ELF-REG: SCALING CONTINUOUS DIFFUSION LANGUAGE MODELS TO REASONING TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Fully continuous diffusion language models (dLMs) denoise continuous representations without intermediate discretization, then decode all response tokens in parallel at the final step. Their performance on challenging reasoning tasks remains less established than that of autoregressive (AR) LLMs and masked dLMs. We scale Embedded Language Flows (ELF) to mathematical reasoning and code generation on GSM8K, MATH-500, HumanEval, and MBPP. We introduce ELF-REG, which improves learning with representation alignment and entanglement (REPA+REG), where a frozen AR teacher supervises intermediate denoiser features and supplies a global representation that is jointly denoised with the response. ELF-REG-L achieves 55.96% pass@1 on GSM8K at 64 network function evaluations (NFE), and 13.39% on MATH-500 and 22.56% on HumanEval at 128 NFE. It outperforms the evaluated comparable-scale dLMs in pass@1 on GSM8K and code, and improves MATH-500 pass@1 from 10.55% for the ELF-L baseline to 13.39% with ELF-REG-L. Without few-step training, the same task-specific checkpoints support strong low-NFE performance through early-stop, which decodes an intermediate clean prediction without completing the denoising trajectory. At 16 NFE, ELF-REG-L reaches 41.21% HumanEval pass@10, outperforming recent continuous dLMs of comparable scale.

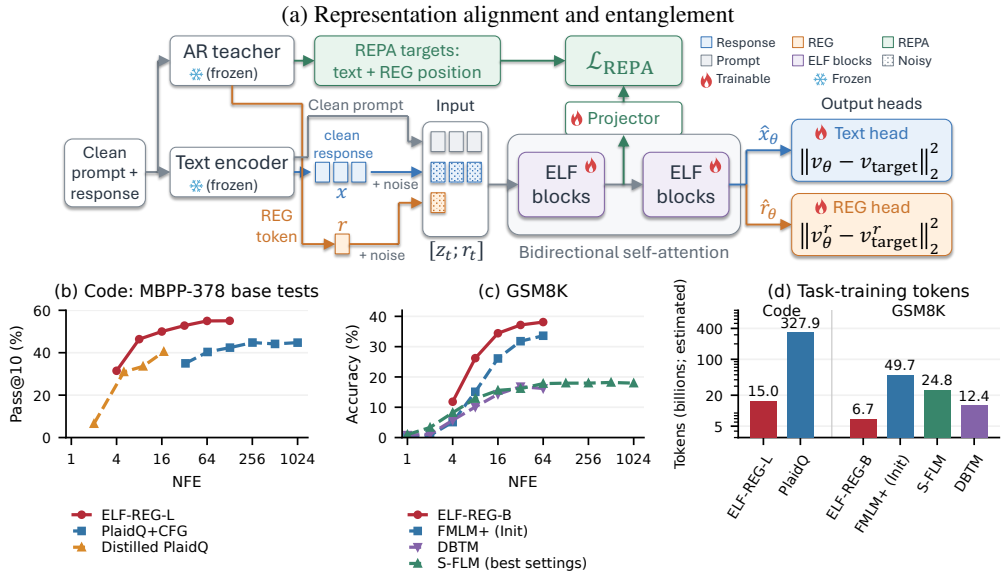

Figure 1: **ELF-REG improves generation with fewer task-training tokens.** (a) A frozen AR teacher supervises intermediate features through REPA and a global REG token denoised jointly with the response. (b,c) ELF-REG-L and ELF-REG-B with early-stop ($\rho = 8$) show superior performance across NFE, compared with PlaidQ (Peng et al., 2026b), FMLM+ (Agarwal et al., 2026), S-FLM (Deschenaux & Gulcehre, 2026), and DBTM (Tang & Wang, 2026). Code uses pass@10 on MBPP-378; GSM8K uses mean pass@1. (d) Non-padding training token budgets. See Appendix B for details.

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.

1 INTRODUCTION

Diffusion language models (dLMs) offer an efficient approach to parallel text generation by iteratively refining a corrupted sequence, in contrast to autoregressive (AR) LLMs that decode tokens in sequence. Continuous dLMs carry out this refinement in a representation space, bringing techniques from continuous diffusion to language and improving parallelism by updating all token positions simultaneously during denoising. Much of the development of continuous dLMs has focused on language modeling and conditional text generation. Plaid (Gulrajani & Hashimoto, 2023) studies likelihood-based language modeling, while LangFlow, LDLM, and Embedded Language Flows (ELF) evaluate unconditional generation using perplexity and entropy (Chen et al., 2026; Meshchaninov et al., 2026; Hu et al., 2026). Conditional applications include machine translation with CDCD (Dieleman et al., 2022), simplification and paraphrasing with DiffuSeq (Gong et al., 2023), and translation and summarization with ELF (Hu et al., 2026). Together, these studies establish continuous denoising as a viable approach to producing and transforming text.

Masked dLMs such as LLaDA and Dream already demonstrate high-performing reasoning and code generation (Nie et al., 2025; Ye et al., 2025), while recent continuous dLM works' performance on GSM8K and code generation (S-FLM, FMLM+, MLFM, PlaidQ (Deschenaux & Gulcehre, 2026; Agarwal et al., 2026; Azangulov et al., 2026; Peng et al., 2026b)) demonstrates progress in mathematical reasoning with continuous dLMs, giving reason to pursue reasoning capabilities of continuous dLMs. These advances show promise for continuous dLMs on reasoning tasks and motivate improving their accuracy at low compute scales, including with limited denoising budget.

In this work, we extend ELF's (Hu et al., 2026) fully continuous dLM to mathematical reasoning on GSM8K and MATH-500 and to code generation on HumanEval and MBPP. We use representation alignment and entanglement (REPA+REG) to improve performance: a frozen AR teacher supervises intermediate features through REPA (Yu et al., 2025) and supplies a global representation jointly denoised with the response through REG (Wu et al., 2025). We call this method **ELF-REG** and evaluate two sizes, ELF-REG-B and ELF-REG-L. At 64 NFE, our ELF-REG-B reaches 38.11% GSM8K pass@1, exceeding FMLM+ (Agarwal et al., 2026) at the same NFE and MDLM (Sahoo et al., 2024) and Duo (Sahoo et al., 2025) at 1024 NFE with comparable parameter count. ELF-REG-L reaches 55.96% at 64 NFE (Table 2). On code, ELF-REG-L achieves the highest pass@1 among the comparable-scale dLMs on all four benchmarks at 128 NFE (Table 1). On MBPP-500 and MBPP-378, ELF-REG-L at 32 NFE already surpasses PlaidQ+CFG's (Peng et al., 2026b) results at 257 NFE (Table 14). ELF-REG-L also improves MATH-500 accuracy over ELF-L baseline, demonstrating its performance on more complex tasks.

Strong performance also extends to low NFE without dedicated few-step training. Building on prior early stopping works for continuous dLMs (Vaina et al., 2024; Gao et al., 2024; Du & Ma, 2026), we use early-stop to decode an intermediate clean prediction and analyse its benefits. On GSM8K, continuing denoising beyond this exit substantially increases agreement with the final response but adds little aggregate accuracy (Figure 2). Thus, useful task performance can emerge before responses stabilize. We use a flow-map formulation to analyse early-stop endpoint error and apply the decoder-margin bound of Du & Ma (2026, Theorem 1) to study decoded-output agreement. Our contributions are:

- We advance fully continuous dLMs on mathematical reasoning and code, outperforming other dLMs of comparable scale in our comparisons of GSM8K and code pass@1, with further gains over ELF-L baseline on MATH-500.
- Without dedicated few-step training, ELF-REG-L reaches 46.39% pass@10 on MBPP-378 using early-stop at 8 NFE, exceeding PlaidQ-D16's 40.49% at 17 NFE (Table 15).
- On GSM8K, we show that accuracy approaches its final level while responses continue changing. We analyse endpoint error and decoded-output agreement using decoder margins and connections between early-stop and flow maps.

2 Related Works

2.1 Diffusion language models

Discrete dLMs. Discrete diffusion language models (dLMs) reverse token corruption or masking to generate text (Austin et al., 2023; Lou et al., 2024; Sahoo et al., 2024). LLaDA and Dream scale masked denoising to reasoning and code, using bidirectional context to predict tokens (Nie et al., 2025; Ye et al., 2025). Block Diffusion combines autoregressive (AR) generation across blocks with parallel denoising within each block (Arriola et al., 2025).

Continuous dLMs. Early continuous dLMs denoise token embeddings for controllable generation and sequence-to-sequence tasks (Li et al., 2022; Gong et al., 2023). CDCD and Plaid develop continuous formulations for categorical prediction and likelihood-based language modeling (Dieleman et al., 2022; Gulrajani & Hashimoto, 2023). Recent methods improve both the denoising process and its representation space. LangFlow (Chen et al., 2026) connects embedding diffusion to flow matching and learns the noise schedule. LDLM (Meshchaninov et al., 2026) jointly trains a latent encoder and decoder with the diffusion model. ELF (Hu et al., 2026) denoises contextual encoder representations and shares the denoising network with the final token decoder, focusing on unconditional language generation, translation, and summarization. S-FLM (Deschenaux & Gulcehre, 2026) extends continuous generation to mathematical reasoning using hyperspherical flows. Continuous diffusion also appears within AR models: L2D (Cetin et al., 2025) diffuses each next-token embedding, while LaDiR (Kang et al., 2026) denoises latent thought blocks and generates the final answer autoregressively.

Few-step generation. FMLM (Lee et al., 2026) distills continuous flows into learned flow maps for one/few-step generation. FMLM+ (Agarwal et al., 2026) adds conditioning on clean tokens and commits confident predictions through posterior refinement. MLFM (Azangulov et al., 2026) learns continuous flows over masked positions and progressively commits tokens during sampling. PlaidQ (Peng et al., 2026b) applies fully continuous diffusion to code with an AR-initialized denoiser, and trains separate few/one-step generators through few-step training methods. Early stopping provides a sampling-time alternative (Li et al., 2026): Just on Time (Kohut et al., 2026) finalizes individual tokens in masked dLMs based on confidence, whereas continuous dLMs can decode the entire sequence early based on decoder confidence (Du & Ma, 2026).

Early-stop. Prior work establishes early stopping for diffusion text generation through fixed and adaptive exits (Vaina et al., 2024; Gao et al., 2024; Du & Ma, 2026; Li et al., 2026), and for image generation through Truncated Jump Sampling (Peng & Gao, 2026). Prophet observes early answer emergence in masked dLMs and uses answer confidence to commit all remaining tokens (Li et al., 2026). Building on these works, we demonstrate **low-NFE mathematical reasoning and code generation** with early-stop and ELF-REG, without dedicated few-step training. Our GSM8K analysis shows that **ELF-REG-B improves intermediate predictions**, reducing endpoint error and bringing earlier token stability. We further interpret early-stop as approximating the remaining flow map, and analyse early-stop endpoint error and decoded-output agreement.

2.2 Representation alignment

Representation alignment. REPA (Yu et al., 2025) improves diffusion training by aligning noisy intermediate features to clean representations from a frozen teacher. HASTE (Wang et al., 2025) stops the alignment loss during training, while SRA (Jiang et al., 2026a) obtains targets from the diffusion model itself. Language applications differ in where they apply this supervision. Portuguese masked-dLM REPA (Junior et al., 2026) aligns denoiser features with clean text representations from pretrained language encoders. REPR-ALIGN (Peng et al., 2026a) aligns a masked bidirectional denoiser with the frozen AR model used to initialize it. TextLDM (Jiang et al., 2026b) instead aligns the VAE encoder with a frozen language model before training continuous latent diffusion.

Jointly denoised representations. REG (Wu et al., 2025) jointly denoises image latents and a global representation supervised by a pretrained encoder. Related language models combine continuous representations with discrete tokens (Zheng et al., 2025; Zhou et al., 2026).

Our contribution. ELF-REG extends ELF's fully continuous dLM design to complex reasoning. REPA aligns noisy intermediate states of the bidirectional denoiser with clean AR teacher states, while REG adds a jointly denoised global representation. We combine representation alignment with inference-time early-stop to push the efficiency-performance frontier **without dedicated few-step training**, and analyse early-stop properties of the trained dLM. Appendix E extends the discussion of related works.

3 METHODOLOGY

3.1 ELF PRELIMINARIES

We largely follow ELF (Hu et al., 2026) for its architecture, training, and sampling. We provide a brief recap here. ELF (Hu et al., 2026) generates continuous response representations from Gaussian noise while holding the prompt fixed. A frozen Qwen3-0.6B-Base encoder supplies clean text states x. Training corrupts response states with noise scale σ as

$$z_t = tx + (1-t)\sigma\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I), \quad t \in [0, 1]. \tag{1}$$

A bidirectional transformer predicts clean states $\hat{x}_\theta$, giving velocity $v_\theta = (\hat{x}_\theta - z_t)/(1-t)$ for $t < 1$. We retain ELF's self-conditioning guidance, using each clean prediction to condition the next sampling update. The same transformer learns to decode representations through token cross-entropy; these decoding examples and velocity regression define $\mathcal{L}_{\text{ELF}}$. One final decoder evaluation produces all response tokens in parallel.

3.2 REPRESENTATION ALIGNMENT AND ENTANGLEMENT

A separate frozen Qwen3-1.7B-Base AR teacher supplies REPA+REG targets (Figure 1(a)). REPA (Yu et al., 2025) aligns student states $u_i^{(\ell)}$ after block ℓ with clean teacher states h_i at the same token position through a learned projector p_ϕ into the teacher dimension, minimizing $1 - \cos(p_\phi(u_i^{(\ell)}), h_i)$. The student retains bidirectional attention. Inspired by REG (Wu et al., 2025), we standardize the teacher's last content-token state to form a global target r, corrupted as $r_t = tr + (1-t)\sigma\varepsilon_r$ where $\varepsilon_r \sim \mathcal{N}(0, I)$. An additional token exchanges information with text through self-attention and predicts the clean REG state, supervised by velocity regression with the same self-conditioning guidance as text. REPA also supervises this token. We optimize

$$\mathcal{L} = \mathcal{L}_{\text{ELF}} + \lambda_{\text{REPA}}\mathcal{L}_{\text{REPA}} + \lambda_{\text{REG}}\mathcal{L}_{\text{REG}}. \tag{2}$$

Text and REG start from noise at inference; REG remains available to the decoder, which emits only text. The teacher and projector are training-only. Appendix A specifies guidance targets, loss normalization, and numerical implementation; Table 6 gives target layers and defaults.

3.3 PREFIX EARLY-STOP GENERATION

For an actual budget of N network function evaluations (NFE), sample a logit-normal grid of denoising timesteps with $M-1$ updates for master budget $M = \rho N$. Execute its first $N-1$ denoiser updates and decode the latest predicted-clean text and REG states at decoder time 1. We use $\rho = 8$. Full-span generation instead traverses the complete grid for budget N and decodes the final updated states. Early-stop is a plug-and-play inference technique and does not require few-step training. NFE includes one final decoder evaluation; note that ρ relates evaluation budgets in NFE, not fractions of continuous time. Algorithm 1 gives the generation procedure following ELF (Hu et al., 2026, Algorithm 2) with our prefix early-stop modification.

4 ANALYSIS

We relate early-stop decoding to the remaining flow map, then examine endpoint error, decoder stability, and task accuracy along the sampling trajectory. In the result displays, pass@1 is answer accuracy averaged over questions and generation seeds. See Appendix B.6 for evaluation settings.

4.1 EARLY-STOP DECODING AND THE REMAINING FLOW MAP

We begin with a smooth continuous flow, with the prompt and guidance scale fixed. Let y_t denote its generated state: response latents together with REG when present. The prompt is a fixed input, outside this state. The flow map $F_{s,t}$ transports a state from time s to time t under the velocity field v (Boffi et al., 2025a;b; Lee et al., 2026). Starting from Gaussian noise $y_0 = \sigma\varepsilon$, with $\varepsilon \sim \mathcal{N}(0, I)$, we compare the endpoint y_1 at $t = 1$ with the clean prediction $\hat{y}_t$ made at exit time $t < 1$:

$$y_t = F_{0,t}(y_0), \qquad y_1 = F_{t,1}(y_t), \qquad \hat{y}_t = D_t(y_t). \tag{3}$$

Here D_t is the denoiser's clean predictor, including its text and REG outputs. The prediction $\hat{y}_t$ is made at time t and targets the clean endpoint. Let $\mathcal{D}$ denote the deterministic parallel token decoder at decoder time 1. Continuing the flow to $t = 1$ and decoding the endpoint returns $\mathcal{D}(y_1)$, and early-stop returns $\mathcal{D}(\hat{y}_t)$. Thus early-stop replaces the remaining flow map with a clean prediction before applying the same decoder. Flow-map models learn finite-time transport explicitly (Lee et al., 2026); here we examine the error of using the existing predictor for that transport. The continuous formulation uses $D_t(y) = y + (1-t)v(y,t)$. Appendix C.3 treats the self-conditioned sampler. We denote the Euclidean norm over the full generated state with $\|\cdot\|$, including text positions after EOS and REG when present.

Proposition 1 (Endpoint error)**.** *Suppose v is continuously differentiable and its flow exists uniquely through time 1. For a state y at time $t < 1$, write $y_s = F_{t,s}(y)$ and define the average remaining velocity as $\bar{v}_{t:1}(y) = (1-t)^{-1}\int_t^1 v(y_s, s)\, ds$. The flow map satisfies $F_{t,1}(y) = y + \int_t^1 v(F_{t,s}(y), s)\, ds = y + (1-t)\bar{v}_{t:1}(y)$. Hence*

$$e_t(y) := D_t(y) - F_{t,1}(y) = (1-t)\big(v(y,t) - \bar{v}_{t:1}(y)\big). \tag{4}$$

Let $a_s = \frac{d}{ds}v(y_s, s) = \partial_s v(y_s, s) + J_y v(y_s, s)v(y_s, s)$ be the acceleration along this trajectory, where $J_y v$ is the Jacobian of v with respect to the state. If $\|a_s\| \le A$ for every $s \in [t, 1]$, then $\|e_t(y)\| \le A(1-t)^2/2$.

