ANCHORS ENHANCE CONTROLLABILITY AND REASONING OF DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models (DLMs) offer new opportunities for controllable text generation. However, their controllability, defined as the ability to satisfy task constraints while preserving their original capabilities, remains underexplored. DLMs' bidirectional attention allows control tokens to guide generation from arbitrary positions in the output sequence, but effective guidance depends on the compatibility between their content and placement. To this end, we propose AnchorDLM, a framework in which an anchor generator produces an anchor consisting of a short output fragment and its position for each query. The anchor is placed as control tokens at the specified position to initialize the output sequence of a frozen DLM and guide subsequent denoising. We train the anchor generator with reinforcement learning (RL) to jointly optimize anchor content and placement, using rewards computed from the DLM's generated outputs based on answer correctness and constraint satisfaction. Moreover, through the lens of discrete flow matching (DFM), we interpret anchor placement as helping to reshape the initial velocity field that guides generation. Experiments on code generation and mathematical reasoning across five DLM backbones show that AnchorDLM improves controllability and further yields gains in reasoning performance across diverse benchmarks.

1 INTRODUCTION

Many generation tasks in real world demand more than fluency or correctness: they also require the ability to guide generation to satisfy the constraints of a given task, such as specific text structure (Zhang et al., 2025; Jiang et al., 2024) or style (Yang et al., 2025b; Caliskan-Islam et al., 2015). We refer to the ability to satisfy such constraints while preserving the model's original capabilities as controllability (Zhang et al., 2025; Liang et al., 2024). Compared with autoregressive (AR) language models, diffusion language models (DLMs) (Austin et al., 2021a; Li et al., 2022; Gong et al., 2022; Sahoo et al., 2024; Li et al., 2025; Huang et al., 2026) offer greater flexibility for controllable text generation. DLMs generate text through iterative denoising with bidirectional attention, allowing each position to incorporate available context from both directions. This bidirectional generation process allows control signals to be placed at arbitrary positions within the output sequence and to guide the generation of surrounding text. However, such potential remains largely underexplored.

One way to realize this potential is to initialize the output sequence by placing control tokens(Xiong et al., 2025; Lee et al., 2025), which involves two intertwined decisions: what to provide and where to control. Recent methods such as S^3 (Xiong et al., 2025) and Ti (Lee et al., 2025) address these decisions by initializing generation with predefined structural templates containing elements such as JSON keys and delimiters. Although effective for rigid format control, these methods rely on generic structural elements to determine what to provide, offering limited semantic guidance about the desired content or reasoning process. For where to control, these methods rely on manually specified templates to determine token positions, limiting their ability to adapt placement to the control content and the needs of individual queries. Addressing these limitations calls for adapting both control content and placement to each query while ensuring their compatibility, as control tokens become part of the generated output.

To improve controllability by learning what guidance to provide and where to place it, we propose AnchorDLM, a framework that trains an anchor generator to produce an anchor consisting of a

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

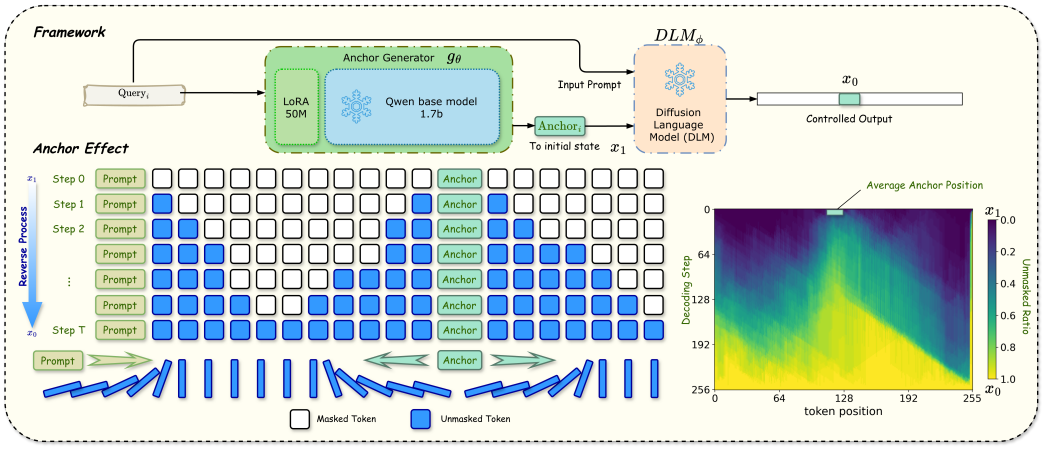

Figure 1: Overview of AnchorDLM framework. (Right) Decoding order averaged over 50 samples.

short output fragment as control tokens and its placement position. We learn the anchor content and its placement together so that each anchor provides semantic guidance suited to its position and can then improve constraint satisfaction without compromising task performance. For each input query, we place the generated anchor at the specified position in the DLM's initial state to guide subsequent iterative denoising. We optimize the anchor generator with reinforcement learning (RL), using rewards computed from the DLM's generated outputs based on answer correctness and constraint satisfaction. We further offer a theoretical interpretation of anchor placement through the lens of discrete flow matching (DFM), in which anchors can be viewed as reshaping the initial velocity field that guides generation. Our contributions are as follows:

1. We are the first to address controllability in DLMs through learnable anchors that provide semantic guidance. We further interpret the role of anchors through the lens of discrete flow matching.
2. We propose AnchorDLM, a model-agnostic framework comprising an anchor generator and a frozen DLM, in which the generator is trained in two stages to produce anchors that guide denoising from the DLM's initial state.
3. Extensive experiments across five DLM backbones show that AnchorDLM improves controllability by 11.30 G-mean points on average over the strongest same-backbone baselines and, when trained for reasoning, matches or exceeds them in all 20 reasoning settings.

2 Preliminary

2.1 Diffusion Language Models

In contrast to autoregressive models, diffusion language models (Austin et al., 2021a; Nie et al., 2025; Ye et al., 2025) capture the text distribution through forward and reverse processes, inspired by their continuous counterparts in image generation (Austin et al., 2021a). We adopt the time convention $t \in [0, 1]$ and let x_t denote the partially noised sequence at time t, with x_0 being the clean sequence and x_1 a fully masked sequence. The forward process independently replaces each token in x_0 with a mask symbol M with probability t, gradually corrupting the sequence until it is fully masked. The reverse process learns a mask predictor $p_\phi(\cdot \mid x_t)$ parameterized by ϕ that recovers data distribution by iteratively predicting masked tokens as t moves from 1 to 0. Training minimizes a weighted cross-entropy loss evaluated only at masked positions (Nie et al., 2025):

$$\mathcal{L}(\phi) \triangleq -\mathbb{E}_{t,x_0,x_t}\left[\frac{1}{t}\sum_{i=1}^{L}\mathbf{1}[x_t^i = \mathrm{M}]\,\log p_\phi(x_0^i \mid x_t)\right], \tag{1}$$

where x_0 is drawn from the training data, L is the sequence length, the factor $1/t$ compensates for the varying masking ratio across time steps, the indicator $\mathbf{1}[\cdot]$ restricts the loss to masked tokens

and x_t^i denotes the i-th token of x_t. We refer to this family of models as diffusion language models (DLMs) throughout this paper.

