ON THE IMPORTANCE OF HIDDEN-STATE REFINEMENT IN MASKED DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Masked diffusion language models (MDLMs) generate text by iteratively denoising masked or corrupted token sequences, allowing token predictions to condition on bidirectional context and update multiple positions in parallel. While each denoising step produces a rich layer-wise hidden-state trajectory, prior work typically focuses only on token-level information and discards these continuous hidden states, leaving their refinement dynamics underexplored. In this work, we analyze hidden-state dynamics in MDLMs and find clear evidence of layer-wise and cross-step refinement: smooth evolution across adjacent layers, greater predictive power in deeper layers, gradual degradation under transformer block removal, and strong final-state alignment across consecutive denoising steps. Motivated by these observations, we propose Cross-Step Recurrent Refinement (CSRR), a mechanism that carries hidden-state information across denoising steps to enhance generation. Extensive experiments show that CSRR improves text generation quality, highlighting the critical role of continuous hidden-state refinement in MDLMs.

1 INTRODUCTION

Masked diffusion language models (MDLMs) (Sahoo et al., 2024; Shi et al., 2024; Nie et al., 2025; Ye et al., 2025) have recently gained attention as an alternative to autoregressive models for text generation. Instead of relying on a strict left-to-right factorization, MDLMs start from a corrupted or masked token sequence and iteratively recover token identities through denoising steps (Austin et al., 2021a; He et al., 2022; Zheng et al., 2023). This approach allows token predictions to condition on bidirectional context and update multiple positions simultaneously. Beyond updating masked tokens, each denoising step produces a full layer-wise trajectory of continuous hidden states, which encode intermediate representations of the sequence before final token probabilities are computed.

Prior work (Sahoo et al., 2024; Shi et al., 2024; Nie et al., 2025; Ye et al., 2025; Wu et al., 2025b;a) primarily focuses on token-level behavior across denoising steps, treating the internal transformer computation as a black box. Most sampling procedures (Arriola et al., 2025; Cheng et al., 2025; Wang et al., 2025; Li et al., 2025; Liu et al., 2025b) also carry only token-level information between steps. As a result, the continuous hidden-state trajectory is discarded after each model call, and the next step mainly receives the updated token sequence. This gives rise to two limitations: (1) layer-wise hidden-state dynamics remain underexplored, leaving it unclear whether transformer layers simply transform representations or progressively refine them toward the final prediction; (2) discarding hidden states limits the ability of subsequent steps to leverage the rich continuous refinement information they contain (Figure 1).

Motivated by these limitations, we investigate whether continuous hidden states can be understood and leveraged as refinement signals in MDLM generation. Specifically, we address two research questions:

- **RQ1.** Do hidden states in MDLMs exhibit layer-wise and cross-step refinement?
- **RQ2.** Can previous-step hidden states be leveraged to improve MDLM generation?

Contribution 1. To address RQ1, we conduct a diagnostic study of hidden-state refinement in MDLMs and highlight four key findings. First, hidden states evolve smoothly across adjacent lay-

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.

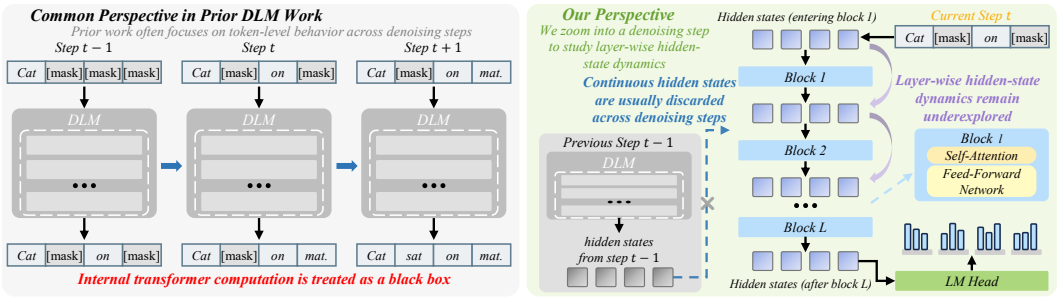


Figure 1: Perspective comparison for MDLM generation. Prior work commonly focuses on token-level behavior across denoising steps and treats the internal transformer computation as a black box. We instead zoom into the current denoising step t to study layer-wise hidden-state dynamics, and ask whether hidden states from the previous step $t-1$ can be leveraged as cross-step refinement signals.

ers, indicating incremental layer-wise refinement rather than abrupt transformations. Second, deeper layers progressively capture more predictive information for the final generated sequence, showing that refinement accumulates toward the output. Third, refinement is distributed across blocks and both self-attention and feedforward sublayers, with no single block dominating the computation. Fourth, final hidden states remain highly consistent across consecutive denoising steps, suggesting that previous-step states provide a reliable context for cross-step reuse. Together, these results demonstrate that MDLM generation involves rich, continuous layer-wise and cross-step refinement beyond token-level updates, highlighting a largely untapped source of information for improving generation.

Contribution 2. To answer RQ2, we propose Cross-Step Recurrent Refinement (CSRR), a lightweight recurrent mechanism that propagates hidden-state information from the previous denoising step to the current one. By injecting prior hidden states into the first block of the current step, CSRR allows the model to leverage accumulated refinement, enhancing the coherence and fidelity of generated sequences. We validate CSRR across multiple MDLM architectures and benchmark datasets, demonstrating consistent improvements in generation quality, convergence speed, and robustness. These results highlight the practical utility of continuous hidden-state refinement and suggest that incorporating such recurrent signals can be a general principle for improving iterative text generation models.

2 Preliminaries for MDLMs

Forward process. Masked diffusion language models define a corruption process over token identities. Given a clean token sequence $x_0 = (x_0^1, \ldots, x_0^N)$, the forward process maps it to a partially masked sequence $\tilde{x}_\tau$ at noise level τ. Let ρ_τ denote the masking rate at this level. A common forward process independently replaces each token with a special mask token $[\texttt{MASK}]$:

$$q(\tilde{x}_\tau^i \mid x_0^i) = \begin{cases} 1-\rho_\tau, & \tilde{x}_\tau^i = x_0^i, \\ \rho_\tau, & \tilde{x}_\tau^i = [\texttt{MASK}]. \end{cases} \tag{1}$$

Reverse process. During generation, let x_s denote the current partially masked sequence at denoising step s. The reverse process uses a learned denoiser $p_\theta(x_0 \mid x_s)$ to predict clean token identities from x_s. The model repeatedly predicts token distributions for currently masked positions, accepts a subset of token predictions according to the sampler, and keeps the remaining positions masked for later denoising steps.

Layer-wise computation. Within a single denoising step, the denoiser first embeds the current token sequence:

$$H_s^0 = \mathrm{Embed}(x_s). \tag{2}$$

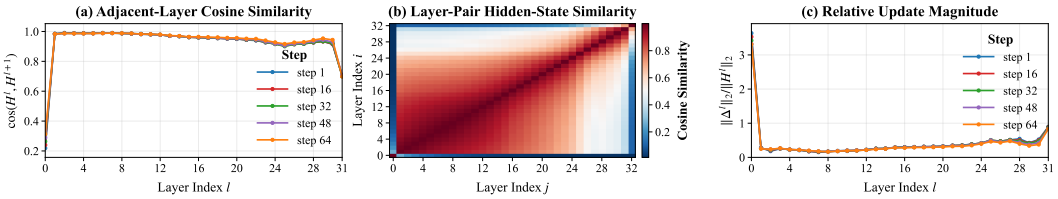


Figure 2: Hidden-state evolution across layers. The implemented figure combines adjacent-layer cosine similarity, layer-pair similarity, and relative update magnitude.

It then applies a stack of L pre-normalized transformer blocks. For block $\ell+1$ with $\ell = 0, \ldots, L-1$, we write the post-self-attention state as

$$A_s^{\ell+1} = H_s^{\ell} + \mathrm{SA}_{\ell+1}\big(\mathcal{N}_{\ell+1}^{\mathrm{attn}}(H_s^{\ell})\big)\,, \tag{3}$$

and the block output after the feed-forward network as

$$H_s^{\ell+1} = A_s^{\ell+1} + \mathrm{FFN}_{\ell+1}\big(\mathcal{N}_{\ell+1}^{\mathrm{ffn}}(A_s^{\ell+1})\big)\,. \tag{4}$$

Here $\mathcal{N}$ denotes the normalization used by the implementation, and FFN denotes the block's feed-forward sublayer. After the last block, a final normalization and the language-model head produce token probabilities:

$$Z_s = \mathcal{N}^{\mathrm{final}}(H_s^L), \tag{5}$$

$$p_\theta(x_0 \mid x_s) = \mathrm{Softmax}(W_{\mathrm{lm}} Z_s)\,. \tag{6}$$

Standard sampling uses the final normalized representation to produce logits, applies token updates, and then recomputes a new hidden-state trajectory at the next denoising step.

3 Understanding Hidden-State Refinement in MDLMs

We first ask whether MDLM generation is iterative not only across denoising steps, but also inside a single model call. We inspect hidden-state trajectories, intermediate predictiveness, block-removal perturbations, and cross-step hidden-state consistency. Together, these diagnostics test whether intermediate hidden states behave like transient activations or like continuous refinement states.