The flow map integrates velocity along the evolving trajectory. The clean predictor instead takes one Euler step across the remaining interval using the current velocity. This step reaches the endpoint at $t = 1$ exactly when the current velocity equals the average remaining velocity. The acceleration bound quantifies the error when the velocity changes along the trajectory. Near the endpoint, the remaining interval is short. At an early exit, the bound is informative when the velocity changes little over the longer remaining interval. Flow-matching training alone does not ensure this condition.

Decoding can preserve a response even when the predicted endpoint differs from the endpoint at $t = 1$. At each response position, the decoder chooses the token with the largest logit. Let m be the smallest gap between that winning logit and any competing logit at y_1, over valid text positions and (if present) the first EOS token. A larger margin m allows more change in the logits before a token changes. Proposition 2 follows from the decoder-margin argument of Du & Ma (2026, Theorem 1).

Proposition 2 (Decoded-output agreement)**.** *Suppose $m > 0$. Let $L \ge 0$ be a finite pairwise-logit Lipschitz constant: each winning-token versus competing-token logit difference changes by at most $L\|a - b\|$ between any two states a, b on the segment joining $\hat{y}_t$ and y_1. If $L\|e_t(y_t)\| < m$, then*

$$\mathcal{D}(\hat{y}_t) = \mathcal{D}(y_1). \tag{5}$$

Corollary 1 (Distributional agreement)**.** *For $t < 1$ let P_t and P_1 be the distributions of $\hat{y}_t$ and y_1 induced by initial noise $\varepsilon \sim \mathcal{N}(0, I)$, with finite second moments. Coupling the endpoints through the same initial noise bounds their 2-Wasserstein distance:*

$$W_2(P_t, P_1) \le \sqrt{\mathbb{E}\|e_t(y_t)\|^2}. \tag{6}$$

Let Q_t, Q_1 be the distributions of $\mathcal{D}(\hat{y}_t), \mathcal{D}(y_1)$. If the margin and Lipschitz assumptions of Proposition 2 hold almost surely, then their total variation satisfies $\mathrm{TV}(Q_t, Q_1) \le \Pr[L\|e_t(y_t)\| \ge m]$.

Corollary 1 bounds latent distributional discrepancy in Wasserstein distance and decoded distributional discrepancy in total variation. The decoder condition permits latent differences that preserve the winning tokens. Appendix C gives the proofs and a bound on the decoder failure probability.

Proposition 2 gives a sufficient condition for preserving the complete endpoint response, whether that response is correct or incorrect. Different responses can nevertheless yield the same numerical answer. We therefore compare extracted answers and changes in correctness on matched question–seed pairs, alongside complete-response agreement.

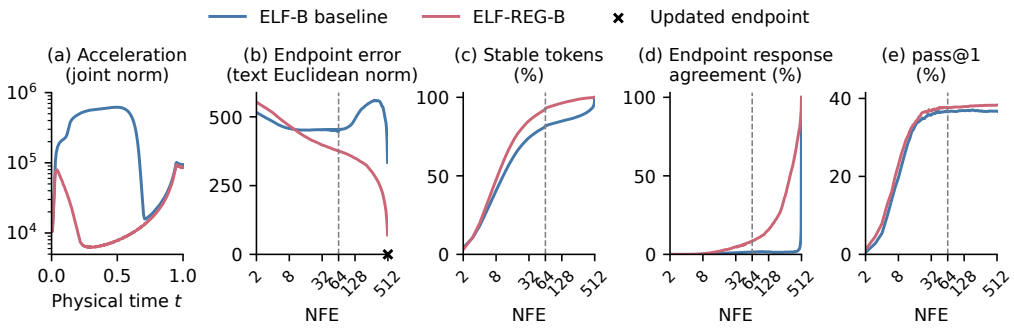

Figure 2: ELF-REG-B has lower discrete acceleration and lower early-stop endpoint error on GSM8K. (a) Median local joint acceleration with uniform time sampling over 512 Euler updates. (b–e) Mean text endpoint error, stable tokens, endpoint response agreement, and pass@1, using logit-normal time samples as specified in Appendix A.4. Stable tokens are those that agree with the endpoint at every later observation; response agreement requires the current sequence, EOS status, and length to match. Dashed lines in (b–e) mark NFE 64; continuing beyond this exit increases agreement substantially but adds little accuracy. Results average over four seeds: the median across questions in (a) and the mean across questions in (b–e). At NFE 512 in (b–e), the updated endpoint gives zero error and full agreement by construction.

4.2 Endpoint error and decoder stability along the sampling trajectory

We compare early-stop predictions with the endpoints of their own sampling trajectories. Figure 2(b–e) follows the ELF-B baseline and ELF-REG-B on GSM8K through and beyond the early-stop exit. Text endpoint error is the Euclidean distance between the predicted-clean text and the final updated text state, over the full response canvas including positions after EOS. This measures the text component of the discrete endpoint discrepancy in equation 14. Intermediate observations decode predicted-clean states; the endpoint decodes the final updated state. A "stable token" agrees with the endpoint at every subsequent observation. Endpoint response agreement instead requires an identical token sequence, EOS status, and length at the current observation.

With $\rho = 8$, the NFE-64 exit occurs early in denoising time t due to the non-uniform denoising schedule: the prediction time averages $t = 0.0825$ across four seeds. Across these schedules, the nominal SNR $t^2/[\sigma^2(1-t)^2]$ averages 2.03×10^{-3} for $\sigma = 2$ (Table 11). Thus the exit corresponds to a noise-dominated training corruption level. Appendix C.4 gives additional details.

At NFE 64, ELF-REG-B has 16.1% lower mean text endpoint error than the baseline (Figure 2b). Its endpoint response agreement is 8.61%, compared with 1.46% for the baseline (Figure 2d). ELF-REG-B also reaches token stability earlier (Figure 2c). Beyond the early exit at NFE 64, its mean text error decreases as response agreement rises, whereas the baseline's mean text error instead rises between NFE 64-257, even as its stable-token fraction increases.

To examine the acceleration term in Proposition 1, we measure consecutive velocity differences divided by time-difference with uniform time sampling over 512 Euler updates. Figure 2(a) shows lower local joint acceleration for ELF-REG-B through the early and middle trajectory. At $t = 5/64$, its median maximum remaining joint acceleration is 84.8% lower than ELF-B baseline, accompanied by 7.5% lower mean joint endpoint error (Table 12). Smaller remaining acceleration thus accompanies smaller endpoint error, consistent with Proposition 1. Note that these measurements are finite differences along the discrete sampler; Proposition 1 concerns a smooth continuous flow. See Appendix C.4 for details regarding the discrete acceleration measurements.

At NFE 64, numerical answers match those at NFE 512 on 80.27% of question–seed pairs for ELF-B baseline and 87.11% for ELF-REG-B (Table 10). Correctness gains and losses during further denoising nearly cancel for ELF-B baseline, while ELF-REG-B gains 0.64 percentage points. Most numerical answers therefore match the endpoint even though complete responses usually differ. Proposition 2 gives a sufficient condition for preserving the complete response. Table 10 numerical answer agreement results show that early-stop often preserves the numerical answer without pre-

serving every token, so complete-response agreement is a stronger requirement than preserving the task answer.

5 EXPERIMENTS

We compare ELF-REG with other dLMs on math reasoning and code generation across NFE budgets, and measure its gains over ELF baseline. Additional MMLU results appear in Appendix D.1. We train separate ELF baseline and ELF-REG variants for GSM8K, code, MATH, and MMLU; the MATH variants are initialized from their corresponding GSM8K weights. Our headline math-reasoning and code results use zero-shot early-stop generation with $\rho = 8$, at NFE 64 for GSM8K (Cobbe et al., 2021) and NFE 128 for MATH-500 (Hendrycks et al., 2021b; Lightman et al., 2023), MBPP-378 & MBPP-500 (Austin et al., 2021; Liu et al., 2023), and HumanEval(+) (Chen et al., 2021; Liu et al., 2023). We report mean pass@1 over sixteen samples per question. Appendix B.2 gives training data, budgets, and initialization; Table 6 and Appendix B.3 specify hyperparameters and evaluation protocols.

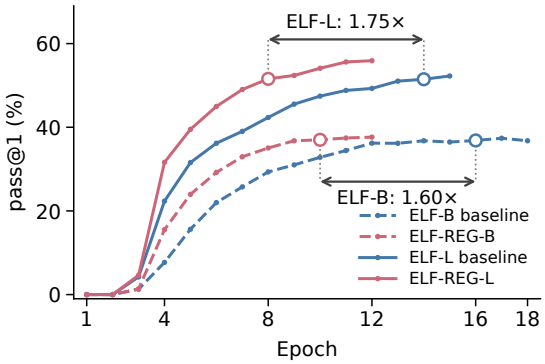

Figure 3: GSM8K pass@1 with early-stop (NFE 64), averaged over 4 seeds. Arrows show reductions in epochs to first attainment of matched accuracy.

We list the performance of various AR LLMs and large dLMs in Tables 1-2. We study task-specific training of small continuous dLMs for math reasoning and code generation. The AR LLMs/large dLMs also receive math and code training within much larger training-data budgets and are not comparable with our method (Yang et al., 2025; Allal et al., 2025; Nie et al., 2025; Ye et al., 2025). Tables 2 and 1 separate these references from comparable-scale dLMs. Other baselines use different settings for prompting, response budgets, and code post-processing; Appendix B.5 details these settings.

Table 1: Zero-shot code generation, pass@1 (%). Our evaluations report means, with standard deviation across 16 seeds in parentheses. Slashes separate base/instruct names and scores; Qwen3-0.6B results use non-thinking mode. Bold marks the best comparable-scale dLM mean per benchmark. Both MBPP columns use base tests; HumanEval+ uses extended tests. Parameter counts follow Table 2. Budgets specify NFE or sampling steps; n.r. means unreported. Sources and protocols appear in Appendix B.

Model	Param.	NFE / steps	MBPP-500	MBPP-378	HumanEval	HumanEval+
Large dLMs/AR LLMs						
LLaDA-8B-Base(Nie et al., 2025)	8.0B	n.r.	38.80	52.60	35.40	30.50
Dream-v0-Base-7B(Ye et al., 2025)	7.6B	n.r.	55.40	71.50	56.70	50.00
Qwen3-0.6B-Base/Qwen3-0.6B(Yang et al., 2025)	0.6B	–	–	–	30.50 / 41.46	– / 37.19
Llama-3.2-1B/Llama-3.2-1B-Instruct(Meta, 2024)	1.2B	–	–	–	18.90 / 33.50	–
SmolLM2-1.7B/SmolLM2-1.7B-Instruct(Allal et al., 2025)	1.7B	–	–	–	22.60 / 28.10	–
Qwen3-0.6B-Base (our evaluation)(Yang et al., 2025)	0.6B	–	–	40.62(2.71)	22.98(2.12)	19.05(1.95)
Comparable-scale dLMs						
Mask DFM(Gat et al., 2024)	1.3B	1,000 steps	6.20	–	9.10	7.90
Edit Flow(Havasi et al., 2025)	1.3B	10,000 steps	10.00	–	12.80	10.40
Open-dCoder(Peng et al., 2025)	494M	128 NFE	16.70	23.90	20.80	17.60
oDLM(Peng et al., 2026a)	596M	128 NFE	15.70	22.99	17.87	16.40
PlaidQ(Peng et al., 2026b)	717M	129 NFE	12.19	22.18	19.02	18.05
PlaidQ+CFG(Peng et al., 2026b)	717M	257 NFE	15.53	24.58	22.04	19.70
ELF-L baseline	637(+157)M	128 NFE	17.19(1.34)	26.75(1.58)	19.66(1.52)	18.75(1.52)
ELF-REG-L (ours)	652(+157)M	128 NFE	**19.10**(1.24)	**28.92**(1.38)	**22.56**(2.25)	**21.46**(2.10)

Mathematical reasoning. On GSM8K and at comparable scales, our ELF-REG-B at NFE 16 (34.42%) already exceeds the NFE 64 FMLM+ (Init) result (33.60%) (Table 13). At NFE 64, ELF-

Table 2: Mathematical reasoning, pass@1 (%). Our evaluations report mean with standard deviation across 16 seeds in parentheses. Slashes separate base/instruct models, scores, and differing shot counts. Qwen3-0.6B MATH-500 accuracy uses thinking mode. Bold marks the best comparable-scale dLM performance per benchmark. We show parameter count as non-decoder(+decoder) for ELF, including REPA+REG components. Budgets specify NFE or sampling steps; shots count demonstrations, and n.r. means unreported. Sources and protocols appear in Appendix B.

Model	Param.	NFE / steps	Shots	pass@1
GSM8K: large dLMs/AR LLMs				
LLaDA-8B-Base(Nie et al., 2025)	8.0B	1,024 steps	4	70.30
Dream-v0-Base-7B(Ye et al., 2025)	7.6B	256 steps	8	77.20
TESS 2 v0.1 (GSM8K fine-tuned)(Tae et al., 2025)	7B	1,000 steps	8	68.9
Qwen3-0.6B-Base/Qwen3-0.6B(Yang et al., 2025)	0.6B	–	4 / 0	59.59 / 79.20
Llama-3.2-1B/Llama-3.2-1B-Instruct(Meta, 2024)	1.2B	–	5 / 8	7.60 / 44.40
SmolLM2-1.7B/SmolLM2-1.7B-Instruct(Allal et al., 2025)	1.7B	–	5	31.10 / 48.80
Qwen3-0.6B-Base (our evaluation)(Yang et al., 2025)	0.6B	–	0	39.53(0.73)
GSM8K: comparable-scale dLMs				
S-FLM(Deschenaux & Gulcehre, 2026)	163M	512 NFE	0	18.18
FMLM+ (Init)(Agarwal et al., 2026)	0.168B	64 NFE	0	33.60
MLFM(Azangulov et al., 2026)	1.35B	256 steps	n.r.	31.24
MDLM(Sahoo et al., 2024)	168M	1,024 NFE	0	33.90
Duo(Sahoo et al., 2025)	168M	1,024 NFE	0	36.02
ELF-B baseline	90(+156)M	64 NFE	0	37.71(0.95)
ELF-REG-B (ours)	104(+156)M	64 NFE	0	38.11(1.07)
ELF-L baseline	637(+157)M	64 NFE	0	51.85(0.88)
ELF-REG-L (ours)	652(+157)M	64 NFE	0	**55.96**(0.94)
MATH-500: large dLMs/AR LLMs				
Qwen3-0.6B-Base/Qwen3-0.6B(Yang et al., 2025)	0.6B	–	4 / n.r.	29.80 / 77.60
Llama-3.2-1B/Llama-3.2-1B-Instruct(Meta, 2024)	1.2B	–	4 / 0	1.60 / 24.80
SmolLM2-1.7B/SmolLM2-1.7B-Instruct(Allal et al., 2025)	1.7B	–	4 / 0	11.60 / 19.20
Qwen3-0.6B-Base (our evaluation)(Yang et al., 2025)	0.6B	–	0	26.30(1.76)
MATH-500: comparable-scale dLMs				
ELF-L baseline	637(+157)M	128 NFE	0	10.55(0.90)
ELF-REG-L (ours)	652(+157)M	128 NFE	0	**13.39**(1.03)

REG-B reaches 38.11%, exceeding S-FLM at NFE 512 and MDLM and Duo at NFE 1024. On GSM8K, ELF-REG-L reaches 55.96% pass@1 at NFE 64. Figure 1 shows the accuracy-vs-NFE comparisons. Relative to ELF-L baseline, ELF-REG-L raises GSM8K pass@1 from 51.85% to 55.96% and MATH-500 pass@1 from 10.55% to 13.39%. ELF-REG-L reaches the 51.48% GSM8K accuracy threshold in 8 epochs versus 14 for ELF-L baseline, requiring approximately 43% fewer epochs (Figure 3).

Code generation. ELF-REG-L at NFE 128 exceeds the comparable-scale PlaidQ+CFG (Peng et al., 2026b) at NFE 257 on all four benchmarks in Table 1. Compared to PlaidQ+CFG (Peng et al., 2026b), our ELF-REG-L pass@1 reaches 28.92% vs 24.58% on MBPP-378 and 22.56% vs 22.04% on HumanEval. ELF-REG-L also outperforms Open-dCoder and oDLM (Peng et al., 2026b, Table 1) at matched NFE 128 on all four benchmarks. ELF-REG-L improves over ELF-L baseline on each benchmark. Appendix D.1 reports the complete comparisons and Appendix B.3 specifies prompting/execution protocols.

Table 3: Mathematical reasoning, pass@k (%), at NFE 64. ELF-REG-L improves ELF-L baseline across the sample budgets on both benchmarks. Results are shown across 16 seeds. Bold marks the best value per benchmark and sample budget.

