UNMASKING NONDETERMINISM IN MASKED DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Greedy decoding of masked diffusion language models (MDLMs) is nominally deterministic, yet the same prompt can yield different outputs when only the batch size or the GPU changes, while accuracy barely moves and so hides the problem. In autoregressive models, the first differing token permanently separates two generations, but MDLMs also decide at every step where to write, and how nondeterminism develops under this second decision is not understood. Tracking the partially decoded response over 512 steps for LLaDA and Dream, we find that forks arise mainly from the position decision. These forks are often transient: on LLaDA, 92.6% of prompts return to an identical intermediate response, and the token conflicts that make a difference permanent appear much later. Treating the two decisions separately, we freeze the position order under periodic LayerCast refresh and recheck only tokens within a few ULPs of a tie, cutting LLaDA's MATH500 output divergence from 69.2% to 1.0% at less than half the cost of LayerCast, and most of Dream's. Freezing changes the decoding rule, however, so the recipe's outputs agree with each other but often not with LayerCast's, and accuracy can shift. Reproducible MDLM evaluation should therefore track states beyond the first fork and report agreement across executions and accuracy separately. More broadly, locating where nondeterminism enters the decoding rule lets a remedy target the fragile decision instead of paying for full precision.

1 INTRODUCTION

Greedy decoding is meant to be deterministic: it uses no intentional sampling. In practice, the same request can still produce different outputs when only the batch size or the GPU changes (Yuan et al., 2025). The cause is numerical: floating-point addition is not associative, so kernels that reduce in a different order return slightly different scores (Goldberg, 1991; Demmel & Nguyen, 2013). Such a perturbation is harmless until it flips an argmax; from that token on, the two runs decode from different states. Outputs that depend on execution settings complicate regression tests, scientific comparisons, and dynamically batched serving, and the problem is easy to miss: aggregate accuracy barely moves even when many outputs change (Dutta et al., 2024; Fang et al., 2026).

This picture comes from autoregressive (AR) models, which make one decision per step: the next token for a fixed prefix. A masked diffusion language model (MDLM) makes two: *where* to write and *which* token to write there. Existing work shows that MDLM outputs likewise vary across batch, hardware, and precision settings even when aggregate scores remain similar (Fang et al., 2026). That observation leaves three questions open: in which of the two decisions the nondeterminism first arises, whether and how it affects the final output, and which intervention removes it efficiently.

We study two released models, LLaDA-8B-Instruct (Nie et al., 2025) and Dream-v0-Instruct-7B (Ye et al., 2025), with a decoder modified only to record every commitment. Repeating a configuration reproduces every token; changing the batch size, the GPU or the arithmetic precision does not. We make three contributions:

- The nondeterminism first appears in the position decision, not the token decision (99.9% of LLaDA's forks, the first steps at which two runs commit differently). This holds across models, tasks and execution configurations, and the first mismatch usually comes early, within the first 5% of steps.

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.

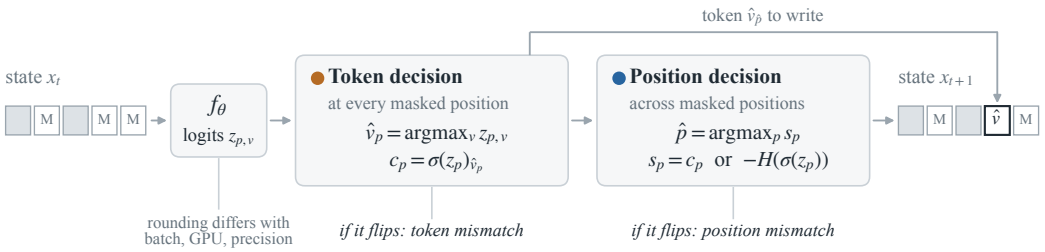

Figure 1: One greedy MDLM step makes two decisions (Equations 1 and 2; step index omitted). The token decision picks a candidate and its confidence at every masked position; the position decision picks where to write, scoring positions by confidence (LLaDA) or negative entropy (Dream). Rounding that differs across batch, GPU or precision perturbs the logits; two executions fork when it flips either decision, with a token mismatch or a position mismatch.

- An early mismatch rarely settles the outcome: in the primary LLaDA comparison, 92.6% of diverged runs later return to an identical partial response, irreversible token conflicts arrive much later if at all, and about a third of prompts end with identical outputs.
- Freezing the position order under periodic LayerCast refresh and ULP-aware token verification remove most of the disagreement at a fraction of the cost of LayerCast (MATH500 output divergence 69.2% to 1.0% at 2.2× instead of 5.1× the BF16 cost).

2 Related Work

Determinism in LLM inference. Floating-point addition is not associative, so the same computation can round differently when kernels, tensor shapes or reduction orders change (Goldberg, 1991; Demmel & Nguyen, 2013; Blanchard et al., 2021). In LLM inference this makes greedy outputs depend on the batch, the hardware and the arithmetic precision (Yuan et al., 2025). Proposed remedies act at the kernel or precision level: batch-invariant kernels fix the reduction structure (He & Thinking Machines Lab, 2025), LayerCast keeps low-precision weights but computes in FP32 (Yuan et al., 2025), and HEAL combines targeted quantization with error-compensated multiplication (Zhu et al., 2026). Fang et al. (2026) show that masked diffusion models are affected as well, with sample-level output changes that leave aggregate accuracy nearly unchanged. We start from that observation and ask where in the MDLM decoding rule the change enters and how it develops.

Masked diffusion decoding and reveal order. Masked diffusion language models (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024; Shi et al., 2024) fill masked positions iteratively, and confidence-guided reveal orders go back to Mask-Predict and MaskGIT (Ghazvininejad et al., 2019; Chang et al., 2022). Choosing where to reveal is known to matter for quality (Kim et al., 2025b; Asano et al., 2026), and much recent work changes the schedule: parallel and cached decoding (Arriola et al., 2025; Wu et al., 2026; Ma et al., 2025; Israel et al., 2025; Ben-Hamu et al., 2025), remasking that revises revealed tokens (Wang et al., 2026a; Huang et al., 2025; Wang et al., 2026b), and rules that exploit the within-run stability of candidates, from stability-weighted decoding to early stopping (Wu & Huang, 2026; Chen et al., 2026; Kohut et al., 2026). Our question is different: we hold the released schedule fixed and ask how two executions of the same greedy rule come to disagree. Appendix F explains why the between-execution reunions we measure differ from the within-run stability these methods exploit.

Verification for deterministic decoding. Selective verification is established for autoregressive decoding: speculative decoding checks proposals against a target model while preserving its distribution (Leviathan et al., 2023), LLM-42 uses fixed-shape verified speculation to make outputs deterministic (Gond et al., 2026), and MarginGate re-evaluates only next-token decisions with a small logit margin (Chu et al., 2026). All three verify the token decision. In an MDLM the position decision comes first, and we find that selecting it by margin cannot isolate the risk for LLaDA, because near-ties among positions are the norm. Our intervention therefore freezes the position order and gates only the token decision.

3 PRELIMINARIES AND SETUP

Decoder. At step t, let M_t be the set of masked response positions. An MDLM reads the partially masked response and produces a logit $z_{t,p,v}$ for every vocabulary item v at every $p \in M_t$, and σ denotes the softmax, $\sigma(z)_v = e^{z_v} / \sum_u e^{z_u}$. Both models' released samplers then make two decisions. They pick a token and its confidence at every masked position,

$$\hat{v}_{t,p} = \arg\max_v z_{t,p,v}, \qquad c_{t,p} = \sigma(z_{t,p})_{\hat{v}_{t,p}}, \tag{1}$$

and pick the position with the highest score,

$$\hat{p}_t = \arg\max_{p \in M_t} s_{t,p}, \qquad s_{t,p} = \begin{cases} c_{t,p} & \text{(LLaDA)}, \\ -H\big(\sigma(z_{t,p})\big) & \text{(Dream)}, \end{cases} \tag{2}$$

where $H(q) = -\sum_v q_v \log q_v$ is the entropy, and commit $\hat{v}_{t,\hat{p}_t}$ there. LLaDA scores a position by the confidence of its token and Dream by negative entropy, the default of each released sampler. Dream commits one token at almost every step (Appendix E.4).

Divergence and forks. We compare two executions of the same prompt through their *trajectories*, the sequences of position–token commitments. They show *trajectory divergence* if the trajectories differ and *output divergence* if the final outputs differ; we compare all 512 output tokens (*strict*) unless stated otherwise. When the trajectories differ but the outputs agree, the executions *re-converge*. The first step at which the trajectories differ is the *fork*, and the state just before it, which both executions share, is the *last shared state*. At the fork, the executions either commit at different positions, a *position mismatch*, or commit different tokens at the same position, a *token mismatch* (Appendix A). Rounding can flip a decision only if it moves a score by at least half the *margin*, the gap between the top two candidates, so forks start at decisions whose margin is small relative to rounding error (Section 6.1).

Experimental setup. We run the released LLaDA-8B-Instruct and Dream-v0-Instruct-7B models greedily, filling 512 response positions in 512 steps, on MATH500 ($n = 500$) (Hendrycks et al., 2021; Lightman et al., 2024) and all 164 HumanEval problems (Chen et al., 2021). Outputs are scored with `math_verify` and the official tests (Appendix B). Dream's greedy outputs collapse to EOS, so we set its EOS logit to $-\infty$ before both decisions. A configuration fixes the model, the precision in which it is loaded and run (BF16, FP16 or FP32), the GPU (RTX PRO 6000, A6000 or RTX 3090) and the batch size (1, 4 or 16). Dream with its entropy scorer is reported in BF16 and FP32 only: this released scorer adds 10^{-10} inside the logarithm, which underflows in FP16, so every position score becomes NaN and decoding falls back to a fixed left-to-right order (Appendix B). The *pair census* is every pair of configurations that differ only in batch size or only in GPU (Appendix C.1). Within it, the *primary pair* compares batch sizes 1 and 4 on the RTX PRO 6000: each prompt's tokens are unchanged, but its batchmates, left padding and tensor shapes are not, as when a serving system regroups requests. Appendix B gives the remaining protocol details and the models' accuracies.

4 WHERE GREEDY DECODING FORKS

4.1 BF16 OUTPUTS CHANGE OFTEN, YET ACCURACY BARELY MOVES

Changing only the batch size or the GPU changes many BF16 outputs in both models (Figure 2): averaged over the pair census, most MATH500 outputs differ for both models, and 23.1% (LLaDA) and 81.3% (Dream) of HumanEval outputs. Higher precision removes most of this: LLaDA's MATH500 divergence falls from 64.6% in BF16 to 19.4% in FP16 and 0.1% in FP32, and Dream shows the same contrast between BF16 and FP32. Trajectories diverge more often than outputs (94.6% for LLaDA MATH500 in BF16) because some forked executions re-converge (Section 4.2). The whiskers are wide because pairs differ in how much they change the execution; Appendix C.1 breaks them down by pair.

Accuracy hides almost all of this disagreement. On the primary pair in BF16, LLaDA solves 114 MATH500 problems at batch one and 116 at batch four although 346 of 500 outputs differ, and Dream solves 219 and 227 although 437 differ (Table 19). Output divergence therefore measures whether a result can be reproduced, not which execution is more correct.

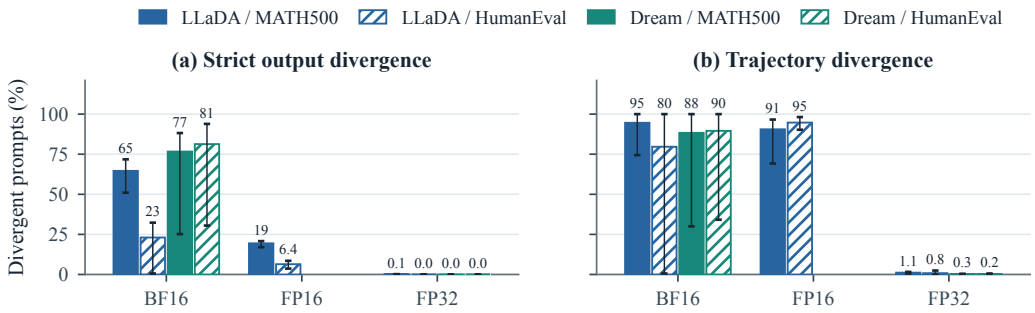

Figure 2: Strict output divergence and trajectory divergence by precision, averaged over the pair census (Section 3). Whiskers span the pairs. Dream has no FP16 bars because its released scorer is undefined in FP16 (Section 3).

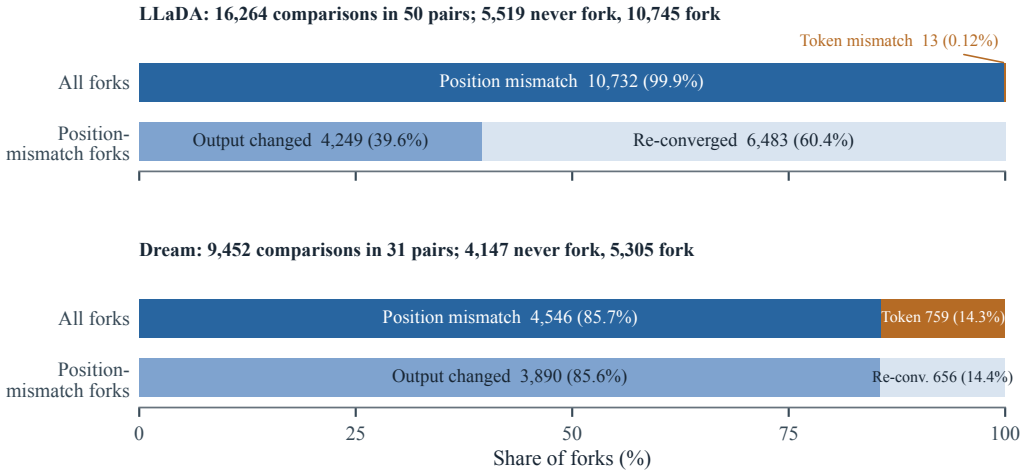

Figure 3: Forks over the pair census (MATH500 and HumanEval, all precisions), one row per model. The lower bar splits the position-mismatch forks by whether the executions re-converge. A prompt can appear in several pairs, so counts are descriptive; Appendix C.1 gives prompt-cluster bootstrap intervals.

4.2 Most forks are position mismatches

Figure 3 classifies every fork in the pair census. In LLaDA almost every fork is a position mismatch: 10,732 of 10,745, with no pair below 99.3%, and the primary BF16 pair begins with 499 position mismatches and one token mismatch. Dream's forks are also mostly position mismatches, 4,546 of 5,305 (85.7%), with no pair below 83%; the rest are token mismatches, which LLaDA almost never produces.

The two kinds of fork end differently. Of the position-mismatch forks, 60.4% re-converge to the same strict output in LLaDA and 14.4% in Dream, but no token-mismatch fork re-converges in either model: commitments are never revised, so two different tokens at the same position stay in the output. Forks also come early: the median fork occurs at 4.7% of the 512-step schedule for LLaDA and 3.3% for Dream (Appendix Table 5), so most of each response is decoded after the executions have separated. Section 5 follows what happens next.

A position mismatch names where the executions first commit differently, not what changed first numerically. At 80.8% of LLaDA's position-mismatch forks, some still-masked position already has a different candidate token at the last shared state, although almost never at a selected position (7 of 10,732; Appendix Table 4).