Diagnostic setup. Unless otherwise noted, diagnostics use LLaDA-8B-Instruct (Nie et al., 2025) on the first 10 GSM8K (Cobbe et al., 2021) test examples, with generation length 128, 64 denoising steps, block length 32, temperature 0, and low-confidence remasking. Layer 0 is the embedding output and layer 32 is the final transformer layer. Detailed diagnostic definitions are provided in Appendix D.

3.1 Hidden States Evolve Smoothly Across Layers

We measure whether representations change abruptly from layer to layer using mean token-wise cosine similarity:

$$\mathrm{Sim}_{s,\ell} = \mathbb{E}_i\left[\cos\left(H_{s,i}^{\ell}, H_{s,i}^{\ell+1}\right)\right], \tag{7}$$

where i indexes token positions. We also compute full layer-pair similarity and relative update norm:

$$r_{s,\ell} = \frac{\|H_s^{\ell+1} - H_s^{\ell}\|_2}{\|H_s^{\ell}\|_2}. \tag{8}$$

Adjacent-layer cosine similarity. Figure 2(a) plots adjacent-layer cosine similarity for selected denoising steps. Across all 64 steps and 10 examples, the mean similarity is 0.930, and middle layers are typically above 0.90. Thus most transformer blocks update representations locally rather than replacing them abruptly, which is consistent with an incremental refinement process. The embedding-to-layer-1 transition is an exception, suggesting that the first block converts token embeddings into an initial contextual representation before later blocks perform incremental refinement. Appendix E.1 provides the all-step heatmap, and Appendix E.2 shows the diagnostic across tasks and model variants.

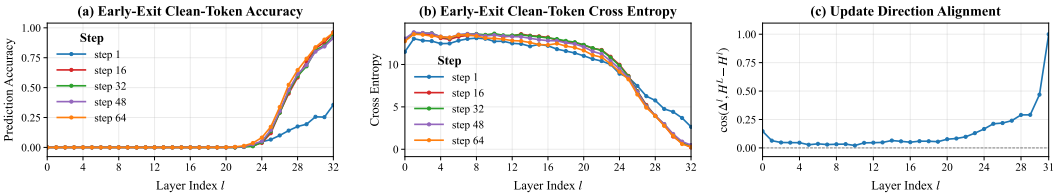


Figure 3: Deeper layers progressively improve denoising predictions. The implemented figure reports early-exit accuracy, cross entropy, and update-direction alignment.

Layer-pair hidden-state similarity. Figure 2(b) extends the measurement to all layer pairs. The strong diagonal band shows that nearby layers remain close while distance grows gradually with depth, matching a continuous refinement trajectory rather than a sequence of unrelated transformations. Appendix E.3 repeats the analysis with linear CKA.

Relative update magnitude. Figure 2(c) measures normalized update size between adjacent layers. High cosine similarity does not imply unchanged states, since each block still contributes a nonzero residual update. The result is therefore not that layers are idle, but that most layers make bounded corrections to an evolving hidden state. Together, the three panels support a smooth refinement view of layer-wise computation.

3.2 Deeper Layers Become More Predictive

Smoothness alone does not show that updates are useful. We therefore apply the model's final normalization and shared LM head to intermediate hidden states, and compare predictions with the final generated sequence used as the pseudo clean target. For layer ℓ,

$$Z_s^\ell = W_{\text{lm}} \mathcal{N}^{\text{final}}(H_s^\ell). \tag{9}$$

After suppressing the mask-token logit, we compute token accuracy against the final generated sequence:

$$\text{Acc}_{s,\ell} = \text{Acc}\left(\arg\max Z_s^\ell, x_{\text{final}}\right). \tag{10}$$

Early-exit accuracy and cross entropy. Panels (a) and (b) of Figure 3 apply the shared LM head to intermediate layers. Early layers have little direct token-prediction ability, while late layers become much more predictive and have lower cross entropy. Thus the smooth trajectory is not merely geometrically smooth; it also accumulates information that becomes increasingly readable by the output head. Appendix E.4 provides top-k accuracy, token-probability, and entropy diagnostics.

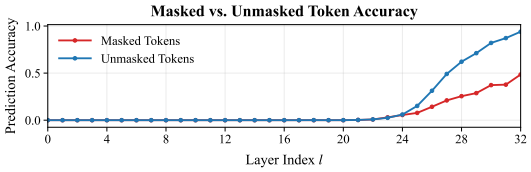


Figure 4: Masked-token and unmasked-token early-exit accuracy curves.

Update-direction alignment. Figure 3(c) measures whether the local update $H^{\ell+1} - H^\ell$ points toward the final-layer residual direction $H^L - H^\ell$. Positive alignment indicates movement toward the final representation rather than orthogonal drift. This directional evidence complements the early-exit results and supports a view in which intermediate layers participate in denoising, rather than only preserving a stable representation. Appendix E.5 decomposes adjacent-layer prediction changes into wrong-to-correct fixes and correct-to-wrong breaks.

Masked vs. unmasked token accuracy. Figure 4 separates still-masked and already-unmasked positions. Both improve with depth, while unmasked positions remain easier because they contain more lexical evidence. The masked-token gains show that late layers recover genuinely hidden content rather than only preserving tokens that are already visible.

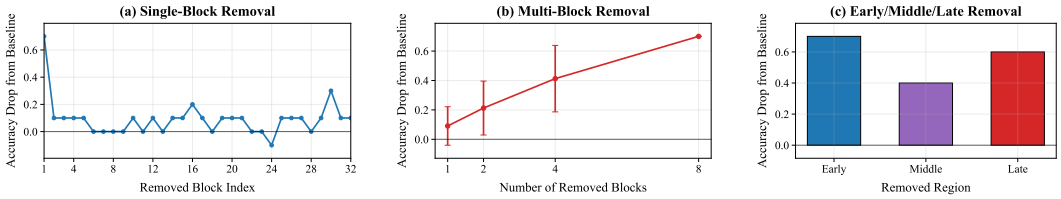


Figure 6: Block-removal perturbations. The diagnostic run has baseline accuracy 70% on 10 GSM8K examples. Single-block and short-window removals often degrade gradually, while larger structured removals can collapse.

Hidden-state trajectory. Figure 5 projects hidden states into a shared two-dimensional PCA space. The trajectory provides a qualitative companion to the accuracy and alignment diagnostics, showing that states move along a structured path across layers rather than scattering randomly. This visualization makes the layer-wise refinement interpretation more explicit.

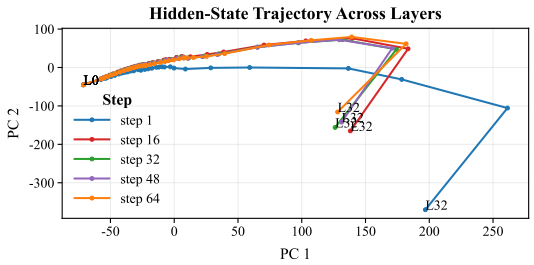


Figure 5: PCA trajectory of hidden states across layers.

3.3 Refinement Is Distributed Across Blocks

If denoising were concentrated in a single critical block, removing most other blocks would have little effect. We test this by replacing selected transformer blocks with the identity during inference, including single blocks, consecutive windows of 1, 2, 4, or 8 blocks, early/middle/late four-block regions, and self-attention-only or FFN-only sublayers.

Single-block removal. Figure 6(a) removes one transformer block at a time. Most removals preserve nonzero performance, with mean accuracy 60.9% versus the 70.0% baseline. Thus no single middle block alone carries the whole denoising computation, although some block positions are more sensitive than others.

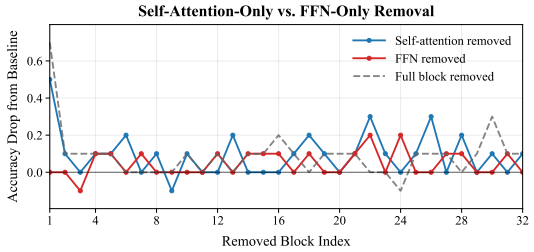


Figure 7: Self-attention-only and FFN-only removal diagnostics.

Consecutive multi-block removal. Figure 6(b) removes consecutive windows of 1, 2, 4, or 8 blocks. Removing two or four blocks lowers mean accuracy to 48.8% and 28.8%, while removing eight collapses the 10-example run. This does not mean layers are redundant. Rather, refinement is distributed across blocks, but still depends on preserving enough of the ordered layer trajectory.

Early, middle, and late region removal. Figure 6(c) removes coarse four-block regions. Removing blocks 1–4 collapses accuracy to 0%, removing blocks 15–18 leaves 30%, and removing blocks 29–32 leaves 10%. Thus early representation formation and late decoding-oriented refinement are both important, while different regions affect the computation in different ways.

Self-attention-only vs. FFN-only removal. Figure 7 removes only the self-attention or FFN sublayer at each block. The pattern again supports distributed refinement across many blocks and both sublayers, rather than a single isolated component carrying all denoising computation. The main exception is the first self-attention sublayer, whose removal causes a disproportionately large degradation, suggesting that most blocks behave like incremental refinement steps while the first self-attention sublayer forms the initial context-mixed state.

