BEYOND MODE-SEEKING RL: TRAJECTORY-BALANCE POST-TRAINING FOR DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models are a promising alternative to autoregressive models, yet post-training methods for them largely adapt reward-maximizing objectives. We identify a central failure mode in this setting that we call *trajectory locking*: sampled reward-driven updates over-concentrate probability mass onto a narrow set of denoising paths, reducing coverage of alternative correct solutions under repeated sampling. To address this, we propose TraFL (**Tra**jectory **F**low ba**L**ancing), a trajectory-balance objective that trains the policy toward a reward-tilted target distribution anchored to a frozen reference model. We make this practical for diffusion language models with a diffusion-compatible sequence-level surrogate and a learned prompt-dependent normalization. Direct measurements of denoising-path diversity show that TraFL preserves reference-level path diversity (101% on average), compared with 87% for ESPO (Ou et al., 2026) and 56% for JustGRPO (Ni et al., 2026). Across mathematical reasoning and code generation benchmarks, TraFL is the only evaluated post-training method that improves over the base model in every benchmark–length setting, with gains that persist as the sampling budget increases. The improvements transfer to held-out evaluations: TraFL stays above the base model on Minerva Math (Lewkowycz et al., 2022) and is the strongest method on every LiveCodeBench (Jain et al., 2025) difficulty split.

1 INTRODUCTION

Recent diffusion language models (dLLMs) have emerged as a compelling alternative to autoregressive models, showing early promise on reasoning and code generation (Zhu et al., 2025; Ye et al., 2025). A key open question for advancing these models further is how to post-train them effectively. From a capability perspective, current post-training methods improve single-sample accuracy but struggle to produce a *diverse* set of correct solutions under repeated sampling—a key requirement when many valid answers exist.

Designing such an objective is challenging because diffusion language models do not expose token-level conditional log-probabilities in the way autoregressive models do. Existing approaches therefore adapt reward-maximizing RL to diffusion generation through surrogate likelihoods, autoregressive reductions, or stepwise approximations (Zhao et al., 2025; Tang et al., 2025; Wang et al., 2026a; Kunde et al., 2026; Ni et al., 2026). Across these methods, we observe a recurring failure mode we call *trajectory locking*: because reward depends only on the final completion and is indifferent to which denoising path produced it, sampled policy-gradient updates reinforce already-favored paths, progressively collapsing probability mass onto a narrow subset of trajectories—and with it, coverage of alternative correct solutions. Distribution-matching methods that train toward a reward-tilted target (Zhu et al., 2026a;b) offer a principled alternative, yet as we show, the way the normalization term is handled can reintroduce the same concentration failure. This raises a basic question: *what is the right post-training objective for diffusion language models that avoids trajectory locking while remaining practical?*

To this end, we propose TraFL (**Tra**jectory **F**low ba**L**ancing), a post-training objective grounded in the trajectory-balance principle from Generative Flow Networks (GFlowNets) (Bengio et al., 2023). Rather than using reward as an unconstrained amplifier of sampled trajectories, TraFL trains the policy toward a reward-tilted target distribution anchored to a frozen reference model. This design directly addresses trajectory locking: the reward determines how much probability mass each completion

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.

receives, while the reference model determines how that mass is distributed across the denoising paths that produce it. As a result, no path is reinforced merely because it was sampled more often. We make this practical for diffusion language models with two ingredients: (i) a diffusion-compatible sequence-level surrogate for comparing fully denoised completions under the current and reference models, and (ii) a learned prompt-dependent normalization, trained jointly with the policy, rather than estimated from the current batch of rollouts. Across mathematical reasoning (GSM8K (Cobbe et al., 2021), MATH-500 Lightman et al. (2023)) and code generation (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021)) benchmarks, TraFL is the only directly evaluated post-training method that improves over the base model in every benchmark-length setting. The gains transfer to held-out evaluations on Minerva Math (Lewkowycz et al., 2022) and LiveCodeBench (Jain et al., 2025), where TraFL is the strongest method on every LiveCodeBench difficulty split. Direct measurements of denoising-path diversity support the proposed mechanism: TraFL preserves reference-level path diversity (101% on average), whereas ESPO and JustGRPO retain only 87% and 56%, respectively.

Our contributions are as follows:

1. We identify *trajectory locking* as a central failure mode of dLLM post-training, where sampled reward-driven updates collapse probability mass onto a narrow set of denoising paths.
2. We propose TraFL (**Tra**jectory **F**low ba**L**ancing), a trajectory-balance objective for post-training diffusion language models that trains toward a reference-anchored reward-tilted target distribution, made practical with a diffusion-compatible sequence-level surrogate and a learned prompt-dependent normalization.
3. TraFL is the only evaluated post-training method that improves over the base model in every benchmark-length setting on GSM8K, MATH-500, HumanEval, and MBPP, outperforming ESPO (Ou et al., 2026) and JustGRPO (Ni et al., 2026) on average.
4. The gains transfer to held-out Minerva Math (Lewkowycz et al., 2022) and LiveCodeBench (Jain et al., 2025), where TraFL is the strongest method on every LiveCodeBench difficulty split, and an LLM-as-judge analysis provides evidence that improvements are associated with broader correct-solution coverage rather than only sharper scoring of a single mode.

2 Related Work

RL post-training for diffusion language models. Recent diffusion language models such as LLaDA (Zhu et al., 2025) and Dream (Ye et al., 2025) have made diffusion-based text generation a viable alternative to autoregressive language models, but reinforcement learning for these models remains challenging because diffusion generation does not expose the same left-to-right token-level conditional factorization used by PPO- or GRPO-style training in autoregressive LLMs. Early work adapted autoregressive RL objectives to diffusion models by introducing likelihood surrogates. In particular, d1 (Zhao et al., 2025) proposes diffu-GRPO, a GRPO-style method for masked dLLMs built on one-step per-token log-probability estimation together with a mean-field approximation to sequence likelihood, and combines it with a preceding supervised fine-tuning stage in its full recipe. wd1 (Tang et al., 2025) removes explicit policy-ratio estimation and instead optimizes a ratio-free weighted log-likelihood objective derived from reverse-KL-regularized policy optimization. SPG (Wang et al., 2026a) addresses the bias induced by one-sided likelihood surrogates by maximizing a lower bound for positive-advantage samples and minimizing an evidence upper bound for negative-advantage samples, together with a block-wise masking strategy for more stable Monte Carlo estimation.

A complementary line of work argues that the central issue is not only the quality of the surrogate, but also the action granularity used by the RL objective. ESPO (Ou et al., 2026) formalizes this view most explicitly: it treats whole-sequence generation as a single action and uses the ELBO as a tractable sequence-level proxy, together with stabilized ratio and KL estimators, arguing that token-level objectives are fundamentally mismatched to non-autoregressive diffusion generation. TraceRL (Wang et al., 2026b) takes yet another perspective, emphasizing alignment between the training objective and the model's preferred inference trajectory. It performs trajectory-aware optimization over denoising traces, and introduces a diffusion-based value model for variance reduction. In contrast, JustGRPO (Ni et al., 2026) argues that preserving arbitrary-order generation during RL can itself be counterproductive for reasoning, and instead constrains training to autoregressive order

so that standard GRPO can be applied directly, while still retaining the parallel decoding benefits of dLLMs at inference time.

Distribution matching and reward-tilted target distributions. Closest in spirit to our work are methods that move beyond pure reward maximization and instead optimize toward reward-tilted target distributions. On the dLLM side, DMPO (Zhu et al., 2026b) formulates post-training as policy distribution matching toward a reward-tilted target distribution and implements this through importance sampling and weighted denoising cross-entropy, i.e., a scalable forward-KL-style approximation to distribution matching. On the autoregressive side, FLOWRL (Zhu et al., 2026a) likewise advocates matching a reward-tilted distribution rather than maximizing reward alone, but does so through a GFlowNet-style formulation for autoregressive reasoning models rather than diffusion language models. Our method is aligned with this broader distribution-matching view, but differs in how it is instantiated for dLLMs: we adopt a trajectory-balance perspective tailored to diffusion generation, combine it with a diffusion-compatible sequence-level surrogate, and learn a prompt-dependent partition function that captures the normalization of the reward-tilted target.

3 TRAJECTORY FLOW BALANCING

We introduce TraFL (**Tra**jectory **F**low ba**L**ancing), a reference-anchored trajectory-balance objective for post-training diffusion language models. We first formalize why terminal rewards can induce trajectory locking, then define the reward-tilted target underlying TraFL.

Setup: Diffusion Trajectories and Terminal Rewards Given a prompt $\mathbf{x}$, a discrete diffusion language model samples denoising trajectories $\tau^{(i)} = (z_T^{(i)}, \dots, z_0^{(i)}) \sim p_\theta(\tau \mid \mathbf{x})$, where z_T and z_0 denote the fully noised and final denoised states, respectively. The reward $r(\mathbf{x}, \mathbf{y})$ is assigned only to this terminal completion, while the model distribution is induced through the full denoising trajectory. This distinction is important because several denoising trajectories can terminate in the same completion $\mathbf{y}$, while different completions can correspond to different valid solution modes. Thus, post-training can affect both the allocation of probability over paths to a fixed answer and the coverage of distinct terminal solution modes.