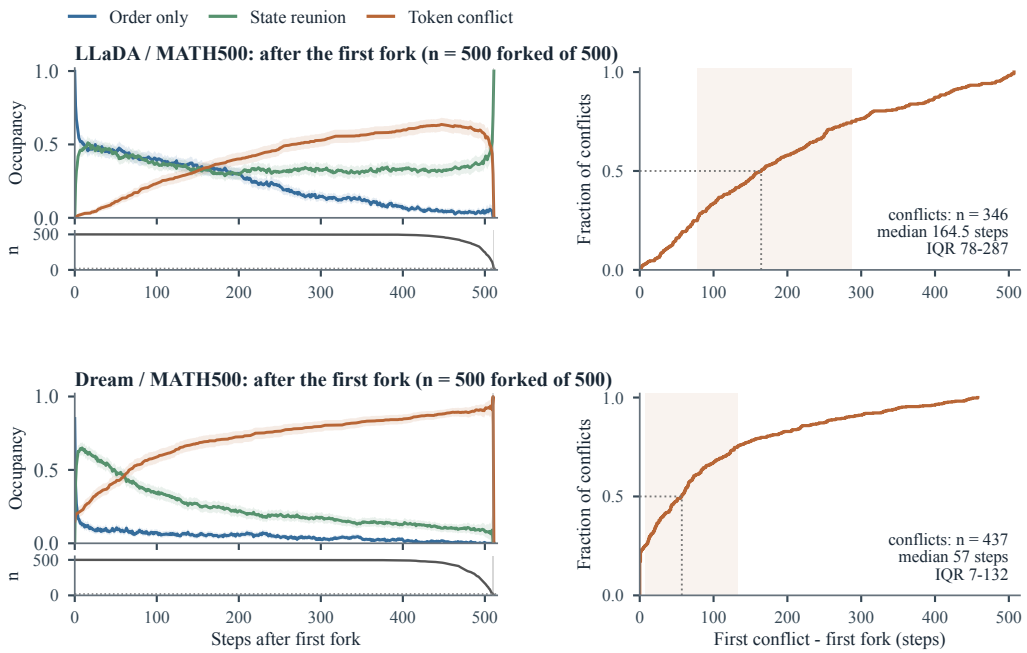

Figure 4: After the fork on the primary BF16 MATH500 pair, for LLaDA (top) and Dream (bottom), aligned to each prompt's fork. Left: share of prompts whose partial responses differ only in which positions are filled ($H > 0$, $C = 0$), are equal ($H = 0$), or contain a token conflict ($C > 0$); fewer prompts remain at long delays because runs end. Curves show the state at each delay, not whether an event ever occurs: because most reunions are temporary, the equal-state share stays well below the 92.6% of LLaDA prompts that reunite at least once. Ribbons are prompt-bootstrap 95% intervals. Right: steps from the fork to the first conflict for the prompts that reach one (346 for LLaDA, 437 for Dream); shading marks the interquartile range. Appendix Figure 10 repeats the figure for HumanEval.

5 What Happens After a Fork

A fork does not by itself fix the output. To see what follows it, we compare the two partial responses, masks included, after every step (Appendix D). After step t, let O_t count the positions committed in only one execution and C_t the positions committed in both with different tokens. Every unequal position is one of the two, and commitments are never revised, so the number of unequal positions H_t satisfies

$$H_t = O_t + C_t, \qquad C_{t+1} \geq C_t. \tag{3}$$

Once $C_t > 0$ the outputs must differ; we call this a *token conflict*. While $C_t = 0$, $H_t = O_t$: the executions differ only in which positions they have filled so far, and they can *reunite*, with H_t returning to zero before the final step. At the final step every position is filled, so $O_t = 0$ and $H_t = C_t$.

LLaDA's forked executions often reunite, and conflicts come late. Every prompt forks on the primary LLaDA MATH500 pair, yet 463 of 500 reunite at least once (Figure 4). Reunion is usually temporary: 460 of them fork again, and 346 prompts eventually reach a token conflict, a median of 164.5 steps after the fork. A fork is therefore not yet a changed output; the output changes only if a conflict arrives before the response is complete. On HumanEval, 162 of 164 prompts reunite and only 43 reach a conflict, a median of 388 steps after the fork (Appendix Table 9).

Dream reunites too, but conflicts sooner. Dream also reunites, on 379 of 500 MATH500 prompts, but 437 reach a conflict, a median of 57 steps after the fork (Figure 4). Its executions already disagree more at the last shared state: they predict different candidate tokens at 82 still-masked positions per fork on average, against 12 for LLaDA in BF16 (Appendix Table 4). When one execution later fills

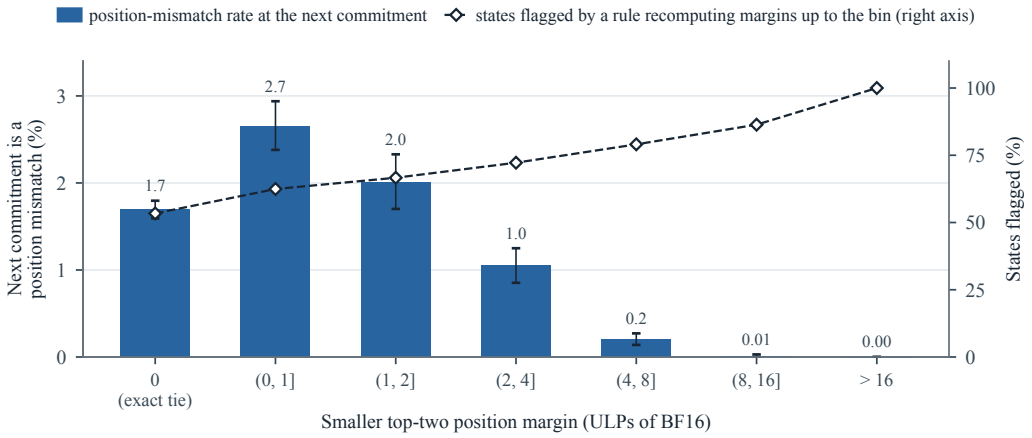

Figure 5: LLaDA BF16 position-fork risk by position margin, pooled over the pair census. For every shared state up to the fork, the margin is the smaller of the two runs' top-two confidence gaps in BF16 ULPs. Bars (left axis): rate at which the next commitment is a position mismatch, with prompt-cluster bootstrap 95% intervals. Dashed line (right axis): cumulative share of shared states whose margin is at most the bin's upper edge, that is, the share a k-ULP recomputation rule would flag.

a position that the other has already filled, the two tokens are then more likely to differ. The same difference appears as the lower re-convergence of Dream's position-mismatch forks in Section 4.2.

6 Order Refresh and ULP-Aware Token Verification

Sections 4 and 5 show that divergence has two stages: it enters mostly through the position decision as an order difference that can still disappear, and it becomes permanent only with a token conflict. A remedy therefore has to keep the executions filling positions in the same order and to catch the token decisions that would create a conflict. We intervene on the two decisions of Equations 1 and 2 separately: the position order comes from a periodic LayerCast pass and is frozen in between (Section 6.1), and the token at each frozen position is recomputed only when its margin is within a few ULPs of the execution dtype (Section 6.2). Both use LayerCast forward passes, which keep the released BF16 weights in storage but evaluate functional linear operations and the residual stream in FP32 (Yuan et al., 2025). Running LayerCast at every step without freezing is our point of comparison, and below *LayerCast* alone refers to this decoder; it matches native FP32 inference token for token where we ran both (Appendix B).

6.1 Freezing the order removes position forks

Section 4 located the fork almost always in the position decision. Recomputing that decision only when its margin is small would not save much for LLaDA (Figure 5). Position forks do concentrate at small margins, but BF16 confidences saturate near one, so small margins are everywhere: a single BF16 run ties exactly at 42.1% of its position decisions (Appendix Table 6). A rule that recomputes a state whenever either run's margin is within 2 ULP would flag 66.6% of shared states to catch 94.4% of position-mismatch forks. We therefore take the position decision out of low precision altogether. A refresh runs one LayerCast forward, ranks the top w masked positions and freezes their order; for the next $w - 1$ steps, a BF16 forward computes only the token at the next frozen position. The order is chosen at the refresh and kept for w steps, although every commitment changes the context of the remaining positions; the unfrozen decoder might rank them differently after those commitments. Freezing therefore changes the decoding rule itself, not only the arithmetic precision of the ranking (Section 6.3). Tokens, by contrast, are always computed from the current state. Appendix E.4 gives the Dream schedule exceptions.

On the primary LLaDA MATH500 pair, freezing brings position-mismatch forks down to the level of LayerCast: from 499 to 9–20 for every window, against 8 under LayerCast itself (Figure 6). What

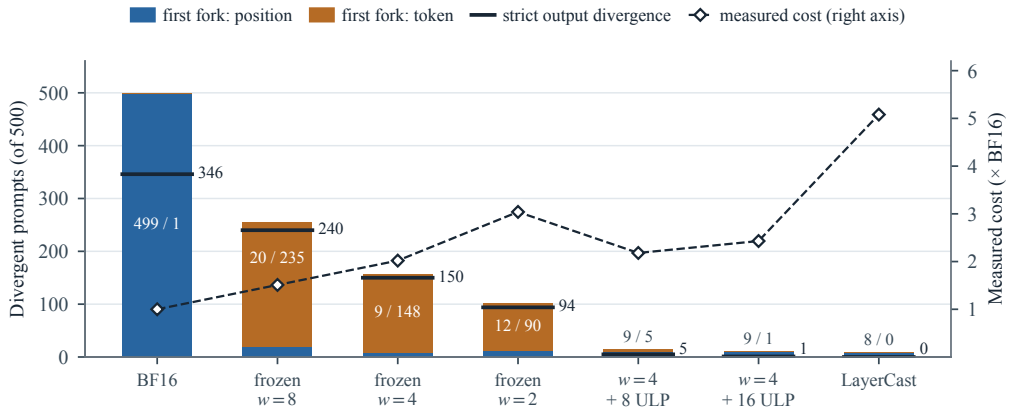

Figure 6: Freezing the order on the primary LLaDA MATH500 pair (RTX PRO 6000, batch $1 \leftrightarrow 4$, $n = 500$). Bars stack the divergent trajectories by the kind of fork (labels: position / token mismatches); the black line marks strict output divergence. Dashed line (right axis): measured per-sample cost relative to BF16 (Appendix E.6).

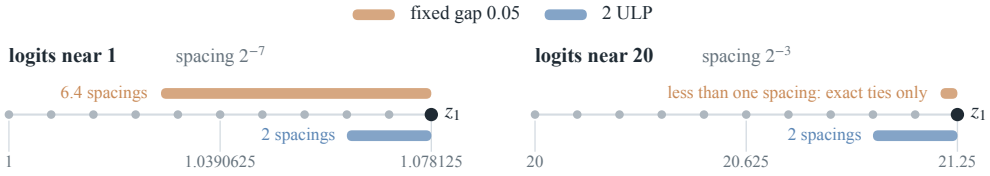

Figure 7: Why the token gate counts ULPs. Dots are the adjacent BF16 values below a top logit z_1 of 1.078125 (left) or 21.25 (right), all exactly representable; a runner-up z_2 under a bar triggers recomputation. A fixed gap of 0.05 spans 6.4 spacings near 1 but less than one spacing near 20, where it catches only exact ties; a gap of k ULPs ($k = 2$ shown) spans the same number of representable values at any magnitude.

remains are token-mismatch forks, and the window sets how many: 235, 148 and 90 at $w = 8$, 4 and 2. We use $w = 4$, the largest window at which position-mismatch forks already reach that level (Appendix Figure 11), at 2.02× the BF16 cost, and leave the token-mismatch forks to the verifier of Section 6.2. Since most forks come early (Section 4.2), computing only the first steps with LayerCast might seem enough. It is not: LayerCast for the first 20% of steps still leaves 316/500 outputs divergent at 1.82×, because BF16 steps later in decoding fork as well (Appendix E.7). The refresh therefore has to recur throughout decoding. Dream behaves the same way. Freezing with $w = 4$ reduces its position-mismatch forks from 425 to 3 on MATH500 and from 148 to 1 on HumanEval, leaving 405 and 139 token-mismatch forks (Appendix E.3).

6.2 Verifying tokens by ULPs removes most of the rest

With the order frozen, the remaining risk is the token decision at each frozen position. We recompute that decision only when it is close to a tie. Closeness has to be judged relative to the spacing of the execution dtype, because that spacing grows with the magnitude of the logits: BF16 keeps eight significant bits, so two distinct logits between 16 and 32 differ by at least $2^{-3} = 0.125$, while two between 1 and 2 can differ by 2^{-7} (Figure 7). Let $z_1 \geq z_2$ be the top two logits at the position about to be committed, computed in the execution dtype d, and $g = z_1 - z_2$ their gap. Let u_d be one ULP of d at $\max(|z_1|, |z_2|)$, the distance from that value to the next number d can represent; we recompute when

$$g < k\, u_d, \qquad u_d = 2^{\lfloor \log_2 \max(|z_1|,|z_2|) \rfloor - m_d}, \tag{4}$$

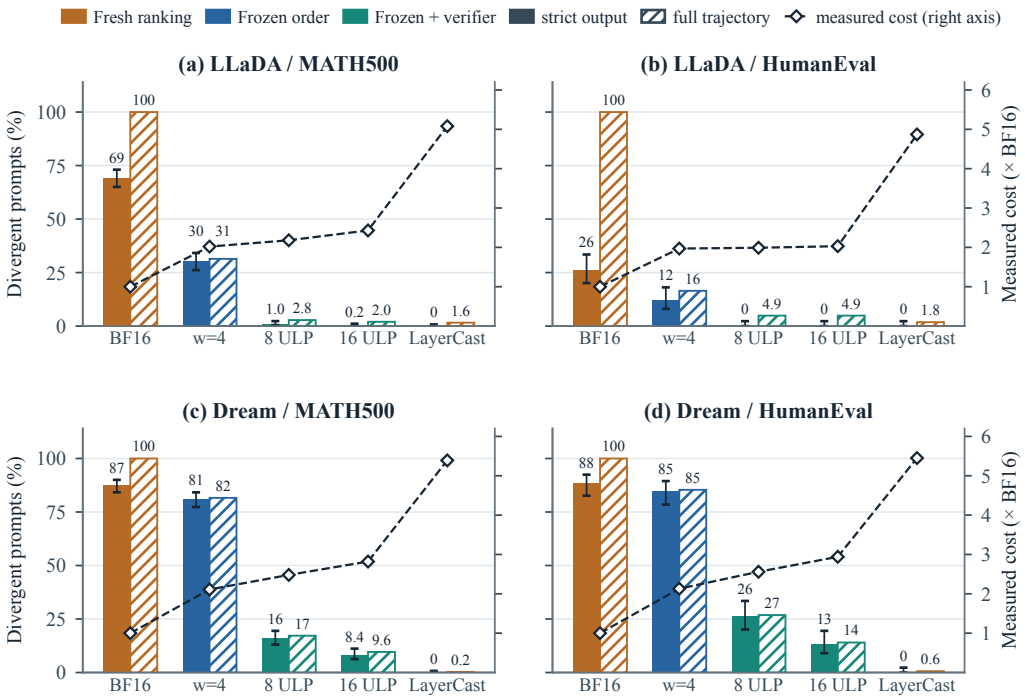

Figure 8: Batch $1 \leftrightarrow 4$ divergence on the RTX PRO 6000 under each mitigation policy. Solid bars: strict output divergence (95% Wilson intervals); hatched bars: trajectory divergence. Dashed line (right axis): measured per-sample cost relative to BF16 (Appendix E.6).

where m_d is the number of fraction bits of d (7 for BF16, 10 for FP16, 23 for FP32). A trigger runs one extra LayerCast forward for that example and replaces only the token at the frozen position; the position stays fixed. The threshold k counts spacings, so it measures how close the decision is to a tie on the dtype's grid, not how large the accumulated numerical error is; exact ties between nonzero logits always trigger. For Dream, the top two logits are taken after EOS suppression, over the tokens the decoder can actually commit (Appendix A).