2.2 Discrete Flow Matching

Discrete flow matching (DFM) (Gat et al., 2024; Campbell et al., 2024) provides a more general framework for generative modeling over discrete state spaces by directly parameterizing transition rates between sequences. Let $\kappa_t \in [0, 1]$ be a monotonically decreasing schedule with $\kappa_0 = 1$ and $\kappa_1 = 0$, representing the weight retained on the clean data at time t, and let $p_{0|t}(x^i \mid x_t)$ denote the posterior over the clean token at position i given the partially noised sequence x_t. Under DFM, the transition rate at position i is

$$\nu_t^i(x^i, x_t) = \frac{-\dot{\kappa}_t}{1 - \kappa_t}\Big[p_{0|t}(x^i \mid x_t) - \delta_{x_t^i}(x^i)\Big], \tag{2}$$

which is non-negative on off-diagonal entries since $\dot{\kappa}_t \leq 0$. A small backward Euler step samples the previous state from

$$X_{t-h}^i \sim \text{Categorical}\Big(\delta_{X_t^i}(\cdot) + h\,\nu_t^i(\cdot, X_t)\Big). \tag{3}$$

where $\dot{\kappa}_t = \mathrm{d}\kappa_t/\mathrm{d}t$ denotes the time derivative, $h > 0$ denotes the small backward Euler step size, $\delta_{x_t^i}(\cdot)$ is the Kronecker delta indicator, equal to 1 when its argument equals x_t^i and 0 otherwise, $X_t \in \mathcal{V}^L$ denotes the random sequence at time t and X_t^i its i-th token, with $\mathcal{V}$ the vocabulary augmented by the mask symbol M. This formulation highlights that DFM evolves probability mass through context-dependent stochastic transition rates over a discrete state space. We will show in Section 3.4 that masked DLMs are a special case of this framework.

3 Methodology

Our framework modifies the DLM's initial state by placing learned anchors, guiding the subsequent denoising toward controlled output. In this section, we first present the overall framework (Section 3.1). We then describe a supervised fine-tuning stage over an augmented training set (Section 3.2), that teaches the anchor generator to produce format valid anchors adaptive to each input query. Building on this, in Section 3.3 we construct an offline reinforcement learning pipeline that uses rewards computed from the DLM's generated outputs based on answer correctness and constraint satisfaction, aligning the generator with the downstream DLM. Finally, in Section 3.4, we provide insight into the effectiveness of anchors through the lens of DFM.

3.1 Framework

Given an input query q and a frozen DLM that generates an output sequence $x \in \mathcal{V}^L$, our goal is to enhance controllability of the DLM without any update to its parameters. To this end, our framework first produces the guidance that is placed into the DLM's initial state ahead of its generation, controlling the subsequent denoising toward the output aligning to the guidance. We refer to these signals as anchors, each a pair $a = (p, c)$, represented as {`"Position": "Content"`}, where the `Position` $p \in \{0, 1, \ldots, L\}$ is a position index in the output sequence and the `Content` c is a multi-token sequence over $\mathcal{V}$ placed starting at that position. As illustrated in Figure 1, the placed anchor affects decoding order and propagates information outward during denoising, influencing both its left and right context.

Our framework consists of two components bridged by an anchor: an anchor generator g_θ with parameters θ, implemented as a lightweight Low-Rank Adaptation(LoRA) adapter on top of a pre-trained language model(Qwen3-1.7B-Base (Yang et al., 2025a)), and a frozen DLM with fixed parameters ϕ. Concretely, given the query q, the anchor generator produces an anchor $a = (p, c) = g_\theta(q)$. We place c at position p of the DLM's initial state, keeping all remaining positions as the mask token M, yielding an anchored initial state $x_1^{(a)} \in \mathcal{V}^L$. The frozen DLM performs standard denoising starting from $x_1^{(a)}$ to produce x_0. The single-step denoising transition can be written as

$$p_\phi(x_{t-h} \mid x_t, q), \quad x_1 = x_1^{(a)}, \tag{4}$$

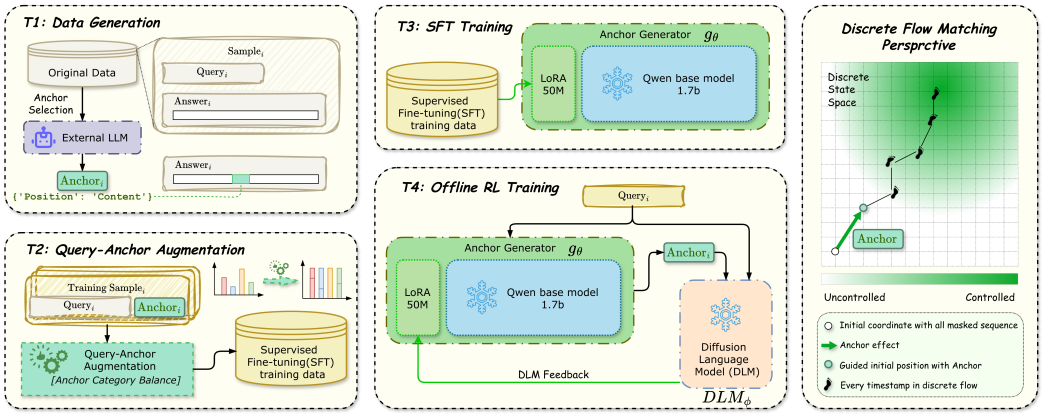

Figure 2: Training pipeline and DFM perspective of AnchorDLM. T1–T2 construct training data. T3 performs supervised fine-tuning(SFT). T4 further refines the adapter with reinforcement learning(RL). (Right) From a discrete flow matching perspective, an anchor reshapes the initial coordinate of the discrete flow, pulling the generation trajectory from the uncontrolled region toward the controlled region of the discrete state space.

where $p_\phi(x_{t-h} \mid x_t, q)$ denotes the single-step denoising transition of the frozen DLM, and the anchor influences the entire generation by determining the initial state $x_1^{(a)}$.

3.2 Supervised Fine-tuning of the Anchor Generator

This stage aims to equip the anchor generator with the ability to produce query adaptive and format valid anchors. To this end, we first use Claude Opus 4.6 to rewrite each ground truth answer into the target style, then extract a distinctive fragment and its position as the anchor target for the corresponding query (Figure 2, T1). To improve coverage of different anchor functions, we group anchors by functional role and rebalance the categories by paraphrasing queries associated with underrepresented categories (Query-Anchor Augmentation; Figure 2, T2).

To teach the generator the required anchor format {`"Position": "Content"`}, we train it on the rebalanced dataset (See details in Appendix A.1 and A.2). However, anchors learned through SFT may still fail to guide the downstream DLM effectively, motivating further optimization.

