ✓ marks a fraction whose interval lies above one half.

Table 3 shows that the overlays differ far more in how many answers they change than in net accuracy. At N=128 CoRe changes 555 of the 13,494 problems with a rescue fraction of 0.677 $[0.637, 0.716]$, while STaRR, T2M and ReMDM-conf change 1,774, 2,276 and 2,332 problems with fractions of 0.503 $[0.479, 0.526]$, 0.514 $[0.494, 0.535]$ and 0.470 $[0.450, 0.491]$; the first two intervals include one half and the third lies below it. Across the three budgets the compute-free overlays change 2.5 to 4.2 times as many answers as CoRe, and 2.2 to 3.7 times at rescue fractions of 0.475 to 0.535 when GSM8K, whose prompts differ, is left out. Per benchmark, CoRe's rescues outnumber its breakages significantly on MATH, BBH and MBPP (Table 1), while on GSM8K and HumanEval too few problems change to support a conclusion. The counts are correctness flips, not remasked tokens: CoRe remasks at most one token per verification call, at most 8 per generation at N=128 and 3.7 to 7.0 on average (Appendix A.5), and we did not log how many tokens the compute-free overlays remask. Because the methods also differ in schedule, thresholds and candidate tokens, the data do not identify why CoRe's fraction is higher; attributing it to the perturbed-context verification would need an ablation that fixes the schedule and intervention rate while removing the verification signal, which we did not run. Prompt length does not stratify the flips reliably (Appendix B.2).

Takeaway 3

Equal net accuracy can hide very different behaviour: the compute-free overlays change several times as many answers as CoRe, at rescue fractions near one half, so the number of changed answers belongs next to the net gain.

5 CONCLUSION

Revision overlays for masked diffusion language models are usually evaluated at one operating point, where a gain can reflect the budget, the compute spent or the sample of problems as much as the method. We compared four overlays and vanilla decoding in one decoder on LLaDA-8B-Base at three budgets, counted forward passes from the implementations, ran equal-cost controls and estimated paired uncertainty for the headline mean. CoRe's gain over vanilla held at every budget and at equal cost, whereas its lead over the compute-free overlays was within the sampling uncertainty at 64 and 128 steps, and the overlays differed most in how many answers they changed. A revision method's gain is a property of the operating point and the backbone as much as of the method, so a report should give the budget, the pass count and the uncertainty of the headline metric with it. The ablation that would explain CoRe's higher rescue fraction, the equal-cost control at the largest budget and further seeds for ReMDM-conf remain future work (Appendix C).

REPRODUCIBILITY STATEMENT

All runs use one LLaDA-8B-Base checkpoint (`GSAI-ML/LLaDA-8B-Base`) with greedy token selection, L=512, a single block, batch size 1 and seed 42, in one `lm-evaluation-harness` adapter in which vanilla and CoRe use the unmodified `UCF-CRCV/core` release. A single run manifest maps every method and budget to its per-problem logs, and one script regenerates every table, figure and in-text statistic from it (Appendix A.1). Appendix A.2 gives each decision rule and Appendix A.3 the protocol and compute accounting. The run drivers, the manifest and the analysis code are available at `https://anonymous.4open.science/r/dLMM-benchmark-iclr27-81D2/`, and the per-problem logs they read (1.2 GB) will be released with them.

ETHICS STATEMENT

This work evaluates decoding methods for publicly released language models on public benchmarks (GSM8K, MATH, BBH, HumanEval and MBPP). It involves no human subjects, no crowdsourcing and no personal or sensitive data, and it introduces no new model or training data. Programs generated for HumanEval and MBPP are executed only inside the evaluation harness's sandbox, and the models are used under their released licenses. The main foreseeable impact is on how revision methods are evaluated and reported: our results argue for quoting the budget, the forward passes and the uncertainty with every gain, which should make comparisons harder to overstate. All experiments are inference-only, and the per-problem logs we will release let others re-analyse the results without spending further compute. The authors have read and adhere to the ICLR Code of Ethics.

USE OF LARGE LANGUAGE MODELS

During the preparation of this paper, we used a large language model to draft and revise text and to improve its grammar, word choice and conciseness. We also used it to help identify relevant literature; the authors read each suggested work and verified every citation against its original source before inclusion. We used Claude Code to assist with implementing the analysis scripts, running the experiments and producing the tables and figures. The authors reviewed the code and verified all reported numerical results against the saved per-problem logs, which will be released with the paper. All claims are the authors' own, and we take full responsibility for the content of this 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 (ICLR)*, 2025.

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 (NeurIPS)*, volume 34, pp. 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.

Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Nazanin Mohammadi Sepahvand, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Tal Arbel, Chris Pal, Gaël Varoquaux, and Pascal Vincent. Accounting for variance in machine learning benchmarks. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 3, pp. 747–769, 2021.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

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, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998. doi: 10.1162/089976698300017197.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. Show your work: Improved reporting of experimental results. 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. 2185–2194. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1224.

Yihong Dong, Zhaoyu Ma, Xue Jiang, Zhiyuan Fan, Jiaru Qian, Yongmin Li, Jianha Xiao, Zhi Jin, and Ge Li. Saber: Efficient sampling with adaptive acceleration and backtracking enhanced remasking for diffusion language model in code generation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3623–3642. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.165.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1383–1392. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1128.

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. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1633.

Jiatao Gu, Changhan Wang, and Jake Zhao. Levenshtein Transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11694.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-In, Narrow-Out: Revokable decoding for efficient and effective DLLMs. In *International Conference on Learning Representations (ICLR)*, 2026.

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 (ICLR)*, 2026.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32819–32848. PMLR, 2024.