3.4 Previous-Step Final States Are Consistent

The previous observations support layer-wise refinement within one denoising step. We next measure whether the final hidden state from one step remains aligned with the next step's final hidden state:

$$\mathrm{CrossSim}_s = \mathbb{E}_i \left[\cos\left(H^L_{s-1,i}, H^L_{s,i}\right) \right]. \tag{11}$$

Cross-step final-state consistency. Figure 8 measures cosine similarity between final hidden states at steps $s-1$ and s. Across steps 2–64, the mean valid-token similarity remains roughly 0.96–0.99. The previous final hidden state is therefore not an unrelated activation, but already close to the next step's refinement state and a plausible source of cross-step continuous context.

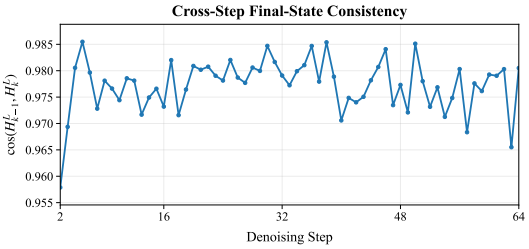


Figure 8: Final hidden states remain highly consistent across consecutive denoising steps.

Takeaway. Together, the diagnostics suggest that MDLMs refine text across denoising steps in token space and across transformer layers in continuous hidden-state space. Standard sampling carries token updates forward but discards the continuous refinement trajectory, motivating a cross-step path for hidden-state refinement.

4 Cross-Step Recurrent Refinement

The observations above suggest that MDLM generation contains both discrete token refinement across denoising steps and continuous hidden-state refinement inside each model call. Standard sampling preserves only the former, since each step receives the updated token sequence but discards the hidden-state trajectory computed in the previous step. Cross-Step Recurrent Refinement (CSRR) adds a lightweight recurrent path for this missing continuous state while leaving the original masked-token update rule unchanged.

4.1 Formulation

At denoising step s, a standard MDLM maps the current partially masked sequence to a layer-wise hidden-state trajectory and then updates tokens according to the sampler:

$$x_s \to H^0_s, \ldots, H^L_s \to p_\theta(x_0 \mid x_s) \to x_{s+1}. \tag{12}$$

After the token update, H^L_s is normally discarded. CSRR instead reuses the detached final-block hidden state from the previous denoising step.

$$M_s = \mathrm{SG}\left(H^L_{s-1}\right), \tag{13}$$

where SG denotes stop-gradient. At the first denoising step, no previous hidden state is available, so the model runs exactly as the original MDLM.

CSRR incorporates the previous-step hidden state into the first transformer block of the current denoising step. The insertion point is placed after the ordinary self-attention sublayer and before the FFN sublayer. Let A^1_s denote the intermediate hidden state after the self-attention update in the first block. CSRR attends from this current-step state to the cached previous-step hidden state.

$$Q^1_s = \mathcal{N}^{\mathrm{attn}}_1(A^1_s) W^{\mathrm{cross}}_Q, \tag{14}$$

$$K_s = \mathcal{N}^{\mathrm{attn}}_1(M_s) W^{\mathrm{cross}}_K, \tag{15}$$

$$V_s = \mathcal{N}^{\mathrm{attn}}_1(M_s) W^{\mathrm{cross}}_V, \tag{16}$$

$$C^1_s = \mathrm{Attn}(Q^1_s, K_s, V_s) W^{\mathrm{cross}}_O. \tag{17}$$

The resulting cross-attention output is added as a scaled residual.

$$\widetilde{A}^1_s = A^1_s + \alpha C^1_s. \tag{18}$$

Table 1: Main reasoning results on LLaDA-8B-Base and LLaDA-8B-Instruct. Unless otherwise specified, CSRR denotes the joint-training setting. The module-only variant trains only the newly introduced modules. All values are percentages. Columns under each benchmark denote generation length. Parenthesized values denote absolute improvements over the corresponding baseline.

NFE/Gen Len	Method	Countdown		GSM8K		SVAMP	
		128	256	128	256	128	256
LLaDA-8B-Base							
	Baseline	7.42	10.16	17.06	60.88	48.00	77.67
1/2	CSRR (module only)	15.23 (+7.81)	13.67 (+3.51)	19.03 (+1.97)	63.38 (+2.50)	56.33 (+8.33)	76.67 (-1.00)
	CSRR	17.58 (+10.16)	17.58 (+7.42)	33.59 (+16.53)	70.66 (+9.78)	66.67 (+18.67)	81.00 (+3.33)
	Baseline	6.25	11.72	16.68	64.75	52.00	80.33
1	CSRR (module only)	16.02 (+9.77)	12.50 (+0.78)	27.07 (+10.39)	70.13 (+5.38)	54.33 (+2.33)	78.33 (-2.00)
	CSRR	18.36 (+12.11)	21.88 (+10.16)	37.38 (+20.70)	79.23 (+14.48)	71.33 (+19.33)	82.67 (+2.34)
LLaDA-8B-Instruct							
	Baseline	17.19	13.67	68.31	76.04	83.67	85.67
1/2	CSRR (module only)	23.83 (+6.64)	20.31 (+6.64)	68.92 (+0.61)	76.35 (+0.31)	86.33 (+2.66)	85.67 (+0.00)
	CSRR	28.52 (+11.33)	24.22 (+10.55)	70.58 (+2.27)	78.54 (+2.50)	86.00 (+2.33)	86.00 (+0.33)
	Baseline	23.44	21.48	70.43	78.62	87.00	86.67
1	CSRR (module only)	23.05 (-0.39)	23.44 (+1.96)	71.95 (+1.52)	80.06 (+1.44)	89.67 (+2.67)	88.33 (+1.66)
	CSRR	38.28 (+14.84)	29.30 (+7.82)	74.00 (+3.57)	83.47 (+4.85)	89.00 (+2.00)	89.00 (+2.33)

The first block then feeds $\widetilde{A}_s^1$ into its ordinary FFN sublayer. The full CSRR inference procedure is summarized in Appendix B.1.

Training follows the original supervised denoising objective while exposing the model to consecutive denoising forward passes. After each forward pass, the detached final-block hidden state is reused by the next forward pass, matching the recurrent computation used at inference time. For a training segment with K supervised denoising forward passes, let $p_\theta^{(k)}$ denote the predicted distribution from the k-th forward pass over supervised token positions, and let x_{gold} denote the corresponding clean targets. We minimize the averaged cross entropy:

$$\mathcal{L}_{\text{CSRR}} = \frac{1}{K} \sum_{k=1}^{K} \text{CE}\left(p_\theta^{(k)}, x_{\text{gold}}\right). \tag{19}$$

4.2 Design Rationale

The implemented configuration uses the previous final-block hidden state and incorporates it after the first self-attention update of the current step. The previous final-block state is motivated by the cross-step consistency diagnostic, where consecutive final hidden states remain highly aligned. The insertion site is motivated by the block- and sublayer-removal results, where the first self-attention sublayer acts as a special context-forming component while later blocks largely follow the incremental refinement pattern. Appendix F.5 further sweeps the source layer used for extraction and the injection layer used for incorporation, showing that CSRR remains effective for many layer pairs and that the default choice is among the stronger settings.

Table 2: Code-generation results on LLaDA-8B-Instruct. NFE equals generation length in all code settings. All values are percentages, and Δ denotes the absolute improvement over the baseline.

Code benchmark	Gen Len 128			Gen Len 256		
	Baseline	CSRR	Δ	Baseline	CSRR	Δ
HumanEval	27.44	29.88	+2.44	37.20	43.90	+6.70
HumanEval+	20.12	22.56	+2.44	29.88	34.15	+4.27
MBPP	49.03	50.58	+1.55	49.03	50.97	+1.94

4.3 Empirical Validation

4.3.1 Experimental Setup

We mainly evaluate CSRR on LLaDA-8B-Base and LLaDA-8B-Instruct (Nie et al., 2025), and further conduct extension tests on Dream-7B-Base and Dream-7B-Instruct (Ye et al., 2025). For

each backbone, we compare against the same sampler without cross-step hidden-state reuse. CSRR is fine-tuned on the S1K dataset (Muennighoff et al., 2025) using consecutive denoising forward passes. We mainly evaluate reasoning benchmarks including Countdown (Pan et al., 2025), GSM8K (Cobbe et al., 2021), and SVAMP (Patel et al., 2021), and additionally use HumanEval (Chen et al., 2021), HumanEval+ (Liu et al., 2023), and MBPP (Austin et al., 2021b) as code evaluation benchmarks. More implementation details and the evaluation prompt templates are listed in Appendices C and G, respectively.

4.3.2 Evaluation Results

Main reasoning results. Table 1 reports the main results on Countdown, GSM8K, and SVAMP using LLaDA-8B-Base and LLaDA-8B-Instruct. Unless otherwise specified, CSRR refers to the joint-training setting that trains both the newly introduced modules and LoRA adapters on the original backbone. CSRR improves the baseline across all reported reasoning configurations, with particularly large gains on Countdown and GSM8K for the base model. We also include a module-only variant, which often improves over the baseline and indicates that the added recurrent path contributes useful information even when only the new modules are trained.

Table 3: Countdown results with KV-cache support on LLaDA-8B-Base and LLaDA-8B-Instruct. All values are percentages, and Δ denotes the absolute improvement over the baseline.