3.1 TRAJECTORY LOCKING: WHY TERMINAL REWARD IS NOT ENOUGH

Maximizing terminal reward alone is blind to how probability mass is allocated across denoising paths that reach the same completion. The objective depends on the total probability assigned to a final completion $\mathbf{y}$, but not on how that probability is split among the denoising trajectories that produce $\mathbf{y}$. We formalize this path-indifference below.

Proposition 1 (Path-indifference of terminal reward)**.** *Fix a prompt $\mathbf{x}$ and a final completion $\mathbf{y}$. Let $\mathcal{T}(\mathbf{y})$ denote the set of denoising trajectories that terminate at $\mathbf{y}$. Any redistribution of probability mass among trajectories in $\mathcal{T}(\mathbf{y})$ that preserves $p_\theta(\mathbf{y} \mid \mathbf{x})$ leaves a terminal-reward objective unchanged.*

Proof. The terminal-reward objective decomposes as

$$J(\mathbf{x}) = \sum_{\mathbf{y}} p_\theta(\mathbf{y} \mid \mathbf{x})\, r(\mathbf{x}, \mathbf{y}), \qquad p_\theta(\mathbf{y} \mid \mathbf{x}) = \sum_{\tau \in \mathcal{T}(\mathbf{y})} p_\theta(\tau \mid \mathbf{x}).$$

The contribution of $\mathbf{y}$ to J depends only on the scalar $p_\theta(\mathbf{y} \mid \mathbf{x})$, not on how that mass is spread across trajectories in $\mathcal{T}(\mathbf{y})$. Therefore, any redistribution of mass within $\mathcal{T}(\mathbf{y})$ that keeps $p_\theta(\mathbf{y} \mid \mathbf{x})$ unchanged leaves $J(\mathbf{x})$ unchanged. □

Proposition 1 is an objective-level statement: reward maximization does not distinguish among paths that reach the same terminal completion. During optimization, however, this flatness becomes unstable. Under policy-gradient updates, the push each trajectory receives is proportional to its current probability, so among trajectories that reach the same rewarding completion, those that are already more likely gain mass faster. Sampling amplifies this effect further: only sampled trajectories

receive direct gradient signal, so a trajectory that is sampled slightly more often early in training receives more positive updates, becomes more likely, and is sampled even more often. We call this self-reinforcing concentration *trajectory locking*. In App. D.2, we prove that this concentration arises even under exact expected policy-gradient updates: for any completion with positive reward advantage, the entropy of its conditional path distribution is non-increasing (Proposition 2).

Trajectory locking matters because trajectory diversity upper-bounds terminal solution coverage. We make this connection explicit next.

Theorem 1 (Trajectory diversity is necessary for mode coverage)**.** *Let $\mathcal{T}$ be a random denoising trajectory taking values in a countable set $\Omega_{\mathcal{T}}$, and let $M = g(\mathcal{T})$ be its terminal solution mode under a deterministic mapping $g : \Omega_{\mathcal{T}} \to \Omega_M$. Then*

$$|\mathrm{supp}(M)| \le |\mathrm{supp}(\mathcal{T})| \qquad \textit{and} \qquad H(M) \le H(\mathcal{T}),$$

where $\mathrm{supp}(\cdot)$ denotes the support of a distribution and $H(\cdot)$ the Shannon entropy. In particular, terminal mode coverage cannot exceed trajectory diversity.

Corollary 1 (Trajectory locking bounds mode coverage)**.** *If trajectory locking reduces the trajectory distribution to a subset $\mathcal{S}$ of all possible paths, then the set of terminal modes that can be covered is at most $g(\mathcal{S})$. In particular, collapse to a single trajectory implies collapse to a single terminal mode.*

Proofs of Theorem 1 and Corollary 1 are given in App. D.1. Together, these statements make the stakes concrete: trajectory locking does not merely change which paths are used; it can limit which solutions the model can produce under sampling. This motivates a reference-anchored objective that uses reward to tilt terminal completions, while preserving a non-degenerate allocation of probability mass over trajectories.

3.2 Reward-Tilted Trajectory Flow

To avoid relying on terminal reward alone to determine trajectory allocation, we define a reward-tilted target distribution over denoising trajectories by reweighting a frozen reference model:

$$p^*(\tau \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} p_{\mathrm{ref}}(\tau \mid \mathbf{x}) \exp\big(\beta r(\mathbf{x}, \mathbf{y})\big), \tag{1}$$

where $r(\mathbf{x}, \mathbf{y})$ is a scalar reward on the final completion, $\beta > 0$ controls reward sharpness, and $Z(\mathbf{x})$ is the prompt-dependent normalization term. Although the reward is still terminal, it does not define the target distribution by itself: it tilts a frozen reference trajectory distribution. Thus, among trajectories with the same terminal reward, the target remains proportional to $p_{\mathrm{ref}}(\tau \mid \mathbf{x})$, rather than being indifferent to how mass is allocated across them. This separates two roles that are coupled in direct reward maximization: the reward controls how much mass high-reward completions should receive, while the reference model anchors the relative allocation of probability over plausible trajectories and completions, rather than letting reward amplify only the sampled support.

Trajectory Flow Balancing Objective We begin from the trajectory-level balance condition associated with the reward-tilted target in Eq. (1). Equating $p_\theta(\tau \mid \mathbf{x})$ with $p^*(\tau \mid \mathbf{x})$ and taking logarithms gives

$$\delta(\tau, \mathbf{x}) = \log p_\theta(\tau \mid \mathbf{x}) - \log p_{\mathrm{ref}}(\tau \mid \mathbf{x}) - \beta r(\mathbf{x}, \mathbf{y}) + \log Z(\mathbf{x}), \tag{2}$$

and we minimize the squared loss,

$$\mathcal{L}_{\mathrm{TraFL}} = \mathbb{E}_{\tau \sim p_\theta(\cdot \mid \mathbf{x})}\big[\delta(\tau, \mathbf{x})^2\big]. \tag{3}$$

At the minimum, the residual is zero when the current trajectory distribution matches the reward-tilted reference distribution. The learned normalization term $Z(\mathbf{x})$ absorbs the prompt-dependent offset in the balance condition, so the policy is trained on relative trajectory-level preferences rather than on an unnormalized reward signal alone. Eqs. 1–3 define the trajectory-level balance condition.

Practical Instantiation Exact trajectory probabilities are not available for diffusion language models, so the trajectory-level balance condition cannot be evaluated directly. Using $\log p(\tau \mid \mathbf{x}) = \log p(\mathbf{y} \mid \mathbf{x}) + \log p(\tau \mid \mathbf{y}, \mathbf{x})$, the exact residual decomposes into a completion-level term and a within-completion path term. The reward-tilted target changes the completion marginal while retaining the reference conditional $p_{\mathrm{ref}}(\tau \mid \mathbf{y}, \mathbf{x})$.

TraFL instantiates this condition through the completion-level residual using the masked-denoising score below. Crucially, this score depends on a rollout only through its final completion and on corruptions sampled independently of the rollout's realized decoding path. Thus, two trajectories that reach the same completion receive the same expected training signal: a path is not reinforced merely because it was sampled more often. This removes the path-frequency feedback identified in Proposition 2. App. D.5 gives the exact decomposition and scope of this statement, while Sec. 5 verifies empirically that TraFL preserves reference-level path diversity.

Masked-Denoising Score Evaluating Eq. (2) requires log-probability terms under both the current and reference models. Diffusion language models do not expose exact trajectory or completion log-probabilities, so we use a tractable masked-denoising score. Given a completion $\mathbf{y} = (y_1, \ldots, y_L)$, we sample $l \sim \mathrm{Uniform}(\{1, \ldots, L\})$, construct a corrupted sequence $\mathbf{y}_l \sim q_l(\cdot \mid \mathbf{y}, \mathbf{x})$ by replacing exactly l uniformly sampled completion tokens with $[\texttt{mask}]$, and define

$$\log \hat{p}_\theta(\mathbf{y} \mid \mathbf{x}) = \mathop{\mathbb{E}}_{l \sim \mathcal{U}(\{1,\ldots,L\})} \mathop{\mathbb{E}}_{\mathbf{y}_l \sim q_l(\cdot \mid \mathbf{y}, \mathbf{x})} \left[\frac{1}{l} \sum_{i=1}^{L} \mathbf{1}[y_i^l = [\texttt{mask}]] \log p_\theta(y_i \mid \mathbf{y}_l, \mathbf{x}) \right]. \tag{4}$$

The indicator picks out the l masked positions, so the bracketed quantity is the average log-probability across those l positions; the outer expectations are over the sampled mask level l and corruption $\mathbf{y}_l$. As shown in App. C, Eq. (4) equals an expected reveal-order log-probability normalized by the completion length, which lower-bounds the log-likelihood of a uniform reveal-order model by Jensen's inequality. It is therefore not an unbiased estimator of the exact completion log-probability under our sampler. We estimate Eq. (4) with Monte Carlo samples and use the same masking pattern for the current and reference models; this reduces the variance of their difference but does not remove approximation bias.