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs created equal? A large-scale study. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

Lovish Madaan, Aaditya K. Singh, Rylan Schaeffer, Andrew Poulton, Sanmi Koyejo, Pontus Stenetorp, Sharan Narang, and Dieuwke Hupkes. Quantifying variance in evaluation benchmarks. *arXiv preprint arXiv:2406.10229*, 2024.

Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. *arXiv preprint arXiv:2411.00640*, 2024.

Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check. In *European Conference on Computer Vision (ECCV)*, volume 12370 of *Lecture Notes in Computer Science*, pp. 681–699. Springer, 2020. doi: 10.1007/978-3-030-58595-2_41.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 56354–56392, 2025. doi: 10.52202/085713-1689.

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 (NeurIPS)*, volume 37, pp. 130136–130184, 2024. doi: 10.52202/079017-4135.

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 (NeurIPS)*, volume 37, pp. 103131–103167, 2024. doi: 10.52202/079017-3277.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

Xinhao Sun, Huaijin Zhao, Maoliang Li, Zihao Zheng, Jiayu Chen, Yun Liang, and Xiang Chen. STaRR: Spatial-temporal token-dynamics-aware responsive remasking for diffusion language models. *arXiv preprint arXiv:2601.04205*, 2026.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.824.

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 (NeurIPS)*, volume 38, pp. 163488–163545, 2025. doi: 10.52202/085713-4927.

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 (ICLR)*, 2026.

Lin Yao. Remask, don't replace: Token-to-Mask refinement in diffusion language models. *arXiv preprint arXiv:2604.18738*, 2026.

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.

Kevin Zhai, Sabbir Mollah, Zhenyi Wang, and Mubarak Shah. CORE: Context-robust remasking for diffusion language models. In *International Conference on Machine Learning (ICML)*, 2026.

A PROTOCOL DETAILS

This appendix gives the details of the protocol of Section 3: the run manifest (Appendix A.1), the decision rule of each method (Appendix A.2), the evaluation protocol and compute accounting (Appendix A.3), and CoRe's mechanism statistics (Appendix A.5).

A.1 RUN MANIFEST AND TABLE REGENERATION

Every number in this paper is computed from per-problem `lm-eval` sample logs through one loader and one manifest. The manifest maps each (method, budget) to a directory of logs: vanilla and CoRe at $N{=}128$ come from the original CoRe reproduction run; vanilla and CoRe at $N{=}64$ and 256 come from the budget sweep (GSM8K, BBH, HumanEval) and a companion run (MATH, MBPP) with the same decoder and settings; ReMDM-conf, STaRR and T2M at all three budgets come from one run of the three rules in the same decoder as vanilla and CoRe; and the equal-cost vanilla cells (68 and 136 steps) and the ReMDM-conf seed-7 cells come from a separate driver with the same settings. Before these runs we re-ran vanilla at $N{=}128$ on HumanEval and obtained identical generations on all 164 problems with a different number of GPUs, so new and old cells are comparable.

The loader reads every sub-task file (MATH has 7 and BBH 27), keys records on (sub-task, problem id) because problem ids restart in each file, and selects the headline filter explicitly (GSM8K logs a strict-match and a flexible-extract record for each problem under the same id). BBH and MATH scores are example-weighted means over 6,511 and 5,000 problems, which equal the harness group scores; the unweighted means over sub-tasks are given for reference in Appendix B.3. One `export` script turns the manifest into every table, the CSV files the figures are drawn from, and the macros for the statistics quoted in the text. The manifest, the loader, the `export` script and the run drivers are available at `https://anonymous.4open.science/r/dLMM-benchmark-iclr27-81D2/`.

A.2 REIMPLEMENTATION DETAILS

All five methods share one decoding loop on the frozen LLaDA-8B-Base: at each of the N steps the model predicts a distribution at every masked position, the low-confidence rule commits the scheduled number of tokens, and each method adds its own rule that may return committed tokens to the masked state. Vanilla applies no remasking. The three compute-free rules use only quantities from the ordinary decoding pass and add no model calls. Table 4 lists the hyperparameters, which are the published values where available.

CoRe (context-robust remasking). We run the unmodified `UCF-CRCV/core` release. At every step i with $i \bmod E = 0$ and $\lfloor N/4 \rfloor \le i < \lfloor 3N/4 \rfloor$, CoRe selects the m committed, non-prompt tokens with the smallest top-2 margin under the current logits (excluding the last 16 positions of the block), masks all of them at once, and runs one forward pass on this perturbed sequence. A candidate is eligible for remasking if its perturbed-context argmax differs from the committed token and that argmax has probability at least 0.3; among eligible candidates, CoRe remasks the one whose committed token has the lowest log-probability under the perturbed context (at most one per call) and refills it with the perturbed-context prediction. This costs one extra forward pass per call and $N/(2E)$ per generation.

ReMDM-conf. At inner step i of N let $\alpha_t = i/N$ and $\alpha_s = (i{+}1)/N$ be the unmasked fractions before steps i and $i{+}1$, and let ψ_ℓ be the committed probability of the token at position ℓ. Each committed token is returned to `[MASK]` independently with probability

$$\sigma_\ell = \big[\mathrm{softmax}_\ell(-\psi)\big] \cdot \sigma_{\max}, \qquad \sigma_{\max} = \min\Big(1,\ \tfrac{1-\alpha_s}{\alpha_t}\Big), \tag{5}$$

so the step-wise remask budget $\sigma_{\max}$ is spread across positions in inverse proportion to confidence. The draws use the seeded random number generator, which makes a run reproducible for a given seed but not seed-invariant. Active while $i < 0.9N$, with the confidence mode $\eta{=}{-}1$.