NFE/Gen Len	Gen Len 128			Gen Len 256		
	Baseline	CSRR	Δ	Baseline	CSRR	Δ
LLaDA-8B-Base						
1/2	8.59	17.19	+8.60	7.81	19.53	+11.72
1	7.03	22.66	+15.63	10.94	21.48	+10.54
LLaDA-8B-Instruct						
1/2	17.19	25.39	+8.20	15.62	21.09	+5.47
1	21.48	26.56	+5.08	16.41	23.44	+7.03

Code benchmarks. Table 2 evaluates CSRR on HumanEval, HumanEval+, and MBPP using LLaDA-8B-Instruct. CSRR consistently improves pass rates across all reported code-generation settings, showing that the proposed hidden-state reuse is not limited to mathematical reasoning tasks.

Extension to Dream models. Figure 9 reports extension results on Dream-7B-Base and Dream-7B-Instruct. CSRR improves the corresponding baseline on all reported reasoning benchmarks, suggesting that the recurrent hidden-state path transfers beyond the LLaDA model family.

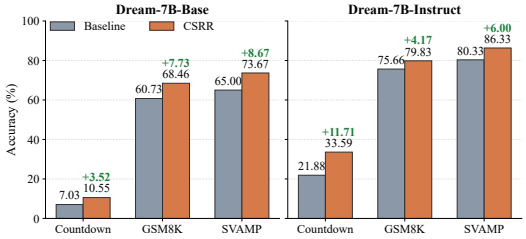


Figure 9: Reasoning results on Dream-7B-Base and Dream-7B-Instruct. All settings use generation length 256 and NFE 128.

KV-cache support. Table 3 evaluates CSRR with KV-cache support on Countdown. The gains remain consistent across model variants and generation budgets, indicating that the recurrent hidden-state path is compatible with the cached inference setting. Full KV-cache results across Countdown, GSM8K, and SVAMP are provided in Appendix F.1.

Parameter and inference overhead. Figure 10 compares the computational cost of CSRR against the original LLaDA-8B-Instruct sampler. The added cross-attention module increases the parameter count from 8.016B to 8.082B, corresponding to only 0.82% additional parameters. With the same prompt and sampling hyperparameters, average inference time increases from 2.969s to 3.082s, a 3.80% overhead. Thus, CSRR adds a small parameter footprint and modest runtime cost.

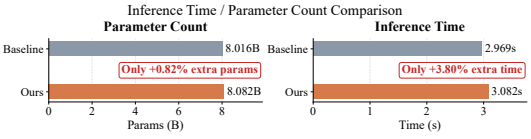


Figure 10: Inference-time and parameter-count comparison between the original LLaDA-8B-Instruct baseline and CSRR. Runtime is measured with the same prompt and sampling hyperparameters, using generation length 256 and NFE 128.

Additional S1K-matched fine-tuning controls, fixed-scale results, block-size results, and source-injection layer sweeps are provided in Appendices F.2, F.3, F.4, and F.5, respectively.

5 Related Work

Masked diffusion language models. Diffusion language modeling (Yu et al., 2025; Wu et al., 2025c; Zhong et al., 2026b) adapts denoising diffusion (Ho et al., 2020; Song & Ermon, 2019; Podell et al., 2023; Zhong et al., 2025) to discrete token sequences. Early text generation models explored categorical corruption processes (Austin et al., 2021a), BERT-style initialization and masked reconstruction (He et al., 2022), reparameterized discrete objectives (Zheng et al., 2023), and semi-autoregressive simplex diffusion (Han et al., 2023). Masked diffusion language models (MDLMs) (Sahoo et al., 2024; Shi et al., 2024) further simplify this paradigm by formulating generation as iterative masked-token denoising, allowing each update to condition on bidirectional context rather than a strict left-to-right prefix. Recent DLMs have been scaled through instruction tuning (Ye et al., 2023), adaptation from autoregressive backbones (Gong et al., 2024), open model releases (Tae et al., 2025; Nie et al., 2025; Ye et al., 2025; Zhu et al., 2025; Bie et al., 2025), and extensions to multimodal, reasoning, and long-context settings (Yang et al., 2025; Zhao et al., 2025; Liu et al., 2025a). These works establish MDLMs as a practical alternative generation paradigm, but they mainly characterize denoising through token-level states or final output distributions.

Inference and latent refinement in DLMs. Another line of work improves the efficiency and flexibility of DLM sampling. Existing methods study blockwise or hybrid generation (Arriola et al., 2025; Cheng et al., 2025), variable-length and sequential denoising (Li et al., 2025; Liu et al., 2025b), faster-than-autoregressive or parallel decoding (Wang et al., 2025; Chen et al., 2025), and cache-based or system-level acceleration (Wu et al., 2025b;a; Liu et al., 2025c; Ma et al., 2025). Recent variants also enrich hard remasking with soft masking (Hersche et al., 2025), coupled discrete-continuous diffusion (Zheng et al., 2025), progressive token evolution (Zhong et al., 2026a), hierarchical semantic prediction (Zhou et al., 2025a), and latent reasoning trajectories (Zhou et al., 2025b; Kang et al., 2025). These approaches modify the sampler, token-transition process, or explicit latent variables. Our work is complementary: we inspect the hidden-state trajectory that is already produced inside each MDLM forward pass, and use it as a continuous cross-step refinement signal without changing the masked-token update rule.

Recurrent and feedback transformers. Transformer recurrence has been explored as a way to increase effective computation, extend memory, or improve iterative reasoning. Universal and depth-recurrent transformers reuse blocks across depth or test-time computation (Dehghani et al., 2018; Geiping et al., 2026; Oncescu et al., 2026), while segment-level and external-memory models carry information across long contexts (Dai et al., 2019; Rae et al., 2019; Wu et al., 2022; Bulatov et al., 2022; Hutchins et al., 2022). Other architectures introduce feedback from previous time steps (Fan et al., 2020) or recurrent-style sequence modeling in transformer-like networks (Sun et al., 2023; Peng et al., 2023). CSRR shares the motivation of reusing continuous computation, but differs in setting and mechanism: it reuses the previous denoising step's hidden states in an MDLM, rather than adding recurrence over autoregressive tokens, long-context segments, or repeated depth alone.

6 Conclusion

In this paper, we studied hidden-state refinement in masked diffusion language models. Our diagnostics show that hidden states evolve smoothly across layers, become more predictive with depth, depend on distributed block-wise computation, and stay aligned across denoising steps. These findings reveal a continuous refinement trajectory that complements discrete token refinement. Motivated by this view, we proposed Cross-Step Recurrent Refinement, which carries previous-step hidden states through a lightweight cross-attention path while preserving masked-token sampling. Extensive experiments across multiple models and benchmarks show that CSRR improves MDLM generation with modest overhead.

AI USE STATEMENT

The authors used large language models to aid and polish the writing of this paper and to assist in implementing portions of the research code. The authors take responsibility for the final manuscript, code, and all scientific claims and results.

REFERENCES

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

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

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

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

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

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. dparallel: Learnable parallel decoding for dllms. *arXiv preprint arXiv:2509.26488*, 2025.

Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*, 2025.

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

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 2978–2988, 2019.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. *arXiv preprint arXiv:2002.09402*, 2020.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2026.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. *arXiv preprint arXiv:2410.17891*, 2024.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11575–11596, 2023.

Zhengfu He, Tianxiang Sun, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. Diffusionbert: Improving generative masked language models with diffusion models. *arXiv preprint arXiv:2211.15029*, 2022.

Michael Hersche, Samuel Moor-Smith, Thomas Hofmann, and Abbas Rahimi. Soft-masked diffusion language models. *arXiv preprint arXiv:2510.17206*, 2025.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. *Advances in neural information processing systems*, 35:33248–33261, 2022.

Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Nicklas Majamäki, Navdeep Jaitly, Yi-An Ma, and Lianhui Qin. Ladir: Latent diffusion enhances llms for text reasoning. *arXiv preprint arXiv:2510.08558*, 2025.

Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. Beyond fixed: Training-free variable-length denoising for diffusion large language models. *arXiv preprint arXiv:2508.00819*, 2025.

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

Xiaoran Liu, Yuerong Song, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. Longllada: Unlocking long context capabilities in diffusion llms. *arXiv preprint arXiv:2506.14429*, 2025a.

Yangzhou Liu, Yue Cao, Hao Li, Gen Luo, Zhe Chen, Weiyun Wang, Xiaobo Liang, Biqing Qi, Lijun Wu, Changyao Tian, et al. Sequential diffusion language models. *arXiv preprint arXiv:2509.24007*, 2025b.

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

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

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. *arXiv preprint arXiv:2502.09992*, 2025.

Costin-Andrei Oncescu, Depen Morwani, Samy Jelassi, Alexandru Meterez, Mujin Kwun, and Sham Kakade. The recurrent transformer: Greater effective depth and efficient decoding. *arXiv preprint arXiv:2604.21215*, 2026.

Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. Tinyzero. `https://github.com/Jiayi-Pan/TinyZero`, 2025. Accessed: 2026-05-22.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, et al. Rwkv: Reinventing rnns for the transformer era. In *Findings of the association for computational linguistics: EMNLP 2023*, pp. 14048–14077, 2023.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Jaesung Tae, Hamish Ivison, Sachin Kumar, and Arman Cohan. Tess 2: A large-scale generalist diffusion language model. *arXiv preprint arXiv:2502.13917*, 2025.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*, 2025.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328*, 2025a.