With $w = 4$ and $k = 8$, strict output divergence on the primary LLaDA MATH500 pair falls from 69.2% to 1.0% (346 to 5 of 500) at 2.18× the BF16 cost, against 5.08× for LayerCast (Figure 8; costs are measured on the A6000, Appendix E.6). The sweep over k (Appendix Figure 12) shows that this holds in all three environments and that the threshold trades cost for agreement smoothly: the number of divergent outputs falls from 74–78 of the 500 prompts at $k = 1$ to 3–5 at $k = 8$ and 0–1 at $k = 16$. Position-mismatch forks stay at 6–9 for every k, the same level as under LayerCast, so what the verifier leaves is not a token decision it misses but the position forks that LayerCast itself does not remove. We chose $k = 8$ post hoc from this MATH500 sweep and kept it fixed for every other setting.

Tuned on LLaDA MATH500, the recipe carries over to another task and to another model without retuning. It makes LLaDA's HumanEval outputs identical across batch sizes in all three environments (26.2% divergent in BF16, 0% with the recipe) at 1.99×. On Dream it reduces output divergence from 87.4% to 16.0% at $k = 8$ and 8.4% at $k = 16$ on MATH500, and from 88.4% to 26.2% and 13.4% on HumanEval, at 2.5–2.9× the BF16 cost, about half the 5.4–5.5× of LayerCast. More of Dream's divergence remains because its token decisions fork more often: with the order frozen but no verifier, a Dream step produces a token-mismatch fork about six times as often as a LLaDA step (Appendix E.3). Section 5 points the same way: at the last shared state, Dream's two executions already predict different tokens at 82 still-masked positions per fork, against 12 for LLaDA. A larger k removes more of it, as the step from $k = 8$ to $k = 16$ shows.

6.3 FREEZING BUYS AGREEMENT, NOT LAYERCAST'S OUTPUTS

Freezing makes the executions agree with each other, but not with the outputs of LayerCast, because it changes the decoding rule: for $w - 1$ steps the order is not re-ranked while the context changes. On MATH500, the two batch sizes now produce the same output for 495 of the 500 prompts, yet only 61 of those outputs are the one LayerCast produces; the other 434 are a different output on which both batch sizes agree. On HumanEval all 164 prompts agree across batch sizes, but only 32 of them on the output of LayerCast (Appendix E.5). Accuracy can therefore move in either direction. LLaDA gains slightly, from 22.8% to 23.6% on MATH500 and from 23.2% to 27.4% on HumanEval, while Dream loses on MATH500, from 43.8% to 40.0% with the verifier. LayerCast keeps Dream at 44.4%, so the loss comes from the frozen order rather than from the FP32 arithmetic (Appendix Table 19 and Figure 13). The recipe guarantees agreement across executions; its effect on accuracy has to be checked on each task.

7 DISCUSSION

How MDLM nondeterminism differs from the autoregressive case. In an autoregressive LLM, nondeterminism enters through one decision per token, and remedies either remove the numerical differences, as batch-invariant kernels and LayerCast do, or recheck near-tied token decisions, as MarginGate does (Section 2). An MDLM adds a second decision, where to write, and in LLaDA executions first split there. Unlike a token difference in an autoregressive model, such a split is usually temporary: it becomes permanent only if a token conflict follows (Section 5). LayerCast fixes both decisions but costs about five times the BF16 step. Selective rechecking, the cheap alternative developed for autoregressive models, covers only token decisions, and moving the margin gate to positions does not help LLaDA, where near-ties among positions are the norm (Section 6.1). The recipe therefore splits the work: it fixes the position order for w steps with a LayerCast ranking and rechecks only the few token decisions that are near a tie.

Limitations. The recipe is weakest where token decisions are noisy: on Dream it leaves part of the divergence even at the largest threshold we tested (Section 6.2), and because freezing changes the decoding rule, it can also cost accuracy (Section 6.3). The evidence is also narrow. We chose $k = 8$ after seeing the MATH500 sweep, and we cover two models with BF16 weights, mainly one decoding schedule (Appendix C.6 adds two), Dream only with its EOS logit suppressed and without FP16, and one run per execution setting. Two comparisons mix several causes. Moving to another GPU changes the hardware, the driver and the kernels at once, so we cannot say which of them causes a divergence. A refresh likewise does two things, computing the ranking in FP32 and reusing it for w steps, and we did not test them separately, so we cannot say how much of its effect on agreement or accuracy comes from each. Finally, our costs are the time per sample on one GPU model (Appendix E.6); they do not show throughput or memory use in a serving system.

8 CONCLUSION

Nondeterminism in masked diffusion LMs enters mainly through the position decision: executions that differ only in batch size or GPU usually first commit at different positions. These position mismatches often reunite, and outputs differ only once a token conflict follows. Mitigating nondeterminism at each decision separately, by freezing the position order under periodic LayerCast refresh and rechecking only near-tied tokens, reduces LLaDA's MATH500 output divergence from 69.2% to 1.0%, close to LayerCast, at less than half its cost. Future work includes extending the analysis to parallel samplers, which also decide how many tokens to commit per step and so add a third decision that may fork, and developing remedies that secure agreement across executions without changing the decoding rule, so that accuracy is preserved.

AI USE STATEMENT

Generative AI tools assisted with methodological feedback, code for experiments and analysis, mathematical and data analysis, literature review, and manuscript preparation. The authors take full responsibility for all content and conclusions.

ETHICS STATEMENT

This study evaluates publicly released models and benchmarks and involves no human participants or private participant data. Asset provenance is discussed in Appendix B. Section 7 states the limits of our evidence, including the two models and mainly one decoding schedule we study, and the cost of the proposed intervention. Agreement across execution conditions is not evidence of correctness, and we report explicitly that Dream is run with its EOS logit suppressed and without FP16.

REFERENCES

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/7ede97c3e082c6df10a8d6103a2eebd2-Paper-Conference.pdf.

Hikaru Asano, Tadashi Kozuno, Kuniaki Saito, and Yukino Baba. Where-to-unmask: Ground-truth-guided unmasking order learning for masked diffusion language models, 2026. URL https://arxiv.org/abs/2602.09501.

Berk Atıl, Sarp Aykent, Alexa Chittams, Lisheng Fu, Rebecca J. Passonneau, Evan Radcliffe, Guru Rajan Rajagopal, Adam Sloan, Tomasz Tudrej, Ferhan Ture, Zhe Wu, Lixinyu Xu, and Breck Baldwin. Non-determinism of "deterministic" LLM system settings in hosted environments. In Mousumi Akter, Tahiya Chowdhury, Steffen Eger, Christoph Leiter, Juri Opitz, and Erion Çano (eds.), *Proceedings of the 5th Workshop on Evaluation and Comparison of NLP Systems*, pp. 135–148, Mumbai, India, December 2025. Association for Computational Linguistics. ISBN 979-8-89176-305-0. doi: 10.18653/v1/2025.eval4nlp-1.12. URL https://aclanthology.org/2025.eval4nlp-1.12/.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas S Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 55981–56007. Curran Associates, Inc., 2025. doi: 10.52202/085713-1874. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/510e22f4a2da5212a64f1591736e2eaf-Paper-Conference.pdf.

Pierre Blanchard, Desmond J Higham, and Nicholas J Higham. Accurately computing the log-sum-exp and softmax functions. *IMA Journal of Numerical Analysis*, 41(4):2311–2330, 10 2021. ISSN 0272-4979. doi: 10.1093/imanum/draa038. URL https://doi.org/10.1093/imanum/draa038.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. Maskgit: Masked generative image transformer. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11305–11315, 2022. doi: 10.1109/CVPR52688.2022.01103.

Kecheng Chen, Ziru Liu, Xijia Tao, Hui Liu, Xinyu Fu, Suiyun Zhang, Dandan Tu, Lingpeng Kong, Rui Liu, and Haoliang Li. Beyond confidence: Adaptive and coherent decoding for diffusion language models, 2025a. URL https://arxiv.org/abs/2512.02044.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Sitan Chen, Kevin Cong, and Jerry Li. Optimal inference schedules for masked diffusion models, 2025b. URL https://arxiv.org/abs/2511.04647.

Yuzhe Chen, Jiale Cao, Xuyang Liu, Jin Xie, Aiping Yang, and Yanwei Pang. Stdec: Spatio-temporal stability guided decoding for dllms, 2026. URL https://arxiv.org/abs/2604.06330.

Kexin Chu, Yang Zhou, and Wei Zhang. Margingate: Sparse margin-triggered verification for batch-invariant llm inference, 2026. URL https://arxiv.org/abs/2605.30218.

James Demmel and Hong Diep Nguyen. Fast reproducible floating-point summation. In *2013 IEEE 21st Symposium on Computer Arithmetic*, pp. 163–172, 2013. doi: 10.1109/ARITH.2013.9.

Abhinav Dutta, Sanjeev Krishnan, Nipun Kwatra, and Ramachandran Ramjee. Accuracy is not all you need. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 124347–124390. Curran Associates, Inc., 2024. doi: 10.52202/079017-3950. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/e0e956681b04ac126679e8c7dd706b2e-Paper-Conference.pdf.

Zhengyu Fang, Zhimeng Jiang, Huiyuan Chen, Xiaoge Zhang, Tianyi Li, Kaiyu Tang, Xiao Li, and Jing Li. Dataset-level metrics attenuate non-determinism: A fine-grained non-determinism evaluation in diffusion language models, 2026. URL https://arxiv.org/abs/2604.13413.

Stipe Frkovic, Metod Jazbec, Dan Zhang, Christian A. Naesseth, Ilija Bogunovic, and Eric Nalisnick. Re-evaluating confidence remasking in masked diffusion language models, 2026. URL https://arxiv.org/abs/2606.12232.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *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, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1633. URL https://aclanthology.org/D19-1633/.

David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991. doi: 10.1145/103162.103163.

Raja Gond, Aditya K Kamath, Ramachandran Ramjee, and Ashish Panwar. Llm-42: Enabling determinism in llm inference with verified speculation, 2026. URL https://arxiv.org/abs/2601.17768.

Aarav Gupta, Gururaj Deshpande, and Chandreyi Chakraborty. On the quantization robustness of diffusion language models in coding benchmarks, 2026. URL https://arxiv.org/abs/2604.20079.

Horace He and Thinking Machines Lab. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250910. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

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 J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

Zemin Huang, Yuhang Wang, Zhiyang Chen, and Guo-Jun Qi. Don't settle too early: Self-reflective remasking for diffusion language models, 2025. URL https://arxiv.org/abs/2509.23653.

Yuki Ichihara, Naoto Iwase, Mohammad Atif Quamar, and Junpei Komiyama. Prefix-denoising consistency: Test-time verification for diffusion language models, 2026. URL https://arxiv.org/abs/2608.25311.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 52870–52888. Curran Associates, Inc., 2025. doi: 10.52202/085713-1763. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4c7d31b2da05a17f15101db5b37a1e24-Paper-Conference.pdf.

Bumjun Kim, Dongjae Jeon, Dueun Kim, Wonje Jeung, and Albert No. Rainbow padding: Mitigating early termination in instruction-tuned diffusion llms, 2025a. URL https://arxiv.org/abs/2510.03680.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: understanding token ordering in masked diffusions. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *PMLR*, pp. 30749–30768. PMLR, 2025b. doi: 10.5555/3780338.3781545. URL https://dl.acm.org/doi/10.5555/3780338.3781545.

Zakhar Kohut, Severyn Shykula, Mykola Vysotskyi, Serhii Dmytryshyn, Dmytro Khamula, Michal Zakrzewski, Damian Rynczak, Jacek Małecki, Taras Rumezhak, and Volodymyr Karpiv. Just on time: Token-level early stopping for diffusion language models, 2026. URL https://arxiv.org/abs/2602.11133.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

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*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32819–32848. PMLR, 2024. URL https://proceedings.mlr.press/v235/lou24a.html.

Omer Luxembourg, Haim Permuter, and Eliya Nachmani. Plan for speed: Dilated scheduling for masked diffusion language models, 2026. URL https://arxiv.org/abs/2506.19037.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 149009–149033. Curran Associates, Inc., 2025. doi: 10.52202/085713-4984. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/db0ee27cb50dd9087b133f6e7d28a90e-Paper-Conference.pdf.

Amr Mohamed, Guokan Shang, and Michalis Vazirgiannis. Less is more: Mutual-stability sampling for diffusion language models, 2026. URL https://arxiv.org/abs/2606.16908.

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 D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 50608–50646. Curran Associates, Inc., 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/48b383b24230e0e6e649d9c98dae4d8c-Paper-Conference.pdf.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 64972–65009, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a365e37c18fb91af547a2f0012a89e98-Paper-Conference.pdf.

Han Peng, Peiyu Liu, Zican Dong, Daixuan Cheng, Junyi Li, Yiru Tang, Shuo Wang, and Wayne Xin Zhao. How efficient are diffusion language models? a critical examination of efficiency evaluation practices, 2025. URL https://arxiv.org/abs/2510.18480.

Shahzeb Qamar, Lorenz Sparrenberg, Christian Bauckhage, Baha Rababah, Carson Leung, Murat Kantarcioglu, Cuneyt Gurcan Akcora, and Rafet Sifa. Accuracy is not enough: A divergence-based approach to evaluate fidelity loss in quantized llms, 2026. URL https://arxiv.org/abs/2609.07664.

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 A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184. Curran Associates, Inc., 2024. doi: 10.52202/079017-4135. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/eb0b13cc515724ab8015bc978fdde0ad-Paper-Conference.pdf.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 103131–103167. Curran Associates, Inc., 2024. doi: 10.52202/079017-3277. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/bad233b9849f019aead5e5cc60cef70f-Paper-Conference.pdf.

Yifan Song, Guoyin Wang, Sujian Li, and Bill Yuchen Lin. The good, the bad, and the greedy: Evaluation of LLMs should not ignore non-determinism. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4195–4206, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.211. URL https://aclanthology.org/2025.naacl-long.211/.

Lipeng Wan, Jianhui Gu, Junjie Ma, Jianguo Huang, Shiguang Sun, Siyuan Li, and Xuguang Lan. Progressive refinement regulation for accelerating diffusion language model decoding, 2026. URL https://arxiv.org/abs/2603.04514.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling, 2026a. URL https://arxiv.org/abs/2503.00307.

Xinyun Wang, Min Zhang, Sen Cui, Zhikang Chen, Bo Jiang, Kun Kuang, and Mingbao Lin. Reversible diffusion decoding for diffusion language models, 2026b. URL https://arxiv.org/abs/2602.00150.

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 C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 57027–57051, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/5d8d4e6061c3ba96c240b7fa1ae3471d-Paper-Conference.pdf.

Yue Wu and Jian Huang. Stability-weighted decoding for diffusion language models, 2026. URL https://arxiv.org/abs/2604.17068.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models, 2025. URL https://arxiv.org/abs/2508.15487.

Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in llm inference. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 169819–169851. Curran Associates, Inc., 2025. doi: 10.52202/085713-5653. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/f80094a824ba5912d4a2de169c404a40-Paper-Conference.pdf.

Zhenting Zhu, Lucas Thai, Shan Yu, Yicheng Liu, Yifan Qiao, Chenxi Wang, Harry Xu, and Junyi Shu. Demystifying numerical instability in llm inference: Achieving reproducible inference for mission-critical tasks with heal, 2026. URL https://arxiv.org/abs/2606.21023.