STaRR (spatial-temporal responsive remasking). STaRR gives each committed position a dynamic threshold that blends a temporal and a spatial term and remasks positions whose current confidence $c_\ell^i = \max_v p_\ell^i(v)$ falls below it:

$$\tau_\ell^i = p \cdot T_\ell^{\text{base}} + (1-p) \cdot T_\ell^{\text{neigh}}, \qquad T_\ell^{\text{base}} = m \cdot \mathrm{var}_i\big(c_\ell; W_t\big), \qquad T_\ell^{\text{neigh}} = \mathrm{dev}_i(\ell), \tag{6}$$

where var_i is the deviation of the current confidence from its mean over the last W_t steps (with a fixed fallback τ_{fixed} before W_t steps of history exist), and $\mathrm{dev}_i(\ell) = c_\ell^i - \sum_{d=1}^{W_n} 2^{-d}\,(c_{\ell-d}^i + c_{\ell+d}^i)/Z$ is the spatial residual against a 2^{-d}-weighted neighborhood mean (edges replicated). The blend weight p is 0.6 near the block boundaries and 0.5 in the interior, and the rule is active while $i < 0.85N$. We reconstructed it from the paper's Eqs. 2–6 and omit its variable-commit speedup, which does not affect a fixed-step accuracy comparison.

T2M (remask, don't replace). T2M remasks any committed token whose own predictive probability falls below τ_{lp}, and a position may be remasked at most $C_{\max}$ times. Active while $i < 0.85N$.

Dream-7B. Dream predicts the token at position ℓ from the logits at position $\ell{-}1$; we shift the logits by one position before every decoding and verification pass, and otherwise run vanilla and CoRe exactly as on LLaDA-8B-Base.

Table 4: Hyperparameters and extra forward passes per generation at step budget N.

Method	Hyperparameters	Extra passes
Vanilla	none	0
CoRe	$m{=}32$, $E{=}8$, window $[N/4, 3N/4)$, at most one remask per call	$N/(2E)$
ReMDM-conf	window $0.9N$, $\eta{=}{-}1$, $\sigma_{\max}$ schedule	0
STaRR	$W_t{=}3$, $W_n{=}3$, $m{=}3$, $\tau_{\text{fixed}}{=}0.9$, window $0.85N$	0
T2M	$\tau_{\text{lp}}{=}0.7$, $C_{\max}{=}3$, window $0.85N$	0

A.3 Evaluation protocol, determinism and compute accounting

Harness and metrics. All numbers come from `lm-evaluation-harness` with a model adapter that wraps the LLaDA-8B-Base diffusion sampler. Few-shot counts follow the CoRe release: HumanEval 0, MBPP 3, GSM8K 4, MATH (`minerva_math`) 4 and BBH 3 with chain of thought. Metrics are pass@1 in the harness sandbox (HumanEval, MBPP), strict-match on the extracted final answer (GSM8K), and exact-match (MATH, and BBH with the `get-answer` filter), with MATH and BBH aggregated as example-weighted group scores. Every run uses greedy token selection (temperature 0, no classifier-free guidance), $L{=}512$, one block of length L and batch size 1. The step budget N is the only quantity varied across the sweep.

Prompts and commands. The harness builds each prompt from few-shot demonstrations followed by the problem. MATH, BBH and MBPP use fixed demonstrations and HumanEval none; GSM8K draws four demonstrations per problem from its training split (Appendix A.4). Figure 3 shows the full four-shot prompt for one GSM8K problem, the one used in the first example of Appendix B.8; the model continues after the final `Answer:`. Figure 4 gives the command for one evaluation cell and the arguments that change between cells.

Determinism and uncertainty. Vanilla, CoRe, STaRR and T2M contain no random draws once tokens are selected greedily. Re-executing vanilla at $N{=}128$ on HumanEval reproduced all 164 generations exactly, and so did re-running CoRe at $N{=}128$ on HumanEval with seed 7 instead of 42. ReMDM-conf does draw at random, so its results are one sample per seed (Appendix B.6). In both cases a single run is not free of uncertainty: a benchmark is a finite sample of problems, and the intervals and tests quantify how much a result could change on a new sample from the same tasks.

GSM8K, four-shot prompt

```
Question: Gloria has five times as many dimes as quarters in her graduation money. She decides
to put 2/5 of the quarters aside for future use. If she has 350 dimes, calculate the combined
number of quarters and dimes she has after putting aside 2/5 of the quarters?
Answer: If the number of dimes is 20 times the number of quarters, she has 350/5 = 70 quarters.
When she puts 2/5 of the quarters away, the number of quarters reduces by 2/5*70 =
<<2/5*70=28>>28
She's now left with 70-28 = <<70-28=42>>42 quarters.
In total, she has 42 quarters + 350 dimes = <<42+350=392>>392
#### 392

Question: Rodney and Todd are rowing down a river that is 50 yards wide at one end. If the
river's width increases from this end uniformly by 2 yards every 10 meters along, and they row
along the river at a rate of 5 m/s, how long (in seconds) will it take them to get to the point
where the river is 80 yards wide?
Answer: The difference in width between where they're starting from and their destination is
80-50 = <<80-50=30>>30 yards
The width increases 2 yards every 10 m along the river so they would have to cover (30/2)*10 =
<<(30/2)*10=150>>150 meters
Traveling at the rate of 5m/s, he will spend 150/5 = <<150/5=30>>30 seconds
#### 30

Question: Susie's pet lizard Moe takes 10 seconds to eat 40 pieces of cuttlebone each day. How
long would it take Moe to eat 800 pieces?
Answer: If it takes Moe 10 seconds to eat 40 cuttlebones, then in one second she eats 40
cuttlebones / 10 seconds = <<40/10=4>>4 cuttlebones/second
Since the cat eats 4 pieces of cuttlebone in 1 second, 800 pieces will be 800 cuttlebones / 4
cuttlebones/second = <<800/4=200>>200 seconds.
#### 200

Question: Maria buys a large bar of French soap that lasts her for 2 months.  She spends $8.00
per bar of soap.  If she wants to stock up for the entire year, how much will she spend on soap?
Answer: 1 bar of soap will last for 2 months and there are 12 months in 1 year so she needs 12/2
= 6 bars of soap
Each bar of soap is $8.00 and she needs 6 bars so that's a total of 8*6 = $<<8*6=48.00>>48.00
#### 48

Question: Rachel bought 23 cookies and Janet gave her 42 cookies. The other day her brother ate
44 of those cookies. How many cookies are left for Rachel?
Answer:
```