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. *arXiv preprint arXiv:2505.22618*, 2025b.

Linyu Wu, Linhao Zhong, Wenjie Qu, Yuexin Li, Yue Liu, Shengfang Zhai, Chunhua Shen, and Jiaheng Zhang. Dmark: Order-agnostic watermarking for diffusion large language models. *arXiv preprint arXiv:2510.02902*, 2025c.

Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022.

Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. Mmada: Multimodal large diffusion language models. *arXiv preprint arXiv:2505.15809*, 2025.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Jiasheng Ye, Zaixiang Zheng, Yu Bao, Lihua Qian, and Quanquan Gu. Diffusion language models can perform many tasks with scaling and instruction-finetuning. *arXiv preprint arXiv:2308.12219*, 2023.

Runpeng Yu, Qi Li, and Xinchao Wang. Discrete diffusion in large language and multimodal models: A survey. *arXiv preprint arXiv:2506.13759*, 2025.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025.

Huangjie Zheng, Shansan Gong, Ruixiang Zhang, Tianrong Chen, Jiatao Gu, Mingyuan Zhou, Navdeep Jaitly, and Yizhe Zhang. Continuously augmented discrete diffusion model for categorical generative modeling. *arXiv preprint arXiv:2510.01329*, 2025.

Lin Zheng, Jianbo Yuan, Lei Yu, and Lingpeng Kong. A reparameterized discrete diffusion model for text generation. *arXiv preprint arXiv:2302.05737*, 2023.

Linhao Zhong, Fan Li, Yi Huang, Jianzhuang Liu, Renjing Pei, and Fenglong Song. Outdreamer: Video outpainting with a diffusion transformer. *arXiv preprint arXiv:2506.22298*, 2025.

Linhao Zhong, Linyu Wu, Bozhen Fang, Tianjian Feng, Chenchen Jing, Wen Wang, Jiaheng Zhang, Hao Chen, and Chunhua Shen. Beyond hard masks: Progressive token evolution for diffusion language models. *arXiv preprint arXiv:2601.07351*, 2026a.

Linhao Zhong, Linyu Wu, Wen Wang, Yuling Xi, Chenchen Jing, Jiaheng Zhang, Hao Chen, and Chunhua Shen. Efficient self-evaluation for diffusion language models via sequence regeneration. *arXiv preprint arXiv:2603.02760*, 2026b.

Cai Zhou, Chenyu Wang, Dinghuai Zhang, Shangyuan Tong, Yifei Wang, Stephen Bates, and Tommi Jaakkola. Next semantic scale prediction via hierarchical diffusion language models. *arXiv preprint arXiv:2510.08632*, 2025a.

Cai Zhou, Chenxiao Yang, Yi Hu, Chenyu Wang, Chubin Zhang, Muhan Zhang, Lester Mackey, Tommi Jaakkola, Stephen Bates, and Dinghuai Zhang. Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner. *arXiv preprint arXiv:2510.03206*, 2025b.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, et al. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

A Appendix Overview

This appendix is organized to provide methodological details, implementation details, diagnostic definitions, additional hidden-state analyses, supplementary experimental results, and evaluation prompts.

- **Appendix B: More Methodological Details**
  - Appendix B.1: CSRR Inference Algorithm
- **Appendix C: More Implementation Details**
  - Appendix C.1: Detailed Descriptions of Training Dataset and Evaluation Benchmarks
  - Appendix C.2: Training Configurations for CSRR
  - Appendix C.3: Inference and Evaluation Setup
- **Appendix D: Detailed Diagnostic Definitions**
  - Appendix D.1: Token Groups and Aggregation
  - Appendix D.2: Hidden-State Similarity Diagnostics
  - Appendix D.3: Early-Exit Prediction Diagnostics
  - Appendix D.4: Correction and Break Diagnostics
- **Appendix E: More Hidden-State Refinement Analyses**
  - Appendix E.1: All-Step Layer Smoothness
  - Appendix E.2: Adjacent-Layer Smoothness Across Tasks and LLaDA Variants
  - Appendix E.3: CKA-Based Layer Similarity
  - Appendix E.4: Early-Exit Top-k, Probability, and Entropy Diagnostics
  - Appendix E.5: Correction and Break Rates
- **Appendix F: More Experimental Results**
  - Appendix F.1: KV-Cache Results Across Benchmarks
  - Appendix F.2: S1K-Matched Fine-Tuning Control
  - Appendix F.3: Fixed-Scale Results
  - Appendix F.4: Results with Different Block Sizes
  - Appendix F.5: Source and Injection Layer Positions
- **Appendix G: Evaluation Prompts**
  - Appendix G.1: Chat-Template Wrapping
  - Appendix G.2: Reasoning Prompts
  - Appendix G.3: Code-Generation Prompts

B More Methodological Details

This section provides additional methodological details for CSRR, focusing on the inference procedure that carries a previous-step hidden state across denoising steps.

B.1 CSRR Inference Algorithm

Algorithm S1 summarizes CSRR inference. The discrete token update is exactly the same sampler used by the corresponding MDLM baseline. The only additional state is the detached final hidden state from the previous denoising step, which is reused by the next step through the cross-step recurrent path described in the main paper.

When M_s is available, $f_\theta(x_s; M_s, \alpha)$ denotes the forward pass that injects M_s after the first self-attention sublayer and before the first FFN sublayer. When M_s is unavailable at the first step, the model reduces to the original MDLM forward pass. The sampler update rule may correspond to the same semi-autoregressive block sampling procedure used in the baseline, including low-confidence remasking and the selected NFE setting.

Algorithm S1 CSRR inference

Input initial masked sequence x_1, denoising steps S, model f_θ, sampler update rule Update, recurrent scale α
$M_1 \leftarrow \varnothing$
for $s = 1, \ldots, S$ **do**
  if $M_s = \varnothing$ **then**
    $(Z_s, H_s^L) \leftarrow f_\theta(x_s)$
  else
    $(Z_s, H_s^L) \leftarrow f_\theta(x_s; M_s, \alpha)$
  end if
  $x_{s+1} \leftarrow \mathrm{Update}(x_s, Z_s)$
  $M_{s+1} \leftarrow \mathrm{SG}(H_s^L)$
end for
return x_{S+1}

C More Implementation Details

This section provides implementation details for the experiments, including the training dataset, evaluation benchmarks, training configuration, and inference-time evaluation setup.

C.1 Detailed Descriptions of Training Dataset and Evaluation Benchmarks

Training dataset. We use a compact supervised dataset to adapt the recurrent hidden-state path while preserving the model's ability to produce explicit reasoning traces.

- **S1K** (Muennighoff et al., 2025). We fine-tune CSRR on the S1K dataset, a compact reasoning dataset containing 1,000 diverse and difficult questions paired with distilled reasoning traces and solutions. We preserve the long-form reasoning format, which matches our goal of adapting the model to multi-step denoising trajectories rather than training it only on short final answers. The targets are formatted with an explicit reasoning field and a boxed final answer so that the model receives both intermediate solution structure and a clear answer boundary.

Reasoning benchmarks. We evaluate mathematical reasoning with benchmarks that require different forms of multi-step inference, including symbolic arithmetic search, natural-language arithmetic, and robustness to problem rewrites.

- **Countdown** (Pan et al., 2025). Countdown evaluates arithmetic planning under explicit compositional constraints. Each instance provides a small set of numbers and a target value, and the model must construct an expression using the provided numbers and basic arithmetic operations to reach the target. This benchmark is useful for testing whether cross-step refinement improves search-like symbolic reasoning, since a valid answer must satisfy both the numerical target and the input-number constraints.
- **GSM8K** (Cobbe et al., 2021). GSM8K is a grade-school math word-problem benchmark with linguistically diverse questions that typically require multi-step arithmetic reasoning. We evaluate on the test split and extract the final numeric answer from the generated solution. GSM8K serves as the main natural-language arithmetic benchmark because it combines textual understanding, intermediate calculation, and final-answer consistency.
- **SVAMP** (Patel et al., 2021). SVAMP is a math word-problem benchmark designed to stress robustness to variations in problem wording and structure. Each example contains a short narrative, a question, an equation, and a numeric answer. Compared with GSM8K, SVAMP places more emphasis on avoiding shallow pattern matching, making it a complementary test of whether the model can maintain the intended semantic relations during iterative generation.

Code benchmarks. We further evaluate code generation to test whether the method transfers from mathematical reasoning to executable program synthesis.

- **HumanEval** (Chen et al., 2021). HumanEval evaluates Python function synthesis from a function signature and natural-language docstring. The model is asked to complete the function, and correctness is measured by executing the generated code against the provided unit tests. We use it to test whether the proposed hidden-state reuse mechanism transfers beyond mathematical reasoning to functional code generation.
- **HumanEval+** (Liu et al., 2023). HumanEval+ follows the HumanEval problem format but evaluates generated programs with a substantially expanded test suite. It is a stricter functional-correctness benchmark because additional tests cover more edge cases and reduce the chance that a solution passes by matching only the original sparse tests.
- **MBPP** (Austin et al., 2021b). MBPP contains short Python programming tasks described in natural language, with reference solutions and executable tests. We use the sanitized split, whose tasks are manually checked for clearer specifications and more reliable tests. MBPP complements HumanEval by emphasizing beginner-to-intermediate programming patterns such as list processing, string manipulation, arithmetic, and simple control flow.