Benchmark	Model	pass@2	pass@4	pass@8	pass@16
GSM8K	ELF-L baseline	62.15	70.43	76.98	82.26
GSM8K	ELF-REG-L (ours)	**65.18**	**72.57**	**78.64**	**83.70**
MATH-500	ELF-L baseline	15.65	23.64	33.36	44.00
MATH-500	ELF-REG-L (ours)	**20.75**	**29.95**	**40.41**	**52.20**

Multiple-sample performance. At NFE 128, ELF-REG-L reaches 55.05% pass@10 on MBPP-378 and 48.87% on HumanEval, compared with 44.79% and 39.87% for PlaidQ+CFG at NFE 257 (Table 16). At NFE 16 and without few-step training, ELF-REG-L already achieves higher pass@10 than PlaidQ+CFG at NFE 257 on both MBPP-378 and HumanEval (Table 15). On math reasoning, ELF-REG-L improves over ELF-L baseline at reported pass@k budgets at NFE 64, including MATH-500 pass@16 increasing from 44.0% (ELF-L baseline) to 52.2% (ELF-REG-L) (Table 3).

5.1 Ablations and additional analyses

Training components. Table 4 compares ELF-B variants at epoch 6. REPA improves over the baseline, and REPA+REG improves further. Stripped REPA+REG uses the REPA+REG weights but does not generate the REG token, retaining most of the gains of REPA+REG. REPA with one learnable token (REPA+OLT) does not reproduce the REG benefit. The comparison therefore identifies a training benefit that persists after REG is removed at inference.

Table 4: REPA+REG training benefits largely survive removal of REG at inference. At epoch 6 on GSM8K, REPA-only improves over Baseline, and REPA+REG improves further. Stripped REPA+REG retains most of this additional gain using the same trained weights as REPA+REG, except the REG component which is removed. REPA+OLT adds one learnable token (OLT) to REPA and does not reproduce the REG benefit. All settings use ELF-B. The REPA+REG setting corresponds to ELF-REG-B (ours).

	Baseline	REPA-only	REPA+REG	Stripped REPA+REG	REPA+OLT
pass@1 (%)	21.74	24.56	28.92	28.66	25.60

Prefix early-stop. At matched NFE, early-stop improves ELF-REG-B accuracy over full-span generation across budgets (Table 5). Early-stop uses the beginning of a longer time grid and decodes predicted-clean states; full-span generation traverses its shorter time grid and decodes the updated latent. Appendix D.5.1 gives a sweep of early-stop ratio ρ.

Table 5: GSM8K pass@1 (%) at matched NFE, ELF-REG-B (ours) at epoch 12, averaged over sixteen generation seeds.

NFE	4	8	16	32	64
Full-span	7.25	15.75	25.27	30.95	34.19
Prefix early-stop ($\rho = 8$)	11.82	26.20	34.42	37.15	38.11

Sensitivity to coarse-sampling errors. We inject errors from coarse (fewer-NFE) trajectories and continue denoising. Errors induced by coarser fewer-NFE sampling are more damaging than random errors of matched magnitude. Appendix D.5.1 and Figure 8 describe the details.

Additional representation analyses. The default Qwen3-0.6B-Base encoder supports more effective learning and gold-answer recovery under corruption than t5-small (Appendix D.3.1). Position-shifted AR teacher targets strengthen forward perturbation influence at the supervised block; the final-block ordering depends on corruption time (Appendix D.3.2).

6 Conclusion and Discussion

We scale ELF's (Hu et al., 2026) fully continuous dLM to mathematical reasoning and code generation using representation alignment and entanglement. We demonstrate superior performance compared to similar-scale dLMs on math (GSM8K) and code generation (HumanEval, MBPP).

Building on prior early stopping work (Vaina et al., 2024; Gao et al., 2024; Du & Ma, 2026), we apply and analyze early-stop for strong low-NFE generation without dedicated few-step training. Early-stop decodes an intermediate clean prediction, allowing the same trained models to generate responses with a limited denoising budget. ELF-REG-L exceeds PlaidQ-D16 on MBPP-378 pass@10 with fewer NFE, despite PlaidQ-D16's dedicated few-step training. Future work will develop methods for improving and scaling continuous dLMs into generalist models, aiming to match AR LLM performance at lower cost.

AI USE STATEMENT

In this work, we used Qwen3-30B-A3B-Instruct-2507 to generate rationales for the MMLU auxiliary training dataset. We have not used generative AI tools for other tasks requiring disclosure. Additionally, we used generative AI tools for literature search, recording and compiling results, and writing supporting experiment code. We have reviewed all AI-assisted work. We checked compiled results against experiment outputs and verified AI-generated code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We upload code in the supplemental materials for reproducing the headline results and data preprocessing. Proofs are provided in Appendix C.

REFERENCES

Manan Agarwal, Sheel Shah, Chanhyuk Lee, Jaehoon Yoo, Jerry Huang, Seunghoon Hong, Aditi Raghunathan, Jinwoo Kim, and Nicholas M. Boffi. Posterior refinement: Fast language generation via any-order flow maps, 2026. URL https://arxiv.org/abs/2606.24773.

Wasi Uddin Ahmad, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Vahid Noroozi, Somshubra Majumdar, and Boris Ginsburg. Opencodeinstruct: A large-scale instruction tuning dataset for code llms, 2025. URL https://arxiv.org/abs/2504.04030.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

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, 2025. URL https://arxiv.org/abs/2503.09573.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces, 2023. URL https://arxiv.org/abs/2107.03006.

Iskander Azangulov, Kianoosh Ashouritaklimi, Leo Zhang, Simon Vary, and Patrick Rebeschini. Masked language flow models, 2026. URL https://arxiv.org/abs/2606.27617.

Nicholas M. Boffi, Michael S. Albergo, and Eric Vanden-Eijnden. Flow map matching with stochastic interpolants: A mathematical framework for consistency models, 2025a. URL https://arxiv.org/abs/2406.07507.

Nicholas M. Boffi, Michael S. Albergo, and Eric Vanden-Eijnden. How to build a consistency model: Learning flow maps via self-distillation, 2025b. URL https://arxiv.org/abs/2505.18825.

Edoardo Cetin, Tianyu Zhao, and Yujin Tang. Large language models to diffusion finetuning, 2025. URL https://arxiv.org/abs/2501.15781.

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.

Yuxin Chen, Chumeng Liang, Hangke Sui, Ruihan Guo, Chaoran Cheng, Jiaxuan You, and Ge Liu. Langflow: Continuous diffusion rivals discrete in language modeling, 2026. URL https://arxiv.org/abs/2604.11748.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Justin Deschenaux and Caglar Gulcehre. Language modeling with hyperspherical flows, 2026. URL https://arxiv.org/abs/2605.11125.

Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H. Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, Curtis Hawthorne, Rémi Leblond, Will Grathwohl, and Jonas Adler. Continuous diffusion for categorical data, 2022. URL https://arxiv.org/abs/2211.15089.

Zhicheng Du and Lan Ma. Continuous language diffusion as a decoder-interface problem, 2026. URL https://arxiv.org/abs/2606.08810.

Zhujin Gao, Junliang Guo, Xu Tan, Yongxin Zhu, Fang Zhang, Jiang Bian, and Linli Xu. Empowering diffusion models on the embedding space for text generation. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4664–4683, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.261. URL https://aclanthology.org/2024.naacl-long.261/.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching, 2024. URL https://arxiv.org/abs/2407.15595.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and Lingpeng Kong. Diffuseq: Sequence to sequence text generation with diffusion models, 2023. URL https://arxiv.org/abs/2210.08933.

Ishaan Gulrajani and Tatsunori B. Hashimoto. Likelihood-based diffusion language models, 2023. URL https://arxiv.org/abs/2305.18619.

Marton Havasi, Brian Karrer, Itai Gat, and Ricky T. Q. Chen. Edit flows: Flow matching with edit operations, 2025. URL https://arxiv.org/abs/2506.09018.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021a. URL https://arxiv.org/abs/2009.03300.

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, 2021b. URL https://arxiv.org/abs/2103.03874.

Keya Hu, Linlu Qiu, Yiyang Lu, Hanhong Zhao, Tianhong Li, Yoon Kim, Jacob Andreas, and Kaiming He. Elf: Embedded language flows, 2026. URL https://arxiv.org/abs/2605.10938.

Dengyang Jiang, Mengmeng Wang, Liuzhuozheng Li, Lei Zhang, Haoyu Wang, Wei Wei, Guang Dai, Yanning Zhang, and Jingdong Wang. No other representation component is needed: Diffusion transformers can provide representation guidance by themselves, 2026a. URL https://arxiv.org/abs/2505.02831.

Jiaxiu Jiang, Jingjing Ren, Wenbo Li, Bo Wang, Haoze Sun, Yijun Yang, Jianhui Liu, Yanbing Zhang, Shenghe Zheng, Yuan Zhang, Haoyang Huang, Nan Duan, and Wangmeng Zuo. Textldm: Language modeling with continuous latent diffusion, 2026b. URL https://arxiv.org/abs/2605.07748.

Adalberto Ferreira Barbosa Junior, Lucas Lima Neves, and Adriano César Santana. Accelerating Portuguese masked diffusion models through representation alignment. In Marlo Souza, Iria de Dios-Flores, Diana Santos, Larissa Freitas, Jackson Wilke da Cruz Souza, and Eugénio Ribeiro (eds.), *Proceedings of the 17th International Conference on Computational Processing of Portuguese (PROPOR 2026) - Vol. 1*, pp. 968–973, Salvador, Brazil, April 2026. Association for Computational Linguistics. ISBN 979-8-89176-387-6. URL https://aclanthology.org/2026.propor-1.97/.

Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Nicklas Majamaki, Navdeep Jaitly, Yi-An Ma, and Lianhui Qin. Ladir: Latent diffusion enhances llms for text reasoning, 2026. URL https://arxiv.org/abs/2510.04573.

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.

Chanhyuk Lee, Jaehoon Yoo, Manan Agarwal, Sheel Shah, Jerry Huang, Aditi Raghunathan, Seunghoon Hong, Nicholas M. Boffi, and Jinwoo Kim. Flow map language models: One-step language modeling via continuous denoising, 2026. URL https://arxiv.org/abs/2602.16813.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Soroush Vosoughi, and Shiwei Liu. Diffusion language models know the answer before decoding, 2026. URL https://arxiv.org/abs/2508.19982.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-lm improves controllable text generation, 2022. URL https://arxiv.org/abs/2205.14217.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

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, 2023. URL https://arxiv.org/abs/2305.01210.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution, 2024. URL https://arxiv.org/abs/2310.16834.

Viacheslav Meshchaninov, Alexander Shabalin, Egor Chimbulatov, Nikita Gushchin, Ilya Koziev, Alexander Korotin, and Dmitry Vetrov. How to train your latent diffusion language model jointly with the latent space, 2026. URL https://arxiv.org/abs/2605.07933.

Meta. Llama-3.2-1b-instruct model card, 2024. URL https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct.

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, 2025. URL https://arxiv.org/abs/2502.09992.

Fred Zhangzhi Peng, Shuibai Zhang, Alex Tong, et al. Open-dllm: Open diffusion large language models. https://github.com/pengzhangzhi/Open-dLLM, 2025.

Fred Zhangzhi Peng, Alexis Fox, Anru R. Zhang, and Alexander Tong. Don't retrain, align: Adapting autoregressive lms to diffusion lms via representation alignment, 2026a. URL https://arxiv.org/abs/2605.06885.

Fred Zhangzhi Peng, Kaiwen Zheng, and Anru R. Zhang. Distilled continuous diffusion language models can write code in few steps—or one, 2026b. URL https://arxiv.org/abs/2609.04531.

Xin Peng and Ang Gao. x-prediction is all you need:training-free accelerated generation via endpoint decodability, 2026. URL https://arxiv.org/abs/2607.06114.

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, 2024. URL https://arxiv.org/abs/2406.07524.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The diffusion duality, 2025. URL https://arxiv.org/abs/2506.10892.

Jaesung Tae, Hamish Ivison, Sachin Kumar, and Arman Cohan. Tess 2: A large-scale generalist diffusion language model, 2025. URL https://arxiv.org/abs/2502.13917.

Sophia Tang and Shiyi Wang. Discrete beckmann transport models for one-step language modeling and reasoning, 2026. URL https://arxiv.org/abs/2609.15903.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data, 2024. URL https://arxiv.org/abs/2410.01560.

Sofia Maria Lo Cicero Vaina, Nikita Balagansky, and Daniil Gavrilov. Diffusion language models generation can be halted early, 2024. URL https://arxiv.org/abs/2305.10818.

Ziqiao Wang, Wangbo Zhao, Yuhao Zhou, Zekai Li, Zhiyuan Liang, Mingjia Shi, Xuanlei Zhao, Pengfei Zhou, Kaipeng Zhang, Zhangyang Wang, Kai Wang, and Yang You. Repa works until it doesn't: Early-stopped, holistic alignment supercharges diffusion training, 2025. URL https://arxiv.org/abs/2505.16792.

Ge Wu, Shen Zhang, Ruijing Shi, Shanghua Gao, Zhenyuan Chen, Lei Wang, Zhaowei Chen, Hongcheng Gao, Yao Tang, Jian Yang, Ming-Ming Cheng, and Xiang Li. Representation entanglement for generation: Training diffusion transformers is much easier than you think, 2025. URL https://arxiv.org/abs/2507.01467.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models, 2024. URL https://arxiv.org/abs/2309.12284.

Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think, 2025. URL https://arxiv.org/abs/2410.06940.

Changsheng Zhao, Ernie Chang, Zechun Liu, Chia-Jung Chang, Wei Wen, Chen Lai, Sheng Cao, Yuandong Tian, Raghuraman Krishnamoorthi, Yangyang Shi, and Vikas Chandra. Mobilellm-r1: Exploring the limits of sub-billion language model reasoners with open training recipes, 2026. URL https://arxiv.org/abs/2509.24945.

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, 2025. URL https://arxiv.org/abs/2510.01329.

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, 2026. URL https://arxiv.org/abs/2510.03206.

Jingwei Zuo, Maksim Velikanov, Ilyas Chahed, Younes Belkada, Dhia Eddine Rhayem, Guillaume Kunsch, Hakim Hacid, Hamza Yous, Brahim Farhat, Ibrahim Khadraoui, Mugariya Farooq, Giulia Campesan, Ruxandra Cojocaru, Yasser Djilali, Shi Hu, Iheb Chaabane, Puneesh Khanna, Mohamed El Amine Seddik, Ngoc Dung Huynh, Phuc Le Khac, Leen AlQadi, Billel Mokeddem, Mohamed Chami, Abdalgader Abubaker, Mikhail Lubinets, Kacper Piskorski, and Slim Frikha. Falcon-h1: A family of hybrid-head language models redefining efficiency and performance, 2025. URL https://arxiv.org/abs/2507.22448.

CONTENTS

1 Introduction **2**

2 Related Works **3**

2.1 Diffusion language models 3

2.2 Representation alignment 3

3 Methodology **4**

3.1 ELF preliminaries 4

3.2 Representation alignment and entanglement 4

3.3 Prefix early-stop generation 4

4 Analysis **4**

4.1 Early-stop decoding and the remaining flow map 5

4.2 Endpoint error and decoder stability along the sampling trajectory 6

5 Experiments **7**

5.1 Ablations and additional analyses 9

6 Conclusion and Discussion **9**

References **10**

A Method details **17**

A.1 Clean encoder and teacher representations 17

A.2 Alignment and loss normalization 17

A.3 Training-time guidance and velocity targets 17

A.4 Sampling behavior 18

B Experimental settings and comparison protocols **18**

B.1 Default hyperparameters 18

B.2 Training data and initialization 19

B.3 Benchmarks, metrics, and evaluation protocols 21

B.4 NFE and parameter accounting 22

B.5 External comparison sources and training budgets 22

B.6 Analysis protocols 24

C Early-stop proofs and the discrete sampler **25**

C.1 Endpoint error 25

C.2 Decoding and distributional agreement 25

C.3 Self-conditioning, REG, and the denominator clamp 26

C.4 Endpoint-error and discrete-acceleration measurements 27

D Additional results and analyses **29**

D.1 Additional headline results 29

D.2 Numerical values underlying Figure 1 31

D.3 Additional representation analyses 31

D.4 Training ablations 35

D.5 Sampling analyses 35

E Comparison with related work **39**

E.1 Early-stop generation 40

F Limitations **41**

G Qualitative Examples **42**

A METHOD DETAILS

A.1 CLEAN ENCODER AND TEACHER REPRESENTATIONS

Table 6 gives the architecture, training, and sampling defaults for the headline configurations. REG uses the teacher state at the last non-special content token. The clean Qwen3 text states and global targets are normalized across features. For a feature vector a, this gives $(a-\bar{a})/\sqrt{s_a^2+10^{-6}}$, where $\bar{a}$ is its mean and s_a^2 its variance. Text REPA targets retain their raw unnormalized feature values. The REPA target at the REG position uses the pooled representation from the REPA teacher layer. The clean REG denoising target uses the REG teacher layer.

A.2 ALIGNMENT AND LOSS NORMALIZATION

The causal teacher sees both clean prompt and response. On denoising examples, the bidirectional dLM student sees clean prompt and noisy response states. REPA uses a position-wise MLP projector. Text alignment excludes special tokens and padding. REG contributes an additional valid position to the REPA mean. Alignment is also applied to decoder examples. Position-shifted REPA targets change only text targets and their valid-position mask; the REG target stays fixed.

Loss normalization. ELF training selects decoder examples with some probability. The token cross-entropy and text velocity losses are summed over their respective examples and divided by the total number of supervised response positions. With EOS padding which we use for all our experiments, this includes post-response EOS positions. REPA takes separate masked means for prompt and response positions before computing a weighted sum; by default, the prompt and response REPA losses have weights 0 and 1, respectively. REG averages squared velocity error over features, sums it over denoising examples, and divides by the expected number of denoising examples in a batch. Decoder examples receive corrupted REG inputs but no REG velocity loss.