Figure 3: GSM8K prompt for one problem, exactly as the model receives it.

run command

```
# one evaluation cell: CoRe at N = 128, E = 8 on GSM8K (four-shot)
REVISE_EVERY=8 CANDIDATE_M=32 \
accelerate launch --num_processes <gpus> --mixed_precision bf16 eval.py \
    --model llada_dist --tasks gsm8k --num_fewshot 4 --llada_seed 42 \
    --log_samples --confirm_run_unsafe_code --output_path <out_dir> \
    --model_args "model_path='GSAI-ML/LLaDA-8B-Base',gen_length=512,\
block_length=512,steps=128,remasking=core"

# every other cell changes only these arguments
#   vanilla                 remasking=low_confidence
#   equal-cost vanilla      remasking=low_confidence, steps=68 or 136
#   compute-free overlays   remasking=remdm | starr | t2m
#   other budgets           steps=64 | 256
#   other benchmarks        --tasks minerva_math | bbh | humaneval | mbpp
#                           with --num_fewshot 4 | 3 | 0 | 3
```

Figure 4: Command for one evaluation cell and the arguments that differ between cells.

They assume problems are independent within a benchmark, which is approximately true for GSM8K, MATH, HumanEval and MBPP and less so for BBH, whose sub-tasks share few-shot contexts.

Compute accounting. Vanilla and the three compute-free overlays use exactly N passes. CoRe adds one pass per verification call and calls verification $N/(2E)$ times, for 68, 136 and 272 passes at N=64, 128, 256 with E=8 and 66, 132, 264 with E=16. The equal-cost vanilla runs use exactly the

first two of these pass counts, so they need no interpolation. The study comprises 75 cells (5 methods, 3 budgets, 5 benchmarks), the nine E=16 cells, the ten equal-cost vanilla cells and the ReMDM-conf seed cells.

A.4 PROMPT AUDIT

Every logged record carries a hash of the prompt the model received, so we can check, problem by problem, whether two cells saw the same input. Table 5 counts, for every cell, the problems whose prompt differs from vanilla's at N=128. Vanilla and CoRe at every budget, and the equal-cost vanilla runs, received identical prompts on all five benchmarks. The logged runs of ReMDM-conf, STaRR and T2M received identical prompts on MATH, BBH, HumanEval, MBPP, but on GSM8K they drew different four-shot demonstrations for 1313 of the 1,319 problems, at all three budgets. The harness version, seed and few-shot configuration recorded in those runs match ours, so the demonstrations depend on how the run was launched rather than on its settings. GSM8K comparisons between these three overlays and vanilla or CoRe therefore mix the overlay with a change of prompt. We mark those cells with † in Tables 2 and 3, and we repeat the comparisons that involve them on the four benchmarks where every prompt matches: CoRe's lead over T2M is $+0.36$, $+0.64$ and $+1.94$ points at N=64, 128 and 256, with intervals that include zero at the first two budgets, and the compute-free overlays change 2.2 to 3.7 times as many answers as CoRe. Neither conclusion depends on GSM8K.

Table 5: Problems whose prompt differs from vanilla's at N=128.

Method	N	GSM8K	MATH	BBH	HumanEval	MBPP
Vanilla	64	0	0	0	0	0
	68	0	0	0	0	0
	128	0	0	0	0	0
	136	0	0	0	0	0
	256	0	0	0	0	0
CoRe	64	0	0	0	0	0
	128	0	0	0	0	0
	256	0	0	0	0	0
ReMDM-conf	64	1313	0	0	0	0
	128	1313	0	0	0	0
	256	1313	0	0	0	0
STaRR	64	1313	0	0	0	0
	128	1313	0	0	0	0
	256	1313	0	0	0	0
T2M	64	1313	0	0	0	0
	128	1313	0	0	0	0
	256	1313	0	0	0	0

A.5 CoRe MECHANISM STATISTICS

Table 6 reports CoRe's logged mechanism statistics at N=128 and E=8. The schedule fires at the multiples of E in $[32, 96)$, i.e. 8 times per generation, and each call masks m=32 candidates. CoRe remasks at most one token per call and on average 3.7 (BBH) to 7.0 (MATH) per generation. The change rate, the fraction of remasked tokens whose perturbed-context prediction differs from the committed token, is 1 by construction. The logger records every generation the evaluation runs, including the duplicates that distributed evaluation adds to pad each sub-task to a multiple of the number of processes, so the number of generations slightly exceeds the number of problems (1,319, 5,000, 6,511).

Table 6: CoRe mechanism statistics per benchmark at N=128 and E=8.