C.2 Training Configurations for CSRR

The main LLaDA backbones are LLaDA-8B-Base and LLaDA-8B-Instruct, and the extension settings additionally cover Dream-7B-Base and Dream-7B-Instruct. We consider two training regimes. In the main setting, we jointly train the newly introduced CSRR modules and lightweight LoRA adapters on the original backbone for 10,000 steps. The LoRA adapters are applied to the query, key, and value projections with rank 128, alpha 256, dropout 0.05, and no bias tuning. We also report a module-only setting, where the pretrained backbone is frozen and only the newly added CSRR parameters are optimized for 20,000 steps. The main CSRR training run uses batch size 4, maximum sequence length 1024, learning rate 1×10^{-5}, weight decay 0.1, and gradient clipping at 1.0. The training collator simulates consecutive denoising forward passes with $K = 2$. It samples a target block of length 512, supervises tokens inside the selected block, updates one predicted token per forward stage, and samples the cross-attention scale from $\{0.5, 1.0, 1.5, 2.0\}$ during training.

C.3 Inference and Evaluation Setup

We evaluate CSRR with the same semi-autoregressive masked-token sampling procedure used by the corresponding baseline. Unless otherwise stated, generation uses block length 32, low-confidence remasking, and generation temperature 0.5. We evaluate cross-attention scales in $\{0.5, 1.0, 1.5, 2.0\}$ and report the best value in the main results. We use NFE, short for number of function evaluations, to denote the total number of model forward evaluations during generation. We consider generation lengths 128 and 256, with NFE equal to either the generation length or half of it. For reasoning tasks, accuracy is computed after task-specific answer extraction. GSM8K and SVAMP use the extracted numeric answer, while Countdown additionally verifies that the generated expression reaches the target value using the provided numbers. For code tasks, we evaluate functional correctness by extracting the generated Python program and executing it against the benchmark tests.

D Detailed Diagnostic Definitions

This section defines the main diagnostic quantities used in the main paper and the supplementary material. The definitions match the diagnostics reported in our analysis and use the final generated sequence as the pseudo clean target for early-exit prediction diagnostics.

D.1 Token Groups and Aggregation

Let P denote the prompt length and let the current semi-autoregressive block span positions $[B_{\text{start}}, B_{\text{end}})$. We use *prompt* for positions before P, *generated* for generated positions before the current block end, *valid* for all positions before the current block end, and *current block* for positions inside $[B_{\text{start}}, B_{\text{end}})$. For early-exit diagnostics, the *all* group refers to generated positions before the current block end, while *masked* and *unmasked* are generated positions that are respectively equal or not equal to the mask token before the current transfer step. We compute token-level

quantities inside the selected group and aggregate them across recorded denoising steps and examples.

D.2 Hidden-State Similarity Diagnostics

For a token group G, adjacent-layer cosine similarity is computed as

$$\mathrm{Sim}_{s,\ell}(G) = \frac{1}{|G|} \sum_{i \in G} \cos\left(H_{s,i}^{\ell}, H_{s,i}^{\ell+1}\right). \tag{S1}$$

The layer-pair heatmap uses the same token-wise cosine computation for all layer pairs:

$$\mathrm{PairSim}_{s,\ell,j} = \frac{1}{N} \sum_{i=1}^{N} \cos\left(H_{s,i}^{\ell}, H_{s,i}^{j}\right). \tag{S2}$$

The relative update magnitude normalizes the update size by the current hidden-state norm:

$$r_{s,\ell} = \frac{\|H_s^{\ell+1} - H_s^{\ell}\|_2}{\|H_s^{\ell}\|_2}. \tag{S3}$$

For CKA, we form centered token-feature matrices $\bar{X}_s^{\ell}$ and $\bar{X}_s^{j}$ from selected token positions, compute Gram matrices $K_\ell = \bar{X}_s^{\ell}(\bar{X}_s^{\ell})^\top$ and $K_j = \bar{X}_s^{j}(\bar{X}_s^{j})^\top$, and report

$$\mathrm{CKA}(\ell, j) = \frac{\langle K_\ell, K_j \rangle_F}{\|K_\ell\|_F \|K_j\|_F}. \tag{S4}$$

D.3 Early-Exit Prediction Diagnostics

For each intermediate hidden state H_s^{ℓ}, we apply the model's final normalization and shared LM head to obtain intermediate logits:

$$Z_s^{\ell} = W_{\mathrm{lm}} \mathcal{N}^{\mathrm{final}}(H_s^{\ell}). \tag{S5}$$

Let y_i denote the token at position i in the final generated sequence and let $p_{s,i}^{\ell} = \mathrm{Softmax}(Z_{s,i}^{\ell})$. Early-exit accuracy and cross entropy over a token group G are

$$\mathrm{Acc}_{s,\ell}(G) = \frac{1}{|G|} \sum_{i \in G} \mathbf{1}\left[\arg\max_v p_{s,i}^{\ell}(v) = y_i\right], \tag{S6}$$

$$\mathrm{CE}_{s,\ell}(G) = -\frac{1}{|G|} \sum_{i \in G} \log p_{s,i}^{\ell}(y_i). \tag{S7}$$

The top-k accuracy, target-token probability, maximum probability, and entropy are

$$\mathrm{TopKAcc}_{s,\ell}^{(k)}(G) = |G|^{-1} \sum_{i \in G} \mathbf{1}\left[y_i \in \mathrm{TopK}_k(p_{s,i}^{\ell})\right], \tag{S8}$$

$$\mathrm{Prob}_{s,\ell}(G) = |G|^{-1} \sum_{i \in G} p_{s,i}^{\ell}(y_i), \tag{S9}$$

$$\mathrm{MaxProb}_{s,\ell}(G) = |G|^{-1} \sum_{i \in G} \max_v p_{s,i}^{\ell}(v), \tag{S10}$$

$$\mathrm{Ent}_{s,\ell}(G) = -|G|^{-1} \sum_{i \in G} \sum_v p_{s,i}^{\ell}(v) \log p_{s,i}^{\ell}(v). \tag{S11}$$

For update-direction alignment, we compare each local layer update with the remaining direction toward the final hidden state:

$$\mathrm{Align}_{s,\ell}(G) = |G|^{-1} \sum_{i \in G} \cos\left(\Delta H_{s,i}^{\ell}, R_{s,i}^{\ell}\right), \tag{S12}$$

$$\Delta H_{s,i}^{\ell} = H_{s,i}^{\ell+1} - H_{s,i}^{\ell}, \quad R_{s,i}^{\ell} = H_{s,i}^{L} - H_{s,i}^{\ell}. \tag{S13}$$

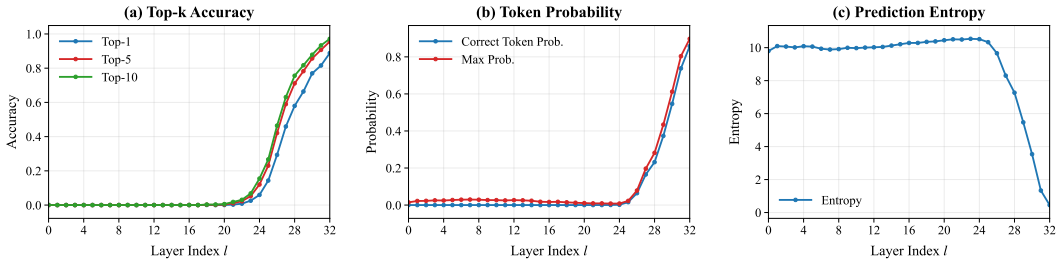


Figure S1: Top-k accuracy, token probability, and entropy diagnostics for intermediate-layer predictions.

D.4 Correction and Break Diagnostics

Let $c_{s,\ell,i} = \mathbf{1}[\arg\max_v p^{\ell}_{s,i}(v) = y_i]$ denote whether layer ℓ predicts the pseudo clean target at token i. The fix rate measures wrong-to-correct transitions between adjacent layers, while the break rate measures correct-to-wrong transitions:

$$\mathrm{Fix}_{s,\ell}(G) = |G|^{-1} \sum_{i \in G} \mathbf{1}[\neg c_{s,\ell,i} \wedge c_{s,\ell+1,i}]\,, \tag{S14}$$

$$\mathrm{Break}_{s,\ell}(G) = |G|^{-1} \sum_{i \in G} \mathbf{1}[c_{s,\ell,i} \wedge \neg c_{s,\ell+1,i}]\,. \tag{S15}$$

We also report the net correction rate $\mathrm{Fix}_{s,\ell}(G) - \mathrm{Break}_{s,\ell}(G)$, which indicates whether an adjacent layer pair repairs more predictions than it disrupts.

E More Hidden-State Refinement Analyses

This section provides additional observation figures that complement the main paper. The main text focuses on a compact set of diagnostics so that the overall argument remains readable. Here, we include supporting analyses that examine the same hidden-state refinement view with broader denoising-step coverage, different tasks and model variants, an alternative representation-similarity metric, and finer-grained early-exit prediction statistics. These supplementary diagnostics broaden the empirical basis of the main observations while keeping the core discussion compact.