Why Distribution-Matching Alone Is Insufficient: The Role of a Learned Z. DMPO (Zhu et al., 2026b) also targets a reward-tilted distribution but does not learn $Z(\mathbf{x})$; it instead estimates the partition function with a softmax over the current rollout buffer. This makes the normalization rollout-local: once sampled completions begin to concentrate, both the estimated target weights and the resulting update are computed over the same narrowed support, and completions outside that support receive no gradient signal. As we discuss in App. D.6, this leads to a self-reinforcing concentration over sampled completions. In the ablation in App. B, the learned $Z_\phi(\mathbf{x})$ achieves an average Pass@5 of 64.6, compared with 63.2 for constant Z and 62.7 for a rollout-group estimate of Z.

4 Experimental Setup

4.1 Training Details

Implementation. Our base model is LLaDA-8B-Instruct (Nie et al., 2025), a masked diffusion language model that generates text by iterative denoising over discrete token sequences. For RL post-training, we apply TraFL directly to the instruction-tuned checkpoint, without an additional SFT stage. We use LoRA (Hu et al., 2022) adapters with rank $r = 128$ and scaling factor $\alpha = 128$ for parameter-efficient fine-tuning. The partition-function head $\log Z(\mathbf{x})$ is parameterized as a 2-layer MLP over mean-pooled prompt hidden states, with backbone features detached so that optimizing the partition head does not alter the backbone's prompt representations. We train with a peak learning rate of 6×10^{-5} under a cosine decay schedule with minimum learning rate 6×10^{-6}, and generate 5 rollouts per prompt. The masked surrogate log-probability $\log \hat{p}_\theta(\mathbf{y} \mid \mathbf{x})$ is estimated using 4 antithetic replicates, i.e., complementary mask pairs, with masking applied only to non-EOS positions.

Training Datasets. We train TraFL on two task families, mathematical reasoning and code generation, with separate task-specific checkpoints for each family. For math, we train on GSM8K (Cobbe

et al., 2021) (7,473 problems) and MATH (Lightman et al., 2023) (7,500 problems) as two independent training runs. For code, we train on a single mixture of filtered subsets from AceCode-89K (Zeng et al., 2025) (11,937 examples) and KodCode-Light-RL-10K (Xu et al., 2025) (2,695 examples). Filtering details are given in App. F, and a comparison of training data used by TraFL and the baselines we evaluate against is provided in Tab. 3. Across both domains, rewards are binary: exact answer matching for math and execution-based test passing for code.

4.2 Evaluation Setup

We evaluate on both mathematical reasoning and code generation to test whether improvements transfer across symbolic reasoning and program synthesis. We use the same prompting format for both training and evaluation; representative examples are shown in App. E.

Metrics. At evaluation time, we generate outputs with maximum lengths of 256 and 512 tokens and a block size of 32. We use the fastdLLM (Wu et al., 2025) parallel decoding sampler with a low-confidence masking strategy: at each denoising step, tokens are selected for unmasking according to model confidence, and decoding proceeds under stochastic nucleus sampling with temperature T. We report results under top-p=0.91 sampling and compute **Pass@**k from n independently sampled generations per problem using the unbiased estimator of HumanEval (Chen et al., 2021): $\text{Pass@}k = 1 - \prod_{i=n-c+1}^{n}(1 - k/i)$, where $c \leq n$ is the number of correct samples among the n generations. This estimator has lower variance than evaluating exactly k samples directly and approaches the true solve rate as n increases. For math, correctness is determined by exact-match accuracy of the extracted boxed answer; for code, by execution against the provided test suite.

Benchmarks. For mathematical reasoning, we evaluate on GSM8K (Cobbe et al., 2021) (1,319 problems), MATH-500 (Lightman et al., 2023) (500 problems), and the algebra split of Minerva Math (Lewkowycz et al., 2022) (1,187 problems). For code generation, we evaluate on HumanEval (Chen et al., 2021) (164 problems), MBPP (Austin et al., 2021) (500 problems), and LiveCodeBench (Jain et al., 2025) (release_v5), which contains 880 problems spanning easy, medium, and hard difficulty levels. Together, these benchmarks cover natural-language mathematical reasoning and executable code synthesis, with Minerva Math and LiveCodeBench serving as held-out evaluations beyond the datasets used for post-training.

Baselines. Our primary direct baselines are JustGRPO (Ni et al., 2026) and ESPO (Ou et al., 2026), two recent post-training methods for diffusion language models with usable public checkpoints. This allows us to evaluate all direct baselines under the same decoding protocol and evaluation pipeline. Other closely related methods, including D1 (Zhao et al., 2025) and DMPO (Zhu et al., 2026b), had not released public checkpoints at the time of our experiments. We therefore include their published numbers in App. G for context. For math, all evaluated methods are trained per-task on GSM8K and MATH separately and reported as matched per-task checkpoints. For code, TraFL trains on a filtered AceCode-89K and KodCode mixture (14,632 examples total); the public JustGRPO and ESPO coding checkpoints we evaluate were trained on substantially larger coding mixtures than TraFL, so coding comparisons are conservative with respect to TraFL. Tab. 3 (App. F) summarizes the training data for each method.

5 Results

We first directly measure trajectory locking, then evaluate TraFL across robustness to sampling budget and temperature, in-distribution math and code performance, held-out transfer, and correct-solution diversity, where an LLM-as-a-judge protocol tests whether gains reflect broader coverage of distinct solution strategies.

5.1 Trajectory Locking Analysis

TraFL preserves the reference model's denoising-path diversity, whereas reward-maximizing baselines lose a substantial fraction of it. Before evaluating final outputs, we directly measure the quantity that trajectory locking concerns: the diversity of the denoising paths themselves.

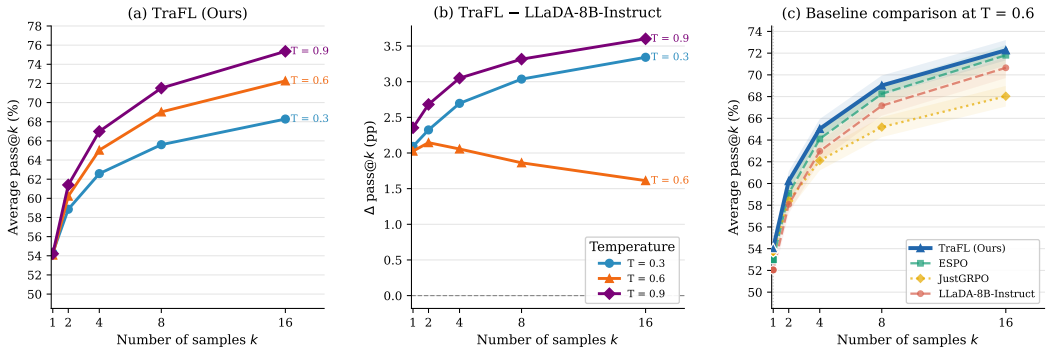


Figure 1: **TraFL improves over the base model and strong prior post-training methods across sampling budgets and temperatures.** (a) Average Pass@k of TraFL on GSM8K, MATH-500, HumanEval, and MBPP for $T \in \{0.3, 0.6, 0.9\}$. (b) Pass@k gap to LLaDA-8B-Instruct under matched decoding. (c) Baseline comparison at $T = 0.6$. TraFL leads ESPO, JustGRPO, and the base model from Pass@1 through Pass@16. All results use $n = 16$ samples and are averaged over generation lengths 256 and 512.

For each prompt, we generate N=16 independent samples and record which token positions each sample unmasks at every denoising step (L=256 generated tokens, T=256 steps). For two samples, we compute the Jaccard overlap $|S_t^i \cap S_t^j|/|S_t^i \cup S_t^j|$ between the sets of positions S_t^i and S_t^j they unmask at the same step t, and average it over all sample pairs and steps to obtain $\bar{J}$. A high $\bar{J}$ means independent generations repeatedly follow the same unmasking schedule, which is the signature of trajectory locking. We then report the fraction of the reference model's path diversity that each method preserves, $\text{Preserved} = \bar{J}_{\text{ref}}/\bar{J}_{\text{model}} \times 100\%$, where the reference is LLaDA-8B-Instruct: 100% means reference-level path diversity is fully preserved, and lower values indicate diversity lost to path concentration.

TraFL fully preserves the reference model's path diversity across benchmarks, averaging 101% (Tab. 1). We read this as reference-level preservation rather than a diversity gain, consistent with anchoring to the reference model. ESPO loses roughly 13% of the reference path diversity, and JustGRPO retains only about 56%. TraFL therefore improves performance while fully preserving the reference model's denoising-path diversity, whereas the other baselines lose a substantial fraction of it.