3.3 Reinforcement Learning of the Anchor Generator

We further optimize the generator through reinforcement learning (RL), using rewards based on answer correctness and constraint satisfaction in the frozen DLM's outputs. Given the cost of DLM inference, we use offline RL with a fixed preference dataset. For each query, we sample k anchors from the generator and evaluate the corresponding DLM outputs using these rewards. Then we construct 80% of preference pairs from queries with both successful and failed outputs, using the corresponding anchors as chosen and rejected responses, respectively. For the remaining 20%, we use queries where all k outputs fail, pairing the supervised anchor as chosen with a sampled anchor as rejected. This mixture combines preferences over sampled anchors with supervision for queries that yield no successful outputs (Li & Khashabi, 2025; Jiao et al., 2024; Wang et al., 2025). To account for the mismatch between the current generator and the data collection policy, we adopt Weighted Preference Optimization (WPO) (Zhou et al., 2024). The WPO objective is:

$$\mathcal{L}_{\text{WPO}} = -\mathbb{E}_{(q,a_w,a_l)\sim\mathcal{D}}\Big[w_\theta(q,a_w)\, w_\theta(q,a_l)\, \log\sigma\Big(\beta\log\frac{\pi_\theta(a_w \mid q)}{\pi_{\text{ref}}(a_w \mid q)} - \beta\log\frac{\pi_\theta(a_l \mid q)}{\pi_{\text{ref}}(a_l \mid q)}\Big)\Big], \tag{5}$$

$$w_\theta(q,a) = \exp\left(\frac{1}{|a|}\sum_{n=1}^{|a|}\left[\log \pi_\theta(a_n \mid q, a_{<n}) - \log \sum_{v\in\mathcal{V}} \pi_\theta(v \mid q, a_{<n})^2\right]\right), \tag{6}$$

where a_w and a_l are the preferred and dispreferred anchors, π_θ is the current anchor generator and π_{ref} the SFT reference, β controls the deviation from the reference, $|a|$ is the anchor's token length, and a_n denotes the n-th token of a (See details in Appendix A.1). The weights $w_\theta(q,a)$ follow WPO's sampled alignment formulation and are detached from gradient computation (Zhou et al., 2024). Experiments show that this RL stage improves the control success rate of generated anchors on the downstream DLM.

3.4 Interpretation from a Discrete Flow Matching Perspective

We analyze AnchorDLM through the standard absorbing DFM with independent masking, exact denoising posteriors, and sampling in continuous time without remasking (Gat et al., 2024). For a fixed query q, let $p_0(\cdot \mid q)$ denote the unanchored terminal distribution. With $\kappa_t = 1 - t$, the probability velocity for reverse updates from t to $t - h$ is

$$\nu_t^i(v, x_t, q) = \frac{1}{t}\left[p^i_{0|t}(v \mid x_t, q) - \delta_{x_t^i}(v)\right], \quad 0 < t < 1, \tag{7}$$

where $p^i_{0|t}$ is the denoising posterior and δ denotes a point mass. The masked prediction objective in Section 2.1 learns the posterior that parameterizes this velocity field.

For an anchor $a = (p, c)$, let S_a denote the positions occupied by c. At revealed positions, the posterior is a point mass at the observed token, so $\nu_t^i = 0$. Anchor tokens and subsequently revealed tokens therefore persist, providing context that can influence velocity evaluations throughout denoising (Figure 2). For any correct controlled target $x^\star$ with $x^\star_{S_a} = c$, initialization also reduces the Hamming distance from $d_H(\mathrm{M}^L, x^\star) = L$ to $d_H(x_1^{(a)}, x^\star) = L - |c|$. Thus, a compatible anchor places the initial state closer to a target while providing persistent guidance.

Define $E_a = \{x : x_{S_a} = c\}$ and assume $p_0(E_a \mid q) > 0$. Fixing the anchor coordinates yields a conditional DFM on the remaining positions, whose terminal law is $p_0^{(a)}(\cdot \mid q) = p_0(\cdot \mid q, E_a)$. Consequently,

$$p_0^{(a)}(x \mid q) = \frac{p_0(x \mid q)\,\mathbf{1}\{x \in E_a\}}{p_0(E_a \mid q)}, \qquad D_{\mathrm{KL}}\Big(p_0^{(a)}(\cdot \mid q) \,\|\, p_0(\cdot \mid q)\Big) = -\log p_0(E_a \mid q). \tag{8}$$

The KL identity follows from the constant probability ratio on E_a and quantifies the information imposed by the anchor.

To connect this redistribution to controllability, let C_q denote the event that an output is correct and satisfies the task constraints, with $p_0(C_q \mid q) > 0$. Bayes' rule gives

$$p_0^{(a)}(C_q \mid q) = p_0(C_q \mid q)\frac{p_0(E_a \mid C_q, q)}{p_0(E_a \mid q)}. \tag{9}$$

Thus, an anchor improves control success when it is more likely among successful outputs than under the overall output distribution. This criterion motivates learning anchor content and placement from the preferences constructed using downstream outcomes in Section 3.3.

4 Experiment

This section begins with the experimental setup in Section 4.1. Section 4.2 presents the improvements of AnchorDLM on DLMs' controllability and reasoning. We then analyze the anchor generator's size (Section 4.3), properties of the generated anchors including content and placement compatibility, diversity, and content ratio (Section 4.4), and the generator's generalization across DLMs (Section 4.5). We further conduct an ablation study on each component (Section 4.6). Finally, Section 4.7 provides a case study illustrating how anchors control DLM generation.

4.1 Experiment Setup

We evaluate AnchorDLM on mathematical reasoning benchmarks, GSM8K (Cobbe et al., 2021) and MATH500 (Lightman et al., 2024), and code generation benchmarks, HumanEval (Chen et al., 2021)

and MBPP (Austin et al., 2021b). We maintain a strict separation between training and evaluation data throughout SFT and RL. For GSM8K and MBPP, we use their official training splits. For HumanEval, we construct training data by reformatting examples from the MBPP training split into the HumanEval task format. For MATH500, we use samples from the MATH dataset after excluding all MATH500 evaluation problems. All supervised anchors and RL preference pairs are constructed exclusively from these training sources.

We compare AnchorDLM against four categories of baselines:(i) the frozen DLM (LLaDA-8B-Base and LLaDA-8B-Ins (Nie et al., 2025), Dream-7B-Base and Dream-7B-Ins (Ye et al., 2025), and LLaDA 1.5-8B(Zhu et al., 2026)); (ii) a prompt baseline that prepends a hand-crafted instruction to the input query; (iii) one recent controllable DLM method, S^3 (Xiong et al., 2025) , which places predefined schema templates into the DLM's initial state, and (iv)another controllable DLM method, Ti (Lee et al., 2025), which places fixed reasoning templates with additionally adjusting template positions.