Benchmark	Generations	Per call		Per generation		
		Verified	Change rate	Calls	Remasked	Avg. Δ (nats)
GSM8K	1,320	32	1.000	8	6.9	6.29
MATH	5,008	32	1.000	8	7.0	6.66
BBH	6,564	32	1.000	8	3.7	5.42
HumanEval	164	32	1.000	8	4.6	5.64
MBPP	500	32	1.000	8	5.2	5.36

B Additional experiments

This appendix collects supporting results for the three RQs: the revision interval (Appendix B.1) and prompt length (Appendix B.2); per-sub-task and per-method breakdowns (Appendices B.3 and B.4); the Dream-7B check and two further seeds for ReMDM-conf (Appendices B.5 and B.6); and a comparison with published numbers and qualitative examples (Appendices B.7 and B.8).

B.1 Revision interval

Setup. We rerun CoRe with E=16, which halves the number of verification passes (66, 132 and 264 passes), on the three budget-sweep benchmarks. In Table 7, Δ is the difference from vanilla at the same N in points, and ✓ marks a significant gain (exact McNemar, $p < 0.05$).

Main result: halving the verification frequency costs accuracy on BBH only. E=16 lowers BBH relative to E=8 at every budget, by 0.71, 0.52 and 0.40 points ($p < 0.001$, 0.02 and 0.09 for the paired E=16 against E=8 comparison), and changes HumanEval and GSM8K by at most 0.61 points ($p \geq 0.23$).

Table 7: CoRe with revision intervals E=8 and E=16.

N	Method	Passes	HumanEval		GSM8K		BBH	
			Accuracy	Δ	Accuracy	Δ	Accuracy	Δ
64	Vanilla	64	0.0549		0.2130		0.4050	
	CoRe E=8	68	0.0854	+3.05 ✗	0.2206	+0.76 ✗	0.4247	+1.97 ✓
	CoRe E=16	66	0.0793	+2.44 ✗	0.2168	+0.38 ✗	0.4176	+1.26 ✓
128	Vanilla	128	0.1220		0.5178		0.4617	
	CoRe E=8	136	0.1402	+1.83 ✗	0.5262	+0.83 ✗	0.4796	+1.80 ✓
	CoRe E=16	132	0.1402	+1.83 ✗	0.5277	+0.99 ✓	0.4744	+1.27 ✓
256	Vanilla	256	0.2500		0.6368		0.4580	
	CoRe E=8	272	0.2988	+4.88 ✓	0.6368	0.00 ✗	0.4753	+1.74 ✓
	CoRe E=16	264	0.2988	+4.88 ✓	0.6406	+0.38 ✗	0.4714	+1.34 ✓

B.2 Prompt-length stratification

Setup. We stratify CoRe's change at N=128 by the number of LLaDA-8B-Base-tokenizer tokens in the full model input (few-shot context and question), with quartiles cut within each benchmark over all its problems. A heterogeneity test permutes quartile labels within the benchmark (20,000 permutations) with the range of the four quartile differences as its statistic; its p value is given under each benchmark name in Table 8.

Main result: no consistent dependence on prompt length. The quartile differences are within noise on MATH (p=0.385), MBPP (p=0.637) and HumanEval (p=0.544, where only 7 problems change), and marginal on GSM8K (p=0.078, driven by the longest quartile, +2.4 points). On BBH they vary strongly ($p < 0.001$) but non-monotonically, and the BBH quartiles are confounded with sub-task, since each sub-task has its own few-shot context: the per-sub-task differences range from

-3.2 to $+16.8$ points. With quartiles cut and permuted within each sub-task, the BBH range falls from 4.7 to 2.4 points (quartile differences $+1.8$, $+0.4$, $+2.3$, $+2.8$; p=0.031), and the MATH range is 0.5 points (p=0.658, against 0.7 pooled). These data do not show prompt length to be a reliable predictor of where revision helps, but they cannot rule out a weak effect.

Table 8: Prompt-length quartiles at N=128 and E=8.

Benchmark	Q	Quartile n	Quartile Tokens	Accuracy Vanilla	Accuracy CoRe	CoRe − vanilla Δ	CoRe − vanilla Rescues / breakages
GSM8K	Q1	330	519–739	0.567	0.564	−0.30 ✗	3/4
perm. p=0.078	Q2	330	739–819	0.518	0.524	+0.61 ✗	3/1
	Q3	330	819–920	0.512	0.518	+0.61 ✗	6/4
	Q4	329	920–1401	0.474	0.498	+2.43 ✓	9/1
MATH	Q1	1250	624–648	0.306	0.315	+0.88 ✓	14/3
perm. p=0.385	Q2	1250	648–663	0.192	0.194	+0.16 ✗	8/6
	Q3	1250	663–687	0.113	0.115	+0.24 ✗	5/2
	Q4	1250	687–2016	0.061	0.065	+0.40 ✗	12/7
BBH	Q1	1628	232–778	0.610	0.653	+4.30 ✓	105/35
perm. p<0.001	Q2	1628	778–875	0.452	0.448	−0.37 ✗	30/36
	Q3	1628	875–999	0.365	0.373	+0.80 ✗	54/41
	Q4	1627	999–2487	0.419	0.444	+2.46 ✓	76/36
HumanEval	Q1	41	44–99	0.317	0.293	−2.44 ✗	1/2
perm. p=0.544	Q2	41	99–127	0.122	0.122	0.00 ✗	0/0
	Q3	41	127–171	0.024	0.073	+4.88 ✗	2/0
	Q4	41	172–412	0.024	0.073	+4.88 ✗	2/0
MBPP	Q1	125	664–684	0.168	0.256	+8.80 ✓	11/0
perm. p=0.637	Q2	125	684–708	0.160	0.280	+12.00 ✓	15/0
	Q3	125	708–756	0.136	0.216	+8.00 ✓	10/0
	Q4	125	757–4320	0.192	0.264	+7.20 ✓	10/1