A.3 TRAINING-TIME GUIDANCE AND VELOCITY TARGETS

We follow ELF (Hu et al., 2026) in applying self-conditioning independently to denoising examples, with indicator $b \in \{0,1\}$. We sample a guidance scale $w > 0$, favoring smaller values and following ELF's sampling strategy. A first auxiliary prediction uses zero response self-conditioning. Its predicted clean states condition a second auxiliary prediction, at the same noisy state, time, and guidance scale, with the prompt held fixed. The zero-conditioned and conditioned text velocities are $v_{\text{no-sc}}$ and v_{sc} in equation 7, respectively; the corresponding REG velocities are $v^r_{\text{no-sc}}$ and v^r_{sc}, where r denotes the REG component. The trainable prediction conditions on the first auxiliary prediction's detached clean states when $b = 1$ and zero response self-conditioning when $b = 0$.

Implemented velocity targets. Following ELF (Hu et al., 2026), conversions from clean predictions to velocities use $c_t = \max(1-t, t_\epsilon)$. In particular, $v_\theta = (\hat{x}_\theta - z_t)/c_t$ and $v^r_\theta = (\hat{r}_\theta - r_t)/c_t$. Here sg stops gradients through the regression target. The detached regression targets are

$$v_{\text{target}} = \text{sg}\left[\frac{x - z_t}{c_t} + b\left(1 - \frac{1}{w}\right)(v_{\text{sc}} - v_{\text{no-sc}})\right], \tag{7}$$

$$v^r_{\text{target}} = \text{sg}\left[\frac{r - r_t}{c_t} + b\left(1 - \frac{1}{w}\right)(v^r_{\text{sc}} - v^r_{\text{no-sc}})\right]. \tag{8}$$

Without self-conditioning ($b = 0$), the targets reduce to the base velocities $(x - z_t)/c_t$ and $(r - r_t)/c_t$. The guided targets train the network to incorporate guidance in a single denoiser evaluation at inference. The clean targets in equation 9 use these same clamped denominators: $x^\star_t = \text{sg}(z_t + c_t v_{\text{target}})$ and $r^\star_t = \text{sg}(r_t + c_t v^r_{\text{target}})$. The identity $\|v_\theta - v_{\text{target}}\|^2 = c_t^{-2}\|\hat{x}_\theta - x^\star_t\|^2$ converts clean-prediction error to the implemented velocity error; there is no additional time weight on the velocity loss. The velocity losses apply only to denoising examples. Decoder examples receive time conditioning $t = 1$, zero self-conditioning, and independently corrupted inputs, with per-token logit-normal corruption scale. The REG decoder input uses a separately sampled corruption scale from the same distribution as the decoder examples.

Expanded training objective. Let the time-weighting coefficient be $\omega(t) = c_t^{-2}$. Let $a \sim \mathrm{Bernoulli}(p_{\mathrm{dec}})$ indicate a decoder example, and write ℓ^{CE} for token cross-entropy. The text and REG feature dimensions are d and d_r, respectively, and $\hat{r}_\theta$ is the REG head's clean prediction. Written out, the loss is

$$\mathcal{L} = \underbrace{\mathbb{E}\left[a\ell^{\mathrm{CE}} + (1-a)\frac{\omega(t)}{d}\|\hat{x}_\theta - x_t^\star\|_2^2\right]}_{\mathcal{L}_{\mathrm{ELF}}} + \lambda_{\mathrm{REPA}} \underbrace{\mathbb{E}\left[1 - \cos\left(p_\phi(u^{(\ell)}), h\right)\right]}_{\mathcal{L}_{\mathrm{REPA}}} + \lambda_{\mathrm{REG}} \underbrace{\mathbb{E}\left[\frac{\omega(t)}{d_r}\|\hat{r}_\theta - r_t^\star\|_2^2 \,\middle|\, a = 0\right]}_{\mathcal{L}_{\mathrm{REG}}}. \quad (9)$$

$\mathcal{L}_{\mathrm{ELF}}$ averages over supervised response positions, using time-weighted clean-prediction error for denoising examples ($a = 0$) and token cross-entropy for decoder examples ($a = 1$). $\mathcal{L}_{\mathrm{REPA}}$ aligns intermediate states with the teacher representation h at valid response content positions and the REG position, including decoder examples. $\mathcal{L}_{\mathrm{REG}}$ trains the model to reconstruct the guided global target and averages over denoising examples only. Both squared errors use the same time weight $\omega(t)$, with d^{-1} and d_r^{-1} averaging over features. The coefficients λ_{REPA} and λ_{REG} weight the auxiliary losses.

A.4 Sampling behavior

The Euler ODE sampler samples denoising time from a logit-normal distribution and appends endpoints 0 and 1. Its time distribution and noise scale match denoising training (Table 6). Decoding uses time 1 and zero self-conditioning. Ordinary REPA+REG retains REG; stripped REPA+REG loads the shared weights into the baseline architecture without the REG token position and associated weights. Please refer to Algorithm 1 for the sampling algorithm with plug-and-play early-stop, adapted from ELF (Hu et al., 2026).

NFE counts denoiser evaluations plus one final decoder evaluation. Full-span denoising, vocabulary projections, and cached AR decoding have different FLOP and latency costs; Appendix B.4 specifies the accounting conventions.

B Experimental settings and comparison protocols

B.1 Default hyperparameters

Table 6 contains the default settings for the headline experiments. Appendix B.6 specifies the analysis and ablation protocols.

Table 6: Default hyperparameters for the headline ELF configurations. GSM8K pairs give ELF-B / ELF-L when the sizes differ; other slashes separate the quantities named in the row. Parameters are non-decoder (+decoder), in millions. REPA+REG denotes ELF-REG-B (ours) or ELF-REG-L (ours), according to model size; the baseline omits both objectives. Training epochs identify the evaluated checkpoints. MATH-500 epochs count task training after GSM8K initialization (Appendix B.2). Shared entries span the task columns.

Setting	GSM8K	Code	MATH-500	MMLU
Architecture				
Model size	ELF-B / ELF-L	ELF-L	ELF-L	ELF-B
Transformer blocks	12 / 32	32	32	12
Hidden width	768 / 1,280	1,280	1,280	768
Attention heads	12 / 16	16	16	12
SwiGLU hidden width	2,048 / 3,413	3,413	3,413	2,048
Vocabulary size		151,669		
Parameters (M), baseline	90 (+156) / 637 (+157)	637 (+157)	637 (+157)	90 (+156)
Parameters (M), REPA+REG	104 (+156) / 652 (+157)	652 (+157)	652 (+157)	104 (+156)
Text latent / decoder width		1024		
Text projection bottleneck		128		
Time / guidance / mode tokens		4 / 4 / 4		
Attention / projection dropout		0.0		

Continued on next page.

Table 6: Default hyperparameters for the headline ELF configurations (continued).

Setting	GSM8K	Code	MATH-500	MMLU
Optimization				
Optimizer		Muon + Nesterov-Adam		
Muon momentum		0.95		
Adam (β_1, β_2); ϵ		$(0.9, 0.999)$; 10^{-8}		
Learning rate		0.002		
Global batch size		512		
Weight decay		0.0		
Gradient clipping norm		1.0		
LR warmup (epochs)	0.5	0.5	0.25	2.5
LR schedule		Linear warmup, then constant		
EMA warmup (updates)	1000	1000	0	1000
Training epochs	baseline: 18 / 15 REPA+REG: 12 / 12	12	8	30
Training objectives				
Encoder		Qwen3-0.6B-Base		
Encoder layer		20		
Denoising logit-time mean / std.		-1.5 / 0.8		
Denoising noise scale σ		2.0		
Decoder-example probability		0.2		
Decoder logit-corruption mean / std.		0.8 / 0.8		
Decoder noise scale		1.0		
Self-conditioning probability		0.5		
Training guidance range w		[0.5, 5.0]		
REPA+REG				
Teacher (REPA and REG)		Qwen3-1.7B-Base		
Teacher layers, REPA / REG		20 / 24		
Teacher target width		2048		
REPA supervised block	4 / 8	8	8	4
Projector layers / hidden width		3 / 2,048		
REPA weight λ_{REPA}		0.5		
REG weight λ_{REG}		0.1		
REPA prompt / response weights		0.0 / 1.0		
Teacher target shift		0		
REPA training cutoff	None	Epoch 8	None	None
Sampling				
Sampler		Euler ODE		
Time grid		Sorted logit-normal		
CFG / SCCFG	1 / 3	1 / 2	1 / 2	1 / 2
Evaluated EMA	0.9999	0.9999	0.9999	0.999
Prompt / sequence limit	256 / 1,024	512 / 1,024	256 / 1,024	800 / 1,056
Denominator clamp t_ϵ		0.05		

B.2 Training data and initialization

Initialization. We determine checkpoints using fixed training budgets. Table 6 gives the training epochs of the evaluated checkpoints, and Table 7 gives the training-token budgets and GPU-hours. We initialize the ELF model from scratch for GSM8K, code, and MMLU. MATH models are fine-tuned for eight epochs from the corresponding GSM8K weights at epoch 7.5, using the EMA 0.999 checkpoint as initialization.

At the GSM8K accuracy thresholds in Figure 3, ELF-REG reduces training GPU-hours relative to the baseline by 14.1% for ELF-B and 31.4% for ELF-L (Table 8).

Table 7: Training epochs, non-padding task-training tokens, and GPU-hours on H200 GPUs for the headline checkpoints. Init. counts inherited task training; Task counts training on the current corpus; Total sums them. Token counts are epoch-based estimates in billions. MATH-500 GPU-hours are given including / excluding GSM8K initialization.

Task	Model	Epoch	Init. (B)	Task (B)	Total (B)	GPU-hours
GSM8K	ELF-B baseline	18	–	10.044	10.044	170.8
GSM8K	ELF-REG-B (ours)	12	–	6.696	6.696	156.5
GSM8K	ELF-L baseline	15	–	8.370	8.370	347.9
GSM8K	ELF-REG-L (ours)	12	–	6.696	6.696	333.9
Code	ELF-L baseline	12	–	14.958	14.958	317.3
Code	ELF-REG-L (ours)	12	–	14.958	14.958	407.9
MATH-500	ELF-L baseline	8	4.185	13.625	17.810	449.6 / 275.7
MATH-500	ELF-REG-L (ours)	8	4.185	13.625	17.810	563.1 / 354.4
MMLU	ELF-B baseline	30	–	1.259	1.259	10.9
MMLU	ELF-REG-B (ours)	30	–	1.259	1.259	16.7

Table 8: Training GPU-hours on H200 GPUs to reach the GSM8K accuracy thresholds in Figure 3. Reduction is relative to the baseline of the same architecture. REPA+REG denotes ELF-REG-B (ours) or ELF-REG-L (ours), according to architecture.

		GPU-hours		
Architecture	Accuracy (%)	Baseline	REPA+REG	Reduction (%)
ELF-B	36.85	151.8	130.4	14.1
ELF-L	51.48	324.7	222.6	31.4

GSM8K training data. We combine the GSM8K-derived portions of OpenMathInstruct-2 (Toshniwal et al., 2024) and MetaMathQA (Yu et al., 2024). We remove examples with missing or unsupported numerical answers, examples exceeding the token limits, and groups of questions that match the GSM8K test set (Cobbe et al., 2021). After grouping related questions, we sample 10% of the source questions from each corpus for validation and hold out their entire groups. Validation uses a fixed subset of 1024 examples from distinct groups. The training dataset contains 2,404,028 examples.

MATH training data. We use MATH-derived question–solution pairs from the five-million-example release of OpenMathInstruct-2 (Toshniwal et al., 2024). We remove examples with malformed answers, groups with conflicting answers, examples exceeding the token limits, and groups matching MATH-500 (Hendrycks et al., 2021b; Lightman et al., 2023). We hold out 10% of the related-question groups for validation and use a fixed subset of 1024 examples from distinct groups. The training dataset contains 3,572,583 examples.

Code training data. We use Python question–solution pairs from OpenCodeInstruct (Ahmad et al., 2025). We filter by the dataset's quality scores, require solutions to parse as Python, and enforce the token limits. We remove training examples matching the HumanEval and MBPP task sets distributed with EvalPlus (Chen et al., 2021; Austin et al., 2021; Liu et al., 2023). Matching uses normalized prompt equality or word-8-gram Jaccard similarity of at least 0.5, as well as exact matches between solution abstract syntax trees. We deduplicate normalized task descriptions. The code dataset contains 2,740,479 examples.

MMLU training data. We use the MMLU auxiliary training dataset (Hendrycks et al., 2021a) with rationales generated by Qwen3-30B-A3B-Instruct-2507. We retain examples whose generated answer matches the gold answer and whose rationale satisfies the format and length checks. The training dataset contains 96,112 examples.

Training-token accounting. We count prompt and response tokens in the processed training datasets, including EOS tokens and excluding padding, and multiply the per-epoch count by the training epochs. Table 7 reports these epoch-based estimates. For MATH, eight fine-tuning epochs

contribute 13.625B tokens; the initial 7.5 GSM8K training epochs contribute 4.185B tokens, giving 17.810B tokens in total. These task-training totals are separate from the pretraining of the frozen Qwen3 encoder and the training-time Qwen3 teacher.

B.3 Benchmarks, metrics, and evaluation protocols

Code benchmarks. MBPP-378 contains the 378 MBPP tasks retained by EvalPlus, evaluated with base tests; MBPP-500 contains the original 500-task MBPP test partition (Austin et al., 2021; Liu et al., 2023). HumanEval contains 164 tasks, and HumanEval+ evaluates the same tasks with the additional EvalPlus tests (Chen et al., 2021; Liu et al., 2023). PlaidQ Appendix B.7 (Peng et al., 2026b) names the corresponding MBPP-378 and MBPP-500 benchmarks "MBPP+" and "MBPP", respectively. Figure 1(b) uses MBPP-378 pass@10.

Metrics. We report mean pass@1 and standard deviation across generation seeds. Each question has sixteen generated samples. For pass@k, if c samples are correct, the question-level estimate is $1-\binom{16-c}{k}/\binom{16}{k}$; we average these subset estimates across questions (Chen et al., 2021). HumanEval and HumanEval+ use the same generated samples; HumanEval+ requires passing both the base and extended tests.

ELF evaluation. All headline ELF results use sixteen generation seeds and the sampling settings in Table 6. GSM8K (Cobbe et al., 2021) includes all 1319 test questions, MATH-500 (Hendrycks et al., 2021b; Lightman et al., 2023) all 500 questions, and MMLU (Hendrycks et al., 2021a) all 14,042 test questions. GSM8K evaluation extracts a numerical answer from generated reasoning. MATH-500 uses symbolic equivalence scoring with `math-verify` 0.9.0. MMLU scores the final answer letter extracted from the generated rationale. The headline mathematical-reasoning and code results use $\rho = 8$ early-stop. We tune self-conditioning CFG (SCCFG) on validation data for GSM8K and MATH, and separately on the MBPP and HumanEval benchmarks for code (we choose SCCFG 2 for both MBPP and HumanEval).

Code prompts and response budgets. ELF requests a complete Python solution using the following template:

```
Write a complete Python solution. Return only code.

{task}

Python solution:
```

For HumanEval(+) and MBPP-378, the task is the prompt supplied by EvalPlus. For MBPP-500, it is the problem statement followed by "Your code should pass these tests:" and the three public assertions; trailing assertions are removed only when necessary to fit the prompt limit. The prompt occupies at most 512 tokens of a 1024-token sequence. The response can occupy all remaining positions, giving a maximum of 1024 minus the actual prompt length. We retain the response up to its first EOS token, or to the end of the sequence if EOS is absent.

Code extraction and execution. We use the EvalPlus framework (Liu et al., 2023) for code execution and its sanitizer to extract Python from each completion. If the expected function name is absent and the extracted code contains exactly one top-level function, we add an alias from the expected name to that function, then sanitize for the expected entry point. HumanEval(+) uses HumanEval+ v0.1.10, and MBPP-378 uses the base tests from MBPP+ v0.2.0. MBPP-500 uses its original three assertions and setup code, with a 20-second program timeout. EvalPlus uses a minimum test timeout of 20 seconds and a reference-runtime multiplier of 10. Each program has a 6 GiB memory limit; execution failures and timeouts count as incorrect.

Our Qwen3 evaluations. The Qwen3-0.6B-Base evaluations use zero-shot evaluation, BF16, no chat template, sixteen samples per question, temperature 0.8, and top-p 0.95. We use the same template for both ELF and Qwen3 evaluation: GSM8K (Cobbe et al., 2021) prompts end with "Answer:", and MATH-500 (Hendrycks et al., 2021b; Lightman et al., 2023) prompts end with

"Step-by-Step Answer:" after the question. Both benchmarks use the same answer scoring as our ELF evaluations. For HumanEval and HumanEval+ (Chen et al., 2021; Liu et al., 2023), we concatenate the benchmark prompt and generated continuation before execution since the HumanEval prompts contain the function header. MBPP-378 (Austin et al., 2021; Liu et al., 2023) requests a complete Python solution and includes the task description and first example assertion. Code extraction follows the procedure above; execution uses HumanEval+ v0.1.10 and MBPP+ v0.2.0 as above. Table 9 records response caps and the observed fraction of generations that reach the caps.

Table 9: Response-token caps and the percentage of Qwen3-0.6B-Base(Yang et al., 2025) generations that reach them in our evaluations. Each question has sixteen samples. HumanEval and HumanEval+ share the same generations.