To evaluate controllability, we select four representative control objectives (Yang et al., 2025b; Jiang et al., 2024; Zhang et al., 2025; Chen & Abedjan, 2023; Buse & Weimer, 2009; Caliskan-Islam et al., 2015; Hart et al., 2023) spanning content, structure, and reasoning logic. For each objective, we define a control score that measures the extent to which the generated output satisfies the specified requirement.

Formal math(Fmt.) (Yang et al., 2025b). The model is required to express solutions through mathematical formulas with minimal natural language. The control score is the proportion of formal mathematical expressions in the generated content.

Backward reasoning(Bwd.) (Jiang et al., 2024). The model is required to reason backward from the final goal to the premises needed to establish it. The control score is the proportion of generated solutions that pass both an LLM assessment of backward reasoning and a check for characteristic reasoning connectives.

Commented code(Cmt.) (Zhang et al., 2025; Buse & Weimer, 2009; Chen & Abedjan, 2023). The model is required to produce code interleaved with comments to improve readability. The control score is the interleaving ratio between comments and code in the generated content.

Structured code(Str.) (Caliskan-Islam et al., 2015; Hart et al., 2023). The model is required to organize code into an initialization block followed by the main logic. The control score is the proportion of generated solutions satisfying this abstract syntax tree(AST) based structural constraint.

To jointly consider whether control succeeds and whether the original task performance is preserved, following Mukherjee et al. (2025)and Yang & Jin (2023), we take the geometric mean(G-mean) of the control score and the corresponding task performance as the final composite controllability score, which is high only when control succeeds and task quality is preserved. Further details of the metrics are provided in the Appendix A.3.

4.2 Main Results

Table 1 reports results with anchor generators trained separately for controllability and reasoning. AnchorDLM achieves the highest G-mean in all 30 controllability settings across five DLM backbones, exceeding the strongest baseline for the same backbone in each setting by 11.30 points on average. The largest average gains occur on GSM8K: 22.41 points for formal mathematical expression and 21.01 points for backward reasoning. When trained for reasoning, AnchorDLM matches or exceeds the strongest baseline for the same backbone in all 20 settings, with an average gain of 0.91 percentage points. Compared with the prompted AR baseline Llama 3-8B-Base, AnchorDLM performs better in all 30 controllability settings and 18 of 20 reasoning settings.

4.3 Anchor Generator Size and Capacity

We compare the downstream performance of anchors extracted from models of different sizes on HumanEval and MBPP. We use the same heuristic to extract anchors from each model's complete outputs. Table 2 shows clear gains from 0.6B to 1.7B, while further improvements vary across benchmarks and model sizes. We adopt Qwen3-1.7B as the default g_θ to balance model size and downstream performance.

Table 1: Main results with the anchor generator trained for controllability (left) and reasoning (right). Controllability is measured by G-mean, and reasoning by accuracy or pass@1. Bold indicates the best result for each backbone.

Backbone	Method	Controllability						Reasoning			
		GSM8K		HumanEval		MBPP		GSM8K	MATH	HE	MBPP
		Fmt.	Bwd.	Cmt.	Str.	Cmt.	Str.	Acc	Acc	Pass@1	Pass@1
Llama 3-8B-Base	Vanilla	13.92	10.02	23.49	44.13	25.82	45.65	43.59	24.80	33.54	41.40
	Prompt	24.66	21.67	36.73	46.97	39.61	47.41	56.03	28.00	36.59	43.80
LLaDA-8B-Base	Vanilla	5.01	5.01	14.59	42.32	29.77	43.16	66.03	30.20	34.15	39.80
	Prompt	7.99	7.40	23.29	45.04	31.16	47.27	66.57	31.80	35.37	40.20
	S^3	36.49	34.55	17.70	35.20	30.30	42.00	52.16	32.20	21.34	16.80
	Ti	38.58	35.19	16.86	45.43	30.11	46.81	49.89	31.60	28.05	17.80
	Ours†	**69.33**	**71.08**	**43.89**	**50.41**	**40.05**	**49.27**	**67.17**	**33.40**	**37.20**	**43.60**
LLaDA-8B-Ins	Vanilla	17.08	30.94	29.83	35.85	28.25	32.06	56.10	33.80	39.02	40.80
	Prompt	16.83	32.02	45.64	47.34	48.35	45.75	57.77	34.60	40.24	40.80
	S^3	45.33	49.15	38.77	44.46	33.11	38.15	52.16	33.20	36.59	25.00
	Ti	45.36	47.85	37.64	47.51	29.52	46.65	58.07	34.40	32.93	18.80
	Ours†	**61.85**	**65.72**	**46.07**	**52.40**	**49.96**	**50.90**	**58.53**	**34.80**	**41.46**	**41.40**
LLaDA 1.5-8B	Vanilla	18.65	31.27	33.81	41.02	30.94	41.14	59.14	33.40	39.63	44.00
	Prompt	23.12	34.28	42.56	46.69	38.01	47.76	61.41	34.20	**40.24**	45.40
	S^3	51.47	53.54	39.02	42.51	37.41	41.63	60.27	33.60	37.80	43.80
	Ti	46.69	50.17	39.37	47.78	34.62	44.46	60.80	34.20	35.98	44.80
	Ours†	**70.08**	**67.42**	**47.11**	**54.53**	**49.65**	**52.48**	**61.56**	**34.80**	**40.24**	**47.40**
Dream-7B-Base	Vanilla	17.52	15.24	18.36	45.27	31.62	42.77	73.69	37.00	52.44	50.80
	Prompt	25.35	23.09	31.01	47.92	38.75	48.27	74.83	40.20	54.88	52.60
	S^3	38.74	37.29	27.64	42.32	37.88	43.87	72.93	39.60	48.78	41.80
	Ti	43.63	41.77	26.15	49.06	35.41	45.59	71.04	40.60	51.83	45.00
	Ours†	**72.61**	**71.70**	**46.79**	**55.12**	**44.05**	**53.24**	**75.36**	**41.80**	**56.10**	**53.20**
Dream-7B-Ins	Vanilla	20.51	33.51	37.13	44.48	29.41	35.81	75.59	38.00	48.78	52.00
	Prompt	29.22	42.16	50.52	55.17	44.15	51.18	76.50	42.40	52.44	54.00
	S^3	49.05	51.13	46.63	49.27	39.59	47.72	74.22	41.80	53.66	48.80
	Ti	48.69	49.43	43.39	53.74	38.61	50.82	75.89	41.80	51.83	50.60
	Ours†	**66.26**	**59.93**	**51.78**	**58.44**	**48.92**	**56.39**	**76.80**	**43.40**	**54.27**	**54.40**

Table 2: Pass@1 (%) with anchors extracted from different models using the same heuristic.

Model	Qwen3					Llama 3.3
Size	0.6B	1.7B	4B	8B	32B	70B
HumanEval	24.39	33.54	38.41	38.41	35.98	40.24
MBPP	28.60	37.80	37.60	38.00	42.20	43.20

4.4 Anchor Analysis

To understand how anchors guide DLM generation, we examine three aspects: (i) the compatibility between anchor content and placement; (ii) the anchor position distribution; and (iii) the proportion of anchor content in the final generation.