B.3 Per-sub-task and per-sub-topic breakdowns

Main result: CoRe's BBH gain is concentrated in a few sub-tasks. Tables 9 and 10 break the BBH and MATH scores at N=128 into sub-tasks, with the best method per row in bold. CoRe's BBH gain over vanilla comes mostly from `sports_understanding` (+16.8 points), `movie_recommendation` (+6.8) and `multistep_arithmetic_two` (+5.6) and is flat or slightly negative on many other sub-tasks, while T2M leads the `tracking_shuffled_objects` sub-tasks. Each table ends with the unweighted mean over sub-tasks and the example-weighted score used in the main text. They differ by at most half a point on BBH but by 1.4 to 1.9 points on MATH, whose largest sub-topics are also its easiest, so we use the example-weighted score throughout and never mix the two.

Table 9: BBH exact-match per sub-task at N=128.

Sub-task	n	Vanilla	CoRe	ReMDM-conf	STaRR	T2M
boolean_expressions	250	0.828	**0.852**	0.804	0.844	0.816
causal_judgement	187	0.465	0.513	**0.519**	0.492	0.433
date_understanding	250	0.804	0.816	**0.824**	0.808	0.812
disambiguation_qa	250	0.492	**0.500**	0.448	0.444	0.444
dyck_languages	250	0.016	0.020	0.020	**0.024**	0.020
formal_fallacies	250	**0.472**	0.468	0.432	0.452	0.464
geometric_shapes	250	0.332	**0.340**	0.328	0.328	0.312
hyperbaton	250	0.520	0.520	**0.528**	0.520	0.516
logical_deduction_five_objects	250	**0.440**	0.408	0.400	0.408	0.404
logical_deduction_seven_objects	250	**0.448**	0.444	**0.448**	0.444	0.424
logical_deduction_three_objects	250	**0.604**	0.592	0.572	0.588	0.588
movie_recommendation	250	0.404	**0.472**	0.392	0.436	**0.472**
multistep_arithmetic_two	250	0.424	**0.480**	0.404	0.440	0.444
navigate	250	0.688	0.688	**0.700**	0.688	0.688
object_counting	250	0.680	0.688	0.696	0.688	**0.708**
penguins_in_a_table	146	0.603	0.610	**0.651**	0.596	0.603
reasoning_about_colored_objects	250	0.576	**0.596**	0.592	0.584	0.568
ruin_names	250	**0.432**	**0.432**	0.400	0.420	0.396
salient_translation_error_detection	250	0.484	**0.508**	0.488	0.484	0.424
snarks	178	0.556	**0.573**	0.539	0.556	0.517
sports_understanding	250	0.344	**0.512**	0.340	0.292	0.420
temporal_sequences	250	0.452	**0.492**	0.416	0.456	0.480
tracking_shuffled_objects_five_objects	250	0.380	0.392	0.368	0.384	**0.452**
tracking_shuffled_objects_seven_objects	250	0.084	0.100	0.096	0.128	**0.256**
tracking_shuffled_objects_three_objects	250	0.484	0.492	**0.504**	0.480	**0.504**
web_of_lies	250	0.488	0.484	0.472	0.488	**0.524**
word_sorting	250	0.052	0.048	0.028	**0.060**	0.056
Unweighted mean		0.465	0.483	0.460	0.464	0.472
Example-weighted		0.462	0.480	0.455	0.461	0.470

Table 10: MATH exact-match per sub-topic at N=128.

Sub-topic	n	Vanilla	CoRe	ReMDM-conf	STaRR	T2M
algebra	1187	0.254	**0.263**	0.226	0.243	0.255
counting_and_prob	474	0.127	0.131	0.124	**0.143**	0.118
geometry	479	0.129	0.129	0.119	**0.142**	0.136
intermediate_algebra	903	0.066	**0.072**	0.066	0.060	0.063
num_theory	540	0.081	0.078	0.087	**0.102**	0.096
prealgebra	871	0.309	**0.315**	0.273	0.279	0.307
precalc	546	**0.081**	**0.081**	0.059	0.075	0.057
Unweighted mean		0.150	0.153	0.136	0.149	0.147
Example-weighted		0.168	0.172	0.152	0.163	0.166

B.4 Per-method churn

Main result: the compute-free overlays change more answers at rescue fractions near one half. Figure 5 draws the N=128 block of Table 3. The compute-free overlays are close to or below one half on GSM8K, MATH and BBH, and above it on MBPP, where vanilla solves few problems and most changes are rescues.

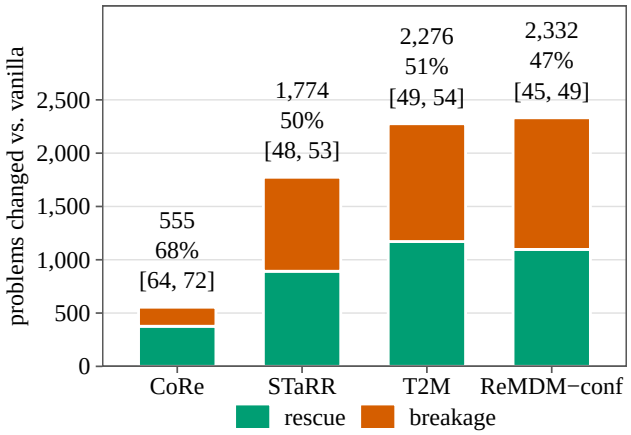


Figure 5: Problems whose correctness differs from vanilla at $N{=}128$, summed over the five benchmarks. Labels give the total and the rescue fraction with its exact 95% interval.