Table 1: TraFL preserves the reference model's denoising-path diversity. Path diversity preserved (%) relative to LLaDA-8B-Instruct (= 100). HE = HumanEval.

Model	HE	MBPP	MATH	GSM	Avg.
TraFL	98	95	104	109	**101**
ESPO	89	86	95	79	87
JustGRPO	59	58	62	47	56

5.2 Robustness to Sampling Budget and Temperature

TraFL improves over the base diffusion model across all sampling budgets and decoding temperatures we test, including the single-sample setting. We evaluate Pass@k for $k \in \{1, 2, 4, 8, 16\}$ and $T \in \{0.3, 0.6, 0.9\}$, separating two effects that a single Pass@k number can obscure: whether post-training sacrifices Pass@1 for multi-sample gains, and how performance scales as the sampling budget increases. Fig. 1 reports average Pass@k across GSM8K, MATH-500, HumanEval, and MBPP.

For TraFL, Pass@k increases steadily with the sampling budget at all three temperatures (Fig. 1(a)). Higher-temperature decoding is especially beneficial as k grows: $T = 0.9$ gives the strongest performance at larger sampling budgets, followed by $T = 0.6$ and $T = 0.3$. This indicates that TraFL benefits from more exploratory sampling rather than saturating after only a few samples.

The gap to the base model is positive at every k and T (Fig. 1(b)), including $k = 1$. Thus, the multi-sample gains do not come from sacrificing single-sample accuracy. The largest margins occur at $T = 0.9$, where the advantage grows with the sampling budget. The base model itself, however, is

strongest at $T = 0.6$. We therefore use $T = 0.6$ for the head-to-head comparison against post-training methods: this gives the base model its best decoding regime and makes the comparison conservative. At this fixed temperature (Fig. 1(c)), TraFL leads ESPO, JustGRPO, and the base model from Pass@1 through Pass@16. The shaded regions show variation across evaluation settings, but TraFL remains the strongest method as the sampling budget increases.

5.3 Pass@5 Results

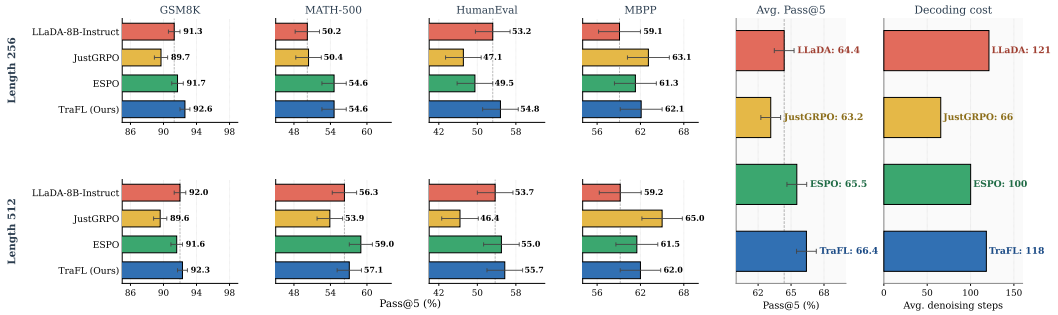


Figure 2: **TraFL improves Pass@5 across math and coding benchmarks at comparable denoising cost.** Pass@5 on GSM8K, MATH-500, HumanEval, and MBPP at maximum completion lengths 256 and 512 ($n = 16$, $T = 0.6$). Error bars show standard error.

TraFL is the only post-training method that improves over the base model in every benchmark-length setting we evaluate. Fig. 2 reports Pass@5 with $n = 16$ samples at $T = 0.6$ for maximum completion lengths 256 and 512, alongside average denoising steps. On math, the largest gain is on MATH-500 at length 256, where TraFL improves Pass@5 from 50.2 to 54.6, with smaller but positive gains on GSM8K at both lengths. On code, TraFL improves HumanEval from 53.2 to 54.8 at length 256 and from 53.7 to 55.7 at length 512, and improves MBPP from 59.1 to 62.1 and 59.2 to 62.0 at the two lengths. Averaged over all eight settings, this is a $+2.0$ point improvement.

The baselines show uneven behavior across benchmarks. ESPO is the strongest prior method overall and improves several settings, especially MATH-500 at length 512, but it drops below the base model on GSM8K-512 and HumanEval-256. JustGRPO improves MBPP substantially but loses performance on GSM8K, HumanEval, and MATH-500 at length 512. TraFL is the only method with positive improvements in all eight settings, so its average gain reflects broad, consistent improvements rather than being concentrated on a single benchmark.

Average denoising steps offer a complementary view (Fig. 2, right). TraFL uses a budget close to the base model (118 vs. 121 steps), whereas JustGRPO terminates much earlier (66 steps) without consistent Pass@5 gains, so fewer steps do not by themselves indicate better generation. This early termination is consistent with the strong path concentration measured for JustGRPO in Sec. 5.1: once mass concentrates on a narrow set of token choices, later denoising steps have little opportunity to explore alternative correct completions. TraFL instead improves multi-sample coverage without altering the decoding cost.

5.4 Held-out Benchmark Evaluation

The gains of TraFL transfer to held-out benchmarks not used during post-training. We evaluate the same checkpoints on the algebra split of Minerva Math (Lewkowycz et al., 2022), an out-of-distribution math benchmark relative to GSM8K and MATH, and LiveCodeBench (Jain et al., 2025), a contamination-free coding benchmark with difficulty-stratified problems. All methods use the same decoding protocol at maximum completion lengths 256 and 512.

On Minerva Math, TraFL stays above the base model at both lengths and substantially outperforms JustGRPO for both GSM8K-trained and MATH-trained checkpoints. ESPO is stronger at length 256 (71.2–71.4 Pass@5 vs. 69.5–70.7 for TraFL), but at length 512 TraFL achieves the best results, reaching 76.3 when trained on GSM8K and 75.8 when trained on MATH.

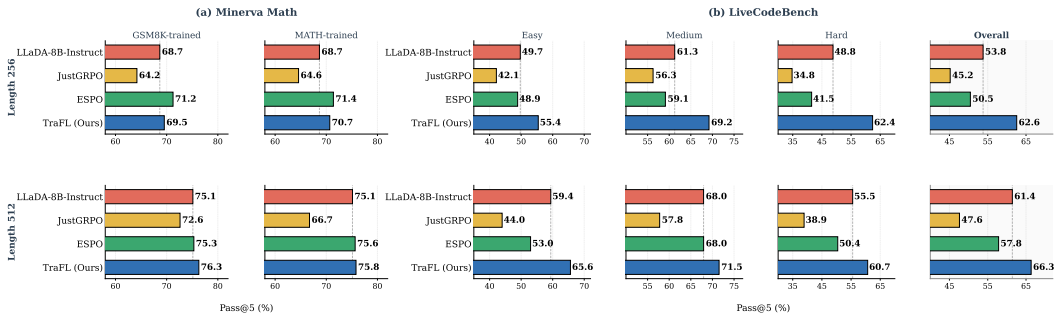


Figure 3: **Gains transfer to held-out math and coding benchmarks.** Pass@5 on Minerva Math (left) using the GSM8K-trained and MATH-trained checkpoints, and on LiveCodeBench (right) by difficulty split, both at maximum completion lengths 256 and 512.

The pattern is stronger on LiveCodeBench. TraFL is the best-performing method on every difficulty split and at both generation lengths, improving over the base model from 53.8 to 62.6 overall at length 256 and from 61.4 to 66.3 at length 512. The largest gains appear on medium and hard problems: at length 256, TraFL improves the base model by 7.9 points on medium and 13.6 points on hard, and outperforms JustGRPO by 12.9 and 27.6 points respectively. ESPO and JustGRPO remain below the base model on the overall LiveCodeBench score at both lengths. The consistent gains observed in the in-distribution analysis therefore generalize beyond the post-training datasets.

5.5 LLM-as-a-Judge Diversity Evaluation

Higher Pass@k does not by itself imply broader mode coverage: a method could improve Pass@k by concentrating probability on a small number of successful strategies. We therefore test whether TraFL's gains are accompanied by broader coverage of distinct correct solution strategies. For each problem in MATH-500 and LiveCodeBench, an LLM judge (GPT-5-4) compares two sets of 16 responses generated under matched decoding and identifies the set containing more substantively different solution approaches, or returns a tie. We restrict the primary analysis to correct responses, retaining comparisons for which both methods produce at least one correct solution.

Among correct solutions, TraFL is preferred over the base model on MATH-500 (12.4% vs. 5.1%, $p = 6.0\times10^{-3}$) and LiveCodeBench (31.2% vs. 19.3%, $p = 4.1\times10^{-5}$). The same pattern holds relative to ESPO on MATH-500 (14.3% vs. 5.6%, $p = 1.0\times10^{-3}$) and LiveCodeBench (38.3% vs. 26.4%, $p = 7.8\times10^{-4}$). In contrast, ESPO is not significantly preferred over the base model on either benchmark. These results are consistent with TraFL's performance gains being accompanied by broader coverage of correct solution strategies rather than only increased concentration on a single mode. Full preference rates including ties, the corresponding all-samples analysis, the complete evaluation protocol, and the judge prompt are provided in App. H and App. I.