Content and placement compatibility. We analyze anchor placement, the proportion of anchor content in generated outputs, and generalization across inputs. To examine the importance of matching anchor content and placement, we use LLaDA-8B-Base as the frozen backbone, hold the anchor content fixed, and shift its position by offsets ranging from -40 to $+40$ tokens. Across these offsets, G-mean varies by 5.3 points on HumanEval and 2.4 points on MBPP (Figure 3). Moving anchors earlier reduces code generation pass@1 by up to 30.5 percentage points, whereas later shifts cause smaller declines and GSM8K accuracy varies by 6.9 percentage points across all offsets. These results show that relocating the same anchor content can impair task performance, highlighting the importance of matching content and placement, particularly for code generation.

Anchor position distribution. Figure 4 visualizes the distribution and density of anchor positions predicted by our generator across datasets. Since our method predicts anchor positions adaptively

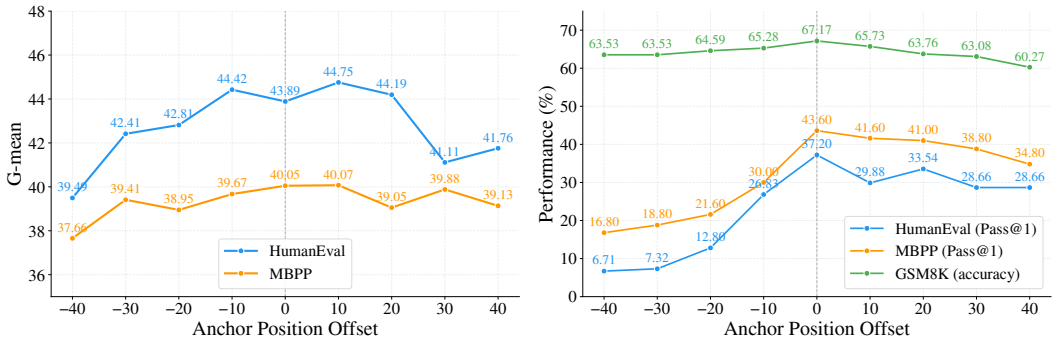

Figure 3: Effect of anchor placement with content held fixed: G-mean (left) and performance (right).

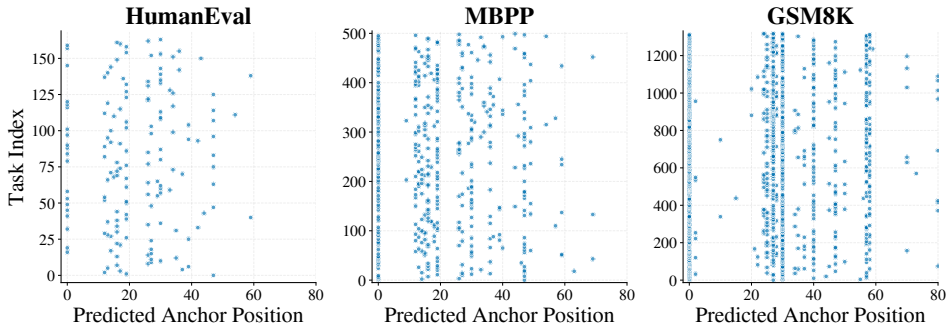

Figure 4: Anchor position distribution and density across datasets.

per query, the distribution is broad, with predicted positions spanning over 80 tokens across samples. This observation confirms that our method offers substantial diversity and flexibility of anchors.

Anchor content ratio. To verify that AnchorDLM's gains do not come from placing large amounts of pre-computed content into the output, we measure the average fraction of output tokens contributed by the anchor on each benchmark. Across benchmarks, anchors account for only 4.33–10.88% of output tokens on average (Appendix A.4).

4.5 Generalization across DLMs

To examine whether the anchor generator generalizes across DLMs, we transfer g_θ, trained using outputs from LLaDA-8B-Base, to LLaDA-8B-Instruct without retraining. On the target DLM, it improves all evaluated metrics over the vanilla baseline, with controllability gains of 15.19 to 42.43 points (Appendix A.5).

4.6 Ablation Study

We ablate three main design choices of AnchorDLM: removing the query-anchor augmentation, removing the RL stage (keeping only SFT), and replacing WPO with vanilla DPO. As shown in Table 3, all three components contribute positively to the final performance. Removing the query-anchor augmentation is by far the most damaging, nearly eliminating the gain of AnchorDLM and even hurting task performance, because the raw anchor distribution is heavily imbalanced across categories, leading generator overfitting to specific cases. Without RL, the generator cannot refine anchors using DLM feedback; replacing WPO with DPO introduces a distributional gap that makes the feedback signal stale.

4.7 Case Study

Figure 5 illustrates anchor effects on MBPP task (idx=389), which requires counting distinct unordered pairs. Without an anchor, the DLM incorrectly treats reversed pairs as distinct and fails the tests. With a comment anchor (Case 1), the DLM generates a correct solution with interleaved com-

Table 3: Ablation study of AnchorDLM on GSM8K and MBPP.

Variant	GSM8K Formal	GSM8K Acc	MBPP Comt.	MBPP Pass@1	Avg. Δ
LLaDA-8B-Base (no anchor)	5.01	66.03	29.77	39.80	–
Full AnchorDLM	**69.33** ↑64.32	**67.17** ↑1.14	**40.05** ↑10.28	**43.60** ↑3.80	**+19.89**
w/o WPO (replace with DPO)	60.42 ↑55.41	66.79 ↑0.76	38.06 ↑8.29	40.20 ↑0.40	+16.22
w/o RL (SFT only)	47.16 ↑42.15	65.96 ↓0.07	36.88 ↑7.11	40.00 ↑0.20	+12.35
w/o query-anchor augmentation	22.97 ↑17.96	58.98 ↓7.05	31.73 ↑1.96	28.00 ↓11.80	+0.27

No Anchor FAIL

```
test_set = set()
for tuple in test_list:
    test_set.add(tuple)
return len(test_set)
```

Case 1: Commented PASS

```
# Create an empty list to
# store unique tuples
unique_tuples = []

# Iterate over each tuple
for tuple in test_list:
    # Sort the tuple
    tuple = sorted(tuple)

    # Check if already present
    if tuple not in unique_tuples:
        unique_tuples.append(tuple)

# Return the number of unique
# tuples present in the list
return len(unique_tuples)
```

Case 2: Reasoning PASS

```
result = {}
for pair in test_list:
    first, second = pair
    if first > second:
        first, second = second, first
    if (first, second) in result:
        result[(first, second)] += 1
    else:
        result[(first, second)] = 1
return len(result)
```

Figure 5: Case study on MBPP sample (idx=389), with anchor highlighted.

ments and code, satisfying both correctness and style constraints. With a reasoning anchor (Case 2), the DLM correctly orders each pair before dictionary lookup to count distinct pairs.