B.5 DREAM-7B

Main result: the mean gain carries over, the per-benchmark profile does not. Table 11 gives the Dream-7B numbers behind the last row of Table 1, with the same metrics and aggregation. The compute-free overlays and other budgets were not run on Dream, so this table supports only the vanilla-versus-CoRe comparison at $N{=}128$.

Table 11: Vanilla and CoRe on Dream-7B at $N{=}128$.

Row	GSM8K	MATH	BBH	HumanEval	MBPP	Mean
Vanilla	0.4511	0.1674	0.4323	0.0122	0.2780	0.2682
CoRe	0.4564	0.1792	0.3990	0.1768	0.3500	0.3123
Δ (points)	+0.53	+1.18	−3.33	+16.46	+7.20	+4.41
Rescues / breakages	30/23	113/54	193/410	27/0	39/3	
McNemar p / 95% CI	0.410	<0.001	<0.001	<0.001	<0.001	[+3.18, +5.71]

B.6 REMDM-CONF SEED SPREAD

Setup. Because ReMDM-conf draws its remasking decisions at random, we re-ran it at $N{=}128$ with seeds 7 and 1337 on HumanEval and MBPP. In Table 12, Δ is the difference from seed 42 in points and "Flips" counts the problems whose correctness changes between that seed and seed 42. We leave out the GSM8K seed runs, because the logged seed-42 run used different demonstrations there (Appendix A.4).

Main result: other seeds move ReMDM-conf by more than its lead on HumanEval. No difference from seed 42 is significant, but seeds 7 and 1337 change the correctness of 15 and 15 HumanEval problems and 68 and 69 MBPP problems, and both raise HumanEval from 0.1707 to 0.2012. On HumanEval the change between seeds (+3.05 points) is larger than ReMDM-conf's lead over T2M at seed 42 (2.44 points), so its single-seed ranking on a benchmark should be read as one draw. The other methods draw no random numbers once tokens are selected greedily.

Table 12: ReMDM-conf at N=128 under three seeds.

Seed	HumanEval Accuracy	HumanEval Δ	HumanEval Flips	MBPP Accuracy	MBPP Δ	MBPP Flips
42	0.1707			0.2100		
7	0.2012	+3.05 ✗	15	0.2260	+1.60 ✗	68
1337	0.2012	+3.05 ✗	15	0.2280	+1.80 ✗	69

B.7 Comparison with published numbers

Main result: our pipeline reproduces the published vanilla and CoRe numbers. Table 13 compares our vanilla and CoRe numbers at N=128 with Table 1 of Zhai et al. (2026), and our equal-cost vanilla at 136 steps with the same control in their Table 3. Vanilla at N=128 agrees within 0.8 points on every benchmark and CoRe within 0.8 points on four of five; the larger HumanEval gap (14.0 against 17.1) is five problems out of 164, which we attribute to `lm-eval` version and sandbox differences since we run the unmodified release. Our equal-cost vanilla at 136 steps is within 1.2 points of theirs on every benchmark, and both runs find that the eight extra steps change vanilla very little. Zhai et al. (2026) also report ReMDM-conf at 136 passes (51.48, 16.96, 46.01, 10.37, 14.00); their implementation moves vanilla less than ours, which shows how sensitive this rule is to its schedule and randomness.

Table 13: Our numbers against Zhai et al. (2026) on LLaDA-8B-Base (accuracy ×100).

Configuration	Source	GSM8K	MATH	BBH	HumanEval	MBPP
Vanilla, 128 steps	Zhai et al. (2026)	51.40	16.72	45.81	12.20	15.60
	ours	51.78	16.80	46.17	12.20	16.40
CoRe, 136 passes	Zhai et al. (2026)	52.69	17.06	47.18	17.07	24.80
	ours	52.62	17.22	47.96	14.02	25.40
Vanilla, 136 steps	Zhai et al. (2026)	52.08	16.72	45.74	12.80	15.40
	ours	51.78	16.78	46.14	12.20	16.60

B.8 Qualitative rescue and breakage examples

This section shows real outputs from the N=128 logs. Each box gives the problem and, for each method, the generation up to the answer line the scorer reads; the generation continues after that line to fill 512 tokens, which we omit. The answer read by the scorer and its verdict are marked after each method's name.

GSM8K. At N=128 there are 21 rescues and 10 breakages, a net of +11 problems (+0.83 points, p=0.071). Some flips change the reasoning: in the first example of Figure 6 vanilla subtracts wrongly and CoRe does not. Others change only the final answer line that strict-match reads. In 5 of the 21 rescues vanilla's output has no parsable `####` answer line, as in the second example, and 7 of the 10 breakages are CoRe outputs that either have no parsable answer line (3) or append digits to a correct answer (4), as in the first example of Figure 7, where `#### 81Question 1` is read as 81 rather than 8. On GSM8K, part of the churn is therefore churn in the answer format rather than in the solution.

MBPP and HumanEval. On MBPP CoRe turns 46 of vanilla's 418 failures into passes against one breakage. Figure 8 shows two rescues in which CoRe's program differs from vanilla's in a single place: a stray `)` **:** that makes vanilla's function a syntax error, and a wrong operand (`base3` for `base2`). On HumanEval the five rescues at N=128 are `will_it_fly`, `fib4`, `same_chars`, `pairs_sum_to_zero` and `median`, and the two breakages are `add` and `prime_length`. The examples illustrate the two kinds of flips; they do not show which remasked token caused each change.