A Measurement and Verification Details

Paired agreement and quality. For prompt i under configurations a, b, let T_i^a, T_i^b be the complete logged commitment traces and y_i^a, y_i^b their final 512-token outputs. Write $q_i^a, q_i^b \in \{0, 1\}$ for their deterministic correctness labels. The prompt-normalized divergence rates are

$$D_{\text{traj}} = n^{-1} \sum_i \mathbf{1}[T_i^a \neq T_i^b], \qquad D_{\text{out}} = n^{-1} \sum_i \mathbf{1}[y_i^a \neq y_i^b], \tag{5}$$

$$D_{\text{corr}} = n^{-1} \sum_i \mathbf{1}[q_i^a \neq q_i^b]. \tag{6}$$

For identical initial requests, complete traces determine the final output; identical outputs determine the score under the same deterministic scorer. Consequently,

$$|\operatorname{Acc}_a - \operatorname{Acc}_b| \leq D_{\text{corr}} \leq D_{\text{out}} \leq D_{\text{traj}}. \tag{7}$$

The first inequality can be strict because gains and losses cancel. The last can be strict because different commitment paths re-converge. These inequalities concern full traces and full outputs; post-hoc final-region projections are separately defined in Appendix C.5.

The quality inequalities above require the same deterministic scoring function of the complete scorer input and gold/tests. All reported accuracies satisfy this condition because every output was re-scored once with one scorer (Appendix B). Effective-prefix equality is weaker than equality of the full decoded string consumed by the scorer.

Scope of the mismatch taxonomy. The position-mismatch and token-mismatch labels of Section 3 observe commitments only. They neither imply equality of the token argmaxes $(\hat{v}_{t,p})_{p \in M_t}$ at unselected positions nor locate the earliest changed internal tensor. For Dream's final step, which commits two tokens, a position mismatch means that the selected position sets differ.

An elementary sufficient condition for argmax stability. Let a be a vector of scores, $j = \arg\max_v a_v$ be unique, $\Delta = a_j - \max_{v \neq j} a_v$, and $\|a' - a\|_\infty \leq \epsilon$. For every competitor $v \neq j$,

$$a'_j - a'_v \geq a_j - a_v - 2\epsilon \geq \Delta - 2\epsilon.$$

Thus $\Delta > 2\epsilon$ preserves the winner. The condition is sufficient, not necessary, and gives no guarantee for ties. Applying it to token logits or position confidences requires a perturbation bound at the same shared state over the same eligible candidate set. Our logged margins do not provide such a bound. After a commitment fork, states themselves may differ, so the argument is not a certificate for the rest of a decoding trajectory.

ULP domain and interpretation. For finite, nonzero normal magnitude $m = \max(|z_1|, |z_2|)$, write $m = f2^e$ with $f \in [1/2, 1)$. BF16 has seven explicit fraction bits, so the spacing in this binade is $2^{e-8} = 2^{\lfloor \log_2 m \rfloor - 7}$, as used in Equation 4. The two logits usually share a binade; when they straddle a power of two, the spacing at the larger magnitude is the coarser one. Rounding each logit moves it by at most half its own spacing, so rounding alone changes the gap by at most that coarser spacing, and the gate compares the gap with k of them. The implementation uses the strict inequality of Equation 4: a gap of exactly k spacings does not trigger. This is a representation-relative heuristic, not a forward-error certificate or a probability calibration. The stated formula does not specify a general-purpose policy for zero, subnormal, or nonfinite logits; such cases require an explicit implementation convention.

The gate is not invariant to a common additive shift. For example, the logits $(1, 1 - 2^{-7})$ have gap 2^{-7} and spacing 2^{-7}, so the strict $k = 1$ test does not trigger. Adding one to both logits preserves their exact softmax and argmax, but changes the larger magnitude's binade: spacing becomes 2^{-6} and the same gap now triggers. Thus applying one threshold across tasks tests an empirical operating point for the tested models, not a model-independent notion of uncertainty.

Order of constraint application. A verifier intended to assess the committed decision must compute its top two candidates after applying the same eligibility restrictions as the final argmax. A gate computed on raw Dream logits, before EOS is suppressed, violates this. Because unsuppressed

Dream ranks EOS first at most masked positions, such a gate compares EOS with the best eligible token. Replaying the per-step gaps of such raw-logit runs against the post-suppression top-two probabilities shows two consequences. When EOS was the runner-up, the gate saw a narrower gap than the committed decision and fired spuriously (2.5–4.3% of its fires). When EOS was first and far above the eligible candidates, the gate saw a wider gap and stayed silent although the eligible gap was below $k\,u$; such steps are 0.2–0.4% of the gated steps, yet 74–92% of the residual first token mismatches occur at them, under both position scorers and on both tasks (Table 18). All Dream verifier rows in this paper therefore apply the suppression before the gate; raw-logit rows appear only in Table 17, for comparison. LLaDA is unaffected because its decoder suppresses nothing.

B Protocol and Execution Environments

Table 1: Execution environments used for the experiments. Reported memory uses the product's nominal capacity.

GPU	memory	capability	driver	confidence block
RTX PRO 6000 Blackwell Max-Q	96 GB	12.0	580.82.09	1024
RTX A6000	48 GB	8.6	580.159.04	512
GeForce RTX 3090	24 GB	8.6	580.173.02	512

All environments use Python 3.11.15, PyTorch 2.12.1+`cu132`, CUDA 13.2, and Transformers 4.49.0. The confidence block partitions the sequence dimension during LLaDA's confidence softmax to fit GPU memory; Dream does not use this block parameter. These three settings in Table 1 therefore form part of the execution environment rather than an isolated hardware comparison.

Decoding. All runs use seed 1234 and temperature zero, with no training or seed averaging. LLaDA follows its released low-confidence sampler: it makes one immutable commitment per step, EOS remains eligible, and all 512 updates are retained. Dream uses the entropy position scorer of its released evaluation code. Greedy zero-shot probes of Dream largely collapsed to EOS, so its EOS logit is set to $-\infty$ before the token and position decisions and fixed-length outputs are scored. Dream receives 512 masks appended to each left-padded prompt, which fixes its reveal schedule across batches; a maximum-probability position scorer, used as a controlled stress test, is reported only in Appendix E.4. Under this protocol, FP32 accuracy is 44.4% on MATH500 and 59.1% on HumanEval for Dream and 23.4% on MATH500 for LLaDA; LLaDA's released semi-autoregressive schedule roughly doubles its MATH500 accuracy without changing the divergence structure (Appendix C.6). TF32 is disabled and FP32 matmul precision is `highest`. Confidence computation and ranking use the stated dtype, with runtime `topk` ties and no canonical rule.

Compared configurations. RTX PRO 6000 comparisons cover both models at BF16, FP16 and FP32 (Dream FP16 excluded; see below); LLaDA additionally uses batch 16 (MATH500 on the RTX PRO 6000 and A6000, HumanEval on the A6000), and A6000 and 3090 comparisons are included where available (Appendix C.1). Environment changes combine hardware, drivers and kernels, preventing causal GPU attribution. Re-running the first batch of a configuration reproduces its tokens exactly, and turning logging off does not change them; the instrumented samplers otherwise match the released code, except for the disclosed Dream EOS suppression.

Divergence measures and statistics. Strict output divergence compares full 512-token tuples; effective divergence truncates before EOS/pad. Trajectory divergence compares ordered per-step position–token mappings. Final-region projection retains only pre-EOS/pad positions; it is not causally closed because discarded positions influenced the bidirectional state. Section 5 defines reunions and token conflicts, and Appendix D gives the formal definitions and describes the response-state replay that validates the step-resolved analyses. Steps are zero-indexed; response-relative positions remove padding offsets. Intervals are Wilson prompt intervals; bootstrap intervals resample prompts within benchmark, retaining all comparisons of each prompt (10,000 draws unless stated otherwise).

LayerCast and native FP32. The released weights are BF16, so native FP32 inference upcasts them without changing their values, and LayerCast, which keeps them in BF16 and computes in FP32, performs the same arithmetic. The two are token-identical in every configuration where we ran both: LLaDA and Dream on MATH500 on the RTX PRO 6000 and LLaDA on HumanEval on the A6000, at batch one and four. Where a native FP32 run is missing, including the RTX 3090, whose 24 GB cannot hold FP32 weights, the pair census uses the LayerCast run as FP32.

Prompts and batching. Samples remain in source-file order without shuffling. After applying the model's chat template, batching takes contiguous chunks of that order and left-pads each batch; the runner uses the tokenizer pad ID, falling back to EOS. Thus changing batch size deterministically changes batchmates and padding. For math tasks, the user message requests a final answer inside `\boxed{}` and appends the problem verbatim. It is passed as one user turn through the model's chat template with a generation prompt. HumanEval requests a complete function inside a Python code block. Dream uses the same prompts.

Scoring. MATH500 is scored with `math-verify` 0.9.0 with boxed-answer priority and first-match extraction; gold answers permit LaTeX or expression extraction. The HumanEval scorer takes the first fenced block that defines the entry point (falling back to the first block), accepts that function or a bare continuation, appends the official tests, runs them in a fresh process with a 10-second timeout, and counts timeouts and crashes as failures. Every configuration reported in this paper is scored with this one scorer, the *unified scorer*, after generation. The scorer consumes the full decoded string, not the effective token prefix. No Dream output contains an EOS or EOT token, so Dream's strict and effective outputs are identical.

Dream in FP16. Dream's released evaluation code scores positions by $\sum_v q_v \log(q_v + 10^{-10})$ with $q = \sigma(z_{t,p})$, in the execution dtype. In FP16 the constant 10^{-10} rounds to zero, every probability that underflows contributes $0 \cdot \log 0$, and every position score is NaN. `topk` over an all-NaN vector returns the lowest indices, so every FP16 run with the entropy scorer, baseline or verifier-only, commits positions 0 to 509 in order and the final two together: the position decision never takes place. The maximum-probability scorer of Appendix E.4 takes no logarithm and is unaffected. The FP16 entropy runs are therefore excluded from the pair census and from Figures 2 and 3.

Asset provenance. The LLaDA model card and HumanEval repository identify MIT terms, while the Dream model card identifies Apache-2.0 terms. The MATH500 mirror does not name a license, while the upstream MATH and PRM800K repositories carry MIT notices. We accessed these assets from their public distributors and do not redistribute weights, prompts, answers, generations, or tests.

C Baseline Divergence Details

C.1 Baseline pair census

Tables 2 and 3 list every pair of baseline runs behind Figures 2 and 3: for each benchmark and precision, all pairs that differ in batch size on one GPU (batch axis) or in GPU at one batch size (GPU axis); runs that differ in both are not paired. Batch 16 exists for LLaDA only, and LLaDA MATH500 is not compared on the RTX 3090, which fixes the pair counts. The two Ampere GPUs (A6000 and 3090) agree with each other far more often than either agrees with the RTX PRO 6000 or with a different batch size on the same GPU; these Ampere pairs set the lower ends of the whiskers in Figure 2. For LLaDA HumanEval in BF16, the two Ampere GPUs differ on 1/164 and 2/164 outputs (batch one and four), against 28–53/164 for every other pair; for Dream MATH500 in BF16 they differ on 126/500 and 276/500, against 424–441/500. They run different drivers (Appendix B), so their agreement is consistent with architecture-dependent kernel selection rather than identical software; because a GPU change also changes hardware, driver and kernels together, our design cannot attribute it causally.

The shares reported in Section 4.2 carry benchmark-stratified prompt-cluster bootstrap 95% intervals (10,000 draws; all comparisons of a resampled prompt are kept together). The position share of forks is 99.88% $[99.77, 99.96]$ for LLaDA and 85.7% $[83.8, 87.5]$ for Dream, and the re-convergence share of position-mismatch forks is 60.4% $[58.4, 62.5]$ for LLaDA and 14.4% $[12.7, 16.3]$ for Dream.

Table 2: LLaDA baseline pair census. Batch-axis rows compare two batch sizes on one GPU; GPU-axis rows compare two GPUs at one batch size (same precision throughout). Counts are prompts out of n (500 for MATH500, 164 for HumanEval): strict-output / full-trajectory divergence, the channel of the first fork, and re-converged trajectories (position-mismatch forks whose strict outputs still agree; no token-mismatch fork re-converges).

benchmark	precision	GPU	batch	out / traj	first fork pos / tok	re-conv.
MATH500	BF16	PRO 6000	1↔4	346 / 500	499 / 1	154
		PRO 6000	1↔16	359 / 500	499 / 1	141
		PRO 6000	4↔16	255 / 372	372 / 0	117
		A6000	1↔4	343 / 500	500 / 0	157
		A6000	1↔16	354 / 500	500 / 0	146
		A6000	4↔16	260 / 384	384 / 0	124
		PRO 6000↔A6000	1	339 / 500	498 / 2	161
		PRO 6000↔A6000	4	320 / 500	499 / 1	180
		PRO 6000↔A6000	16	331 / 500	499 / 1	169
MATH500	FP16	PRO 6000	1↔4	99 / 465	465 / 0	366
		PRO 6000	1↔16	100 / 471	471 / 0	371
		PRO 6000	4↔16	85 / 346	345 / 1	261
		A6000	1↔4	104 / 483	482 / 1	379
		A6000	1↔16	102 / 480	480 / 0	378
		A6000	4↔16	98 / 442	441 / 1	344
		PRO 6000↔A6000	1	90 / 470	470 / 0	380
		PRO 6000↔A6000	4	98 / 460	459 / 1	362
		PRO 6000↔A6000	16	98 / 457	454 / 3	359
MATH500	FP32	PRO 6000	1↔4	0 / 8	8 / 0	8
		PRO 6000	1↔16	1 / 8	8 / 0	7
		PRO 6000	4↔16	1 / 6	6 / 0	5
		A6000	1↔4	0 / 6	6 / 0	6
		PRO 6000↔A6000	1	0 / 3	3 / 0	3
		PRO 6000↔A6000	4	0 / 3	3 / 0	3
HumanEval	BF16	PRO 6000	1↔4	43 / 164	164 / 0	121
		A6000	1↔4	53 / 164	164 / 0	111
		A6000	1↔16	50 / 164	164 / 0	114
		A6000	4↔16	28 / 120	120 / 0	92
		3090	1↔4	52 / 164	164 / 0	112
		PRO 6000↔A6000	1	47 / 164	164 / 0	117
		PRO 6000↔3090	1	47 / 164	164 / 0	117
		A6000↔3090	1	1 / 1	1 / 0	0
		PRO 6000↔A6000	4	46 / 164	164 / 0	118
		PRO 6000↔3090	4	47 / 164	164 / 0	117
		A6000↔3090	4	2 / 4	4 / 0	2
HumanEval	FP16	PRO 6000	1↔4	10 / 152	152 / 0	142
		A6000	1↔4	12 / 161	161 / 0	149
		A6000	1↔16	14 / 161	161 / 0	147
		A6000	4↔16	6 / 148	148 / 0	142
		PRO 6000↔A6000	1	14 / 156	156 / 0	142
		PRO 6000↔A6000	4	7 / 154	154 / 0	147
HumanEval	FP32	PRO 6000	1↔4	0 / 3	3 / 0	3
		A6000	1↔4	0 / 3	3 / 0	3
		3090	1↔4	0 / 4	4 / 0	4
		PRO 6000↔A6000	1	0 / 0	0 / 0	0
		PRO 6000↔3090	1	0 / 0	0 / 0	0
		A6000↔3090	1	0 / 0	0 / 0	0
		PRO 6000↔A6000	4	0 / 0	0 / 0	0
		PRO 6000↔3090	4	0 / 1	1 / 0	1
		A6000↔3090	4	0 / 1	1 / 0	1

C.2 CANDIDATE-TOKEN IDENTITIES AT THE LAST SHARED STATE

Table 4 expands the post-hoc check in Section 4.2 over every position-mismatch fork of the pair census, using the logged per-step candidate maps. "Any candidate" compares the full vector of candidate-token argmax identities over still-masked positions at the last shared state; "selected candidate" tests whether the identity differs at either of the two positions actually selected. The latter is a subset of the former. The masked-position sets agree for every event.