5 Related Works

Diffusion language models. Diffusion language models (DLMs) (Austin et al., 2021a; Li et al., 2022; Gong et al., 2022; Sahoo et al., 2024; Shi et al., 2024; Lou et al., 2023; Khanna et al., 2025) have recently emerged as a parallel, bidirectional alternative to autoregressive generation. Masked DLMs such as LLaDA (Nie et al., 2025; Zhu et al., 2025) and Dream (Ye et al., 2025) iteratively denoise a fully masked sequence under bidirectional attention. These models have shown competitive performance on language modeling, code generation, and mathematical reasoning.

Controllable text generation. Controlling generation has been extensively studied in the autoregressive setting through prompting (Wei et al., 2021), classifier guidance (Sitdikov et al., 2022), and constrained decoding (Geng et al., 2023; Mündler et al., 2025; Sanchez et al., 2023). For DLMs, recent work has begun to explore control mechanisms tailored to bidirectional, masked decoding. S^3 (Xiong et al., 2025) injects predefined schematic templates (e.g., JSON keys, brackets, or fixed scaffolds) into the initial state to enforce structural formats, and Ti (Lee et al., 2025) similarly relies on hand-designed structural markers with adjustable positions. While effective on rigid format control tasks, these approaches use fixed tokens, providing little semantic guidance that the DLM can utilize. In contrast, our framework learns both the position and the semantic content of the placed anchor end-to-end with feedback from the downstream DLM.

6 Conclusion

In this work, to address challenge of controllability of DLM, we introduced AnchorDLM, a model agnostic framework that jointly learns anchor content and placement for each query to guide generation from the initial state of a frozen DLM. Our DFM analysis characterizes the persistent influence of anchors on the velocity field and their effect on the terminal distribution under idealized assumptions. Across five DLM backbones, training for the respective objectives improves controllability by 11.30 G-mean points on average and matches or exceeds the strongest same-backbone baseline in every reasoning setting.

AI USE STATEMENT

We used Claude Opus 4.6 (Anthropic) to assist in constructing synthetic training data. We also used AI tools for language polishing to improve the grammar, wording, and readability of the manuscript. The authors review AI-assisted outputs and take full responsibility for the generated data, the research design, the experimental results and their interpretation, and the final manuscript.

ETHICS STATEMENT

This work investigates the controllability and reasoning capabilities of diffusion language models using existing mathematical reasoning and code generation benchmarks and synthetic training data. Synthetic data and model-based evaluations may inherit errors and biases from the underlying models. Improved control over output style and structure does not guarantee factual correctness, code safety, or the absence of harmful content. Moreover, controllable generation could be misused to steer models toward undesirable outputs. Applications beyond the evaluated research settings therefore require additional validation and appropriate safeguards.

REPRODUCIBILITY STATEMENT

To promote reproducibility, we document the design choices and experimental settings underlying our results. Section 3.1 describes the AnchorDLM framework and its supervised fine-tuning and reinforcement learning objectives, while section 3.4 states the assumptions underlying the discrete flow matching analysis. Section 4.1 introduces the evaluation benchmarks, baselines, and metrics. Appendix A.1 reports training hyperparameters and computational resources. Appendix A.2 details the data construction procedure, anchor categorization, category rebalancing, and anchor construction prompts. Appendix A.3 further specifies the controllability metrics and the model-based evaluation procedure. These details offer a practical basis for independently implementing the method and assessing its performance under the reported conditions.

REFERENCES

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

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

Raymond PL Buse and Westley R Weimer. Learning a metric for code readability. *IEEE Transactions on software engineering*, 36(4):546–558, 2009.

Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. De-anonymizing programmers via code stylometry. In *24th USENIX security symposium (USENIX Security 15)*, pp. 255–270, 2015.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *arXiv preprint arXiv:2402.04997*, 2024.

Binger Chen and Ziawasch Abedjan. Duetcs: Code style transfer through generation and retrieval. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 2362–2373. IEEE, 2023.

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

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

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *Advances in Neural Information Processing Systems*, 37: 133345–133385, 2024.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured nlp tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, 2023.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and LingPeng Kong. Diffuseq: Sequence to sequence text generation with diffusion models. *arXiv preprint arXiv:2210.08933*, 2022.

Rowan Hart, Brian Hays, Connor McMillin, El Kindi Rezig, Gustavo Rodriguez-Rivera, and Jeffrey A Turkstra. Eastwood-tidy: C linting for automated code style assessment in programming courses. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pp. 799–805, 2023.

Zheng Huang, Kiran Ramnath, Yueyan Chen, Aosong Feng, Sangmin Woo, Balasubramaniam Srinivasan, Zhichao Xu, Kang Zhou, Shuai Wang, Haibo Ding, et al. Diffusion language model inference with monte carlo tree search. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 3493–3512, 2026.

Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James Kwok. Forward-backward reasoning in large language models for mathematical verification. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 6647–6661, 2024.

Fangkai Jiao, Geyang Guo, Xingxing Zhang, Nancy F Chen, Shafiq Joty, and Furu Wei. Preference optimization for reasoning with pseudo feedback. *arXiv preprint arXiv:2411.16345*, 2024.

Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv e-prints*, pp. arXiv–2506, 2025.

Junhoo Lee, Seungyeon Kim, and Nojun Kwak. Unlocking the potential of diffusion language models through template infilling. *arXiv preprint arXiv:2510.13870*, 2025.

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. *arXiv preprint arXiv:2508.19982*, 2025.

Tianjian Li and Daniel Khashabi. Simplemix: Frustratingly simple mixing of off-and on-policy data in language model preference learning. *arXiv preprint arXiv:2505.02363*, 2025.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in neural information processing systems*, 35: 4328–4343, 2022.

Xun Liang, Hanyu Wang, Yezhaohui Wang, Shichao Song, Jiawei Yang, Simin Niu, Jie Hu, Dan Liu, Shunyu Yao, Feiyu Xiong, et al. Controllable text generation for large language models: A survey. *arXiv preprint arXiv:2408.12599*, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

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

Sourabrata Mukherjee, Atul Kr Ojha, John Philip McCrae, and Ondřej Dušek. Evaluating text style transfer evaluation: Are there any reliable metrics? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop)*, pp. 418–434, 2025.

Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, and Martin Vechev. Type-constrained code generation with language models. *Proceedings of the ACM on Programming Languages*, 9(PLDI):601–626, 2025.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

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

Guillaume Sanchez, Honglu Fan, Alexander Spangher, Elad Levi, Pawan Sasanka Ammanamanchi, and Stella Biderman. Stay on topic with classifier-free guidance. *arXiv preprint arXiv:2306.17806*, 2023.

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

Askhat Sitdikov, Nikita Balagansky, Daniil Gavrilov, and Alexander Markov. Classifiers are better experts for controllable text generation. *arXiv preprint arXiv:2205.07276*, 2022.