Benchmark	Response cap (tokens)	Cap reached (%)
GSM8K	2,048	0.76
MATH-500	4,096	9.50
MBPP-378	2,048	1.37
HumanEval / HumanEval+	2,048	21.46

B.4 NFE AND PARAMETER ACCOUNTING

NFE includes a final decoder call when one is required. Where exact NFE is unresolved, tables report sampling steps which may not be equivalent to NFE. The conventions are:

- Our results use $N - 1$ denoiser evaluations and one final decoder evaluation, for N NFE total. Self-conditioning guidance is learned during training and does not add an inference evaluation at CFG 1; note, we use CFG 1 throughout the paper.
- PlaidQ uses $K + 1$ evaluations for K denoising steps and a final decode. Applying CFG to PlaidQ adds one additional model evaluation at every denoising step, giving $2K + 1$ (PlaidQ Appendices B.4–B.5 (Peng et al., 2026b)). We note that PlaidQ distills a distinct model for each low-NFE evaluation budget (where distillation applies).
- FMLM+ (Agarwal et al., 2026), DBTM (Tang & Wang, 2026), and S-FLM (Deschenaux & Gulcehre, 2026) all report NFE in their headline results.
- MLFM uses a configured budget of 256 sampling steps (Azangulov et al., 2026). We report steps because the exact NFE for the reported result is unresolved.
- LLaDA and Dream describe sampling steps/diffusion timesteps (Nie et al., 2025; Ye et al., 2025), which we report without converting them to NFE.
- Edit Flow uses 10000 sampling steps for pass@1 and 5000 for pass@10; Mask DFM uses 1000 for each (Havasi et al., 2025). We retain these step budgets without converting them to NFE.

These counts cover generative denoiser and decoder evaluations; the fixed prompt-encoder pass is outside our NFE count. They do not equate FLOPs or latency across architectures.

Parameter counts. ELF's decoder comprises a projection from transformer width to width 1024, followed by the vocabulary projection and their biases. ELF-REG parameter counts include the REPA projector and both REG projections. MLFM's total includes its frozen backbone and trainable adapters (Azangulov et al., 2026); Open-dCoder's count includes shared parameters once. Our frozen Qwen3-0.6B-Base encoder supplies prompt representations at inference. The Qwen3-1.7B-Base teacher supplies training targets and is not used during inference. The additional parameter counts from the frozen Qwen3 encoder/teacher are separate from the ELF parameter counts.

B.5 EXTERNAL COMPARISON SOURCES AND TRAINING BUDGETS

Figure 1 compares continuous and continuous/masked hybrid dLMs using results from PlaidQ, FMLM+, S-FLM, and DBTM (Peng et al., 2026b; Agarwal et al., 2026; Deschenaux & Gulcehre, 2026; Tang & Wang, 2026). Training corpora, tokenizers, and evaluation protocols differ across methods.

Comparison curves. We extract results from FMLM+ (Init) and DBTM using the initialization and refinement-in-loop rows, respectively, from Table 8 of each paper (Agarwal et al., 2026; Tang & Wang, 2026). For S-FLM, we use its best result at each NFE across the architecture, temperature, and velocity-decoding sweeps in Figures 1, 3, and 6–9 (Deschenaux & Gulcehre, 2026); the resulting curve combines different configurations. PlaidQ+CFG uses Figure 5(b), and distilled PlaidQ uses Section 3.3 and Figure 6(b) (Peng et al., 2026b). The Duo (Sahoo et al., 2025) model's GSM8K results that we cite is trained on TinyGSM and evaluated on GSM8K by the S-FLM authors (Deschenaux & Gulcehre, 2026). The MDLM and Duo results in Table 2 are extracted from S-FLM Figure 3(b) at $T = 0.1$ and 1024 NFE (Deschenaux & Gulcehre, 2026). Values extracted from figures are approximate.

Baseline training budgets. PlaidQ Appendix B.1 reports 327.90B non-padding tokens for base training. Its bar shows this base-training budget only; the additional distillation budget is unreported and excluded from our comparisons. For the TinyGSM-trained methods (note: S-FLM, DBTM, and FMLM+ all report training for 250k steps), we estimate non-padding tokens as optimization steps times global batch size times the reported mean length of 194 tokens under the SmolLM tokenizer (S-FLM Figure 5). This rounded mean is measured before filtering sequences longer than 512 tokens, so these budgets are approximate. S-FLM uses $250{,}000 \times 512 \times 194 \approx 24.8$B tokens. FMLM+ (Init) requires an MDLM run and a subsequent FMLM+ run, each with this budget, giving approximately $24.8 + 24.8 = 49.7$B (FMLM+ Appendix B.3 and Table 11). Repeated forwards for self-distillation are not counted as additional training data. DBTM (Tang & Wang, 2026) reports batch size 256 for TinyGSM training, giving $250{,}000 \times 256 \times 194 \approx 12.4$B (note: DBTM reports conflicting batch sizes in different places for TinyGSM training; we take 256 as a potential underestimate for DBTM's training budget).

Scope of training budgets. The bars in Figure 1(d) count task-training tokens, including required task-specific initialization. Foundation pretraining such as the pretraining cost of Qwen3 weights used by PlaidQ and our frozen encoder/teacher is separate and not included in the training budget. Token counts do not measure training FLOPs.

AR mathematical-reasoning sources. Slashes in Table 2 consistently mean base/instruct. Qwen3 Table 8 (Yang et al., 2025) supplies the four-shot GSM8K base score; MobileLLM-R1 Table 9 (Zhao et al., 2026) supplies the zero-shot instruct score without explicitly identifying thinking mode. SmolLM2 Table 4 (Allal et al., 2025) supplies the five-shot GSM8K base scores for Llama-3.2-1B and SmolLM2-1.7B. The Llama-3.2 model card (Meta, 2024) supplies its eight-shot instruct score, and SmolLM2 Table 5 supplies the five-shot SmolLM2 instruct score. For MATH-500, the base scores and the Llama/SmolLM2 instruct scores come from MobileLLM-R1 Tables 8–9, using four and zero demonstrations, respectively. Qwen3 Table 19 supplies its thinking-mode MATH-500 instruct score; the report does not specify its shot count, which we mark n.r.

AR code sources. SmolLM2 Tables 4–5 and Sections 4.7 and 5.4 (Allal et al., 2025) provide the base/instruct HumanEval comparisons for Llama-3.2-1B and SmolLM2-1.7B. MobileLLM-R1 Table 8 (Zhao et al., 2026) supplies the Qwen3-0.6B-Base HumanEval score. The Qwen3-0.6B HumanEval/HumanEval+ pair comes from the same Falcon-H1 evaluation (Zuo et al., 2025), which explicitly disables thinking.

TESS 2. The GSM8K result is approximately 68.9% at 1000 diffusion steps, extracted from TESS 2 Figure 3(b) (Tae et al., 2025). This 7B-scale model uses Mistral-v0.1 initialization, diffusion adaptation, instruction tuning, and GSM8K-specific fine-tuning; TESS 2 Appendix D specifies eight chain-of-thought demonstrations. We note that TESS 2 is not comparable to our method due to its scale.

dLM code evaluations. The LLaDA-8B-Base, Dream-v0-Base-7B, and comparable-scale rows follow PlaidQ Table 1 (Peng et al., 2026b). Open-dCoder (Peng et al., 2025) and oDLM (Peng et al., 2026a) use 128 NFE for the documented P2 configuration with 128 sampling steps and a 128-token response budget. NFE for the LLaDA and Dream code scores are not reported.

Code comparison settings. PlaidQ Appendix B.7 (Peng et al., 2026b) specifies a 128-token completion budget, with total sequence length capped at 2048 tokens. Our ELF evaluations allow the response to fill the remaining positions of a 1024-token sequence, as described in Appendix B.3. PlaidQ uses the HumanEval prompt verbatim and prepends it to the completion before code extraction; ELF instead requests a complete solution. For MBPP-500, PlaidQ includes the problem statement and three public assertions, followed by an opening Python code fence. Its MBPP-378 prompt uses the same format with the task set's base assertions.

PlaidQ truncates completions at benchmark-specific stop strings, removes code fences, and normalizes whitespace. It then retains the longest contiguous span among the first 100 lines that parses as Python and executes the resulting program with a 15-second timeout. Our code extraction uses EvalPlus sanitization with the single-function alias rule and execution limits described above. We use PlaidQ's reported values.

B.6 Analysis protocols

GSM8K generation. The component comparison in Table 4 uses ELF-B epoch 6, early-stop NFE 64, ratio $\rho = 8$, and 16 seeds. Learning curves in Figure 3 evaluate every integer epoch: baseline epochs 1–18 for ELF-B and 1–15 for ELF-L, and epochs 1–12 for both ELF-REG-B and ELF-REG-L. Each point uses early-stop NFE 64, ratio 8, and 4 generation seeds. Both analyses cover all 1319 test questions with the GSM8K sampling settings in Table 6, EOS padding, no model attention mask, and no output truncation. The stripped and unstripped REPA+REG variants share trained weights.

Training progress (GSM8K). One epoch contains 4696 optimizer steps in the GSM8K learning curves, so epoch e corresponds to $4696e$ steps. The speed-up comparison in Figure 3 uses the first evaluated checkpoint that reaches a given accuracy to ensure fairness. Generation seeds share trained weights and do not represent independent training runs. Table 7 reports the corresponding non-padding token budgets, and Figure 1(d) compares task-training budgets across methods.

Continuing trajectories and error injection. Decoder stability (Figure 2(b–e)) and numerical-answer agreement (Table 10) use ELF-B baseline epoch 18 and ELF-REG-B epoch 12, four generation seeds, SCCFG 3, and a master-NFE-512 trajectory. We observe updates 1–64, then 72–504 every eight updates, and update 511. Intermediate observations decode predicted-clean states; the endpoint decodes the updated state. The headline early-exit is update 63, NFE 64. NFE accounting excludes 119 additional diagnostic decoder calls. Stability is averaged over questions within each seed and then across seeds, and we do not measure changes between observations. Coarse-error injection (Figure 8) uses the same checkpoints and four seeds with a 64-NFE reference and a nested 16-NFE coarse grid. Full-span answer retention (Table 28) uses these checkpoints and sixteen generation seeds. Other generation settings follow the GSM8K protocol above.

Encoder comparisons. The GSM8K encoder learning curves (Figure 4, top) use t5-small epochs 1–14 and Qwen3 epochs 1–18, with early-stop NFE 64, ratio 8, EMA 0.9999, and SCCFG 3. The MMLU curves (Figure 4, top) use epochs 10/20/30, NFE 64, EMA 0.999, and SCCFG 2 on 14,042 test questions; whole-sequence (which includes both prompt and response) and prompt token limits are 1,056 and 800 tokens. Both datasets use 4 generation seeds. Recovery (Figure 4, bottom) uses GSM8K epoch 6 and MMLU epoch 30, with each dataset's own EMA and guidance settings. Clean reconstruction uses one decoder call. Corrupted recovery uses one denoiser call and one decoder call for each of 4 seeds.

Perturbation, settling, and training ablations. The single-latent perturbation (Figures 5 and 6) and token-settling analyses (Figure 7) use all three ELF-B conditions at epoch 6, EMA 0.9999, 1 generation seed, and 1024 validation examples. Token-settling uses 64 denoiser updates, CFG 1, and SCCFG 3, with diagnostic decoding every four updates. The secondary training ablations (Tables 25 and 26) use 1 generation seed, CFG 1, SCCFG 3, full-span 64 updates plus one decode (65 NFE), and output truncation. They report validation accuracy on 1024 examples from the validation set and test accuracy on all 1319 GSM8K questions. ELF-B uses epoch 6 and EMA 0.9999; ELF-L uses epoch 2.5 and EMA 0.999.

C EARLY-STOP PROOFS AND THE DISCRETE SAMPLER

C.1 ENDPOINT ERROR

Proof of Proposition 1. Integrating the velocity gives $F_{t,1}(y) = y + \int_t^1 v(y_s, s)\, ds$. Subtracting it from $D_t(y) = y + (1-t)v(y,t)$ proves equation 4. Along the trajectory, $\frac{d}{ds} v(y_s, s) = a_s$. Therefore,

$$e_t(y) = -\int_t^1 \big(v(y_s, s) - v(y,t)\big)\, ds = -\int_t^1 \int_t^s a_u \, du \, ds \tag{10}$$

$$= -\int_t^1 (1-u) a_u \, du. \tag{11}$$

The triangle inequality yields $\|e_t(y)\| \le \int_t^1 (1-u) A \, du = A(1-t)^2/2$. □

C.2 DECODING AND DISTRIBUTIONAL AGREEMENT

The 2-Wasserstein distance W_2 is the minimum root mean squared Euclidean distance over all couplings of the latent distributions. Total variation, TV, is the largest difference in probability assigned to the same event. Small root mean squared endpoint error therefore places the latent distributions close in Wasserstein distance. The decoder condition separately requires the effect on logit differences to remain below the winning-token margin; latents being exactly the same is unnecessary. These bounds concern agreement with the outputs of full-span generation. Whether changed responses improve task accuracy depends on their correctness.

Proof of Proposition 2. We apply the pairwise-logit margin argument of Du & Ma (2026, Theorem 1) at each response position. Let $b = y_1$, $a = \hat{y}_t$, and let $\ell_{i,k}(y)$ denote the logit of token k at position i. If k_i wins at b, define $g_{i,k}(y) = \ell_{i,k_i}(y) - \ell_{i,k}(y)$ for each competitor $k \neq k_i$. By assumption, $g_{i,k}(b) \ge m$ and $|g_{i,k}(a) - g_{i,k}(b)| \le L\|a - b\|$. Consequently, $g_{i,k}(a) \ge m - L\|e_t(y_t)\| > 0$. Every endpoint winner through the first EOS is unchanged. The EOS position and the decoded response therefore agree. If the endpoint has no EOS, the argument applies to the full response canvas. The decoder input is the full latent state, including positions after EOS that can influence earlier logits through attention. □

Proof of Corollary 1. The joint law of $(\hat{y}_t, y_1)$ is a coupling of P_t and P_1. The definition of Wasserstein distance gives

$$W_2^2(P_t, P_1) \le \mathbb{E}\|\hat{y}_t - y_1\|^2 = \mathbb{E}\|e_t(y_t)\|^2.$$

Applying the deterministic decoder gives a coupling of Q_t and Q_1. The coupling inequality and Proposition 2 imply

$$\mathrm{TV}(Q_t, Q_1) \le \Pr[\mathcal{D}(\hat{y}_t) \neq \mathcal{D}(y_1)] \le \Pr[L\|e_t(y_t)\| \ge m].$$

The margin and local Lipschitz constant may depend on the sampled trajectory. They must satisfy the proposition almost surely. For any accuracy function taking values in $[0, 1]$, the difference in expected accuracy is at most $\mathrm{TV}(Q_t, Q_1)$. □

Bounding the decoder failure probability. Under the assumptions of Corollary 1, fix the prompt, guidance scale, and continuous exit time $t < 1$. All probabilities and expectations below are over initial noise. For any fixed $\eta > 0$,

$$\Pr[L\|e_t(y_t)\| \ge m] \le \Pr[m \le L\eta] + \frac{\mathbb{E}\|e_t(y_t)\|^2}{\eta^2}.$$

To see this, if $\|e_t(y_t)\| < \eta$ and $m > L\eta$, then $L\|e_t(y_t)\| \le L\eta < m$ because $L \ge 0$. Thus

$$\{L\|e_t(y_t)\| \ge m\} \subseteq \{m \le L\eta\} \cup \{\|e_t(y_t)\| \ge \eta\}.$$

The union bound and Markov's inequality applied to $\|e_t(y_t)\|^2$ give

$$\Pr[L\|e_t(y_t)\| \ge m] \le \Pr[m \le L\eta] + \Pr[\|e_t(y_t)\| \ge \eta]$$

$$\le \Pr[m \le L\eta] + \frac{\mathbb{E}\|e_t(y_t)\|^2}{\eta^2}.$$

The second moment is finite because $\|e_t(y_t)\|^2 \le 2\|\hat{y}_t\|^2 + 2\|y_1\|^2$ and both endpoint distributions have finite second moments. The margin m and sensitivity L may depend on the trajectory; no independence is required. The first term measures the probability that the margin is small relative to decoder sensitivity at discrepancy scale η. The second controls the probability of an endpoint error at least η. Corollary 1 therefore gives a small TV bound when both terms are small.

C.3 Self-conditioning, REG, and the denominator clamp

Algorithm 1 ELF inference with prefix early-stop. Adapted from Algorithm 2 of ELF (Hu et al., 2026). We apply prefix early-stop to ELF's generation procedure by executing only the prefix of a longer sampling time grid and decoding the latest predicted-clean states. Full-span generation denoises the full time grid and decodes the updated endpoint z. Both full-span and early-stop use $N-1$ denoiser evaluations and one decoder evaluation.

```
# N: total NFE, including one decoder call (N >= 2)
# rho: > 1 for prefix early-stop; 1 for full-span generation
# z: generated text and REG states; baseline includes only text
# shape: shape of these states; operations apply to both text and REG
# sigma: noise scale; t_eps: denominator clamp
# The prompt stays fixed; net includes the configured guidance.

master_nfe = rho * N
ts = [0] + sort(sample_logit_normal(master_nfe - 2)) + [1]
z = sigma * randn(shape)
x_pred_prev = zeros_like(z)

for i in range(N - 1):
    t = ts[i]
    dt = ts[i + 1] - t
    x_pred = net(z, t, self_cond=x_pred_prev, mode="denoise")
    # Convert clean prediction to velocity and update the state.
    v = (x_pred - z) / max(1 - t, t_eps)
    z = z + dt * v
    x_pred_prev = x_pred

# Early-stop decodes the latest clean prediction.
if rho > 1:
    z = x_pred

# Decode text hidden states, with REG available as context.
h = net(z, t=1, self_cond=0, mode="decode")
token_logits = unembed(h)
tokens = argmax(token_logits, axis=-1)
return tokens
```

The implemented sampler keeps more state than the response latents. Let y_j collect the current text and REG latent states at grid time t_j, and let q_j collect their clean predictions from step $j-1$, so that $q_j = d_{j-1}$ for $j \ge 1$, with $q_0 = 0$. The baseline state contains only text. Here j indexes updates, whereas the analysis elsewhere uses continuous time. With the prompt fixed, let $d_j = D_\theta(y_j, q_j, t_j)$ be the clean prediction including the self-conditioning guidance. Following Algorithm 1, for $c_j = \max(1 - t_j, t_\epsilon)$ and $\Delta_j = t_{j+1} - t_j$, Euler sampling is

$$v_j = \frac{d_j - y_j}{c_j}, \qquad y_{j+1} = y_j + \Delta_j v_j, \qquad q_{j+1} = d_j. \tag{12}$$

Thus (y_j, q_j) determines the remaining discrete trajectory. A flow map on y_j alone would discard self-conditioning information.