Table 3: Dream baseline pair census. Batch-axis rows compare two batch sizes on one GPU; GPU-axis rows compare two GPUs at one batch size (same precision throughout). Counts are prompts out of n (500 for MATH500, 164 for HumanEval): strict-output / full-trajectory divergence, the channel of the first fork, and re-converged trajectories (position-mismatch forks whose strict outputs still agree; no token-mismatch fork re-converges).

benchmark	precision	GPU	batch	out / traj	first fork pos / tok	re-conv.
MATH500	BF16	PRO 6000	1↔4	437 / 500	425 / 75	63
		A6000	1↔4	439 / 500	416 / 84	61
		3090	1↔4	441 / 500	416 / 84	59
		PRO 6000↔A6000	1	441 / 500	423 / 77	59
		PRO 6000↔3090	1	440 / 500	422 / 78	60
		A6000↔3090	1	126 / 150	128 / 22	24
		PRO 6000↔A6000	4	425 / 500	416 / 84	75
		PRO 6000↔3090	4	424 / 500	415 / 85	76
		A6000↔3090	4	276 / 324	276 / 48	48
MATH500	FP32	PRO 6000	1↔4	0 / 1	1 / 0	1
		A6000	1↔4	0 / 1	1 / 0	1
		PRO 6000↔A6000	1	0 / 2	2 / 0	2
		PRO 6000↔A6000	4	0 / 2	2 / 0	2
HumanEval	BF16	PRO 6000	1↔4	145 / 164	148 / 16	19
		A6000	1↔4	150 / 164	151 / 13	14
		3090	1↔4	154 / 164	147 / 17	10
		PRO 6000↔A6000	1	143 / 164	153 / 11	21
		PRO 6000↔3090	1	146 / 164	150 / 14	18
		A6000↔3090	1	111 / 118	103 / 15	7
		PRO 6000↔A6000	4	149 / 164	148 / 16	15
		PRO 6000↔3090	4	152 / 164	147 / 17	12
		A6000↔3090	4	50 / 56	53 / 3	6
HumanEval	FP32	PRO 6000	1↔4	0 / 1	1 / 0	1
		A6000	1↔4	0 / 0	0 / 0	0
		3090	1↔4	0 / 0	0 / 0	0
		PRO 6000↔A6000	1	0 / 1	1 / 0	1
		PRO 6000↔3090	1	0 / 1	1 / 0	1
		A6000↔3090	1	0 / 0	0 / 0	0
		PRO 6000↔A6000	4	0 / 0	0 / 0	0
		PRO 6000↔3090	4	0 / 0	0 / 0	0
		A6000↔3090	4	0 / 0	0 / 0	0

Table 4: Candidate-token identities at the last shared state of every position-mismatch fork in the pair census. "any candidate": at least one still-masked position has a different candidate argmax in the two runs; "selected candidate": the candidate differs at either position actually selected; percentages condition on the events in each row.

model	precision	events	any candidate	selected candidate	mean differing identities/event
LLaDA	BF16	5,687	97.59%	0.12%	12.32
	FP16	4,999	62.45%	0.00%	1.56
	FP32	46	0.00%	0.00%	0.00
	all	10,732	80.81%	0.07%	7.25
Dream	BF16	4,537	100.00%	2.76%	82.19
	FP32	9	0.00%	0.00%	0.00
	all	4,546	99.80%	2.75%	82.02

C.3 When the first fork occurs

Table 5 summarizes when the first trajectory mismatch occurs over the pair census.

Table 5: First-fork timing over the pair census. $A = t^*/512$ is the normalized shared prefix before the first trajectory mismatch at zero-indexed step t^*; exact trajectory agreement is assigned $A = 1$ and excluded from the divergent-only columns. Prompts recur across pairs, so the columns describe distributions rather than independent trials.

comparison	exact trajectory	median A [IQR]	mean A	within 1%	within 5%
LLaDA, all pairs	5,519/16,264	4.69% [1.37, 14.65]	12.35%	22.1%	51.5%
LLaDA, primary BF16 b=1–4	–	2.25%	4.09%	–	–
Dream, all pairs	4,147/9,452	3.32% [1.37, 6.84]	4.91%	20.2%	64.5%
Dream, primary BF16 b=1–4	–	3.52%	5.20%	–	–

C.4 Position margins and ties

Aggregate probability gaps. Table 6 uses all 500 MATH500 batch-one trajectories on the RTX PRO 6000. Each prompt contributes 511 position margins (there is no runner-up at the last step) and 512 committed-token gaps.

Table 6: Aggregate probability gaps at batch size one on MATH500. Position statistics contain 255,500 decisions per precision; token statistics contain 256,000. The two gap types are subject to different selection effects.

	Position margin			Committed-token gap		
compute	median	exactly 0	$< 10^{-3}$	median	exactly 0	$< 10^{-3}$
BF16	0.00391	42.08%	42.08%	0.99850	0.050%	0.050%
FP16	0.00586	24.70%	36.58%	0.99640	0.007%	0.007%
FP32	0.00603	1.50%	33.44%	0.99631	0.000%	0.001%

In BF16, the position margin has median 0.00390625 and is exactly zero on 42.08% of decisions, whereas the committed-token gap has median 0.9985 and is zero on 0.05%. The descriptive separation remains in FP16 and FP32, but is asymmetric by construction: the token gap is measured at a position selected for high confidence, while the position margin is a top-two order statistic over many positions. Quantization can also collapse nearby values into ties. These aggregates do not isolate either effect or establish position ranking as the numerical source.

Restricting BF16 to the 186,797 decisions whose selected positions precede the first EOS/pad boundary and whose position margin has a runner-up yields a position-margin median of 0.0078125 with 38.45% exact ties. The corresponding committed-token gap has median 0.99534 and 0.063% exact ties. Post-boundary positions have 35,701/68,703 (51.96%) exact position ties, so the all-step estimate is partly tail-weighted.

Event-conditioned position margins. Over the pair census, at least one execution has an exact-zero runtime top-two position margin at 3,949/5,687 LLaDA BF16 position-mismatch forks (69.4%) and both do at 1,817 (32.0%); FP16 gives 3,839/4,999 and 1,726/4,999, and Dream BF16 416/4,537 and 2/4,537. The corresponding primary-pair counts are 313/499 (62.73%) and 129/499 (25.85%). From the recorded FP16 confidence maps, the pooled cross-run rank of the two selected positions has median 2 and 90th percentile 7; both are in the other run's top two in 307/499 events, but the event-level maximum reaches 205.

The top-confidence sets reconstructed from the recorded FP16 maps overlap in 299/499 primary events, but are never identical; among the 313 events with an exact runtime tie in either run, their median size is two in each execution. Selecting the lowest response position reproduces each execution's local logged tie choice at every examined shared state, but the executions have different top sets, so applying that rule separately to the recorded maps aligns 0/499 mismatch events. Sets within $1/256$ of each run's top overlap in 411/499 events and their lowest positions align in 258, but that rule also changes at least one logged choice in 1,543/10,470 stable states. These are one-step retrospective set comparisons, not simulated decoders; near-plateau collapsing can introduce new disagreements downstream.

To compare the observed cross-run confidence perturbation with the runtime margin, we divide the maximum absolute confidence-map difference over shared masked positions by the smaller positive position margin. At the 186 primary position-mismatch forks with a positive margin, this ratio is at least one in every case (median 7.5; 90th percentile 13). Among 6,705 stable shared states with a positive margin, 3,119 (46.5%) reach one (median 0.8; 90th percentile 8). The other 313 event states and 3,765 stable states have an exact-zero smaller margin. Confidence maps were stored in FP16 while margins retain their runtime values, so this normalization is descriptive and storage quantization can affect the ratio; it is not a ULP-normalized trigger analysis.

Figure 9 expands the single-run sensitivity check. Each execution triggers only from its own observed position margin; "union" asks whether either execution would trigger, solely to score the matched records. The positive class is the first position mismatch. These retrospective diagnostics do not simulate verification and therefore cannot establish residual disagreement or runtime. Percentile intervals resample the 500 prompts with replacement, keep every first-event-censored state within a resampled prompt, and recompute the pooled ratio (50,000 replicates).

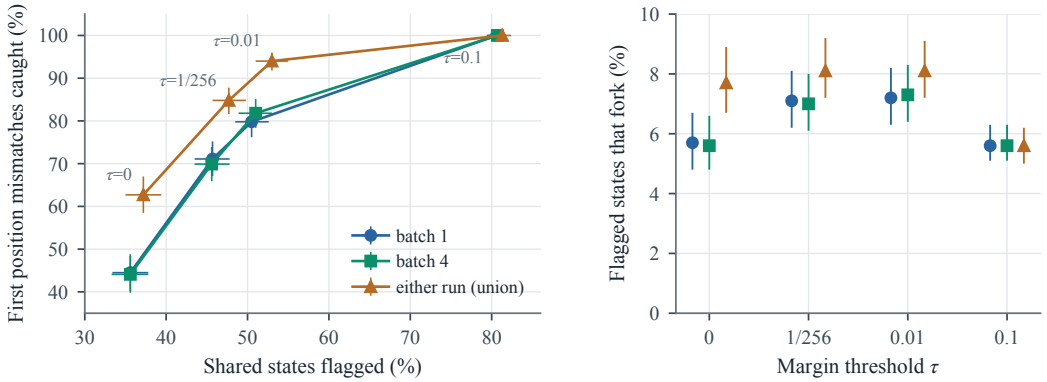

Figure 9: Own-run position-margin triggers through the first mismatch in the LLaDA BF16 batch-1 $\leftrightarrow$ 4 comparison, at thresholds $\tau \in \{0, 1/256, 0.01, 0.1\}$. Left: share of shared states flagged against the share of first position mismatches caught. Right: the fraction of flagged states that are first position mismatches (PPV). Bars are prompt-cluster bootstrap 95% intervals.

Across these thresholds, PPV never exceeds 8.08%. Under a cost-only counterfactual that first runs BF16 and then invokes a one-sample LayerCast forward for every own-run trigger, the A6000 forward-equivalent estimate at $\tau = 0.01$ is 3.43×/3.80× for batch one/four; at $\tau = 0.1$ it is 4.88×/5.43×. Because no position verifier was run, these figures have no associated residual output or trajectory disagreement and are not evidence of an effective policy. The exact numbers of states where the two executions trigger asymmetrically (and position-mismatch forks among them) are 352 (184), 458 (142), 492 (132), and 140 (0) in threshold order.

C.5 Final-output-region trajectory projection

For each execution, we retain in commitment order only position–token pairs whose response-relative position is strictly before that execution's final EOS/pad boundary; a projected trajectory divergence is a first mismatch among the retained pairs. As sensitivity variants, an intersection projection applies $\min(L_a, L_b)$ to both executions, while a union projection applies $\max(L_a, L_b)$ and therefore includes positions visible in either output. Neither common-support variant is causally closed: discarded commitments were present in the bidirectional state, so the projection is an output-visibility summary, not a causal-support estimate. Table 7 reports the run-specific projection for the primary MATH500 pairs.

The seven late FP32 projected divergences all re-converge to the same effective output. Hence those events are mostly procedural, but only one of the eight full FP32 divergences is removed as tail-only; lateness alone does not make the remaining seven post-boundary artifacts.

Table 7: MATH500 batch-1 $\leftrightarrow$ 4 trajectories after projecting each run onto its own final output region. "Touches" conditions on a full mismatch and asks whether either selected position is effective; "tail" means neither is. Projected re-convergence conditions on projected trajectory divergence and asks whether the effective outputs agree.

compute	full traj.	projected traj.	first touches	first tail	projected re-conv.
FP32	8/500	7/500	7/8	1/8	7/7
FP16	465/500	356/500	341/465	124/465	257/356
BF16	500/500	425/500	376/500	124/500	80/425

C.6 Decoding-schedule generalization

All other LLaDA results use one block of 512 positions and 512 steps with one commitment per step. Table 8 repeats the RTX PRO 6000 batch-1 $\leftrightarrow$ 4 comparison under LLaDA's released semi-autoregressive schedule (16 blocks of 32 positions, 512 steps) and under an accelerated schedule that commits two tokens per step (256 steps). Both reproduce the structure of the main setting: the precision gradient, a first mismatch that is a position mismatch in 99% of divergent trajectories, near-total trajectory divergence in BF16 and FP16 with partial re-convergence, and at most 8/500 position-only trajectory forks with at most one output change in FP32. Restricting candidates to a 32-position block lowers divergence (324/500 against 346/500 in BF16, 62/500 against 99/500 in FP16) and delays the median first fork (steps 19 and 103 against 12 and 74), whereas committing two tokens per step raises divergence (393/500 and 162/500) and moves the first fork earlier (steps 5 and 40 of 256), because each step now selects the top two positions and exposes two ranking boundaries. The semi-autoregressive schedule roughly doubles accuracy (208–209/500 against 114–117/500) without changing the divergence structure, so the low absolute accuracy of the main setting reflects the single-block schedule rather than the nondeterminism analysis.

Table 8: LLaDA MATH500 batch-1 $\leftrightarrow$ 4 divergence on the RTX PRO 6000 under three decoding schedules ($n = 500$): the main single-block schedule with one commitment per step, LLaDA's released semi-autoregressive schedule (16 blocks of 32 positions) and an accelerated schedule that commits two tokens per step. "first fork" is the median zero-indexed step of the first trajectory mismatch out of the schedule's step count; "re-conv." counts divergent trajectories with identical strict outputs.

schedule	precision	output / trajectory	first pos. / tok.	first fork	re-conv.	correct b=1 / b=4
block 512, 512 steps	BF16	346 / 500	499 / 1	12 / 512	154	114 / 116
	FP16	99 / 465	465 / 0	74 / 512	366	115 / 113
	FP32	0 / 8	8 / 0	368 / 512	8	117 / 117
block 32, 512 steps	BF16	324 / 499	494 / 5	19 / 512	175	208 / 209
	FP16	62 / 427	423 / 4	103 / 512	365	204 / 209
	FP32	0 / 8	8 / 0	202 / 512	8	208 / 208
block 512, 256 steps	BF16	393 / 500	500 / 0	5 / 256	107	118 / 111
	FP16	162 / 461	459 / 2	40 / 256	299	116 / 117
	FP32	1 / 4	4 / 0	20 / 256	3	115 / 115

D Response-State Replay and Additional Results

D.1 States, events, and validation identities

We reconstruct the response buffer in spatial token order after every original decoding step. A position contains either the mask sentinel or its committed token. Steps are zero-indexed; the remaining-mask count at first fork is measured before that step, whereas O_t, C_t and H_t are measured after it. Commitments within a multi-position step are compared as position–token mappings, not as an arbitrary serialization. Empty steps remain explicit.

Each unequal position falls into exactly one of two disjoint cases: it is committed in only one run, contributing to O_t, or in both with unequal tokens, contributing to C_t. Hence $H_t = O_t + C_t$. Replay checks that every position is initially masked, assigned exactly once, and never overwritten. A

conflicting pair of committed tokens then remains conflicting, so C_t cannot decrease. At completion both buffers are fully assigned and $O_t = 0$; therefore $C_t > 0$ is equivalent to strict final-token disagreement. These properties validate the measurement; they are not an empirical mechanism or predictive achievement.