Yunan Wang, Jijie Li, Bo-Wen Zhang, Liangdong Wang, and Guang Liu. Inco-dpo: Balancing distribution shift and data quality for enhanced preference optimization. *arXiv preprint arXiv:2503.15880*, 2025.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Zhen Xiong, Yujun Cai, Zhecheng Li, and Yiwei Wang. Unveiling the potential of diffusion large language model in controllable generation. *arXiv preprint arXiv:2507.04504*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Dingyi Yang and Qin Jin. Attractive storyteller: Stylized visual storytelling with unpaired text. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11053–11066, 2023.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin E Lauter, Swarat Chaudhuri, and Dawn Song. Position: Formal mathematical reasoning—a new frontier in ai. In *Forty-second International Conference on Machine Learning Position Paper Track*, 2025b.

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

Dutao Zhang, Nicolas Rafael Arroyo Arias, YuLong He, and Sergey Kovalchuk. Style2code: A style-controllable code generation framework with dual-modal contrastive representation learning. *arXiv preprint arXiv:2505.19442*, 2025.

Wenxuan Zhou, Ravi Agrawal, Shujian Zhang, Sathish Reddy Indurthi, Sanqiang Zhao, Kaiqiang Song, Silei Xu, and Chenguang Zhu. Wpo: Enhancing rlhf with weighted preference optimization. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8328–8340, 2024.

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

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11425–11460, 2026.

A Appendix

A.1 Implementation Details

We instantiate the frozen DLM as LLaDA-8B-Base and LLaDA-8B-Instruct Nie et al. (2025), and the anchor generator g_θ as a Qwen3-1.7B backbone with a LoRA adapter of 50M trainable parameters.

Supervised fine-tuning. We adopt a two-stage SFT schedule. Stage 1 trains a prefix anchor on SETUP-only data with $p^* = 0$ for 5 epochs at learning rate 2×10^{-4}, where p^* denotes the supervised target position of the anchor and $p^* = 0$ corresponds to placing the anchor as a prefix of the generation. Stage 2 continues training on the full rebalanced data for 3 epochs at learning rate 1×10^{-4}. Both stages share the same LoRA configuration, batch size 4 with gradient accumulation 2, maximum sequence length 1536, linear schedule with 10% warmup, and `bf16` precision.

Reinforcement learning. We use k=8 rollouts, with temperature = 1.0. Starting from the SFT checkpoint, we further train with WPO for 4 epochs at learning rate 1×10^{-5} and $\beta = 0.3$, using batch size 4 with gradient accumulation 2, maximum sequence length 1536, linear schedule with 10% warmup, and `bf16` precision.

Compute resources. All training and evaluation are conducted on NVIDIA L40 GPUs. The dominant cost is DLM inference during evaluation and during preference-data collection for RL: a single LLaDA-8B inference sample takes on average roughly 20 seconds on coding benchmarks (HumanEval, MBPP) and roughly 30 seconds on the mathematical reasoning benchmark (GSM8K, MATH500).

A.2 Data Construction Details

We provide additional details on anchor categorization and the Query-Anchor Augmentation introduced in Section 3.2. We use a four-category scheme for both mathematical reasoning (GSM8K, MATH500) and code generation (HumanEval, MBPP), with the categories themselves defined separately for each task family.

Math: anchor categories. Each anchor is assigned to one of four categories based on its functional role in the reasoning chain:

- **SETUP**: the set up reasoning step that interprets the question.
- **TRANSFORM**: an intermediate computation that converts existing values into a new value.
- **ACCUM**: an aggregation step that combines values, indicated by cues such as *total*, *altogether*, *combined*.
- **DIFF**: a comparison or remainder step, indicated by cues such as *more than*, *less than*, *remaining*, *difference*.

Math: category rebalancing. The raw distribution is highly imbalanced: SETUP (53.7%), TRANSFORM (30.4%), ACCUM (12.5%), DIFF (3.4%). We rebalance all four categories to a uniform target of 889 samples each (the original size of ACCUM): SETUP and TRANSFORM are downsampled, ACCUM is kept, and DIFF is upsampled via Query-Anchor Augmentation.

Math: Query-Anchor Augmentation. For the under-represented DIFF category, we generate paraphrased queries while keeping the original anchor and answer unchanged. The paraphrasing applies lightweight, semantics-preserving substitutions on question wording, including: *How many* $\leftrightarrow$ *What is the total number of*; *How much* $\leftrightarrow$ *What is the total amount*; *in total* $\leftrightarrow$ *altogether*; *remaining* $\leftrightarrow$ *left over*; *bought* $\leftrightarrow$ *purchased*; *needs to* $\leftrightarrow$ *has to*.

Code: anchor categories. For code generation, anchors are categorized by their syntactic role in the program:

- **COND**: a key decision point, e.g., an `if` or `while` condition.
- **EXPR**: the core computation expression, typically near the top of the function body.
- **BUILD**: a step that constructs the result, e.g., `append`, `add`, `extend`, `yield`.
- **UPDATE**: a state update inside a loop, e.g., accumulators, assignments, or DP transitions.

Code: Query-Anchor Augmentation. For code, we augment the query side by paraphrasing docstrings, varying parameter names, and shuffling doctest order, again without modifying the anchor or the reference solution. After augmentation, the four categories are balanced to the size of the largest category.

Anchor construction prompts. The supervised anchors used in Stage 1 and Stage 2 of SFT are obtained by prompting an external LLM (Claude Opus 4.6) to (i) rewrite each reference answer in the target style and (ii) extract the most distinctive sentence/line as the anchor. We use one prompt per (benchmark, control-objective) pair. All prompts share the same shape: *system instruction* $\rightarrow$ *style description* $\rightarrow$ *rewriting and extraction rules*. We only show the style description and the rewriting/extraction rules below; the anchor format {`"Position": "Content"`} is appended to every prompt as the required output schema.

GSM8K / MATH500 — BACKWARD REASONING.

> **Style.** Backward reasoning solves a problem by starting from the final goal and recursively decomposing it into the sub-goals or quantities required to obtain it, only resolving each sub-goal once its dependencies have been identified. Rewrite the solution in this goal-first, backward style: explicitly state the final goal and the quantities needed to obtain it before doing any computation, using phrases such as "To find X, we need Y" or "Working backwards from the goal". Do *not* use forward connectives such as "First / Then / Finally".
> **Rewrite rule.** Keep the final numerical answer unchanged and the math correct. Do not introduce information beyond the reference answer.
> **Anchor extraction.** Pick the single most pivotal step in the backward decomposition (the one whose sub-goal most directly determines the rest of the chain) as the anchor.

GSM8K / MATH500 — FORMAL MATH.

> **Style.** Rewrite the solution as a sequence of symbolic equations of the form `Let` $x_n = \ldots$, with minimal natural language. Each intermediate quantity is bound to a named variable and reused by subsequent lines.
> **Rewrite rule.** Preserve the reference numerical answer. Do not skip computation steps and do not introduce variables that are never used.
> **Anchor extraction.** Pick the single `Let` $x_n = \ldots$ line that captures the central transformation of the problem as the anchor.

HUMANEVAL / MBPP — COMMENTED CODE.