6 Conclusion

We presented TraFL, a trajectory-balance post-training objective for diffusion language models that trains toward a reward-tilted target distribution rather than directly maximizing reward. Reward-maximizing objectives are blind to how probability is allocated across denoising paths, making them vulnerable to trajectory locking, a self-reinforcing concentration of path diversity that limits solution coverage. TraFL avoids this by anchoring to a reference model and learning a prompt-dependent partition function $Z_\phi(\mathbf{x})$ jointly with the policy. It preserves the reference model's denoising-path diversity, which ESPO and JustGRPO lose, and is the only evaluated method that improves over the base model in every math and code setting, with gains that persist as the sampling budget increases and transfer to held-out Minerva Math and LiveCodeBench. An LLM-as-judge analysis further indicates broader coverage of correct solution strategies. We hope these results encourage further exploration of distribution-matching objectives for diffusion language model post-training.

AI USE STATEMENT

In this work, we used generative AI tools for one research subtask with required disclosure: an LLM (GPT-5.4) serves as the judge in our solution-diversity evaluation (Sec. 5.5), which falls under supporting qualitative analysis of model outputs. The full judge prompt is given in App. I. We have not used generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, provide ingredients for or write proofs, propose or refine hypotheses, design research methodology or experiments, implement methods, or interpret results, and generating synthetic data, translation, and cleaning or reformatting datasets are not applicable to this work. Additionally, we used generative AI tools to edit the manuscript for grammar, spelling, and readability. We have reviewed all AI-assisted work: the judge was blind to method identity, answer-set order was randomized, and all significance tests were pre-specified before judging; all AI-edited text was checked by the authors, and every reference was manually verified against its original source. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies post-training objectives for diffusion language models on mathematical reasoning and code generation. It involves no human subjects and no personal or sensitive data; all training and evaluation datasets and models are publicly available and used under the licenses listed in App. K. Improvements in code synthesis could be misused to generate malicious scripts, automate parts of cyberattacks, or lower the barrier to producing harmful software. Similarly, more capable reasoning models can be used to generate misleading technical explanations, polished but incorrect solutions, or other content that appears trustworthy despite being wrong. Even when used as intended, errors in model output may still mislead users in high-stakes settings if generations are accepted without verification.

REPRODUCIBILITY STATEMENT

We have made several efforts to ensure reproducibility. The complete training procedure is given in Algorithm 1 (App. A), and all hyperparameters are reported in Sec. 4.1. The derivation of the masked-denoising score is in App. C, and complete proofs of all theoretical results, with their assumptions stated explicitly, are in App. D. Dataset filtering procedures and the training-data comparison across methods are described in App. F, prompt templates in App. E, the LLM-judge prompt in App. I, and compute requirements in App. J. All baselines are evaluated from their public checkpoints under a single shared decoding and evaluation pipeline (Sec. 4.2). Source code and trained checkpoints will be publicly released upon acceptance.

REFERENCES

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

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*, 2021.

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J. Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023. URL `http://jmlr.org/papers/v24/22-0364.html`.

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.

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

W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, 3 edition, 1999. ISBN 978-0-471-16068-7.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, Sept 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Vishnu Teja Kunde, Fatemeh Doudi, Mahdi Farahbakhsh, Dileep Kalathil, Krishna Narayanan, and Jean-Francois Chamberland. Reinforcement learning for diffusion llms with entropy-guided step selection and stepwise advantages. *arXiv preprint arXiv:2603.12554*, 2026.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam

Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

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.

Zanlin Ni, Shenzhi Wang, Yang Yue, Tianyu Yu, Weilin Zhao, Yeguo Hua, Tianyi Chen, Jun Song, Cheng Yu, Bo Zheng, and Gao Huang. The flexibility trap: Why arbitrary order limits reasoning potential in diffusion language models. *arXiv preprint arXiv:2601.15165*, 2026.

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.

Jingyang Ou, Jiaqi Han, Minkai Xu, Shaoxuan Xu, Jianwen Xie, Stefano Ermon, Yi Wu, and Chongxuan Li. Principled RL for diffusion LLMs emerges from a sequence-level perspective. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=S5YeC9llIL.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL https://arxiv.org/abs/2308.12950.

Xindi Tang, Radu Dolga, Sangwon Yoon, and Ilija Bogunovic. wd1: Weighted policy optimization for reasoning in diffusion language models. *arXiv preprint arXiv:2507.08838*, 2025.

Chenyu Wang, Paria Rashidinejad, DiJia Su, Song Jiang, Sid Wang, Siyan Zhao, Cai Zhou, Shannon Zejiang Shen, Feiyu Chen, Tommi Jaakkola, Yuandong Tian, and Bo Liu. Spg: Sandwiched policy gradient for masked diffusion language models, 2026a. URL https://arxiv.org/abs/2510.09541.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=KNAyc9DMe3.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding, 2025. URL https://arxiv.org/abs/2505.22618.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. *arXiv*, 2025. URL https://arxiv.org/abs/2503.02951.

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.

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. Acecoder: Acing coder rl via automated test-case synthesis, 2025. URL https://arxiv.org/abs/2502.01718.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=7ZVRlBFuEv.

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.

Xuekai Zhu, Daixuan Cheng, Dinghuai Zhang, Hengli Li, Kaiyan Zhang, Che Jiang, Youbang Sun, Ermo Hua, Yuxin Zuo, Xingtai Lv, Qizheng Zhang, Lin Chen, Fanghao Shao, Bo Xue, Yunchong Song, Zhenjie Yang, Ganqu Cui, Ning Ding, Jianfeng Gao, Xiaodong Liu, Bowen Zhou, Hongyuan Mei, and Zhouhan Lin. FlowRL: Matching reward distributions for LLM reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=lObnTKbm9U.

Yuchen Zhu, Wei Guo, Jaemoo Choi, Petr Molodyk, Bo Yuan, Molei Tao, and Yongxin Chen. Enhancing reasoning for diffusion llms via distribution matching policy optimization, 2026b. URL https://arxiv.org/abs/2510.08233.

Overview of Appendix

A **Training Algorithm**

B **Ablation of the Learned Normalization**

C **Derivation of the Masked-Denoising Score**

D **Theory: Path Diversity, Mode Coverage, and Trajectory Balance**

E **Prompt Templates and Sample Rollouts**

F **Dataset Filtering and Training-Data Comparison**

G **Additional Baseline Comparison Details**

H **LLM-as-a-Judge Diversity Evaluation: Full Protocol and Results**

I **LLM-as-a-Judge Diversity Prompt**

J **Training Compute**

K **Dataset and Model Licenses**

A Training Algorithm

Algorithm 1 summarizes the training procedure for TraFL. We use a fully online RL setup: at each step, we sample rollouts from the current policy, compute rewards and surrogate scores on the same batch, and perform exactly one gradient update.

Algorithm 1: TraFL training algorithm

Input: initial policy $p_{\theta_{\text{init}}}$; frozen reference p_{ref}; reward r; prompts $\mathcal{D}$; hparams β, G, K, M
Output: trained policy p_θ and normalization predictor Z_ϕ
$p_\theta \leftarrow p_{\theta_{\text{init}}}$, initialize Z_ϕ;
for *step* $= 1$ ***to*** M **do**
  Sample prompt batch $\mathcal{D}_b \sim \mathcal{D}$;
  foreach $\mathbf{x} \in \mathcal{D}_b$ **do**
    Sample $\{\mathbf{y}^{(g)}\}_{g=1}^{G} \sim p_\theta(\cdot \mid \mathbf{x})$;
    Compute $r^{(g)} \leftarrow r(\mathbf{x}, \mathbf{y}^{(g)})$ and $\widetilde{r}^{(g)} \leftarrow r^{(g)} - \frac{1}{G}\sum_{g'=1}^{G} r^{(g')}$;
    Estimate $\log \hat{p}_\theta(\mathbf{y}^{(g)} \mid \mathbf{x})$ and $\log \hat{p}_{\text{ref}}(\mathbf{y}^{(g)} \mid \mathbf{x})$ with Eq. (4) using K mask samples;
    Compute

$$\delta^{(g)} = \log \hat{p}_\theta(\mathbf{y}^{(g)} \mid \mathbf{x}) - \log \hat{p}_{\text{ref}}(\mathbf{y}^{(g)} \mid \mathbf{x}) - \beta\, \widetilde{r}^{(g)} + \log Z_\phi(\mathbf{x}).$$

  Update (θ, ϕ) by minimizing

$$\mathcal{L}_{\text{TraFL}} = \frac{1}{|\mathcal{D}_b| G} \sum_{\mathbf{x} \in \mathcal{D}_b} \sum_{g=1}^{G} \left(\delta^{(g)}\right)^2.$$

B Ablation of the Learned Normalization

We isolate the contribution of the learned prompt-dependent normalization $Z_\phi(\mathbf{x})$ by comparing three variants that differ only in how the normalization term is obtained, with all other components, hyperparameters, and training data held fixed (MATH training set): (i) **Constant** Z: $\log Z$ is fixed to zero, so the residual is driven by the log-probability ratio and the reward alone; (ii) **Batch-estimated** Z: $\log Z$ is estimated per step from the current rollout group rather than learned, making the normalization rollout-local, as in DMPO (Zhu et al., 2026b); (iii) **Learned** $Z_\phi(\mathbf{x})$ (TraFL): the prompt-dependent partition head described in Sec. 4.1. We report Pass@5 ($n = 16$, $T = 0.6$) on MATH-500 (in-distribution) and the Minerva Math algebra split (held-out) at maximum generation lengths 256 and 512.