An intermediate response-state reunion is an $H > 0$ to $H = 0$ transition after the first commitment fork, strictly before the last step. It does not erase the distinct commitment histories. A subsequent positive H is a refork. For native runs, even identical response buffers leave batch context and numerical execution different. Interventions can additionally retain different frozen position orders. Response-buffer equality is consequently weaker than complete decoder-state equality.

The first excursion ends at the first reunion or the first positive C, whichever occurs first; its duration is the ending step minus the first-fork step. The delay $c - f$ instead ends at the first token conflict, even if several reunions intervene. It is undefined when no conflict occurs through completion. A singleton position-mismatch fork necessarily has $C_f = 0$, so a one-step minimum delay is structural. The single token-mismatch primary case has zero delay, consistent with these definitions.

D.2 Coverage, matching, and uncertainty

The step-resolved analyses of Section 5 use the primary LLaDA MATH500 BF16 and FP16 pairs and the BF16 pairs of Table 9. Their counts are recomputed from the logged commitments of every prompt (500 or 164 per pair) and agree with the original replay of the primary pair. There are no sample-level exclusions inside these comparisons, and missing references are not borrowed from another environment.

Canonical IDs, prompt-text SHA1 and length, active dataset templates, and gold/source metadata match within included pairs. Original chat-token arrays and tokenizer revisions were not saved, so the prompt-token identity of the original runs cannot be independently certified. The available source tree has no version-control record of the original execution; current source-file hashes are recorded rather than presented as the original execution commit.

The replay validates complete steps, unique assignments, spatial final-buffer reconstruction, and the effective-output prefix. The supplied audit checks the 500 prompts of the primary pair with an independent dense-state reconstruction and checks state-derived event summaries over all exported steps. Independent aggregation reproduces the primary event counts. The accompanying derived records permit checks of events and statistical summaries; they do not replace the server-only original generated token sequences for raw-to-normalized reconstruction.

Intervals use 10,000 percentile bootstrap draws with seed 20260914. Canonical prompts are resampled within benchmark; all configuration comparisons and equality spells of a sampled prompt share its multiplicity. Ratios are recomputed within each draw. Spells, steps and repeated pairs are never treated as independent trials. Curve intervals are pointwise. First-fork alignment has a changing observed population because remaining run lengths differ; terminated observations are not extended. The plotted denominator includes already-conflicted prompts until their own completion, so "observed prompts" is an occupancy denominator, not a conflict-free survival risk set.

Scope. The analysis uses saved commitment records without new model generations, rescoring or confidence-map reranking. Full logits are unavailable, and recorded confidence maps have FP16 storage. It does not reconstruct a first tensor error, simulate a different verifier's continuation, repair the Dream gate by reinterpretation, or validate accelerated parallel reveal.

D.3 Intermediate reunions and conflict delays

Table 9 collects the replay rows and the BF16 recomputation of Figure 10. For the LLaDA MATH500 BF16 and FP16 and Dream (max-prob) rows the reunion-rate intervals are $[90.2, 94.8]\%$, $[88.82, 93.85]\%$ and $[55.2, 63.6]\%$, and among prompts with a reunion, eventual strict disagreement occurs in 309/463, 59/425 and 275/297 cases, respectively. These contrasts combine task, precision and model/protocol differences and do not identify an architectural cause. The Dream (max-prob) row is the EOS-suppressed maximum-probability-scorer stress test of Appendix E.4, not a corrected ULP-verifier run.

Table 9: Response-state dynamics for RTX PRO 6000 batch-1 $\leftrightarrow$ 4 pairs. BF16 rows without a marker are recomputed from the logged commitments with the definitions of Section 5; rows marked ‡ come from the original replay and report only the quantities it recorded. Reunion counts condition on a fork; the conflict delay $c - f$ conditions on an observed conflict, with a prompt-bootstrap 95% interval for the median and the interquartile range; "excursion" is the median length of the first excursion, which ends at the first reunion or conflict.

setting	forks / n	reunions (reforks)	conflicts	median $c - f$ [95%]	IQR	excursion
LLaDA / MATH500 / BF16	500/500	463 (460)	346	164.5 [146,186]	78–287	6
LLaDA / MATH500 / FP16‡	465/500	425 (–)	99	87 [58,111]	–	–
LLaDA / HumanEval / BF16	164/164	162 (160)	43	388 [330,449]	280–468	195
Dream (entropy) / MATH500 / BF16	500/500	379 (379)	437	57 [43,64]	7–132	1
Dream (entropy) / HumanEval / BF16	164/164	116 (116)	145	167 [143,182]	75–253	2
Dream (max-prob) / MATH500 / BF16‡	500/500	297 (–)	478	47 [42,52]	–	–

In the primary pair, 460/463 reunited prompts refork (99.35%, interval $[98.50, 100]\%$). The 309 that subsequently end with different strict outputs give 66.74% of reunited prompts (interval $[62.31, 70.97]\%$). Strict and effective final differences are 346/500 and 345/500: one prompt's strict disagreement is hidden by effective-output truncation. No prompt reunites only at the final step; the one-sided 95% binomial upper bound for that zero count is 0.597%.

The conditional primary $c - f$ distribution has IQR 78–287, 90th percentile 421, 95th percentile 482 and maximum 507. The five largest delays contribute 3.74% of their total; excluding those observed IDs leaves median 160 steps, and excluding the ten largest leaves 156.5. These are descriptive sensitivity checks, not post-selection hypothesis tests. The first-excursion duration has median six steps (interval $[4, 8]$; IQR 2–97.25).

Completed reunion spells span 2,876 episodes across 460 primary prompts. The number of consecutive equal post-step states before the next refork has median 16 (interval $[15, 17]$; IQR 6–36). Another 154 spells extend through completion and have no observed later refork; they are not treated as completed waiting times. These statistics explain why the first fork and the eventual conflict can be widely separated without one uninterrupted period of unequal buffers.

Replication on the other BF16 settings. Table 9 repeats the response-state analysis of Section 5 for the other three BF16 settings, recomputed from the logged commitments, and Figure 10 shows the two HumanEval settings (Dream MATH500 is in Figure 4); the same computation reproduces every primary LLaDA MATH500 count above (463 reunions, 460 reforks, 346 conflicts, median $c - f$ of 164.5 steps). Every prompt forks in all four settings. LLaDA HumanEval behaves like LLaDA MATH500 with more re-convergence: 162/164 buffers reunite and 160 refork, yet only 43 reach a token conflict, the first excursion lasts a median of 195 steps, and the conflicts that do occur are late (median 388 steps after the fork). Dream shows the opposite profile in both tasks: the first excursion ends after one or two steps, reunions still occur (379/500 and 116/164) but almost every prompt conflicts (437/500 and 145/164), and the conflict follows the fork after a median of 57 and 167 steps. Transient equality is therefore common in every setting, but how long it lasts and whether it ends in a conflict depends on the model and the task.

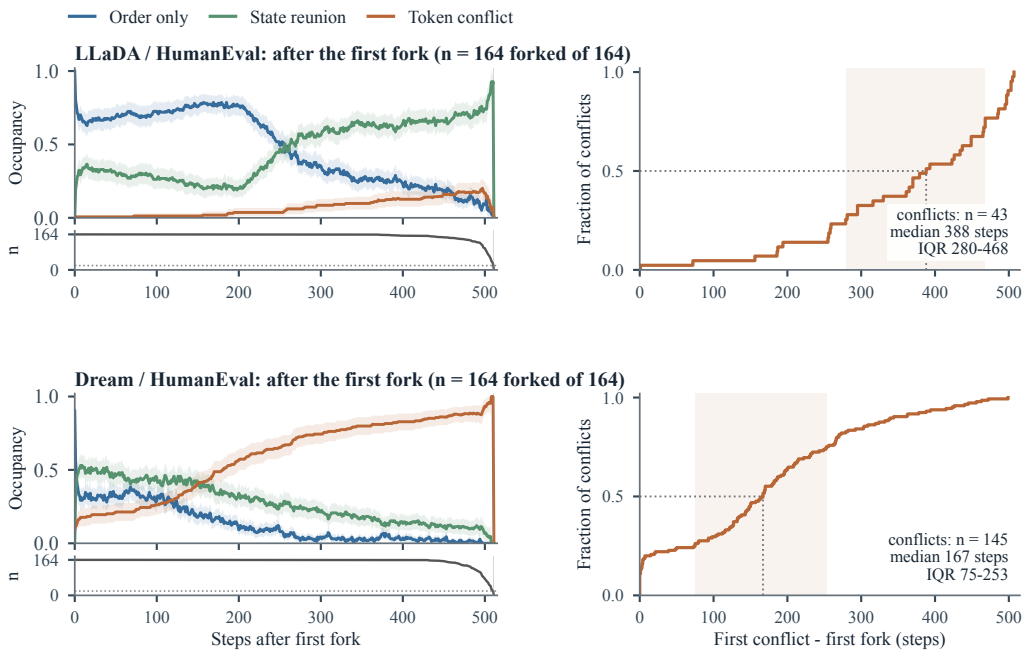

Figure 10: Figure 4 repeated for the RTX PRO 6000 BF16 batch-1 $\leftrightarrow$ 4 pair of LLaDA HumanEval and Dream HumanEval (entropy scorer). Left: occupancy of order-only discrepancy, equal response state and token conflict, aligned to each prompt's first fork, with pointwise prompt-bootstrap 95% ribbons; the population shrinks as runs end (shading: fewer than 20 prompts). Right: delay to the first conflict, conditional on the prompts that experience one.

D.4 First-fork timing is a descriptive covariate

Table 10: Fixed timing bins in the primary BF16 comparison. The covariate is the fraction of positions committed immediately before the first fork. The outcome is strict final-token disagreement. Intervals are prompt-bootstrap 95% intervals; empty and sparse bins are retained.

pre-fork committed fraction	different / prompts	percent [95% interval]
[0, .01)	87/169	51.5 [43.75,58.95]
[.01, .025)	66/87	75.9 [66.67,84.62]
[.025, .05)	80/105	76.2 [67.92,84.16]
[.05, .10)	64/80	80.0 [70.83,88.51]
[.10, .25)	48/58	82.8 [72.41,92.00]
[.25, .50)	1/1	insufficient for generalization
[.50, 1]	0/0	unavailable

Table 10 bins the primary BF16 comparison by the fraction of positions committed before the first fork. The earliest bin does not have the greatest observed disagreement. Prompt difficulty and trace structure can confound this association; it does not show that delaying a fork increases risk. For singleton updates, remaining masks before the first fork equal $512 - f$, so we use only one of these equivalent covariates. Dream uses its actual cumulative commitment count. We do not replace within-pair comparisons with a pooled precision correlation or call these retrospective slices a prediction benchmark.

E Mitigation Results

Unless stated otherwise, each table in this appendix compares batch one with batch four on the same prompts ($n = 500$ for MATH500, 164 for HumanEval). "Output / trajectory" counts prompts whose

strict output or full trajectory differs; "first position / token" splits the divergent trajectories by the kind of first fork; "correct b=1 / b=4" counts correct answers at each batch size under the unified scorer. "Frozen + k ULP" is four-step order freezing with the k-ULP token gate.

E.1 Complete LLaDA ULP sweep

Table 11 reports every completed threshold comparison and Table 12 the verifier trigger rate and measured cost by threshold. Each entry matches the same 500 prompts generated at batch sizes one and four; all 36 underlying generation configurations contain 500 samples, 500 trajectories, and 500 score records. Output divergence falls monotonically with the threshold in every environment, whereas the remaining first position-mismatch count stays nearly constant.

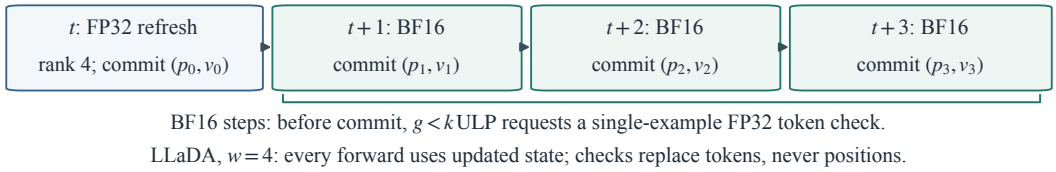

Figure 11: LLaDA four-step order freezing. The LayerCast refresh ranks positions $(p_0, \ldots, p_3)$ and commits token v_0. Each subsequent BF16 forward uses the updated state to compute v_i at frozen position p_i, and the next refresh comes at step $t{+}4$. Circles mark the ULP token gate (Equation 4): a small gap recomputes that example's step in LayerCast, which can replace v_i but not p_i.

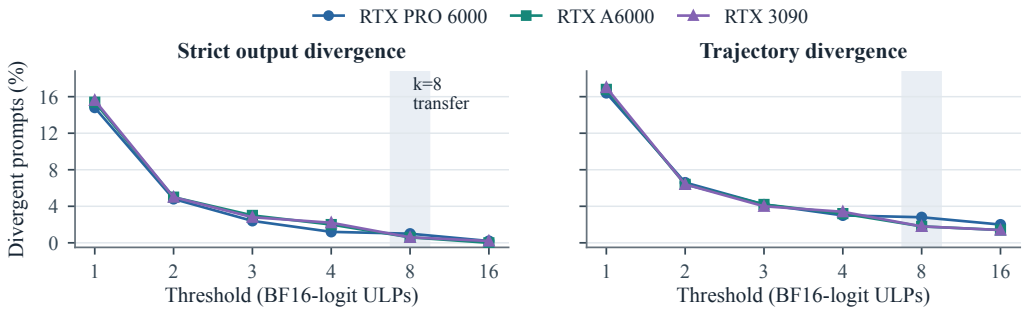

Figure 12: LLaDA MATH500 divergence for four-step order freezing with BF16-logit ULP verification ($n = 500$ per point; batch $1 \leftrightarrow 4$). The combined verifier trigger rate rises from 0.15% of sample-steps at $k = 1$ to 3.34% at $k = 8$ and 8.71% at $k = 16$. Lines connect evaluated thresholds.

At $k = 8$, the output-divergence Wilson intervals are $[0.4, 2.3]\%$, $[0.2, 1.7]\%$, and $[0.2, 1.7]\%$ in table order. Pairing batch-one outputs across environments gives 5/500 divergences for RTX PRO 6000 versus either Ampere environment and 0/500 for A6000 versus 3090. The corresponding trajectory counts are 10, 9, and 3; all five residual output differences in the first two comparisons include a token mismatch.

The model combines scheduled LayerCast refreshes, BF16 steps, and single-example verifier calls from isolated forward timings; the measured ratios are end-to-end generation times with trajectory logging disabled. They agree within 4% at batch one. At batch four LLaDA's measured ratios are lower because its BF16 baseline is itself slower per sample at batch four (Appendix E.6). Neither estimates optimized serving throughput.