> **Style.** Rewrite the reference solution so that comments are interleaved with code throughout the function body, not dumped as a leading block. Each non-trivial code line should be preceded or followed by a short, meaningful comment that describes intent.
> **Rewrite rule.** Keep the function signature, behavior, and test results identical to the reference solution. Do not insert decorative or redundant comments.
> **Anchor extraction.** Pick the single `[code, comment]` pair that most clearly conveys the algorithmic intent of the solution as the anchor.

HUMANEVAL / MBPP — STRUCTURED CODE.

> **Style.** Rewrite the reference solution so that the function body begins with a contiguous variable initialization block, followed by the main logic. A function docstring and necessary early-exit guards may precede this block. Use literal values or empty built-in containers for initialization where appropriate.

Rewrite rule. Keep the function signature, behavior, and test results identical to the reference solution. Initialize variables used by the algorithm; do not introduce unused variables or redundant assignments merely to satisfy the style. Preserve data dependencies and keep initializations that must occur within loops or branches in their original context.
Anchor extraction. Select the leading initialization block together with the first subsequent main-logic statement as a single anchor span. Preserve their order and indentation.

GSM8K / MATH500 — MATH REASONING ANCHOR.

Style. Rewrite the solution as a short, simple chain of arithmetic steps, with at most one operation per step. Avoid algebraic manipulation that requires more than one variable at a time, and avoid abbreviations.
Rewrite rule. Preserve the reference numerical answer. Each step should be directly verifiable from the previous one.
Anchor extraction. Pick the single step that is most important to the final answer as the anchor, excluding the final answer itself.

HUMANEVAL / MBPP — CODE REASONING ANCHOR.

Style. Rewrite the reference solution using only basic Python control flow (`for`, `while`, `if`, `return`) and elementary data structures (`list`, `dict`, `set`, `tuple`). Avoid advanced language features such as list/dict/set comprehensions, generator expressions, lambdas, decorators, or one-liners that fuse multiple operations.
Rewrite rule. Keep the function signature and behavior identical to the reference solution and pass all unit tests.
Anchor extraction. Pick the single line that most directly captures the core algorithmic step (e.g., the key conditional, accumulator update, or result-construction line) as the anchor.

A.3 METRIC DETAILS

We provide details for the four controllability metrics summarized in Section 4.1. For each metric, the resulting control score is then combined with the corresponding task performance via the geometric mean to form the composite controllability score reported in the main paper.

Commented code (coding). This metric measures how densely and meaningfully comments are interleaved with code, rather than dumped as a leading block. After applying the same code post-processing used for pass@1, we tag each line as either `code`, `comment`. After collapsing consecutive identical tags, the control score is the fraction of adjacent positions at which the tag switches between `code` and `comment`. A higher value indicates better interleaving.

Structured code (coding). We evaluate structural compliance using the abstract syntax tree (AST) of the target function. After removing the function docstring and skipping permitted leading early-exit guards, we inspect the ordered statements directly within the function body. An initialization is an assignment or annotated assignment to a single variable whose value is a supported literal or a zero-argument built-in constructor, such as `x = 0`, `res = []`, or `seen = set()`. Assignments nested within loops, conditionals, or helper functions do not count toward the leading initialization block.

A solution passes if the remaining function body begins with at least one qualifying initialization and the contiguous initialization block is followed by a non-initialization statement other than a docstring or `pass`. A single initialization is sufficient, and additional initializations receive no extra credit. Let $s_i \in \{0, 1\}$ indicate whether the i-th solution satisfies this constraint. The control score is

$$S_{\text{str}} = \frac{1}{N} \sum_{i=1}^{N} s_i,$$

where N is the total number of generated solutions. Empty or unparseable outputs receive a score of zero. This metric checks structural compliance; it does not assess whether the initialized values are semantically necessary.

Table 4: Average anchor length and content ratio across benchmarks.

	HumanEval	MBPP	GSM8K	MATH500
Anchor length (tokens)	11.08	11.45	27.85	26.63
Anchor content ratio (%)	4.33	4.47	10.88	10.86

Table 5: Transfer from LLaDA-8B-Base to LLaDA-8B-Instruct without retraining the anchor generator. Formal, Backw., and Cmt. report controllability measured by G-mean; Acc. and Pass@1 report task performance. Parentheses indicate absolute gains over each DLM's vanilla baseline.

DLM	Method	GSM8K			HumanEval		MBPP	
		Formal	Backw.	Acc.	Cmt.	Pass@1	Cmt.	Pass@1
LLaDA-8B-Base	Vanilla	5.01	5.01	66.03	14.59	34.15	29.77	39.80
	AnchorDLM	**69.33** (+64.32)	**71.08** (+66.07)	**67.17** (+1.14)	**43.89** (+29.30)	**37.20** (+3.05)	**40.05** (+10.28)	**43.60** (+3.80)
LLaDA-8B-Instruct	Vanilla	17.08	30.94	56.10	29.83	39.02	28.25	40.80
	AnchorDLM (transferred)	**59.51** (+42.43)	**62.34** (+31.40)	**58.23** (+2.13)	**45.02** (+15.19)	**39.63** (+0.61)	**47.80** (+19.55)	**41.00** (+0.20)

Backward reasoning (math). This metric is designed for GSM8K and detects whether a solution is written in a goal-first, backward-reasoning style (e.g., "To find X, we need Y"), as opposed to standard forward chain-of-thought ("First..., Then..., Finally..."). After isolating the current answer block, we apply two complementary gates and count a sample as backward only if it passes both:

- **Rule gate** (G_1): a keyword-based check that fires when the response contains backward cues such as "to find ... we need ..." or "working backwards ...".
- **LLM-judge gate** (G_2): an LLM judge (Qwen3-14B with 3-seed majority vote) that verifies the semantic reasoning logic genuinely starts from the goal and traces back to the required premises, rather than relying on superficial cue words.

The control score is the intersection pass rate, i.e., the fraction of samples for which $G_1 = 1$ and $G_2 = 1$. Combining the two gates avoids both shallow keyword matching and overly lenient semantic judgments.

Formal math (math). This metric is also designed for GSM8K and rewards solutions written as symbolic, equation-driven derivations (e.g., `Let` $x_1 = \ldots$, `Let` $x_2 = \ldots$) rather than natural-language explanations. After the same cleaning and answer isolation as above, each non-empty line is classified into one of: `let`, `assign`, `eqonly`, `answer`, `eqprose`, `prose`, or `noise`. The control score is the fraction of formal lines:

$$F_1 \;=\; 100 \cdot \mathrm{mean}\left(\frac{\#\texttt{let} + \#\texttt{assign} + \#\texttt{eqonly} + \#\texttt{answer}}{\#\text{total lines}}\right). \tag{10}$$

A.4 Anchor Content Ratio

Table 4 reports the average anchor length and the average fraction of output tokens contributed by anchors on each benchmark.

A.5 Generalization across DLMs

Table 5 reports results on the source and target DLMs using the anchor generator trained with LLaDA-8B-Base outputs, without further training on LLaDA-8B-Instruct.