Let M denote the master NFE budget where M determines the denoising time grid, as in Section 3. The grid has $M-1$ denoiser updates and ends at $t_{M-1} = 1$; its final decoder evaluation completes the NFE budget. After k updates, early-stop decodes d_{k-1}, which was predicted at t_{k-1} before the

last update. Full-span generation decodes y_{M-1}. Writing $j = k - 1$ and summing the updates in equation 12 gives

$$d_j - y_{M-1} = c_j v_j - \sum_{\ell=j}^{M-2} \Delta_\ell v_\ell \tag{13}$$

$$= \big(c_j - (1 - t_j)\big) v_j + \sum_{\ell=j}^{M-2} \Delta_\ell (v_j - v_\ell). \tag{14}$$

The first term is the denominator-clamp correction. The second measures changes in velocity along the actual self-conditioned trajectory. The decoder-agreement proposition applies directly to the resulting endpoint discrepancy because both decoder calls use time 1 and zero self-conditioning inputs.

Numerical prefix error. In the smooth, unclamped model, let $\tilde{y}_t$ be a numerical approximation to y_t. If D_t is K_D-Lipschitz between these states, then

$$\|D_t(\tilde{y}_t) - F_{t,1}(y_t)\| \le K_D \|\tilde{y}_t - y_t\| + \|e_t(y_t)\|.$$

This separates error accumulated before the exit from replacement of the remaining flow. Our trajectory measurement instead compares states on the same numerical trajectory, as in equation 14.

C.4 Endpoint-error and discrete-acceleration measurements

Endpoint-error measurement. We use the checkpoints and master-NFE-512 grid specified in Appendix B.6. A first deterministic pass obtains the endpoint latent state. A second pass from the same initial text and REG noise records each clean prediction and its discrepancy from that endpoint. Text error includes the entire generated canvas because the decoder attends beyond the first EOS. Token stability is computed backwards from the endpoint: a final-response token position is stable only if its token matches at the current and every later saved observation. Missing positions count as mismatches. The stable-token fraction is the proportion of final-response positions, including the first EOS and excluding padding, whose tokens are stable. We average this fraction over questions and seeds. Figure 2(b) reports the mean text Euclidean error, averaging questions within each seed and then equally over four seeds. Decoded-response agreement includes EOS status and length. The last predicted-clean observation we make is at NFE 505; the NFE-512 observation uses the updated endpoint itself. The extra reference pass and diagnostic decoder calls are analysis-related costs, separate from the reported exit NFE.

Numerical-answer agreement and correctness transitions. Table 10 compares NFE 64 with NFE 512 on the same question–seed pairs. We extract answers by taking the first number after the last answer marker, or the last number when no marker is present. Missing extracted answers count as disagreement and as incorrect. The difference between improvement and regression rates is the change in accuracy.

Prediction time and training-interpolant SNR. For the NFE-64 early-stop with $\rho = 8$, the master time grid has 511 denoiser updates, 510 interior time steps drawn from a logit-normal distribution, and endpoints 0 and 1. After 63 updates, early-stop decodes the latest clean prediction, produced at denoising grid index 62 before the last update; the updated state is at index 63. Each generation seed supplies one schedule shared by both models. Table 11 summarizes the four distinct prediction times.

For the training interpolant in equation 1, the signal is the clean encoder state x, scaled by t, and the noise is $(1 - t)\sigma\varepsilon$. With approximately unit-variance clean encoder states, nominal power SNR is $t^2/[\sigma^2(1 - t)^2]$, with $\sigma = 2$. The clean encoder standardizes each state across features using variance. We compute SNR separately at each saved prediction time, then report the mean and range over schedules. This calculation describes the training corruption level at the exit time; it does not decompose the generated ODE state into signal and noise because the generated trajectory need not follow the training noise-data interpolation.

Table 10: Numerical-answer agreement and correctness transitions from early-stop NFE 64 to the endpoint at NFE 512 on GSM8K. Each early-stop prediction is paired with the endpoint of its own master-NFE-512 trajectory. ELF-B baseline uses epoch 18 and ELF-REG-B epoch 12. Each column covers 5,276 question–seed pairs: all 1,319 test questions and four generation seeds. Most numerical answers agree despite low complete-response agreement; correctness gains and losses nearly cancel for ELF-B baseline.

Measurement	ELF-B baseline	ELF-REG-B (ours)
Numerical-answer agreement (%)	80.27	87.11
Complete-response agreement (%)	1.46	8.61
Incorrect → correct, count (%)	79 (1.50)	70 (1.33)
Correct → incorrect, count (%)	78 (1.48)	36 (0.68)
Accuracy at NFE 64 (%)	36.71	37.64
Accuracy at NFE 512 (%)	36.73	38.29
Net accuracy change (percentage points)	+0.02	+0.64

Table 11: Prediction time and nominal training-interpolant power SNR at the NFE-64 exit ($\rho = 8$) in Figure 2. Means and ranges are over four saved logit-normal schedules, shared by ELF-B baseline and ELF-REG-B (ours). SNR is computed separately for each schedule before averaging.

Quantity	Mean	Range
Prediction time t	0.0825	0.0755–0.0858
Power SNR ($\times 10^{-3}$)	2.0272	1.6667–2.2043

Discrete acceleration measurement. We reuse the checkpoints and GSM8K settings of the endpoint-error measurement, replacing logit-normal time sampling with **deterministic uniform time grid** over 512 Euler updates. From the sampling velocities in equation 12, we compute

$$t_j = \frac{j}{512}, \qquad \hat{a}_j = \frac{v_{j+1} - v_j}{1/512}, \qquad \hat{A}_j = \max_{j \le r < 512} \|\hat{a}_r\|, \qquad 0 \le j < 512. \tag{15}$$

The discrete acceleration $\hat{a}_j$ measures the change in sampling velocity per unit time between consecutive grid points. The remaining maximum $\hat{A}_j$ is the largest acceleration norm from step j through step 511. The joint norm includes the full response canvas, including positions after EOS, and REG when present, and excludes the prompt. One additional denoiser evaluation at $t = 1$ retains the final self-conditioning and supplies v_{512} for the purposes of measuring the acceleration without changing the endpoint. Velocities use the denominator clamp $c_j = \max(1 - t_j, 0.05)$. The two-pass procedure above measures endpoint errors on these same uniform time-grid trajectories. Due to the different time-grid (uniform vs logit-normal), the prediction at $t = 5/64$ uses 41 denoiser evaluations and one decode, giving NFE 42; the reference pass and diagnostic evaluations are separate analysis costs.

Table 12 compares the remaining maxima and endpoint errors. The maximum $\hat{A}_j$ measures finite velocity differences along the discrete self-conditioned trajectory. Proposition 1 instead assumes a bound on acceleration at every time along a smooth continuous flow. Since we compute a discrete approximation of the acceleration, substituting $\hat{A}_j$ for A does not give a certified continuous endpoint-error bound. Appendix C.3 accounts for self-conditioning and the denominator clamp.

Table 12: Remaining joint acceleration and joint endpoint error at $t = 5/64$ on GSM8K, using 512 Euler updates with uniform time sampling. Each early-stop prediction is compared with its own trajectory endpoint. Acceleration is the median over all 1,319 questions within each generation seed; endpoint error is the mean Euclidean norm. Both statistics then average four generation seeds. ELF-B baseline uses epoch 18 and ELF-REG-B epoch 12.

Condition	Median $\hat{A}$ ($\times 10^5$)	Mean joint endpoint error
ELF-B baseline	6.215	530.78
ELF-REG-B (ours)	0.946	490.80

D ADDITIONAL RESULTS AND ANALYSES

D.1 ADDITIONAL HEADLINE RESULTS

GSM8K across NFE. Table 13 compares the two ELF sizes with FMLM+ (Init) and DBTM at matched NFE. All ELF rows use early-stop. Our numerical observations underlying Figure 1(c) appear in Table 22.

Table 13: GSM8K mean pass@1 (%) across NFE. ELF-REG-L achieves the highest accuracy; at comparable parameter counts, ELF-REG-B also outperforms FMLM+ (Init) and DBTM at every NFE. FMLM+ (Init) and DBTM values are from their respective Table 8 (Agarwal et al., 2026; Tang & Wang, 2026); ELF values are our evaluations. Bold marks the best value in each NFE column within each group.

Model	4	8	16	32	64
ELF-L baseline	27.64	43.20	48.58	50.82	51.85
ELF-REG-L (ours)	**31.76**	**47.34**	**52.89**	**55.04**	**55.96**
ELF-B baseline	10.24	25.88	33.85	36.81	37.71
ELF-REG-B (ours)	**11.82**	**26.20**	**34.42**	**37.15**	**38.11**
FMLM+ (Init)(Agarwal et al., 2026)	5.10	15.10	26.10	31.80	33.60
DBTM(Tang & Wang, 2026)	5.70	10.10	14.30	16.80	16.20

Code across inference and sample budgets. Tables 14 and 15 report pass@1 and pass@10 at each method's sampling budget. Each PlaidQ-D row uses the student trained for that particular denoising budget. Table 16 gives the ELF-L sample-budget sweep at NFE 128 and the available PlaidQ comparisons. ELF-REG-L reaches 55.05% pass@10 on MBPP-378 and 46.69% on HumanEval+, compared with 52.90% and 42.84% for the ELF baseline. On MBPP-500, the baseline reaches 40.50% and ELF-REG-L reaches 39.92%.

Table 14: Code pass@1 (%) across inference budgets. ELF rows are our early-stop evaluations; external values are from PlaidQ Table 1 and Section 3.3 (Peng et al., 2026b). Budget entries specify NFE or sampling steps; dashes denote unavailable measurements. Bold marks the maximum in each benchmark column over all methods and budgets.

Model	NFE / steps	MBPP-500	MBPP-378	HumanEval	HumanEval+
ELF-L baseline	4 NFE	4.20	8.10	5.68	5.30
ELF-L baseline	8 NFE	9.68	16.19	10.44	9.76
ELF-L baseline	16 NFE	12.89	20.42	14.10	13.38
ELF-L baseline	32 NFE	15.00	23.84	17.19	16.27
ELF-L baseline	64 NFE	15.95	25.69	18.33	17.49
ELF-L baseline	128 NFE	17.19	26.75	19.66	18.75
ELF-REG-L (ours)	4 NFE	6.46	10.14	7.20	6.71
ELF-REG-L (ours)	8 NFE	12.18	18.96	12.84	12.39
ELF-REG-L (ours)	16 NFE	15.03	23.26	16.69	16.08
ELF-REG-L (ours)	32 NFE	16.93	25.79	19.25	18.25
ELF-REG-L (ours)	64 NFE	18.05	28.19	21.19	20.08
ELF-REG-L (ours)	128 NFE	**19.10**	**28.92**	**22.56**	**21.46**
Mask DFM(Gat et al., 2024)	1,000 steps	6.20	–	9.10	7.90
Edit Flow(Havasi et al., 2025)	10,000 steps	10.00	–	12.80	10.40
Open-dCoder(Peng et al., 2025)	128 NFE	16.70	23.90	20.80	17.60
oDLM(Peng et al., 2026a)	128 NFE	15.70	22.99	17.87	16.40
PlaidQ(Peng et al., 2026b)	129 NFE	12.19	22.18	19.02	18.05
PlaidQ+CFG(Peng et al., 2026b)	257 NFE	15.53	24.58	22.04	19.70
PlaidQ-D1(Peng et al., 2026b)	2 NFE	–	–	7.07	–

MATH-500. Table 17 reports mean pass@1 across NFE, and Table 18 reports pass@k at NFE 64. ELF-REG-L reaches 13.31% mean pass@1 at NFE 64 and 13.39% at NFE 128. Its pass@10 at NFE 64 is 44.04%, compared with 36.73% for the baseline.

Table 15: Code pass@10 (%) across inference budgets, using the same benchmark mapping and NFE / steps convention as Table 14. The guided PlaidQ Table 1 row uses NFE 257. Bold marks the maximum in each benchmark column over all methods and budgets.

Model	NFE / steps	MBPP-500	MBPP-378	HumanEval	HumanEval+
ELF-L baseline	4 NFE	16.30	26.84	16.39	15.25
ELF-L baseline	8 NFE	28.13	42.86	30.43	28.53
ELF-L baseline	16 NFE	34.75	47.57	35.67	34.15
ELF-L baseline	32 NFE	37.57	50.43	42.93	40.09
ELF-L baseline	64 NFE	38.41	52.88	45.04	42.28
ELF-L baseline	128 NFE	**40.50**	52.90	45.93	42.84
ELF-REG-L (ours)	4 NFE	22.81	31.48	20.79	19.80
ELF-REG-L (ours)	8 NFE	31.45	46.39	32.95	32.13
ELF-REG-L (ours)	16 NFE	36.43	49.99	41.21	39.86
ELF-REG-L (ours)	32 NFE	38.08	52.78	46.04	42.90
ELF-REG-L (ours)	64 NFE	38.85	54.98	48.14	44.26
ELF-REG-L (ours)	128 NFE	39.92	**55.05**	**48.87**	**46.69**
Mask DFM(Gat et al., 2024)	1,000 steps	25.00	–	17.60	13.40
Edit Flow(Havasi et al., 2025)	5,000 steps	36.40	–	24.30	20.70
Open-dCoder(Peng et al., 2025)	128 NFE	38.40	53.60	38.40	35.20
oDLM(Peng et al., 2026a)	128 NFE	37.00	49.47	35.98	31.10
PlaidQ(Peng et al., 2026b)	129 NFE	21.44	34.33	25.72	24.03
PlaidQ+CFG(Peng et al., 2026b)	257 NFE	32.48	44.79	39.87	36.74
PlaidQ-D1(Peng et al., 2026b)	2 NFE	–	6.52	8.53	–
PlaidQ-D4(Peng et al., 2026b)	5 NFE	–	30.94	15.85	–
PlaidQ-D8(Peng et al., 2026b)	9 NFE	–	33.56	23.19	–
PlaidQ-D16(Peng et al., 2026b)	17 NFE	–	40.49	31.78	–

Table 16: Pass@k (%) on MBPP-500, MBPP-378, HumanEval, and HumanEval+. ELF-L baseline and ELF-REG-L use NFE 128; we report sampling NFE for PlaidQ and PlaidQ+CFG. Bold marks the best result for each benchmark and sample budget across all methods.

Model	NFE	pass@1	pass@2	pass@4	pass@8	pass@10	pass@16
MBPP-500							
ELF-L baseline	128	17.19	24.10	31.28	**38.34**	**40.50**	**44.80**
ELF-REG-L (ours)	128	**19.10**	**25.91**	**32.41**	38.18	39.92	43.40
PlaidQ(Peng et al., 2026b)	129	12.19	–	–	–	21.44	–
PlaidQ+CFG(Peng et al., 2026b)	257	15.53	–	–	–	32.48	–
MBPP-378							
ELF-L baseline	128	26.75	35.63	43.58	50.72	52.90	57.14
ELF-REG-L (ours)	128	**28.92**	**37.88**	**45.82**	**52.89**	**55.05**	**59.52**
PlaidQ(Peng et al., 2026b)	129	22.18	–	–	–	34.33	–
PlaidQ+CFG(Peng et al., 2026b)	257	24.58	–	–	–	44.79	–
HumanEval							
ELF-L baseline	128	19.66	26.79	34.59	43.11	45.93	51.83
ELF-REG-L (ours)	128	**22.56**	**30.47**	**38.35**	**46.42**	**48.87**	**53.66**
PlaidQ(Peng et al., 2026b)	129	19.02	–	–	–	25.72	–
PlaidQ+CFG(Peng et al., 2026b)	257	22.04	–	–	–	39.87	–
HumanEval+							
ELF-L baseline	128	18.75	25.47	32.66	40.32	42.84	48.17
ELF-REG-L (ours)	128	**21.46**	**29.02**	**36.45**	**44.22**	**46.69**	**51.83**
PlaidQ(Peng et al., 2026b)	129	18.05	–	–	–	24.03	–
PlaidQ+CFG(Peng et al., 2026b)	257	19.70	–	–	–	36.74	–

Table 17: MATH-500 mean pass@1 (%) across NFE, ELF-L baseline and ELF-REG-L with early-stop generation. Bold marks the best value in each NFE column.

Model	4	8	16	32	64	128
ELF-L baseline	5.47	7.58	8.56	9.65	9.65	10.55
ELF-REG-L (ours)	**7.04**	**9.76**	**11.53**	**12.40**	**13.31**	**13.39**

Table 18: MATH-500 pass@k (%) at NFE 64, ELF-L baseline and ELF-REG-L with early-stop generation. Bold marks the best value at each sample budget.