Table 11: Complete LLaDA MATH500 ULP sweep, by environment and threshold k. Columns as defined at the start of Appendix E.

environment	k	output / trajectory	first position / token	correct b=1 / b=4
RTX PRO 6000	1	74 / 82	9 / 73	115 / 115
	2	24 / 33	9 / 24	116 / 116
	3	12 / 21	9 / 12	117 / 116
	4	6 / 15	9 / 6	117 / 116
	8	5 / 14	9 / 5	118 / 117
	16	1 / 10	9 / 1	116 / 116
RTX A6000	1	77 / 84	7 / 77	114 / 116
	2	25 / 32	7 / 25	115 / 115
	3	15 / 21	6 / 15	115 / 115
	4	10 / 16	6 / 10	115 / 115
	8	3 / 9	6 / 3	116 / 116
	16	0 / 7	7 / 0	116 / 116
RTX 3090	1	78 / 85	7 / 78	114 / 116
	2	25 / 32	7 / 25	115 / 115
	3	14 / 20	6 / 14	115 / 115
	4	11 / 17	6 / 11	115 / 116
	8	3 / 9	6 / 3	116 / 116
	16	1 / 7	6 / 1	116 / 116

Table 12: ULP trigger rate and cost for LLaDA MATH500. Trigger rate is pooled across the three environments and counts all sample-steps. Cost is A6000 per-sample wall-clock relative to BF16 with instrumentation off, mean of two repetitions on 30 prompts (Appendix E.6); the model column is the calibrated forward-count model with A6000 constants. Thresholds below 8 were not measured ("–").

k	trigger rate	model b=1	measured b=1	measured b=4
1	0.15%	1.95×	–	–
2	0.49%	1.97×	–	–
3	0.87%	1.99×	–	–
4	1.28%	2.01×	–	–
8	3.34%	2.11×	2.18×	1.84×
16	8.71%	2.36×	2.43×	2.03×

E.2 LLaDA HumanEval

Table 13 gives the complete HumanEval policy comparison. The 8-ULP setting removes all observed output divergence in each environment, but 8–10 position-only trajectory forks remain and later re-converge. Its trigger rate is 0.741–0.742% of sample-steps across environments, substantially below MATH500's 3.34%. On RTX PRO 6000, the final-output-region trajectory counts for BF16, frozen order, 8 ULP, and LayerCast are 146, 21, 2, and 3; the two 8-ULP cases re-converge.

For zero observed 8-ULP output differences, the Wilson upper bound is 2.29% in each environment. The decoder intentionally differs from native BF16, so agreement does not imply a quality improvement: under the unified scorer, frozen + 8 ULP solves 45/164 problems in every environment at both batch sizes, against 38/164 for BF16 at batch one and 37/164 for LayerCast, a 7-problem difference that this study does not test for significance.

Table 13: LLaDA HumanEval under the MATH500 recipe, without task-specific retuning. Columns as defined at the start of Appendix E.

environment	policy	output / trajectory	first position / token	correct b=1 / b=4
RTX PRO 6000	BF16	43 / 164	164 / 0	38 / 38
	FP16	10 / 152	152 / 0	38 / 38
	frozen order	20 / 27	8 / 19	45 / 47
	frozen + 8 ULP	0 / 8	8 / 0	45 / 45
	frozen + 16 ULP	0 / 8	8 / 0	45 / 45
	LayerCast	0 / 3	3 / 0	37 / 37
RTX A6000	BF16	53 / 164	164 / 0	38 / 34
	FP16	12 / 161	161 / 0	38 / 38
	frozen order	29 / 39	10 / 29	44 / 45
	frozen + 8 ULP	0 / 10	10 / 0	45 / 45
	LayerCast	0 / 3	3 / 0	37 / 37
RTX 3090	BF16	52 / 164	164 / 0	38 / 34
	frozen order	30 / 38	8 / 30	44 / 45
	frozen + 8 ULP	0 / 8	8 / 0	45 / 45
	LayerCast	0 / 4	4 / 0	37 / 37

E.3 DREAM ENTROPY SCORER

Tables 14 and 15 report the entropy scorer, which is the position rule of the model's released evaluation code and the Dream setting of the main text, on MATH500 and HumanEval. All gated rows use the corrected post-suppression gate. "Frozen order" rows freeze the position order every four steps without a verifier; "frozen + k ULP" rows add the token verifier at the stated threshold. "Verifier-only" rows use no order refresh: the execution dtype selects the positions at every step; if the top-two gap at any selected position satisfies Equation 4 in ULPs of that dtype, the sample's step is recomputed once in LayerCast and the FP32 argmax is written at the same positions. This variant never touches the position channel, so it can only succeed where that channel is already stable in the execution dtype.

Table 14: Dream MATH500 under the entropy scorer. Columns as defined at the start of Appendix E.

environment	policy	output / trajectory	first position / token	correct b=1 / b=4
RTX PRO 6000	BF16	437 / 500	425 / 75	219 / 227
	FP32	0 / 1	1 / 0	222 / 222
	frozen order	405 / 408	3 / 405	192 / 190
	frozen + 8 ULP	80 / 86	6 / 80	200 / 201
	frozen + 16 ULP	42 / 48	6 / 42	199 / 202
	BF16 verifier-only 8 ULP	332 / 500	500 / 0	222 / 224
	LayerCast	0 / 1	1 / 0	222 / 222
RTX A6000	BF16	439 / 500	416 / 84	215 / 237
	FP32	0 / 1	1 / 0	222 / 222
	frozen order	419 / 419	1 / 418	194 / 195
	frozen + 8 ULP	91 / 98	8 / 90	202 / 200
	frozen + 16 ULP	52 / 59	9 / 50	200 / 201
	BF16 verifier-only 8 ULP	323 / 500	499 / 1	238 / 229
RTX 3090	BF16	441 / 500	416 / 84	218 / 227
	frozen order	413 / 413	0 / 413	192 / 188
	frozen + 8 ULP	89 / 93	5 / 88	203 / 198
	BF16 verifier-only 8 ULP	316 / 500	499 / 1	234 / 229

Under the entropy scorer, exact top-score ties occur at 0.3% of BF16 batch-one position decisions (max-prob, Appendix E.4: 33.4%; LLaDA: 42.1%), yet BF16 first mismatches remain position-dominant (425/75), so its position decision forks through small score differences rather than exact ties. Once margins are measured in ULPs of the entropy score, these forks are concentrated: over the pair census, a rule that flags states whose smaller top-two position margin is at most 8 ULP recalls 79.7% of first position mismatches while flagging 6.4% of shared states, and 16 ULP recalls 93.7% for 11.6%. We did not run a position-channel verifier; the recipes below gate only tokens.

The corrected-gate frozen rows on the RTX PRO 6000 (16.0% and 8.4% at $k = 8$ and $k = 16$) leave 80 and 42 first token mismatches with 6 position mismatches each; the A6000 gives 18.2% and 10.4% and the 3090 17.8% at $k = 8$, so the residual is similar across environments. Per shared step up to the fork, Dream's MATH500 token-mismatch forks occur at 43.4 per 10^4 steps under four-step freezing

alone, against 6.9 for LLaDA, and at 3.6 against 0.20 with the 8-ULP verifier; the verified LLaDA rate rests on five forks.

Verifier-only controls. Verifier-only decoding in BF16 fails for Dream (66.4% output divergence with 500/500 first position mismatches), because it never corrects the BF16 position choice. Verifier-only decoding on LLaDA in FP16 also fails (18.2% against 19.8% for FP16 alone, with 465 first position mismatches at a 0.10% trigger rate), because LLaDA's position rule still breaks by ties there. Table 16 summarizes the BF16 recipes by how the position decision forks.

Table 15: Dream HumanEval under the entropy scorer. Columns as defined at the start of Appendix E.

environment	policy	output / trajectory	first position / token	correct b=1 / b=4
RTX PRO 6000	BF16	145 / 164	148 / 16	89 / 95
	frozen order	139 / 140	1 / 139	96 / 90
	frozen + 8 ULP	43 / 44	1 / 43	93 / 90
	frozen + 16 ULP	22 / 23	1 / 22	93 / 90
	BF16 verifier-only 8 ULP	113 / 164	164 / 0	95 / 100
	LayerCast	0 / 1	1 / 0	97 / 97
RTX A6000	BF16	150 / 164	151 / 13	94 / 94
	frozen order	136 / 136	0 / 136	90 / 93
	frozen + 8 ULP	38 / 38	0 / 38	88 / 89
	frozen + 16 ULP	26 / 26	0 / 26	90 / 88
	BF16 verifier-only 8 ULP	109 / 164	164 / 0	92 / 97
	LayerCast	0 / 0	0 / 0	97 / 97
RTX 3090	BF16	154 / 164	147 / 17	91 / 93
	frozen order	136 / 136	0 / 136	87 / 92
	frozen + 8 ULP	41 / 41	0 / 41	88 / 90
	frozen + 16 ULP	27 / 28	1 / 27	88 / 91
	BF16 verifier-only 8 ULP	117 / 164	164 / 0	96 / 96
	LayerCast	0 / 0	0 / 0	97 / 97

Table 16: Recipes by position-channel failure mode. Entries are MATH500 batch-1 $\leftrightarrow$ 4 strict output / full trajectory divergence on the RTX PRO 6000 and the verifier trigger rate (fraction of sample-steps). The tie rate is the share of BF16 batch-one position decisions with an exact top-score tie. All recipes run in BF16.

model / scorer	tie rate	failure mode	recipe	out. / traj. div.	trigger
LLaDA / confidence	42.1%	ties	frozen + 8 ULP	1.0 / 2.8	3.34%
			frozen + 16 ULP	0.2 / 2.0	8.71%
Dream / max-prob	33.4%	ties	frozen + 8 ULP	20.2 / 21.2	5.65%
Dream / entropy	0.3%	perturbation	frozen + 8 ULP	16.0 / 17.2	7.17%
			frozen + 16 ULP	8.4 / 9.6	13.82%

On HumanEval the picture repeats: the BF16 verifier-only control fails as on MATH500 (164/164 first position mismatches), and LayerCast solves 97/164 problems.

E.4 Dream with the maximum-probability scorer (stress test)

This subsection reports Dream under the maximum-probability position scorer ("maskgit-plus" in the released code; "max-prob" below), $\hat{p}_t = \arg\max_{p \in M_t} c_{t,p}$, which we use as a controlled stress test of a tie-driven position rule on a second model; the main text uses Dream's entropy scorer (Section 3). Under max-prob, exact top-score ties occur at 33.4% of BF16 batch-one position decisions, and the batch-1 $\leftrightarrow$ 4 precision gradient on the RTX PRO 6000 is 1/500, 216/500, and 478/500 output divergences with 7/500, 487/500, and 500/500 trajectory divergences. Across seven batch-1 $\leftrightarrow$ 4 comparisons (BF16 and FP16 on each of the three GPUs, and FP32 on the RTX PRO 6000), 2,916 of 2,966 first mismatches (98.31%) change the selected position set and 50 change only a token; for RTX BF16 the split is 492/8. Intermediate reunions occur for 297/500 forked BF16 MATH500 pairs with a conditional conflict-delay median of 47 steps (Appendix D). FP32 accuracy under our prompt is 38.6%.

Dream uses 510 singleton commitments, one zero-commit step, and one final two-position commitment per prompt. For each 512-step decoding batch, the zero-commit first step makes four-step

freezing execute 129 scheduled LayerCast forwards (an initial unproductive forward followed by refreshes at steps 1, 5, . . . , 509), not the nominal 128; verifier subforwards are additional. Table 17 reports the complete comparison. The "frozen + 8 ULP" rows use the corrected verifier, whose gap is computed after EOS suppression (Appendix A); the "raw-logit gate" rows compute the gate before the suppression and are shown for comparison.

Table 17: Dream MATH500 controlled stress test. Columns as defined at the start of Appendix E. "raw-logit gate" rows computed the ULP gap before EOS suppression; "frozen + 8 ULP" rows use the corrected post-suppression gate.

environment	policy	output / trajectory	first position / token	correct b=1 / b=4
RTX PRO 6000	BF16	478 / 500	492 / 8	173 / 167
	FP16	216 / 487	478 / 9	190 / 184
	FP32	1 / 7	7 / 0	193 / 192
	frozen order	366 / 367	4 / 363	139 / 121
	raw-logit gate	143 / 147	6 / 141	140 / 131
	frozen + 8 ULP	101 / 106	7 / 99	136 / 128
	LayerCast	1 / 7	7 / 0	193 / 192
RTX A6000	BF16	473 / 500	492 / 8	176 / 187
	FP16	215 / 488	482 / 6	190 / 186
	frozen order	353 / 354	4 / 350	140 / 138
	raw-logit gate	145 / 148	4 / 144	137 / 138
	frozen + 8 ULP	81 / 87	7 / 80	139 / 134
	LayerCast	0 / 3	3 / 0	192 / 192
RTX 3090	BF16	471 / 500	488 / 12	170 / 188
	FP16	215 / 484	477 / 7	189 / 185
	frozen order	358 / 360	3 / 357	134 / 142
	raw-logit gate	141 / 146	5 / 141	131 / 141
	frozen + 8 ULP	81 / 87	6 / 81	140 / 136
	LayerCast	0 / 5	5 / 0	192 / 192

The corrected verifier fires on 5.64–5.65% of all sample-steps, the same rate as the raw-logit gate (5.62–5.66%): the correction changes which steps are verified, not how many. It reduces the residual first token mismatches from 141–144 to 80–99 and the output divergence from 28.2–29.0% to 16.2–20.2%. Table 18 summarizes the post-hoc audit of the raw-logit runs. The remaining disagreement is similar across the three environments and consists mostly of first token mismatches (80–99 per environment, against 6–7 position mismatches). LayerCast outputs on A6000 and 3090 are token-identical to the RTX FP32 batch-four output; the RTX LayerCast run also matches FP32 within each batch.

Table 18: Post-hoc audit of the raw-logit gate on Dream (batch $1 \leftrightarrow 4$). Gated steps are the BF16 steps of both runs (383,000 per MATH500 comparison, 125,624 per HumanEval comparison); counts in the first three numeric columns are over unfired gated steps, fired steps, and unfired gated steps respectively. A blind-spot step is an unfired step at which EOS entered the raw top two while the post-suppression gap was below $k\,u$.

scorer / task / k	environment	EOS in raw top 2	spurious fires	blind spots	token mismatches at blind spots
max-prob / MATH500 / 8	RTX PRO 6000	11,296 (2.95%)	1,206 (4.2%)	1,057	110 / 141 (78%)
	RTX A6000	11,180 (2.92%)	1,230 (4.3%)	1,068	123 / 144 (85%)
	RTX 3090	11,111 (2.90%)	1,254 (4.3%)	1,075	116 / 141 (82%)
entropy / MATH500 / 8	RTX PRO 6000	27,110 (7.08%)	1,062 (2.9%)	996	132 / 155 (85%)
entropy / MATH500 / 16	RTX PRO 6000	26,345 (6.88%)	1,782 (2.6%)	1,377	134 / 145 (92%)
entropy / HumanEval / 8	RTX PRO 6000	2,023 (1.61%)	377 (3.4%)	260	39 / 53 (74%)
entropy / HumanEval / 16	RTX PRO 6000	1,788 (1.42%)	532 (2.5%)	392	43 / 50 (86%)

E.5 Cross-task summary: agreement, accuracy and LayerCast

Table 19 summarizes the RTX PRO 6000 batch-$1 \leftrightarrow 4$ comparison for the policies of Section 6, with accuracy under the unified scorer, and Figure 13 adds, for every policy, agreement with LayerCast.