Table 2: Ablation of the normalization term. Pass@5 (%) with $n = 16$ samples at $T = 0.6$; all variants are trained on MATH with otherwise identical settings.

	MATH-500		Minerva Math		
Normalization	256	512	256	512	Avg.
Constant Z	51.1	56.5	69.5	**75.8**	63.2
Batch-estimated Z	52.7	55.9	69.5	72.5	62.7
Learned $Z_\phi(\mathbf{x})$ (TraFL)	**54.6**	**57.1**	**70.7**	**75.8**	**64.6**

The learned $Z_\phi(\mathbf{x})$ achieves the highest average Pass@5. It outperforms the constant variant in three of four settings and ties it on Minerva Math at length 512, and it outperforms the batch-estimated variant in every setting. These results support both design choices in Sec. 3: the normalization benefits from being prompt-dependent rather than constant, and learning it is more effective than estimating it from the current rollout group.

C Derivation of the Masked-Denoising Score

We relate Eq. (4) to the standard masked-diffusion likelihood bound and characterize what it estimates.

Setup. Let $\mathbf{y} = (y_1, \dots, y_L)$ be a completion of length L. For each mask level $l \in \{1, \dots, L\}$, $q_l(\mathbf{y}_l \mid \mathbf{y})$ selects l completion-token positions uniformly without replacement and replaces them with $[\texttt{mask}]$, so

$$q_l(\mathbf{y}_l \mid \mathbf{y}) = \binom{L}{l}^{-1}$$

on consistent corruptions and zero elsewhere. The model predicts all masked positions in one forward pass, with

$$p_\theta(\mathbf{y} \mid \mathbf{y}_l, \mathbf{x}) = \prod_{i:\, y_i^l = [\texttt{mask}]} p_\theta(y_i \mid \mathbf{y}_l, \mathbf{x}).$$

One-step reconstruction lower bound. Define the one-step masked reconstruction marginal at level l as

$$p_\theta^{(l)}(\mathbf{y} \mid \mathbf{x}) = \mathbb{E}_{\mathbf{y}_l \sim q_l(\cdot \mid \mathbf{y})}\big[p_\theta(\mathbf{y} \mid \mathbf{y}_l, \mathbf{x})\big].$$

Jensen's inequality gives

$$\log p_\theta^{(l)}(\mathbf{y} \mid \mathbf{x}) \geq \mathbb{E}_{\mathbf{y}_l \sim q_l(\cdot \mid \mathbf{y})}\left[\sum_{i=1}^{L} \mathbf{1}[y_i^l = [\texttt{mask}]] \log p_\theta(y_i \mid \mathbf{y}_l, \mathbf{x})\right]. \tag{5}$$

The indicator picks out exactly l terms, so this lower bound is the expected sum of l masked-token log-probabilities.

Relation to reveal-order likelihoods. Writing $\frac{1}{l}\sum_i = \frac{1}{L} \cdot \frac{L}{l}\sum_i$ gives

$$\log \hat{p}_\theta(\mathbf{y} \mid \mathbf{x}) = \frac{1}{L}\,\mathcal{L}_\theta(\mathbf{y} \mid \mathbf{x}), \qquad \mathcal{L}_\theta(\mathbf{y} \mid \mathbf{x}) = \mathbb{E}_l\, \mathbb{E}_{\mathbf{y}_l}\left[\frac{L}{l}\sum_{i=1}^{L} \mathbf{1}[y_i^l = [\texttt{mask}]] \log p_\theta(y_i \mid \mathbf{y}_l, \mathbf{x})\right], \tag{6}$$

where $\mathcal{L}_\theta$ is the standard masked-diffusion evidence lower bound. Sampling a mask level l, a uniformly random mask set of size l, and one of its masked positions is equivalent to sampling a uniformly random reveal order σ and a reveal step. Defining the probability that the model assigns to $\mathbf{y}$ under reveal order σ as $p_\theta^\sigma(\mathbf{y} \mid \mathbf{x}) = \prod_{k=1}^{L} p_\theta(y_{\sigma_k} \mid \mathbf{y}_{\sigma_{<k}}, \mathbf{x})$, we obtain

$$\mathcal{L}_\theta(\mathbf{y} \mid \mathbf{x}) = \mathbb{E}_\sigma\big[\log p_\theta^\sigma(\mathbf{y} \mid \mathbf{x})\big] \leq \log \bar{p}_\theta(\mathbf{y} \mid \mathbf{x}), \qquad \bar{p}_\theta(\mathbf{y} \mid \mathbf{x}) = \mathbb{E}_\sigma\big[p_\theta^\sigma(\mathbf{y} \mid \mathbf{x})\big],$$

by Jensen's inequality. Hence $\log \hat{p}_\theta(\mathbf{y} \mid \mathbf{x})$ is an expected reveal-order log-probability normalized by the completion length, and it lower-bounds the (normalized) log-likelihood of the uniform reveal-order model $\bar{p}_\theta$. The gap vanishes only if $p_\theta^\sigma(\mathbf{y} \mid \mathbf{x})$ is invariant across reveal orders, which masked diffusion models do not generally satisfy.

What the score does and does not estimate. The Monte Carlo estimator of Eq. (4) is unbiased for the expected score $\frac{1}{L}\mathbb{E}_\sigma[\log p_\theta^\sigma(\mathbf{y} \mid \mathbf{x})]$. It is not, in general, an unbiased, consistent, or order-preserving estimator of the exact completion log-probability under our sampler, whose low-confidence remasking does not follow a uniform reveal order. The score enters the loss only through the policy–reference difference. Using identical masks for both models correlates their estimation noise and reduces the variance of this difference, but it does not remove the approximation bias, which need not vanish and can have either sign.

D Theory: Path Diversity, Mode Coverage, and Trajectory Balance

This appendix provides full proofs of the main-paper results and extended discussion. The argument in this section proceeds in three steps: (1) terminal reward is indifferent to path allocation (Proposition 1); (2) policy-gradient updates, even exact expected ones, concentrate the path distribution within each rewarding completion (Proposition 2), and sampling amplifies this concentration into trajectory locking; (3) trajectory locking directly limits terminal mode coverage (Theorem 1 and Corollary 1). The subsections below prove each step in turn and then explain how the trajectory-balance perspective motivates TraFL.

D.1 Full proofs of the main-paper results

Proposition 1 (Path-indifference of terminal reward). Fix a prompt $\mathbf{x}$ and a final completion $\mathbf{y}$. Let $\mathcal{T}(\mathbf{y})$ denote the set of denoising trajectories that terminate at $\mathbf{y}$. Any redistribution of probability mass among trajectories in $\mathcal{T}(\mathbf{y})$ that preserves $p_\theta(\mathbf{y} \mid \mathbf{x})$ leaves a terminal-reward objective unchanged.

Proof. The terminal-reward objective decomposes as

$$J(\mathbf{x}) = \sum_{\mathbf{y}} p_\theta(\mathbf{y} \mid \mathbf{x})\, r(\mathbf{x}, \mathbf{y}), \qquad p_\theta(\mathbf{y} \mid \mathbf{x}) = \sum_{\tau \in \mathcal{T}(\mathbf{y})} p_\theta(\tau \mid \mathbf{x}).$$

The contribution of $\mathbf{y}$ to J depends only on the scalar $p_\theta(\mathbf{y} \mid \mathbf{x})$. Therefore, any redistribution of mass among trajectories in $\mathcal{T}(\mathbf{y})$ that preserves this sum leaves the objective unchanged. □

Theorem 1 (Trajectory diversity is necessary for mode coverage). Let $\mathcal{T}$ be a random denoising trajectory taking values in a countable set $\Omega_\mathcal{T}$, and let $M = g(\mathcal{T})$ denote its terminal solution mode, where $g : \Omega_\mathcal{T} \to \Omega_M$ is a deterministic mapping. Then

$$|\mathrm{supp}(M)| \le |\mathrm{supp}(\mathcal{T})| \qquad \text{and} \qquad H(M) \le H(\mathcal{T}),$$

where $\mathrm{supp}(\cdot)$ denotes the support of a distribution and $H(\cdot)$ the Shannon entropy. In particular, terminal mode coverage cannot exceed trajectory diversity.