Model	pass@1	pass@2	pass@4	pass@8	pass@10	pass@16
ELF-L baseline	9.65	15.65	23.64	33.36	36.73	44.00
ELF-REG-L (ours)	**13.31**	**20.75**	**29.95**	**40.41**	**44.04**	**52.20**

MMLU. ELF-REG-B improves ELF-B mean pass@1 at every NFE budget (Table 19), reaching 46.71% at NFE 64 compared with 45.34% for the baseline.

Table 19: MMLU mean pass@1 (%), ELF-B baseline and ELF-REG-B with full-span generation. Bold marks the best value in each NFE column.

Model	4	8	16	32	64
ELF-B baseline	28.67	44.00	45.23	45.32	45.34
ELF-REG-B (ours)	**30.27**	**45.60**	**46.66**	**46.69**	**46.71**

Early stopping in FMLM+. Table 20 compares the FMLM+ (Init) results with our reproduction of full-span (which is the default setup of FMLM+) and early-stop evaluations of the same checkpoint. Both local rows average the same four generation seeds over all 1319 GSM8K questions, with Python execution scoring. Full-span generation allocates its token-commit schedule to full NFE budget; early-stop generation follows a master schedule of NFE 64 and reads its complete prediction at each exit. Early stopping lowers accuracy at NFE 4–32 for FMLM+. Its posterior-refinement sampler commits confident token predictions and re-noises uncommitted positions. The accuracy loss likely reflects truncation of the 64-NFE token-commit schedule before completion. At NFE 64, both settings share the full-span endpoint.

Table 20: FMLM+ (Init) GSM8K mean pass@1 (%) across NFE. Reference values are from Table 8 in (Agarwal et al., 2026); our two rows use 4 generation seeds.

Setting	4	8	16	32	64
FMLM+ (Init)(Agarwal et al., 2026)	5.10	15.10	26.10	31.80	33.60
FMLM+ (Init) (our reproduction)(Agarwal et al., 2026)	6.05	15.85	25.38	31.37	33.36
FMLM+ (Init) early-stop, master NFE 64(Agarwal et al., 2026)	2.31	6.25	16.83	29.64	33.36

D.2 Numerical values underlying Figure 1

Tables 21 and 22 report our numerical results underlying Figure 1(b–c). Table 23 gives the training-token comparison in panel (d), including the external methods.

D.3 Additional representation analyses

D.3.1 Clean encoder comparisons

Clean encoder configurations. Figure 4 compares ELF-B baseline using Qwen3-0.6B-Base or t5-small as the text encoder. The configurations differ in data preprocessing as well as the clean text encoder due to tokenizer differences.

Gold-answer recovery. We test on epoch 6 and 30 checkpoints for GSM8K and MMLU, respectively, for gold-answer recovery. GSM8K supplies full gold worked solutions for both text encoder configurations. MMLU supplies only the gold final-answer letter at epoch 30. Gold-answer recovery without corruption decodes encoder states directly with one decoder-mode forward pass. For gold-answer recovery with corrupted inputs, we choose t per example to match the requested noise/signal RMS ratio in equation 1, then use one denoiser evaluation followed by decoding. The ratio is $\mathrm{RMS}((1-t)\sigma\varepsilon)/\,\mathrm{RMS}(tx)$ over valid response positions, including the first EOS. For a requested noise/signal RMS ratio q, let $X = \mathrm{RMS}(x)$ and $E = \mathrm{RMS}(\varepsilon)$ over these positions; we use

Table 21: Figure 1(b): MBPP-378 pass@10 (%).

ELF-REG-L (ours)						
NFE	4	8	16	32	64	128
pass@10	31.48	46.39	49.99	52.78	54.98	55.05

Table 22: Figure 1(c): GSM8K mean pass@1 (%).

ELF-REG-B (ours)					
NFE	4	8	16	32	64
Mean pass@1	11.82	26.20	34.42	37.15	38.11

Table 23: Figure 1(d): task-training tokens. Token counts for methods that use TinyGSM are estimated using the reported mean sequence length; PlaidQ reports its base-training tokens.

Task	Model	Tokens (B)
Code	ELF-REG-L (ours)	14.958
Code	PlaidQ(Peng et al., 2026b)	327.900
GSM8K	ELF-REG-B (ours)	6.696
GSM8K	FMLM+ (Init)(Agarwal et al., 2026)	49.664
GSM8K	S-FLM(Deschenaux & Gulcehre, 2026)	24.832
GSM8K	DBTM(Tang & Wang, 2026)	12.416

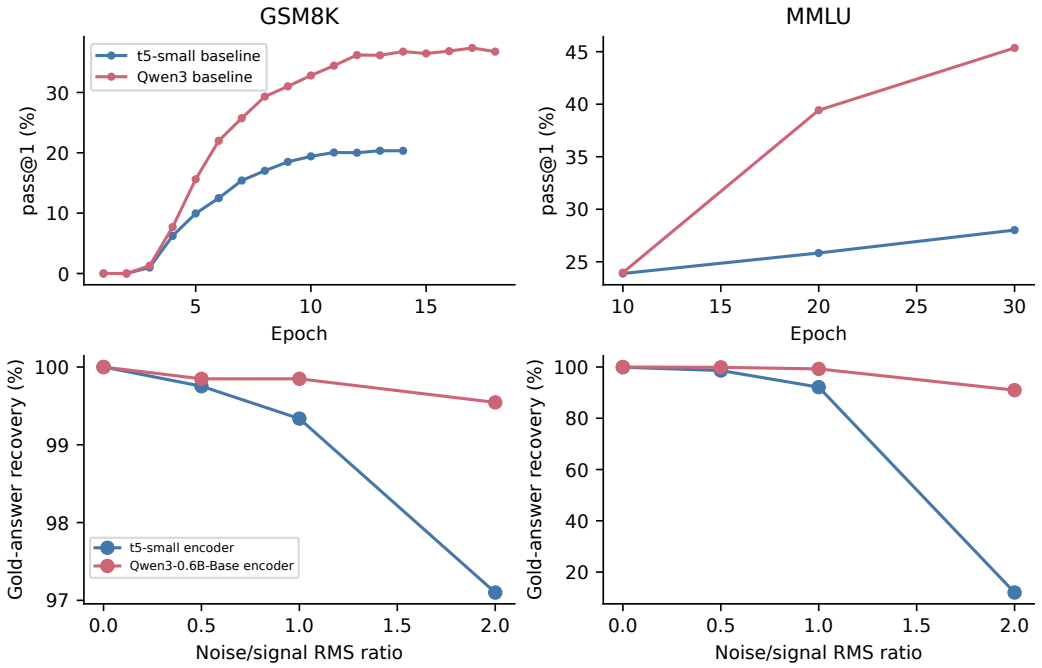

Figure 4: As a text encoder, Qwen3-0.6B-Base supports stronger learning and stronger gold-answer recovery than t5-small under comparable training and evaluation configurations. Top: baseline learning curves on GSM8K and MMLU; the MMLU curves use full-span generation at NFE 64. Bottom: gold-answer recovery after corrupting representations of supplied gold answers. Both configurations achieve reliable gold-answer recovery without corruption, while Qwen3-0.6B-Base retains higher gold-answer recovery when inputs are corrupted, especially on MMLU. Generation and gold-answer recovery with corrupted inputs average four seeds; gold-answer recovery without corruption is evaluated once.

$t = \sigma E/(qX + \sigma E)$. Recovery scores the final numerical answer on GSM8K and the final-answer letter on MMLU, averaging questions within each seed and then across seeds. This evaluates each encoder together with its trained denoiser and decoder.

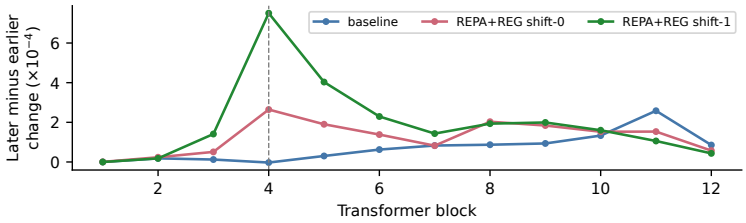

Figure 5: REPA target position shift-1 strengthens forward influence at the supervised block on GSM8K. We replace the clean latent at a token representing a single digit with the latent obtained after substituting a different digit. We measure hidden-state change between the original and perturbed passes as $1-$cosine similarity, then subtract the mean change over positions before the edited token from the mean change over positions after it. Positive values indicate stronger forward influence. At $t = 0.7$, shift-1 has a larger effect than shift-0 at the REPA-supervised block (dashed line), but this advantage disappears at the final block. ELF-B checkpoints are from epoch 6. REPA+REG shift-0 and shift-1 are ELF-REG-B settings.

D.3.2 Single-latent perturbation across corruption times

Zero REPA target position shift (which is the default setup) aligns student position i to teacher position i; REPA target position shift-1 aligns student position i to teacher position $i-1$. The student remains bidirectional. Figure 5 shows stronger forward influence for shift-1 at the supervised block at $t = 0.7$; this advantage disappears at the final block.

We select a token representing a single digit near the midpoint of a GSM8K gold rationale and replace that digit with a different digit that also occupies one token. We encode the modified response and copy only the clean latent at the replacement token's position into the original response; all other clean latents stay fixed. The original and perturbed denoiser passes share the noise, prompt, REG state, and self-conditioning inputs. At the hidden states following each block, we measure hidden-state change as $1-$cosine similarity. The earlier and later windows exclude the edited position and have equal length, given by the smaller distance from the edit to either end of the valid response. Let δ_{edit} be the change at the edited position, and let δ_- and δ_+ be the mean changes in these earlier and later windows. Positive $\delta_+ - \delta_-$ indicates stronger forward influence. Each plotted value averages this difference over the same 1024 validation examples, separately for each block, corruption time, and condition.

Figure 6 extends the $t = 0.7$ comparison to $t = 0.05$ and 0.2. Shift-1 has a larger later-minus-earlier change than shift-0 at block 4 for all three times. At block 12, shift-1 has a slightly larger difference at $t = 0.05$, but a smaller difference at $t = 0.2$ and 0.7. The final-block pattern at $t = 0.7$ therefore depends on the corruption time.

Table 24 separates perturbation magnitude from direction at blocks 4 and 12. For each example, normalized direction is $\nu = (\delta_+ - \delta_-)/(\delta_+ + \delta_-)$ when the denominator is positive, and zero otherwise. We average this ratio across examples. Table 24 also reports the edited-position change, both window means, and the difference between window means. All statistics use the same single-latent interventions as Figure 6.

D.3.3 Position-dependent token settling

We decode predicted-clean states every four updates along a trajectory with 64 denoiser updates, and decode the final updated state at update 64. A position's settling time is the last observed update in the trajectory where the token changes (in other words, the token does not change at subsequent observed updates), with update 4 assigned if it never changes. Figure 7 averages this count within ten response-position bins, excluding prompt and padding. REPA+REG shift-0 settles earlier in every bin. Settling does not progress monotonically from earlier to later positions.

Table 24: Perturbation magnitude and direction on GSM8K at the supervised and final blocks. Values average paired single-latent interventions for the three ELF-B settings at epoch 6. REPA+REG shift-0 and shift-1 are ELF-REG-B (ours) settings. Shift-1 has a larger later-minus-earlier change than shift-0 at block 4 at each corruption time. At block 12, this ordering reverses at $t = 0.2$ and 0.7, while the difference at $t = 0.05$ is small. δ_{edit} measures change at the edit, δ_- and δ_+ are means in matched earlier and later windows, and ν is the mean per-example normalized difference. Hidden-state change is $1-$cosine similarity.

		Hidden-state change ($\times 10^{-4}$)				
Block	Condition	δ_{edit}	δ_-	δ_+	$\delta_+ - \delta_-$	ν
$t = 0.05$						
4	baseline	1.562	0.120	0.123	0.003	0.016
4	REPA+REG shift-0	2.422	0.426	0.444	0.019	0.019
4	REPA+REG shift-1	1.867	0.407	0.449	0.042	0.045
12	baseline	0.176	0.068	0.095	0.027	0.117
12	REPA+REG shift-0	0.191	0.067	0.085	0.019	0.095
12	REPA+REG shift-1	0.170	0.066	0.086	0.020	0.091
$t = 0.2$						
4	baseline	76.897	0.365	0.300	-0.066	-0.090
4	REPA+REG shift-0	129.010	1.008	1.397	0.390	0.126
4	REPA+REG shift-1	100.008	0.799	1.947	1.148	0.318
12	baseline	6.017	0.139	0.267	0.128	0.120
12	REPA+REG shift-0	4.714	0.118	0.225	0.107	0.115
12	REPA+REG shift-1	5.356	0.142	0.232	0.090	0.107
$t = 0.7$						
4	baseline	694.913	0.259	0.230	-0.029	-0.052
4	REPA+REG shift-0	1240.237	0.853	3.497	2.645	0.513
4	REPA+REG shift-1	582.228	0.651	8.156	7.504	0.822
12	baseline	134.540	0.180	1.033	0.854	0.461
12	REPA+REG shift-0	92.139	0.134	0.708	0.574	0.450
12	REPA+REG shift-1	63.242	0.137	0.571	0.434	0.362

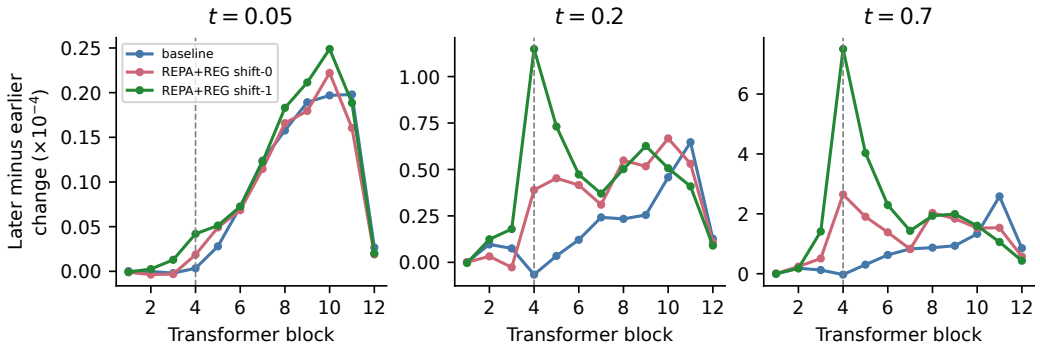

Figure 6: On GSM8K, the direction of perturbation influence depends on the block and corruption time. The $t = 0.7$ panel shows the same measurements as Figure 5. Each panel shows later-minus-earlier hidden-state change, measured as $1-$cosine similarity in equally sized windows around one edited latent. Positive values indicate stronger forward influence. Shift-1 exceeds shift-0 at the REPA-supervised block (dashed line) at all three time t values, while their final-block ordering varies with time. We choose epoch 6 ELF-B checkpoints. REPA+REG shift-0 and shift-1 are ELF-REG-B settings.

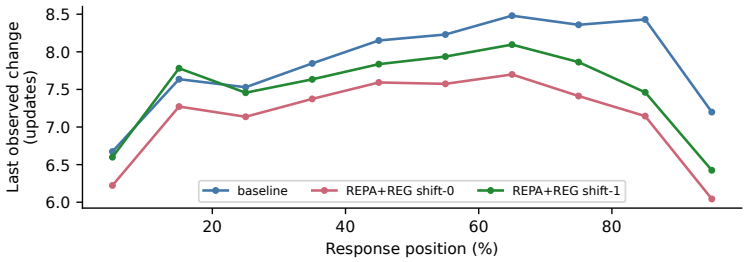

Figure 7: On GSM8K, REPA+REG shift-0 settles earlier in every response-position bin. Settling is the last observed update at which a token changes; lower values indicate earlier settling. Positions are grouped within the final response for three ELF-B conditions at epoch 6. The pattern is not a monotonic progression from earlier to later tokens. REPA+REG shift-0 and shift-1 are ELF-REG-B settings.

D.4 Training ablations

Training settings. Tables 25 and 26 compare the performance of different training settings within each model size, at epoch 6 for ELF-B and epoch 2.5 for ELF-L. Appendix B.6 specifies their evaluation protocol. We use the validation pass@1 accuracy in Tables 25 and 26 to determine the settings for our headline results.

D.5 Sampling analyses

D.5.1 Effect of the early-stop ratio

The early-stop sweep evaluates $\rho \in \{1, 2, 4, 8, 16\}$ using the same ELF-REG-B checkpoint and all 16 generation seeds. For $\rho > 1$, generation decodes the predicted-clean states after $N - 1$ updates of the master grid. The $\rho = 1$ control is full-span generation and decodes the updated terminal states. Table 27 reports mean pass@1. We choose $\rho = 8$ to balance stronger low-NFE performance with gains at high NFE. Figure 1 and our headline results on math reasoning and code use $\rho = 8$ at every budget.

Sensitivity to coarse-sampling errors. We compare a full-span 64-NFE reference denoising trajectory with a nested 16-NFE coarse trajectory. At updates 16, 32, and 48, we inject the difference

Table 25: GSM8K training ablations for ELF-B at epoch 6. Reference configurations appear in the top rows. Each remaining row follows the default REPA+REG setup (alignment depth 4, last-token REG target, response-only text REPA, and $\lambda_{\mathrm{REG}} = 0.1$), except for the stated difference in setting. The time interval restriction for REPA loss applies to text positions, excluding REG, on denoising examples; decoder-example alignment is unchanged. Values are single-seed pass@1 (%) at 65 NFE. Bold marks the best validation and test values across all rows.