E.1 All-Step Layer Smoothness

Figure S2 reports adjacent-layer cosine similarity across all denoising steps rather than only the selected steps shown in the main text. Each cell aggregates the token-wise cosine similarity between H^{ℓ}_s and $H^{\ell+1}_s$ at a particular denoising step and adjacent layer pair. This view is useful because selected-step curves can hide transient behavior. If the layer-wise refinement pattern only appeared at a few steps, the heatmap would show isolated high-similarity regions instead of a coherent structure over the full trajectory. In practice, the high-similarity band persists across most denoising steps, especially through the middle layers. The main deviation remains the transition from the embedding state into the first layer, which is consistent with the main-text interpretation that the first block has a stronger role in forming an initial contextual state before later layers perform more incremental updates.

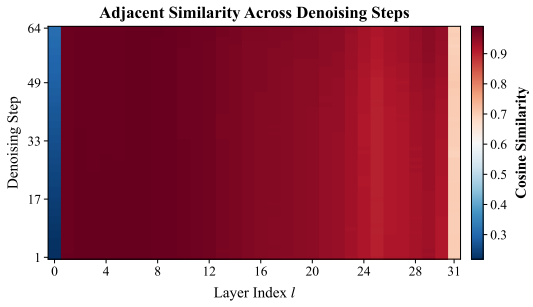


Figure S2: Adjacent-layer cosine similarity across all denoising steps.

E.2 Adjacent-Layer Smoothness Across Tasks and LLaDA Variants

Figure S3 extends the adjacent-layer similarity diagnostic beyond the primary setup used in the main observation, which uses LLaDA-8B-Instruct on GSM8K. We use these additional runs to examine whether the main smoothness pattern persists across different prompt distributions and model variants. The task panel compares GSM8K, Countdown, and SVAMP under the same observation protocol, covering natural-language arithmetic, arithmetic search, and robustness-oriented word problems. The model panel compares LLaDA-8B-Base with LLaDA-8B-Instruct, which isolates whether instruction tuning substantially changes the layer-wise continuity pattern. Across these settings, the curves remain high and have broadly similar shapes, suggesting that adjacent-layer smoothness is not a peculiarity of one benchmark split or one model variant.

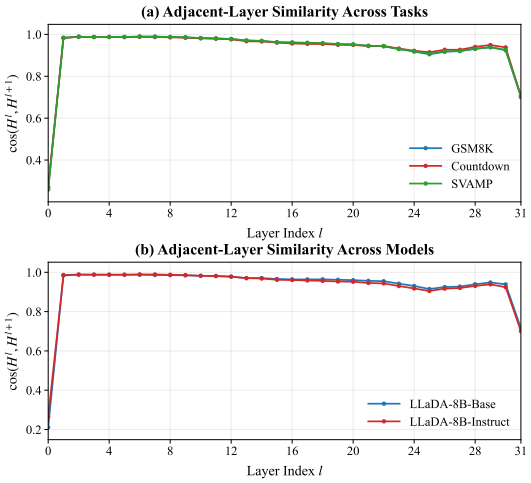


Figure S3: Adjacent-layer cosine similarity across tasks and model variants.

E.3 CKA-Based Layer Similarity

Figure S4 provides a complementary representation-similarity view using linear CKA. Unlike token-wise cosine similarity, which compares each token representation directly, CKA compares the geometry of token-feature matrices after centering. This makes it a useful sanity check because it asks whether layers preserve similar relational structure among token representations, not only whether corresponding token vectors point in similar directions. The diagnostic computes CKA at selected denoising steps and averages the resulting layer-pair similarities. The resulting heatmap again has a strong diagonal structure, with nearby layers more similar than distant layers. This agrees with the cosine-based heatmap in the main paper, suggesting that the smooth layer-wise trajectory is visible under both direct vector similarity and a representation-level similarity measure.

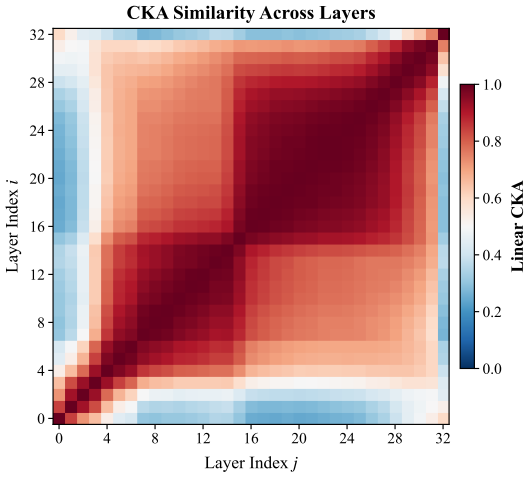


Figure S4: Linear CKA similarity across transformer layers.

E.4 Early-Exit Top-k, Probability, and Entropy Diagnostics

Figure S1 adds distribution-level statistics to the early-exit analysis. For each intermediate layer, we apply the final normalization and shared LM head used by the model output layer, and compare predictions with the final generated sequence used as the pseudo clean target. The top-k curves measure how early the target token becomes highly ranked, the probability curves track both target-token probability and maximum prediction probability, and the entropy curve measures the sharpness of the intermediate output distribution. These diagnostics give a more detailed picture than accuracy alone. Late layers do not merely flip a small number of argmax decisions. They also raise the rank and probability of the target token while making the distribution more decisive, supporting a gradual transformation from weak intermediate representations to output-ready denoising states, with the clearest change appearing in the final layers.

E.5 Correction and Break Rates

Table S1: Fixed-scale reasoning results with $\alpha = 0.5$ on LLaDA-8B-Base and LLaDA-8B-Instruct. Columns under each benchmark denote generation length. All values are percentages. Parenthesized values denote absolute improvements over the corresponding baseline.

NFE/Gen Len	Method	Countdown		GSM8K		SVAMP	
		128	256	128	256	128	256
LLaDA-8B-Base							
1/2	Baseline	7.42	10.16	17.06	60.88	48.00	77.67
	CSRR ($\alpha = 0.5$)	15.23 (+7.81)	17.58 (+7.42)	33.43 (+16.37)	70.66 (+9.78)	64.33 (+16.33)	81.00 (+3.33)
1	Baseline	6.25	11.72	16.68	64.75	52.00	80.33
	CSRR ($\alpha = 0.5$)	18.36 (+12.11)	21.88 (+10.16)	37.38 (+20.70)	79.23 (+14.48)	70.00 (+18.00)	82.67 (+2.34)
LLaDA-8B-Instruct							
1/2	Baseline	17.19	13.67	68.31	76.04	83.67	85.67
	CSRR ($\alpha = 0.5$)	28.12 (+10.93)	20.70 (+7.03)	69.67 (+1.36)	78.54 (+2.50)	83.00 (-0.67)	85.67 (+0.00)
1	Baseline	23.44	21.48	70.43	78.62	87.00	86.67
	CSRR ($\alpha = 0.5$)	38.28 (+14.84)	29.30 (+7.82)	74.00 (+3.57)	83.47 (+4.85)	89.00 (+2.00)	88.67 (+2.00)

Figure S5 decomposes adjacent-layer prediction changes into fix and break events. A fix occurs when a token prediction is incorrect at layer ℓ but correct at layer $\ell + 1$, while a break occurs when the reverse transition happens. This diagnostic is useful because an increase in final accuracy could in principle come from large unstable changes, where layers both fix many wrong tokens and break many correct tokens. The measured curves show a different pattern. In the later layers, the fix rate rises more strongly than the break rate, and the net correction curve becomes positive. This indicates that the layer-wise updates are not simply random reshuffling of token predictions. They are biased toward repairing intermediate mistakes while largely preserving predictions that have already become correct, which further supports the interpretation of the upper layers as refinement-oriented computation.

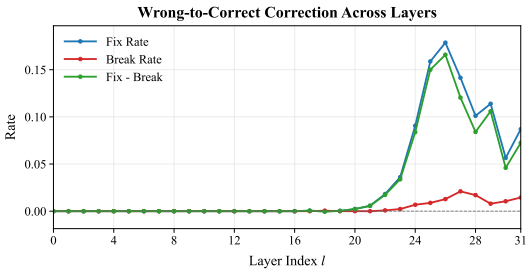


Figure S5: Wrong-to-correct fix rate, correct-to-wrong break rate, and their difference across adjacent layers.

F More Experimental Results

This section provides additional experimental results that complement the main paper. We first report full KV-cache results across reasoning benchmarks, then compare CSRR with a standard fine-tuning control using the same S1K data and training budget. The main results select the best recurrent scale from the evaluated candidates, while the fixed-scale study below asks whether CSRR remains effective without per-setting scale selection. We also vary the semi-autoregressive block size to test whether the gains depend on a particular block granularity.

F.1 KV-Cache Results Across Benchmarks

Table S2 extends the KV-cache evaluation in the main paper from Countdown to all three reasoning benchmarks. CSRR consistently improves over the corresponding cached baseline across both LLaDA variants, both NFE settings, and all reported benchmarks. These results indicate that the recurrent hidden-state path remains compatible with cached inference beyond the Countdown setting shown in the main text.