Proof. Because $M = g(\mathcal{T})$ is a deterministic function of $\mathcal{T}$, every mode $m \in \mathrm{supp}(M)$ must be the image of at least one trajectory $\tau \in \mathrm{supp}(\mathcal{T})$. Hence $\mathrm{supp}(M) = g(\mathrm{supp}(\mathcal{T}))$, which gives $|\mathrm{supp}(M)| \le |\mathrm{supp}(\mathcal{T})|$.

For entropy, since M is a deterministic function of $\mathcal{T}$, we have $H(M \mid \mathcal{T}) = 0$. By the chain rule,

$$H(\mathcal{T}, M) = H(\mathcal{T}) + H(M \mid \mathcal{T}) = H(\mathcal{T}).$$

On the other hand, $H(\mathcal{T}, M) = H(M) + H(\mathcal{T} \mid M) \ge H(M)$. Combining gives $H(M) \le H(\mathcal{T})$. □

Corollary 1 (Trajectory locking upper-bounds mode coverage). If optimization reduces the trajectory distribution to a subset $\mathcal{S} \subseteq \Omega_\mathcal{T}$, then the set of terminal modes that can be covered

is at most $g(\mathcal{S}) \subseteq \Omega_M$. In particular, collapse to a single trajectory implies collapse to a single terminal mode.

Proof. Immediate from Theorem 1, since $M = g(\mathcal{T})$. □

D.2 Policy-gradient dynamics and trajectory locking

Proposition 1 is an objective-level statement: terminal reward is flat over the subspace of trajectory distributions that assign the same mass to each completion. We now show that gradient-based optimization does not stay neutral on this flat subspace: even exact expected policy-gradient updates concentrate the path distribution within each rewarding completion.

Proposition 2 (Policy gradient concentrates paths within a completion)**.** *Consider a softmax policy over trajectories, $p_i \propto \exp(\phi_i)$, trained by exact gradient ascent on the logits of $J = \sum_i p_i r_i$, where $r_i = r(\mathbf{x}, \mathbf{y}(\tau_i))$. Fix a completion $\mathbf{y}$ and suppose that, at the current time, it has positive reward advantage $A_{\mathbf{y}} = r(\mathbf{x}, \mathbf{y}) - \bar{r} > 0$. Let q be the conditional distribution over $\mathcal{T}(\mathbf{y})$. Then (i) for $\tau_i, \tau_j \in \mathcal{T}(\mathbf{y})$ with $q_i > q_j$, the ratio q_i/q_j increases; and (ii) $H(q)$ is non-increasing, and strictly decreasing whenever q is non-uniform.*

Proof. Since $\partial p_k / \partial \phi_i = p_k(\delta_{ki} - p_i)$, exact gradient ascent on the logits gives

$$\dot{\phi}_i = \frac{\partial J}{\partial \phi_i} = p_i\,(r_i - \bar{r}), \qquad \bar{r} = \sum_k p_k r_k.$$

Let $p_{\mathbf{y}} = \sum_{i \in \mathcal{T}(\mathbf{y})} p_i$ and $q_i = p_i / p_{\mathbf{y}}$ for $i \in \mathcal{T}(\mathbf{y})$. All trajectories in $\mathcal{T}(\mathbf{y})$ share the reward $r(\mathbf{x}, \mathbf{y})$, so

$$\dot{\phi}_i = p_{\mathbf{y}}\, A_{\mathbf{y}}\, q_i, \qquad i \in \mathcal{T}(\mathbf{y}).$$

(i) Ratio dynamics. Since q is the softmax of ϕ restricted to $\mathcal{T}(\mathbf{y})$,

$$\frac{d}{dt} \log \frac{q_i}{q_j} = \dot{\phi}_i - \dot{\phi}_j = p_{\mathbf{y}}\, A_{\mathbf{y}}\, (q_i - q_j),$$

which is positive whenever $A_{\mathbf{y}} > 0$ and $q_i > q_j$.

(ii) Entropy dynamics. The restricted softmax gives $\dot{q}_i = q_i\big(\dot{\phi}_i - \sum_k q_k \dot{\phi}_k\big) = p_{\mathbf{y}} A_{\mathbf{y}}\, q_i\big(q_i - \mathbb{E}_q[q]\big)$. Using $\sum_i \dot{q}_i = 0$,

$$\frac{d}{dt} H(q) = -\sum_i \dot{q}_i \log q_i = -p_{\mathbf{y}} A_{\mathbf{y}}\, \mathrm{Cov}_q\big(q_i, \log q_i\big).$$

Because $\log$ is strictly increasing, q_i and $\log q_i$ are comonotone, so $\mathrm{Cov}_q(q_i, \log q_i) \geq 0$, with equality if and only if q is uniform on its support. Hence $\frac{d}{dt} H(q) \leq 0$ when $A_{\mathbf{y}} > 0$, strictly whenever q is non-uniform. □

All trajectories reaching $\mathbf{y}$ receive the same reward, yet the update to each logit is proportional to that trajectory's current probability, so trajectories that are already more likely gain mass faster. The concentration is therefore driven by the policy-gradient dynamics themselves, not only by sampling noise. Sampling amplifies it further: with finite rollouts, only sampled trajectories receive direct gradient signal, so trajectories sampled more often early in training are reinforced more and sampled even more often subsequently.

Proposition 2 establishes the direction of the dynamics in a tabular softmax model, not convergence to a single trajectory. Complete asymptotic collapse would require additional persistence assumptions, and shared neural parameters, KL regularization, clipping, entropy bonuses, and finite-sample optimization can all modify the dynamics. We therefore use *trajectory locking* descriptively, for sustained concentration of the path distribution in neural diffusion policies, and verify it empirically in Sec. 5.

D.3 A GENERALIZED SIMPLEX VIEW OF TRAJECTORY CONCENTRATION

The discussion above concerns strict terminal reward, under which all trajectories leading to the same terminal completion receive the same reward. The following analysis considers a more general setting in which optimization induces effective trajectory-level scores, for example through sampling asymmetries, surrogate objectives, or path-dependent credit assignment. It should therefore be viewed as an auxiliary lens on trajectory concentration rather than a literal description of pure terminal reward. In particular, under the additive dynamics below, trajectories with equal scores keep a constant probability ratio, so this view does not by itself capture concentration among trajectories reaching the same completion under pure terminal reward; that case is covered by the exact Euclidean analysis in Sec. D.2.

Fix a completion $\mathbf{y}$, and let

$$\mathcal{T}(\mathbf{y}) = \{\tau_1, \dots, \tau_K\}.$$

Define the conditional trajectory distribution

$$q_i := p_\theta(\tau_i \mid \mathbf{y}, \mathbf{x}), \qquad q = (q_1, \dots, q_K) \in \Delta^K,$$

and suppose each trajectory τ_i is assigned an effective scalar score r_i. Consider the linear objective

$$J(q) = \sum_{i=1}^{K} q_i r_i. \tag{7}$$

Because Eq. (7) is linear in q, its maximizers lie on extreme points of the simplex when one score dominates, and on a face of the simplex when several scores tie. Thus the objective itself does not prefer dispersed trajectory distributions.

Now parameterize q by logits ϕ through

$$q_i = \frac{e^{\phi_i}}{\sum_j e^{\phi_j}}.$$

The Euclidean gradient of J with respect to ϕ is

$$\frac{\partial J}{\partial \phi_i} = q_i(r_i - \bar{r}), \qquad \bar{r} = \sum_j q_j r_j. \tag{8}$$

Eq. (8) already shows that higher-than-average reward increases the corresponding logit. To obtain the standard replicator dynamics in probability space, however, one must specify a particular continuous-time flow in logit space, and this flow is *not* the ordinary Euclidean gradient flow on the logits. Instead, consider *additive payoff dynamics*,

$$\dot{\phi}_i = r_i. \tag{9}$$

This choice corresponds to mirror-descent-style updates on the simplex, and equivalently to a natural-gradient geometry in probability space (Amari, 1998; Beck & Teboulle, 2003). Under this assumed logit flow, the induced dynamics of q_i are obtained by the chain rule:

$$\dot{q}_i = \sum_j \frac{\partial q_i}{\partial \phi_j}\, \dot{\phi}_j, \qquad \frac{\partial q_i}{\partial \phi_j} = q_i(\delta_{ij} - q_j).$$

Substituting $\dot{\phi}_j = r_j$ gives

$$\dot{q}_i = \sum_j q_i(\delta_{ij} - q_j)\, r_j = q_i\big(r_i - \bar{r}\big),$$

which is the replicator equation:

$$\dot{q}_i = q_i(r_i - \bar{r}). \tag{10}$$

More generally, under $\dot{\phi}_i = \eta r_i$, the induced flow $\dot{q}_i = \eta\, q_i(r_i - \bar{r})$ is proportional to the replicator vector field and differs only by a rescaling of time. The solution to Eq. (10) is

$$q_i(t) = \frac{q_i(0)\, e^{r_i t}}{\sum_{j=1}^{K} q_j(0)\, e^{r_j t}}, \tag{11}$$

so relative mass evolves as

$$\frac{q_i(t)}{q_j(t)} = \frac{q_i(0)}{q_j(0)}\, e^{(r_i - r_j)t}.$$

When $r_i > r_j$, this ratio grows exponentially, driving the conditional trajectory distribution toward a simplex vertex or a low-dimensional face. This makes precise the sense in which reward-style optimization can be mode-seeking when effective path-level growth rates differ.