Condition	Validation pass@1 (%)	Test pass@1 (%)
ELF-B baseline	21.58	17.89
ELF-B baseline, Qwen3-1.7B-Base encoder	10.16	10.39
ELF-REG-B (ours) (default)	**32.32**	25.02
Text REPA during denoising: $0 \leq t \leq 0.4$	31.05	**25.17**
REG target: mean pooling	27.05	23.65
REG only	25.39	17.97
Equal prompt and response REPA weights	27.93	21.91
Alignment depth: 3	29.20	21.61
Alignment depth: 5	28.22	22.06
$\lambda_{\mathrm{REG}} = 0.03$	30.18	**25.17**

Table 26: GSM8K alignment-depth ablations for ELF-L at epoch 2.5. Reference configurations appear in the top rows. All settings follow the default ELF-L setup except for the stated difference in alignment depth; the default is block 8. Values are single-seed pass@1 (%) at 65 NFE. Other defaults and evaluation settings follow Appendix B.6. Bold marks the best validation and test values across all rows.

Condition	Validation pass@1 (%)	Test pass@1 (%)
ELF-L baseline	33.50	27.37
ELF-REG-L (ours) (default)	**44.04**	**34.04**
Alignment depth: 6	42.09	33.13
Alignment depth: 10	41.21	33.28
Alignment depth: 11	40.04	33.13

Table 27: GSM8K accuracy (%) for the early-stop ratio sweep, ELF-REG-B (ours) at epoch 12.

ρ / NFE	4	8	16	32	64
1	7.25	15.75	25.27	30.95	34.19
2	10.60	23.22	30.89	34.08	35.84
4	11.62	25.75	33.43	35.69	37.62
8	11.82	26.20	34.42	37.15	38.11
16	11.22	25.82	35.30	37.55	38.32

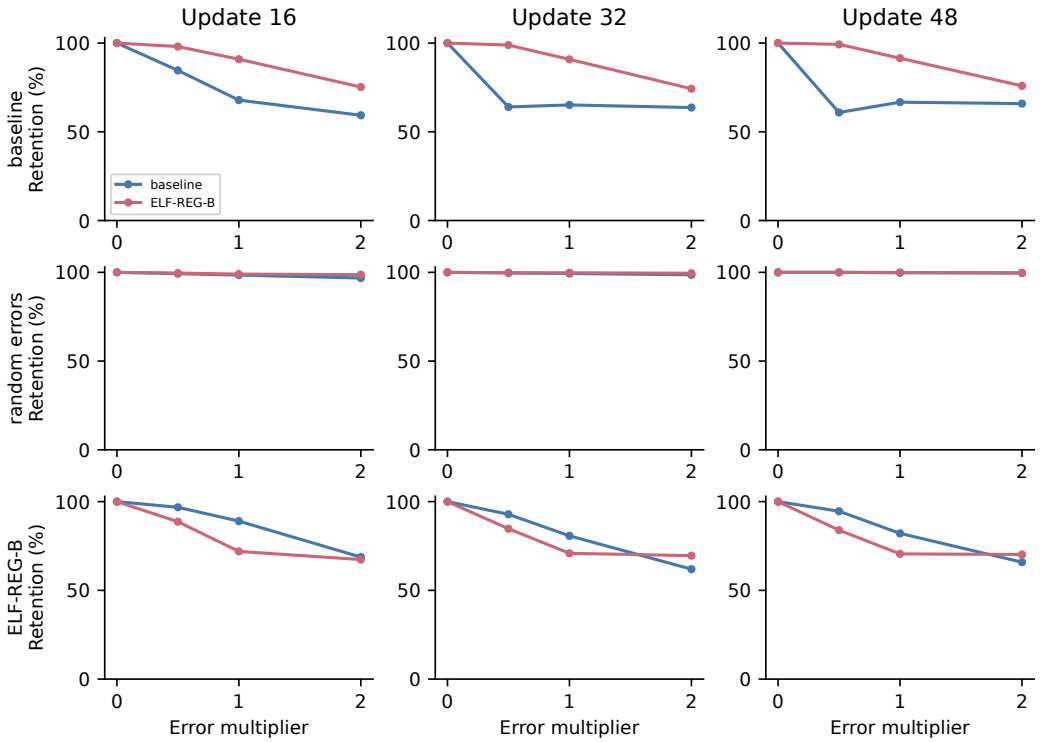

Figure 8: Coarse-sampling errors are more damaging than random errors of matched size. Retention measures correctness on the 1,219 question–seed pairs both models solve before injection (this is the number of questions solved across 4 seeds). Rows identify the error source and columns the injection update; each curve identifies the receiving model. The multiplier scales the injected error. ELF-REG-B retains more answers under its own coarse error, while the recipient ordering changes with the error source.

between the 64-NFE and 16-NFE states (specifically, the noisy text and text self-conditioning) into the 64-NFE reference, then finish generation after the error injection. REG is not affected and evolves normally. Figure 8 injects each model's own errors, errors from the other model, and random errors. Cross-model and random errors are normalized at each response position to the 64-NFE recipient's own combined text-state error. ELF-REG-B retains more shared solutions under its own error. The ordering depends on the error source, and random errors of matched size are less damaging.

The 64-NFE reference time grid has 63 denoiser updates. The coarse 16-NFE time grid retains reference indices $0, 4, 8, \ldots, 48, 53, 58, 63$, giving 15 updates and the same endpoint time. At each injection time, the error is the coarse state minus the reference state. We concatenate the noisy-text error and previous-clean-prediction error at each response position. Cross-model errors are rescaled to the recipient's own error norm at that position. Random controls draw Gaussian noise in this concatenated space and rescale it to the same norm. We split the vector back into the noisy-text error and previous-clean-prediction error. We scale both components of error by a multiplier from $\{0, 0.5, 1, 2\}$ and add them to the corresponding components of the reference state. We then continue generation using the remaining reference time grid. Prompt positions stay fixed, and the reference REG state and its self-conditioning are not modified by injection.

Retention is the percentage of question–seed pairs whose numerical answer remains correct among the question–seed pairs that both unperturbed 64-NFE reference runs solve. We pool these shared successes over 4 generation seeds. Thus the comparison holds the evaluated solutions fixed across error sources and 64-NFE recipients. Appendix B.6 gives checkpoint and sampling settings.

Full-span answer retention. We select the question–seed pairs that both models solve at full-span NFE 64, then measure correctness on those pairs at lower full-span budgets. Table 28 reports this conditional retention. These are separate full-span trajectories at each budget, whereas Figure 2 follows one continuing trajectory.

Table 28: ELF-REG-B retains more shared solutions at every reduced full-span budget. Retention (%) is measured on the 4,693 question–seed pairs both ELF-B baseline and ELF-REG-B solve at NFE 64. Each lower budget uses its own full-span trajectory; retention at NFE 64 is 100% by construction.

NFE	ELF-B baseline	ELF-REG-B (ours)
4	13.87	17.86
8	39.59	44.62
16	65.93	71.34
32	81.95	85.06
64	100.00	100.00

E COMPARISON WITH RELATED WORK

Table 29: Selected methods for language generation and representation supervision. Panel A distinguishes the generated state, generation structure, and training or sampling approach. Panel B distinguishes intermediate-feature alignment from jointly denoised representations. AR denotes autoregressive generation.

A. Language generation

Method	Generation state	Response generation	Training or sampling approach
LangFlow (Chen et al., 2026)	Token embeddings	Parallel sequence denoising	Flow matching with a learned noise schedule.
LDLM (Meshchaninov et al., 2026)	Learned contextual latents	Parallel denoising, then token decoding	Joint training of the latent encoder, denoiser, and decoder.
ELF (Hu et al., 2026)	Contextual encoder representations	Parallel denoising; final token decoding	Flow matching with a shared denoiser and decoder.
S-FLM (Deschenaux & Gulcehre, 2026)	Hyperspherical token embeddings	Parallel sequence denoising	Riemannian flow matching on learned embeddings.
FMLM (Lee et al., 2026)	Noisy one-hot token vectors	Parallel sequence generation	Distillation of a continuous flow into a flow map.
FMLM+ (Agarwal et al., 2026)	Noisy one-hot vectors and committed tokens	Any-order token commitment	Flow-map self-distillation; posterior refinement during sampling.
MLFM (Azangulov et al., 2026)	Token embeddings and committed tokens	Continuous updates with token commitment	Mask-conditioned flow training; confidence-based token commitment.
PlaidQ (Peng et al., 2026b)	Token embeddings	Parallel denoising; final token decoding	AR initialization; distribution-matching distillation for few-step generation, paired trajectories for one-step generation.
L2D (Cetin et al., 2025)	Next-token embedding	AR token sequence	A trainable diffusion path uses a frozen AR backbone.
LaDiR (Kang et al., 2026)	Latent thought blocks	Blockwise latent denoising; AR final answer	Latent reasoning followed by answer generation conditioned on the latent thoughts.
CADD (Zheng et al., 2025)	Discrete tokens and token embeddings	Joint continuous and discrete updates	Continuous token embeddings guide masked-token prediction.
CCDD (Zhou et al., 2026)	Discrete tokens and contextual latents	Joint continuous and discrete updates	Joint denoising of tokens and encoder representations.
ELF-REG (ours)	Encoder representations and global REG state	Parallel full-response denoising; final token decoding	REPA+REG training and prefix early-stop sampling.

Table 29: Selected methods for language generation and representation supervision (continued).

B. Representation supervision

Method	Supervised representation	Supervision source	Role during generation
REPA (Yu et al., 2025)	Intermediate image-denoiser features	Frozen visual encoder	Alignment is used during training; inference denoises image latents.
REG (Wu et al., 2025)	Global representation alongside image latents. REPA is also applied.	Pretrained visual encoder's class token	The global representation is jointly denoised with image latents.
Portuguese masked-dLM REPA (Junior et al., 2026)	Intermediate masked-denoiser features	Clean text representations from pretrained encoders	Alignment is used during training; inference uses masked token denoising.
REPR-ALIGN (Peng et al., 2026a)	Intermediate masked-denoiser features	Frozen AR model used for initialization	Alignment is used during training; inference uses masked token denoising.
TextLDM (Jiang et al., 2026b)	VAE encoder features	Frozen AR language model	Continuous latent diffusion uses the trained VAE decoder for token generation.
CCDD (Zhou et al., 2026)	Per-token contextual latents	Fixed pretrained text encoder	Contextual latents are jointly denoised with discrete tokens.
ELF-REG (ours)	Intermediate continuous-denoiser features and global REG state	Clean AR teacher states	The global REG state is jointly denoised with the continuous response. REPA supervises intermediate features during training.

E.1 Early-stop generation

Early-stop methods differ in the evidence used to stop denoising and the predictions they finalize. For continuous text diffusion, Vaina et al. (2024) compare fixed exits with adaptive criteria based on prediction confidence or changes between steps. Differmer (Gao et al., 2024) motivates early stopping by showing that intermediate clean predictions can yield better text than later predictions. In masked dLMs, confidence can determine either individual token commitments or termination of the remaining generation. Just on Time (Kohut et al., 2026) finalizes confident tokens individually; Prophet (Li et al., 2026) uses confidence over an answer region to commit all remaining tokens and observes that answer tokens can emerge before the surrounding response settles.

For continuous generation, stopping also requires choosing the state returned by the sampler. Truncated Jump Sampling (TJS) (Peng & Gao, 2026) truncates image-generation trajectories and returns a clean prediction, sharing the principle used by our prefix early-stop sampler. Its mean-squared error decomposition concerns prediction of the clean data sample under the corruption distribution. Our endpoint discrepancy compares the prediction with the endpoint of the same generated flow. The curvature-independent decomposition and our acceleration-based bound therefore concern different errors. FMLM (Lee et al., 2026) already interprets the clean predictor as one Euler step over the remaining interval and learns flow maps for finite-time transport. We use the existing clean predictor for that approximation. Token decoding introduces a further distinction: latent discrepancies need not change the decoded tokens. The decoder-margin argument of Du & Ma (2026, Theorem 1), applied in Proposition 2, gives a sufficient condition for preserving the complete endpoint response, including an incorrect response.

We use REPA+REG to improve intermediate clean predictions in fully continuous diffusion language models and demonstrate strong low-NFE mathematical reasoning and code generation through prefix early-stop, without dedicated few-step training. On GSM8K, ELF-REG-B reduces

endpoint error and brings earlier token stability. Our analysis characterizes the discrepancy between an intermediate clean prediction and the endpoint of the same generated flow, bounds this discrepancy through remaining acceleration, and applies the decoder-margin argument to complete-response agreement (Section 4.1). We separately derive the endpoint discrepancy for the implemented self-conditioned sampler (Appendix C.3).

F LIMITATIONS

We train separate models for GSM8K, MATH, code, and MMLU. The code model is shared across the MBPP/HumanEval code benchmarks, and the MATH models are initialized from the corresponding GSM8K weights. Joint training of a single generalist model across these tasks remains future work. Our code training and evaluation also focus on Python, so the study does not cover other programming languages. Extending the training data and benchmarks to additional programming languages is a promising further direction. ELF is a continuous dLM trained on a fixed-length response canvas; adapting it to different canvas lengths without performance reductions requires finetuning and we leave this as future work. Due to compute constraints, we do not repeat training with multiple random seeds.

G QUALITATIVE EXAMPLES

We show one correct and one incorrect response from ELF-REG-L on each benchmark, using the checkpoints and settings of Tables 2 and 1: 64 NFE for GSM8K and 128 NFE for MATH-500, HumanEval, and MBPP-378, with $\rho = 8$ early-stop. Prompts and generated responses are reproduced verbatim, with reference answers for incorrect responses. Appendix B.3 gives the evaluation protocols.

GSM8K **Correct**

Prompt

```
Question: If Ann is 9 years old and her brother is twice her age, how
old will her brother be in 3 years?
Answer:
```

Generated response

```
 If Ann is 9 years old, then her brother is twice her age, which is 9 x
2 = 18 years old.
In 3 years, her brother will be 18 + 3 = 21 years old.
#### 21
```

GSM8K **Incorrect**

Prompt

```
Question: Lori wants to buy a $320.00 pair of shoes and a matching belt
that is $32.00.  Her part-time job pays her $8.00 an hour.  How many
hours will she have to work before she can make her purchase?
Answer:
```

Generated response

```
 Lori's total expenses for the shoes and the belt are $320 + 32 = 352$.
Since she can make $8 per hour, she needs to work $352 / 8 = 43$ hours.
Thus Lori will have to work \boxed{43} hours to make her purchase.
#### 43
```

Reference answer

```
44
```

MATH-500 **Correct**

Prompt

```
Question: If $3x + 2(1 + x) = 17$, what is the value of $6x + 5$?
Step-by-Step Answer:
```

Generated response

```
 First, solve the equation for $x$:
\[ 3x + 2 + 2x = 17 \Rightarrow 5x + 2 = 17 \Rightarrow 5x = 15 \
Rightarrow x = 3 \]

Now substitute the value of into the equation:
\[ 6x + 5 = 6(3) + 5 = 18 + 5 = \boxed{23} \]
Final answer: $\boxed{23}$
```

MATH-500 **Incorrect**

Prompt

```
Question: If $a\star b = 9a+2b-ab+5$, what is the value of $5\star1$?
Step-by-Step Answer:
```

Generated response

```
 To find the value of $5\star1$, we need to substitute $a=5$ and $b=1$
into the given equation.

\[ a\star b = 9a+2b-ab+5 \]

\[ 5\star1 = 9(5)+2(1)-(5)(1)+5 \Rightarrow 5\star1 = 45+2-5+5 \
Rightarrow 5\star1 = 45 \]

So the answer is $\boxed{45}$.
Final answer: $\boxed{45}$
```

Reference answer

```
47
```

HumanEval **Correct**

Prompt

```
Write a complete Python solution. Return only code.

def get_positive(l: list):
    """Return only positive numbers in the list.
    >>> get_positive([-1, 2, -4, 5, 6])
    [2, 5, 6]
    >>> get_positive([5, 3, -5, 2, -3, 3, 9, 0, 123, 1, -10])
    [5, 3, 2, 3, 9, 123, 1]
    """

Python solution:
```

Generated response

```
def get_positive(l: list) -> list:
    """Return only positive numbers in the list l."""
    return [num for num in l if num > 0]
```

HumanEval **Incorrect**

Prompt

```
Write a complete Python solution. Return only code.

def triangle_area(a, h):
    """Given length of a side and high return area for a triangle.
    >>> triangle_area(5, 3)
    7.5
    """

Python solution:
```

Generated response

```
def triangle_area(a, h):
    """
    Calculate the area of a triangle given side and high.

    Parameters:
    a (float): Length of the side.
    h (float): High of the side.

    Returns:
    float: Area of the triangle.
    """
    return a * h ** 2
```

Reference answer

```
def triangle_area(a, h):
    """Given length of a side and high return area for a triangle.
    >>> triangle_area(5, 3)
    7.5
    """


    return a * h / 2
```

MBPP-378 **Correct**

Prompt

```
Write a complete Python solution. Return only code.

"""
Write a function to find the volume of a cube given its side length.
assert volume_cube(3)==27
"""

Python solution:
```

Generated response

```
def volume_cube(side_length):
    """
    Calculate the volume of a cube given its side length.

    Parameters:
    side_length (float): The side length of the cube.

    Returns:
    float: The volume of the cube.
    """
    return side_length ** 3
```

MBPP-378 **Incorrect**

Prompt

```
Write a complete Python solution. Return only code.

"""
Write a function to find the nth octagonal number.
assert is_octagonal(5) == 65
"""

Python solution:
```

Generated response

```
def is_octagonal(n):
    """
    Returns the nth octagonal number.

    Parameters:
    n (int): The position in the octagonal sequence.

    Returns:
    int: The nth octagonal number.
    """
    return n * (n + 1) // 2
```

Reference answer

```
def is_octagonal(n):
    return 3 * n * n - 2 * n
```