F.2 S1K-Matched Fine-Tuning Control

Table S3 compares CSRR with a standard fine-tuning control trained on the same S1K data for the same number of steps. The FT control improves over the untuned baseline, confirming that

Table S2: KV-cache reasoning results on LLaDA-8B-Base and LLaDA-8B-Instruct. Columns under each benchmark denote generation length. All values are percentages. Parenthesized values denote absolute improvements over the corresponding baseline.

NFE/Gen Len	Method	Countdown		GSM8K		SVAMP	
		128	256	128	256	128	256
LLaDA-8B-Base							
1/2	Baseline	8.59	7.81	15.62	52.92	52.67	70.33
	CSRR	17.19 (+8.60)	19.53 (+11.72)	28.35 (+12.73)	66.19 (+13.27)	60.67 (+8.00)	71.67 (+1.34)
1	Baseline	7.03	10.94	16.45	62.85	53.33	78.67
	CSRR	22.66 (+15.63)	21.48 (+10.54)	29.87 (+13.42)	74.98 (+12.13)	70.33 (+17.00)	81.67 (+3.00)
LLaDA-8B-Instruct							
1/2	Baseline	17.19	15.62	66.41	74.15	82.33	80.67
	CSRR	25.39 (+8.20)	21.09 (+5.47)	67.17 (+0.76)	75.59 (+1.44)	86.67 (+4.34)	88.00 (+7.33)
1	Baseline	21.48	16.41	69.14	78.01	85.33	84.67
	CSRR	26.56 (+5.08)	23.44 (+7.03)	73.31 (+4.17)	81.65 (+3.64)	89.33 (+4.00)	90.33 (+5.66)

the training data itself is beneficial. CSRR remains stronger across all three benchmarks and both LLaDA variants, suggesting that the gains are not explained solely by matched supervised fine-tuning.

F.3 Fixed-Scale Results

Table S1 reports LLaDA-8B-Base and LLaDA-8B-Instruct results when the recurrent scale is fixed to 0.5. This setting removes the scale-selection step used in the main table and therefore provides a more constrained evaluation of the recurrent hidden-state path. CSRR remains consistently effective across most reasoning settings, indicating that the improvement is not solely an artifact of choosing the best scale separately for each benchmark configuration. The gains are especially clear for LLaDA-8B-Base and remain positive in most LLaDA-8B-Instruct settings.

Table S3: S1K-matched fine-tuning control. Accuracy is reported in percentages with generation length $=$ NFE $=256$. FT denotes standard fine-tuning on the same S1K training set for the same number of steps.

Model	Method	Countdown	GSM8K	SVAMP
LLaDA-8B-Base	Baseline	11.72	64.75	80.33
LLaDA-8B-Base	FT	14.84	71.34	82.33
LLaDA-8B-Base	**CSRR**	**21.88**	**79.23**	**82.67**
LLaDA-8B-Instruct	Baseline	21.48	78.62	86.67
LLaDA-8B-Instruct	FT	25.39	83.24	88.67
LLaDA-8B-Instruct	**CSRR**	**29.30**	**83.47**	**89.00**

F.4 Results with Different Block Sizes

Table S4 evaluates CSRR under different semi-autoregressive block sizes with generation length 256 and NFE 128. Changing the block size alters how many tokens are denoised together within each local generation segment, which can affect both the baseline sampler and the usefulness of previous-step hidden states. CSRR improves over the corresponding baseline across all reported block-size settings, indicating that the method is compatible with different block granularities rather than being tuned to a single default block size.

Table S4: Reasoning results with different block sizes on LLaDA-8B-Base and LLaDA-8B-Instruct. All settings use generation length 256 and NFE 128. All values are percentages. Parenthesized values denote absolute improvements over the corresponding baseline.

Block size	Method	Countdown	GSM8K	SVAMP
LLaDA-8B-Base				
8	Baseline	7.81	56.18	71.00
	CSRR	15.23 (+7.42)	70.66 (+14.48)	77.33 (+6.33)
16	Baseline	8.59	59.36	74.67
	CSRR	14.84 (+6.25)	71.27 (+11.91)	77.00 (+2.33)
32	Baseline	10.16	60.88	77.67
	CSRR	17.58 (+7.42)	70.66 (+9.78)	81.00 (+3.33)
LLaDA-8B-Instruct				
8	Baseline	10.94	73.09	82.00
	CSRR	19.14 (+8.20)	77.71 (+4.62)	89.00 (+7.00)
16	Baseline	17.19	75.82	83.67
	CSRR	27.34 (+10.15)	77.71 (+1.89)	86.33 (+2.66)
32	Baseline	13.67	76.04	85.67
	CSRR	24.22 (+10.55)	78.54 (+2.50)	86.00 (+0.33)

F.5 Source and Injection Layer Positions

Figure S6 studies how the choice of source layer and injection layer affects CSRR. The source layer denotes the transformer block output used as the previous-step hidden state, and the injection layer denotes the block into which this state is incorporated during the current denoising step. The default CSRR configuration extracts the hidden state after the final block and injects it inside the first block. We conduct this sweep using LLaDA-8B-Base with generation length 256, NFE 128, and recurrent scale 1.0. The results show that most evaluated source-injection pairs improve over the corresponding baseline, supporting the robustness of the recurrent hidden-state path across different layer choices. The default pair remains among the stronger settings, while several alternative pairs achieve even higher accuracy, suggesting that layer-position selection is a meaningful design dimension that deserves further study.

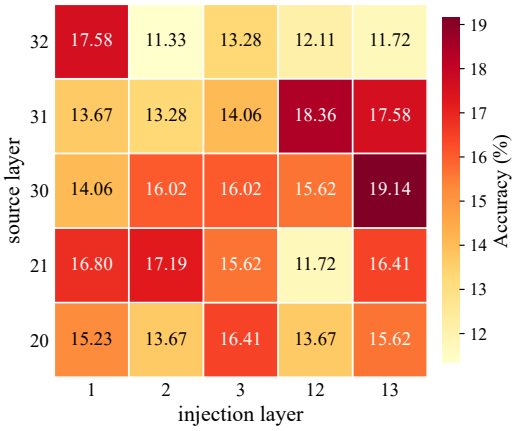


Figure S6: Accuracy under different source-layer and injection-layer choices for CSRR. The sweep uses LLaDA-8B-Base with generation length 256, NFE 128, and recurrent scale 1.0.

G Evaluation Prompts

This section reports the prompt templates used in evaluation. The templates are shown with placeholders for instance-specific fields, and line breaks are inserted for readability. The actual input sequence is obtained by applying the corresponding model tokenizer's chat template to the user message, followed by an optional assistant-side prefill for reasoning tasks.

G.1 Chat-Template Wrapping

For LLaDA-8B-Base and LLaDA-8B-Instruct, the tokenizer wraps each user message in the following structure:

```
<|startoftext|>
<|start_header_id|>user<|end_header_id|>

{user message}<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>

{optional assistant prefill}
```

For Dream-7B-Base and Dream-7B-Instruct, the tokenizer additionally inserts the default system message used by the model tokenizer:

```
<|im_start|>system
You are a helpful assistant.<|im_end|>
<|im_start|>user
{user message}<|im_end|>
<|im_start|>assistant
{optional assistant prefill}
```

For Countdown, GSM8K, and SVAMP, the optional assistant prefill is `<reasoning>`. For HumanEval, HumanEval+, and MBPP, no assistant prefill is used.

G.2 Reasoning Prompts

The reasoning benchmarks use task-specific instructions and require the final answer to be wrapped in `\boxed{}`. Countdown uses the arithmetic-expression prompt below:

```
Using only the provided numbers, create an
arithmetic expression that evaluates to
exactly the provided target
number. You may use the operations +, -, *,
and / as needed, but each number must be
used exactly once. Think step-by-step.
After reasoning, provide only your final
expression inside \boxed{} tags without
including an equals sign or the target
number. For example: \boxed{a + b * c}
Respond in the following format:
<reasoning>
Your reasoning here
</reasoning>
<answer>
\boxed{...}
</answer>

Numbers: {numbers}
Target: {target}
```

GSM8K uses the following mathematical-reasoning prompt:

```
You are a math expert. You will be given a
question to solve. Solve it step by step.
Wrap the final answer in a \boxed{}.
Respond in the following format:
<reasoning>
Your reasoning here
</reasoning>
<answer>
\boxed{...}
</answer>

{question}
```

SVAMP uses the same response format as the mathematical-reasoning prompt, with the problem text formed from the dataset body and question fields:

```
You are a math expert. You will be given a
question to solve. Solve it step by step.
Wrap the final answer in a \boxed{}.
Respond in the following format:
<reasoning>
Your reasoning here
</reasoning>
<answer>
\boxed{...}
</answer>

{body}{question}
```

G.3 Code-Generation Prompts

HumanEval and HumanEval+ share the same prompt template, where {function_prompt} contains the function signature and docstring from the benchmark instance:

```
Complete the following Python function:
{function_prompt}
```

MBPP uses a task prompt together with the provided unit tests:

```
You are a coding expert. You will be given a
coding problem to solve. Write a Python
function to solve the problem. Output only
```

```
the function code wrapped in ```python```
blocks. Do not include explanations or extra
text.

{problem}

Your code should pass the following tests:

{unit_tests}
```