Figure 13 plots each policy's batch-axis agreement against its agreement with LayerCast at the same batch size. The two measures are different quantities and often move in opposite directions. Every verified frozen policy on LLaDA MATH500 matches the LayerCast output on exactly 61/500 prompts

Table 19: Batch-1 $\leftrightarrow$ 4 results on RTX PRO 6000. Divergence entries are strict output/full trajectory percentages. The last column is accuracy (%) at batch one / batch four under the unified scorer. LayerCast uses FP32 arithmetic over BF16-stored weights. Dream's verifier gates the post-suppression decision.

model/task	policy	out./traj. div.	accuracy (%) b=1 / b=4
LLaDA/MATH500	BF16	69.2 / 100.0	22.8 / 23.2
	frozen + 8 ULP	1.0 / 2.8	23.6 / 23.4
	LayerCast	0.0 / 1.6	23.4 / 23.4
LLaDA/HumanEval	BF16	26.2 / 100.0	23.2 / 23.2
	frozen + 8 ULP	0.0 / 4.9	27.4 / 27.4
	LayerCast	0.0 / 1.8	22.6 / 22.6
Dream (entropy)/MATH500	BF16	87.4 / 100.0	43.8 / 45.4
	frozen + 8 ULP	16.0 / 17.2	40.0 / 40.2
	frozen + 16 ULP	8.4 / 9.6	39.8 / 40.4
	LayerCast	0.0 / 0.2	44.4 / 44.4
Dream (entropy)/HumanEval	BF16	88.4 / 100.0	54.3 / 57.9
	frozen + 8 ULP	26.2 / 26.8	56.7 / 54.9
	frozen + 16 ULP	13.4 / 14.0	56.7 / 54.9
	LayerCast	0.0 / 0.6	59.1 / 59.1
Dream (max-prob)/MATH500	BF16	95.6 / 100.0	34.6 / 33.4
	frozen + 8 ULP	20.2 / 21.2	27.2 / 25.6
	LayerCast	0.2 / 1.4	38.6 / 38.4

for $k = 2$ to 16 while its batch agreement rises from 476/500 to 499/500; native BF16 matches on 150/500 and FP16 on 399/500. The frozen order is a different decoder: it reproduces the LayerCast trajectory on 0/500 prompts at every threshold, and so does every other non-LayerCast policy on every task. Only LayerCast agrees across batch sizes while keeping the unfrozen decoding rule. Accuracy is a third quantity: at batch one, the verified LLaDA frozen policies gain 7 HumanEval problems and 1–4 MATH500 problems over BF16 while matching LayerCast on 32/164 and 61/500, whereas Dream's frozen policies under the entropy scorer lose 19–27 MATH500 problems (25–37 at batch four). Agreement across execution contexts is the property that regression testing needs; matching LayerCast is a separate property that these recipes do not provide.

E.6 Cost measurement and calibration

The instrumented generation runs log per-step trajectories and confidence maps, which adds a fixed per-step overhead and compresses every cost ratio toward one; we therefore do not report their wall-clock. Cost is instead measured by rerunning each policy with logging disabled on the first 30 prompts of each task at batch sizes one and four, twice, on the A6000 (every policy in Table 20) and on the 3090 (a seven-policy subset). The reported number is the per-sample generation time divided by that of the baseline decoder in the same precision on the same GPU, task, and batch size. On the A6000 the two repetitions differ by at most 0.8% in every cell; on the 3090 the LLaDA repetitions differ by 28–41%, so only A6000 values are reported.

A forward-count model predicts the same ratios from two calibrated constants (Table 21): R, the LayerCast-to-native time ratio of one forward pass, and $s(b)$, the time of a single-example forward relative to a batch-b forward. Four-step freezing then costs $R/4 + 3/4$, plus $b\, f\, s(b)\, R$ for a verifier that fires on a fraction f of sample-steps; verifier-only decoding costs $1 + b\, f\, s(b)\, R$; LayerCast costs R. Table 20 lists measured and modeled ratios. At batch one they agree within 6.1% for every policy of both models. At batch four LLaDA's measured ratios fall below the model because LLaDA's BF16 baseline is itself slower per sample at batch four than at batch one (89.8 against 65.7 seconds on the A6000): its full-vocabulary confidence softmax over all positions and the left padding remove the batching gain, while only the FP32 matmuls scale with R. Dream's baseline is batch-insensitive (57.7 against 56.6 seconds) and its ratios match the model at both batch sizes.

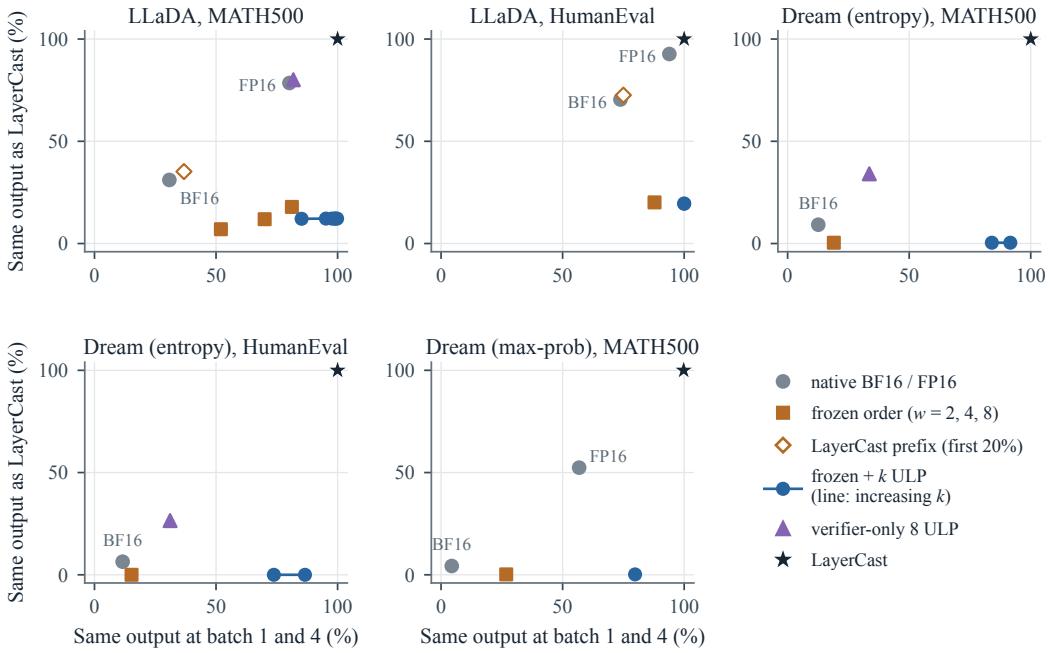

Figure 13: Agreement across batch sizes against agreement with LayerCast on the RTX PRO 6000. The horizontal axis is the share of prompts whose strict output is identical at batch one and four; the vertical axis is the share whose strict output equals LayerCast's at the same batch size, averaged over batch one and four. Frozen policies move right without moving up. No policy other than LayerCast reproduces the LayerCast trajectory on any prompt.

Table 20: Measured cost with instrumentation off (A6000, 30 prompts, mean of two repetitions) and the calibrated model, relative to the baseline decoder in the same precision. Trigger rates are from the full instrumented runs. Verifier-only rows are the appendix-only policies of Appendix E.3.

model / task	policy	trigger	measured b=1	measured b=4	model b=1
LLaDA / MATH500	BF16	–	1.00	1.00	1.00
	LayerCast prefix (first 20%)	–	1.82	1.56	1.76
	frozen order w=8	–	1.51	1.35	1.47
	frozen order w=4	–	2.02	1.70	1.95
	frozen order w=2	–	3.04	2.41	2.89
	frozen + 8 ULP	3.34%	2.18	1.84	2.11
	frozen + 16 ULP	8.71%	2.43	2.03	2.36
	LayerCast	–	5.08	3.81	4.79
LLaDA / HumanEval	BF16	–	1.00	1.00	1.00
	frozen order w=4	–	1.97	1.69	1.95
	frozen + 8 ULP	0.74%	1.99	1.71	1.98
	frozen + 16 ULP	1.96%	2.03	1.75	2.04
	LayerCast	–	4.87	3.77	4.79
Dream (entropy) / MATH500	BF16	–	1.00	1.00	1.00
	frozen order w=4	–	2.11	2.08	2.14
	frozen + 8 ULP	7.17%	2.48	2.46	2.54
	frozen + 16 ULP	13.82%	2.82	2.83	2.92
	BF16 verifier-only 8 ULP	6.58%	1.32	1.39	1.37
	LayerCast	–	5.39	5.27	5.58
Dream (entropy) / HumanEval	BF16	–	1.00	1.00	1.00
	frozen order w=4	–	2.13	2.08	2.14
	frozen + 8 ULP	6.54%	2.56	2.53	2.51
	frozen + 16 ULP	12.30%	2.94	2.93	2.83
	BF16 verifier-only 8 ULP	7.78%	1.48	1.39	1.42
	LayerCast	–	5.45	5.30	5.58
Dream (max-prob) / MATH500	BF16	–	1.00	1.00	1.00
	frozen order w=4	–	2.12	2.08	2.14
	frozen + 8 ULP	5.65%	2.41	2.42	2.46
	LayerCast	–	5.42	5.28	5.58

Table 21: Calibration constants at sequence length 768 (prompt plus 512 response positions): R is the LayerCast-to-native ratio of one forward pass at batch size b, and $s(b)$ is the single-example forward time relative to the batch-b forward. Medians of 12 synchronized forwards. Batch 16 does not fit on the 3090.

GPU	model	$R(1)$	$R(4)$	$R(16)$	$s(4)$	$s(16)$
RTX PRO 6000	LLaDA	5.70	5.74	5.23	0.274	0.063
	Dream	6.11	5.96	5.87	0.276	0.067
RTX A6000	LLaDA	4.79	5.10	5.15	0.287	0.075
	Dream	5.58	5.50	5.48	0.267	0.068
RTX 3090	LLaDA	3.90	4.21	–	0.291	–
	Dream	3.84	4.11	–	0.272	–

E.7 Earlier refresh-window ablations

The window ablation of Figure 6 ($w = 8$, 4 and 2 at 1.51×, 2.02× and 3.04×; Appendix E.6) was run before the ULP sweep. In the same setting, computing the first 20% of steps with LayerCast and then returning to native BF16 leaves 316/500 (63.2%) outputs divergent at 1.82×. These exploratory results motivate periodic order refresh, but compare different decoding policies and do not isolate arithmetic precision from order staleness.

F Extended Comparison with Related Work

F.1 Different sources of variation require different comparisons

Sampling variability, hidden service behavior, model compression and numerical execution changes are distinct experimental interventions. Song et al. (2025) compare greedy decoding with repeated sampling and show why a single decoding choice can give an incomplete performance assessment. Atıl et al. (2025) examine nominally deterministic hosted settings and distinguish raw-response agreement from extracted-answer agreement. Their hosted observations do not isolate a particular kernel or floating-point cause. Our native pairs instead specify local execution configurations, remove intentional token sampling, and compare per-step commitments. They still confound batchmates, padding and tensor shape, and one recorded run per configuration cannot characterize repeated-run variance.

Compression studies motivate separating accuracy from agreement with a reference. Dutta et al. (2024) show that opposing correctness changes can cancel in aggregate accuracy and examine low-margin answer flips. More recently, Qamar et al. (2026) evaluate full-vocabulary distributional fidelity to a BF16 reference with total variation and Jensen–Shannon divergence. Gupta et al. (2026) study GPTQ and modified Hessian-aware quantization on a diffusion coding model, reporting task accuracy and resource trade-offs. Such weight-compression results do not establish exact output invariance under rebatching or changes in arithmetic precision. Our joint reference categories compare generated token arrays, not full-vocabulary distributions; the saved confidence maps cannot support the latter measurement. Our accuracies come from one re-scoring pass (Appendix B) and are reported beside cross-context agreement and agreement with LayerCast, not as a quality benchmark.

Fang et al. (2026) expose sample-level variation in MDLMs across batch, hardware and precision at the level of final predictions; we extend such comparisons to intermediate reunions and later token conflicts. MarginGate's synchronous AR prefix analysis stops at the first divergence (Chu et al., 2026), whereas an MDLM's two decisions per step make the path after the first divergence informative.

F.2 What is stable, and what can change?

Our validation identities are specific to immutable assignments: a conflicting committed token cannot be erased, so C_t is nondecreasing. A revision-enabled sampler can remove that conflict by remasking or replacement; the same monotonicity need not hold. Conversely, our temporary reunions need no corrective operation: executions can fill compatible positions in different orders. Neither event identifies an incorrect answer without a separate scoring rule.

ReMDM adds remasking transitions, RemeDi learns remasking policies, and RDD rolls back stalled block states (Wang et al., 2026a; Huang et al., 2025; Wang et al., 2026b). These methods revise revealed tokens. Our reunions occur without revision; the conflict monotonicity we use is specific to immutable assignments, and restoring an earlier state within one run differs from two unrevised executions naturally reaching equal buffers. We do not interpret such equality as correction of an erroneous token.

Within-run temporal information also has direct precedents: SWD penalizes changing predictive distributions (Wu & Huang, 2026), STDec uses candidate-ID persistence and neighboring decoded positions (Chen et al., 2026), PRR learns control from trajectory-derived convergence labels (Wan et al., 2026), LESS exploits mutual candidate stability (Mohamed et al., 2026), CCD uses historical contextual consistency (Chen et al., 2025a), JoT accelerates reveal through confidence-ratio early exit (Kohut et al., 2026), and PDC evaluates answer consistency under prefix regeneration (Ichihara et al., 2026). Our target is equality and renewed disagreement between committed response buffers across executions, rather than an early-exit or revision policy. Candidate identity can persist before commitment, while committed buffers can reunite despite distinct candidate histories and execution contexts, so within-run persistence and between-run response-state equality are different events even when both are called stability.

F.3 Reveal schedules, caches and measurement scope

The diffusion literature changes several parts of the inference algorithm. MD4 formalizes masked-data learning (Shi et al., 2024); RADD exposes clean-data conditionals and permits output reuse while the noisy input remains unchanged (Ou et al., 2025). Block Diffusion changes the factorization across blocks (Arriola et al., 2025). Our frozen position order instead persists after response updates and is therefore stale by construction. Recomputing token predictions does not make that order fresh or recover another decoder's trajectory. Similarly, the oracle-based schedule analysis of Chen et al. (2025b) concerns sampling-distribution error; it is not an optimality result for our empirical ULP gate.

Temporal methods also make different algorithmic choices. CCD rectifies sampling using contextual history (Chen et al., 2025a); JoT adapts acceptance using a top-two confidence ratio and spatial context (Kohut et al., 2026). These are genuine inference interventions. Our recorded delay from first fork to first conflict is a retrospective statistic conditioned on eventual conflict, not a tested trigger or an available repair budget.

Parallel and cached decoders change other parts of the algorithm. Fast-dLLM and dKV-Cache reuse intermediate representations (Wu et al., 2026; Ma et al., 2025); Plan for Speed spaces simultaneous commitments (Luxembourg et al., 2026); APD uses an auxiliary AR model to adapt parallel sampling (Israel et al., 2025); EB-Sampler controls parallel unmasking with an entropy-based dependence-error proxy (Ben-Hamu et al., 2025). Our frozen order instead retains position indices across changing states and recomputes tokens at every step; it neither caches KV states nor reduces denoising steps. The distinction also matters for verification: entropy-based parallelism controls an approximation from revealing dependent tokens together, whereas our ULP gate checks a local numerical choice at a preselected position, and neither criterion supplies a guarantee for the other.

Finally, protocol matching matters to comparisons of these methods. Frkovic et al. (2026) find setting-dependent gains from confidence remasking, Kim et al. (2025a) show that EOS padding can alter termination, and Peng et al. (2025) analyze the influence of workload, batching and implementation on efficiency conclusions. We therefore state our 512-step main schedule, the Dream EOS constraint and the 30-prompt cost protocol as explicit scope limits. The semi-autoregressive and two-tokens-per-step schedules of Appendix C.6 reproduce the divergence structure, but no result here establishes parallel-reveal quality, lower deployment latency, or superiority over these samplers.