GSM8K rescues, N = 128

```
Question: Rachel bought 23 cookies and Janet gave her 42 cookies. The other day her brother ate
44 of those cookies. How many cookies are left for Rachel?
Gold answer: 21

Vanilla output (answer read by the scorer: 11, scored wrong):
 Rachel had 23 cookies +42 cookies = <<23+42=65>>65 cookies. After her brother ate 44 cookies,
there are 65 cookies - 44 = 11 cookies..
#### 11

CoRe output (answer read by the scorer: 21, scored correct):
 Rachel had 23 cookies +42 cookies = <<23+42=65>>65 cookies. After her brother ate 44 cookies,
there are 65 cookies - 44 = 21 cookies left.
#### 21

-----------------------------------------------------------------------------------------------

Question: While walking down the street with his 3 younger siblings, Greg found $20. To be fair
to his siblings, he decided to split the money equally. How much money did each of them get?
Gold answer: 5

Vanilla output (answer read by the scorer: [invalid], scored wrong):
 Greg has 20/4 =$<<20/4=5>>5.
Each 5

CoRe output (answer read by the scorer: 5, scored correct):
 Greg has 20/4 =$<<20/4=5>>5.
#### 5
```

Figure 6: GSM8K problems that vanilla fails and CoRe solves at $N{=}128$.

GSM8K breakages, N = 128

```
Question: Raymond had $21. Then he saved $11 from his allowance and spent $5 on a comic book and
$19 on a puzzle. How much money does Raymond have left?
Gold answer: 8

Vanilla output (answer read by the scorer: 8, scored correct):
 Raymond had $21 + $11 = $<<21+11=32>>32.
He spent $19 + $5 = $<<19+5=24>>24.
He has $32 - $24 = $<<32-24=8>>8 left.
#### 8

CoRe output (answer read by the scorer: 81, scored wrong):
 He had $21 + $11 = $<<21+11=32>>32.
He spent $19 + $5 = $<<19+5=24>>24.
He has $32 - $24 = $<<32-24=8>>8 left.
#### 81Question 1

-----------------------------------------------------------------------------------------

Question: Carol spends 4 hours writing a song, half that much time recording it, and 90 minutes
editing it. What percentage of her total work time did she spend editing?
Gold answer: 20

Vanilla output (answer read by the scorer: 20, scored correct):
 She spends 4/2=<<4/2=2>>2 hours recording
So
She spends 4+2=<<4+2=6>>6 hours on and recording
She spends 90/60=<<90/60=1.5>>1.5 hours editing
So she spends 6+1.5=<<6+1.5=7.5>>7.5 hours working
So spends editing 1.5/7.5=<<1.5/7.5=0.2>>0.2

So she spends 0.2*100=<<0.2*100=20>>20% of her time editing
#### 20#

CoRe output (answer read by the scorer: 100, scored wrong):
 She spends 4/2=<<4/2=2>>2 hours recording
So
She spends 4+2=<<4+2=6>>6 hours writing and recording
She spends 90/60=<<90/60=1.5>>1.5 hours editing
So she spends 6+1.5=<<6+1.5=7.5>>7.5 hours working
So she spends 1.5/7.5=<<1.5/7.5=0.2>>0.2 editing
So she spent 0.2*100=<<0.2*100=20>>20% of her time editing
#### 100
```

Figure 7: GSM8K problems that vanilla solves and CoRe fails at $N=128$.

MBPP rescues, N = 128

```
Task: Write a function to find the perimeter of a square.
Tests:
  assert square_perimeter(10)==40
  assert square_perimeter(5)==20
  assert square_perimeter(4)==16

Vanilla program (fails the tests):
def square_perimeter(s):):
    return 4 * s

CoRe program (passes the tests):
def square_perimeter(s):
  return 4 * s

-----------------------------------------------------------------------------------------------

Task: Write a function to find the median of a trapezium.
Tests:
  assert median_trapezium(15,25,35)==20
  assert median_trapezium(10,20,30)==15
  assert median_trapezium(6,9,4)==7.5

Vanilla program (fails the tests):
def median_trapezium(base1, base2, base3):
  median = (base1 + base3) / 2
  return median

CoRe program (passes the tests):
def median_trapezium(base1, base2, base3):
  median = (base1 + base2) / 2
  return median
```

Figure 8: MBPP problems that vanilla fails and CoRe solves at $N{=}128$.

C SCOPE AND LIMITATIONS

The study uses three step budgets, so the budget patterns are descriptive rather than fitted trends. The equal-cost control covers $N{=}64$ and 128 but not $N{=}256$. ReMDM-conf is stochastic, its main results use one seed, Appendix B.6 adds two further seeds on two benchmarks, and our intervals for CoRe against ReMDM-conf cover the sampling of problems but not its seed variance. On GSM8K the logged runs of the three compute-free overlays used different demonstrations from vanilla and CoRe (Appendix A.4). We use greedy token selection only. The overlay comparison covers one backbone, and the Dream check covers one budget and two methods. ReMDM-conf, STaRR and T2M are our reimplementations with hyperparameters fixed across budgets, and they may differ from their authors' tuned versions. Forward passes ignore the small per-step cost of the compute-free rules and wall-clock effects. We did not run the ablation that would explain the difference in rescue fractions.

One hypothesis we did not test is confidence gating. Breakages fall on problems that vanilla solves, so one could skip revision when vanilla appears confident. Relative to always revising, skipping helps only if the skipped problems contain more would-be breakages than rescues, which the counts in Table 3 cannot decide, and a vanilla pre-pass costs N additional passes. Evaluating it would require a specified selector, a held-out threshold and an accuracy–coverage curve at total cost.