Remark on standard Euclidean gradient ascent. For completeness, we note that ordinary Euclidean gradient ascent on the logits, $\dot{\phi}_i \propto \partial J/\partial \phi_i = q_i(r_i - \bar{r})$, instead induces the cubic dynamic

$$\dot{q}_i \propto q_i \left[q_i(r_i - \bar{r}) - \sum_j q_j^2 (r_j - \bar{r}) \right],$$

which does not admit the closed-form exponential reweighting in Eq. (11). We use the replicator form here because it admits a clean exponential solution; the exact Euclidean dynamics for the terminal-reward case are analyzed in Sec. D.2. The qualitative concentration conclusion—that higher-scoring trajectories acquire more mass relative to lower-scoring ones—is robust to this choice: both flows drive q toward simplex vertices or low-dimensional faces when scores differ. The replicator equation should therefore be viewed as a simplifying assumption rather than the actual logit dynamics, chosen for analytical tractability rather than as the literal flow induced by standard policy-gradient updates.

D.4 Exact trajectory balance with a backward distribution

This subsection describes the exact trajectory-balance condition that motivates our design. TraFL is inspired by this principle, but in practice uses a diffusion-compatible sequence-level surrogate and a learned prompt-dependent normalization term rather than exact trajectory probabilities.

Let $P_F(\tau)$ denote a forward trajectory distribution and let $P_B(\tau \mid \mathbf{y}, \mathbf{x})$ denote a backward distribution over trajectories conditioned on a terminal completion $\mathbf{y}$. In an exact trajectory-balance formulation, one enforces

$$\log Z(\mathbf{x}) + \log P_F(\tau \mid \mathbf{x}) = \beta r(\mathbf{x}, \mathbf{y}) + \log P_B(\tau \mid \mathbf{y}, \mathbf{x}), \qquad \forall \tau \in \mathcal{T}(\mathbf{y}). \tag{12}$$

Equivalently,

$$P_F(\tau \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\big(\beta r(\mathbf{x}, \mathbf{y})\big)\, P_B(\tau \mid \mathbf{y}, \mathbf{x}). \tag{13}$$

Summing over $\tau \in \mathcal{T}(\mathbf{y})$ yields

$$P_F(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\big(\beta r(\mathbf{x}, \mathbf{y})\big). \tag{14}$$

Therefore, reward determines only the *total* mass assigned to terminal completion $\mathbf{y}$. Conditioned on $\mathbf{y}$, we obtain

$$P_F(\tau \mid \mathbf{y}, \mathbf{x}) = P_B(\tau \mid \mathbf{y}, \mathbf{x}). \tag{15}$$

This factorization separates terminal weighting from within-terminal path allocation. Unlike direct reward amplification, it does not repeatedly favor one trajectory over another merely because they terminate at the same rewarding output.

We stress again that Eqs. (12) to (15) describe the exact trajectory-balance condition. They are intended as conceptual motivation for TraFL, not as exact identities satisfied by the practical surrogate objective in Sec. 3.

D.5 Relation to the practical TraFL objective

The exact trajectory-balance construction above specifies the target property we seek: reward controls terminal mass, while the reference distribution anchors how that mass is allocated across trajectories. We now show how the practical TraFL objective instantiates this principle without requiring intractable trajectory probabilities.

Reference model instead of explicit backward model. The exact TB factorization in Sec. D.4 uses a backward distribution $P_B(\tau \mid \mathbf{y}, \mathbf{x})$ to govern within-completion path allocation. In TraFL, this role is played by the frozen reference model: the reward-tilted target

$$p^*(\tau \mid \mathbf{x}) \propto p_{\mathrm{ref}}(\tau \mid \mathbf{x}) \exp\big(\beta r(\mathbf{x}, \mathbf{y})\big)$$

factors, after marginalization over $\tau \in \mathcal{T}(\mathbf{y})$, into

(i) a completion-level marginal proportional to $\exp\big(\beta r(\mathbf{x}, \mathbf{y})\big)\, p_{\mathrm{ref}}(\mathbf{y} \mid \mathbf{x})$, which determines how much total mass each completion receives, jointly through the reward tilt and the reference model's marginal over completions; and

(ii) a within-completion conditional path distribution $p_{\mathrm{ref}}(\tau \mid \mathbf{y}, \mathbf{x})$, which governs how that mass is spread across the denoising trajectories that reach the same completion.

This is precisely the separation that policy-gradient reward maximization lacks: reward acts only on (i), while (ii) is left unconstrained by the objective and is concentrated by the policy-gradient dynamics themselves (Proposition 2), enabling trajectory locking.

Path invariance of the practical residual. Diffusion language models do not expose exact trajectory probabilities, so TraFL replaces the corresponding log-probability terms by the masked-denoising score in Eq. (4). Let m denote the random masking pattern used to score a completion and write the practical residual as

$$\widehat{\delta}(\mathbf{y}, \mathbf{x}; m).$$

For any two realized trajectories $\tau, \tau' \in \mathcal{T}(\mathbf{y})$, the practical residual satisfies

$$\widehat{\delta}(\tau, \mathbf{x}; m) = \widehat{\delta}(\tau', \mathbf{x}; m) = \widehat{\delta}(\mathbf{y}, \mathbf{x}; m),$$

because the realized denoising path does not enter the score and m is sampled independently of the policy's decoding order. Consequently, within a fixed completion, a trajectory receives no additional training signal merely because that trajectory occurred more frequently in the rollout batch. This is exactly the path-frequency feedback responsible for the concentration in Proposition 2.

This path-invariance property does not require that $p_\theta(\tau \mid \mathbf{y}, \mathbf{x})$ remain exactly equal to $p_{\mathrm{ref}}(\tau \mid \mathbf{y}, \mathbf{x})$ throughout training: shared parameter updates can change the conditional distribution indirectly. The claim of the practical objective is instead that it removes direct path-dependent reinforcement. Whether this is sufficient to preserve path diversity in the neural policy is an empirical question; Sec. 5 shows that TraFL preserves reference-level path diversity in practice.

Centered rewards as variance reduction. In the implementation, we replace $r(\mathbf{x}, \mathbf{y})$ by rewards centered within each prompt group. Because the centering term depends on the sampled group rather than only on $\mathbf{x}$, it cannot in general be absorbed exactly into the deterministic normalization term $Z(\mathbf{x})$. This centering should therefore be interpreted as a stochastic variance-reduction heuristic for optimization, not as an exact rewriting of the target distribution.

D.6 Why DMPO's batch-softmax $\hat{Z}$ still induces trajectory locking

DMPO (Zhu et al., 2026b) targets the same reward-tilted distribution p^* as TraFL but estimates $Z(\mathbf{x})$ without a learned network, instead normalizing importance weights by a softmax over the current training buffer. Concretely, for each rollout $o^{(n)}$ it defines

$$w^{(n)} = \frac{\exp(\ell^{(n)})}{\sum_k \exp(\ell^{(k)})}, \qquad \ell^{(n)} = \frac{r^{(n)}}{\alpha} + \log \frac{p_{\mathrm{ref}}(o^{(n)} \mid \mathbf{x})}{p_v(o^{(n)} \mid \mathbf{x})},$$

and minimizes $\sum_n w^{(n)} \mathcal{L}_{\mathrm{DCE}}(o^{(n)})$. This batch-softmax step is an empirical estimator of $\hat{Z}(\mathbf{x})$.

Treating the weights $w^{(n)}$ as stop-gradient targets (as in DMPO's implementation), the WDCE objective drives the buffer logits $\phi^{(n)} = \log p_\theta(o^{(n)} \mid \mathbf{x})$ according to additive payoff dynamics

$$\dot{\phi}^{(n)} \propto w^{(n)}.$$

This is precisely the additive payoff regime analyzed in Sec. D.3: under additive logit dynamics, the chain rule through the softmax Jacobian yields the replicator equation on the conditional buffer

distribution,

$$\dot{q}^{(n)} = q^{(n)}\big(w^{(n)} - \bar{w}\big), \qquad q^{(n)} = \frac{p_\theta(o^{(n)} \mid \mathbf{x})}{\sum_k p_\theta(o^{(k)} \mid \mathbf{x})},$$

with solution

$$q^{(n)}(t) = \frac{q^{(n)}(0)\, e^{w^{(n)} t}}{\sum_k q^{(k)}(0)\, e^{w^{(k)} t}}.$$

The highest-weight completion therefore acquires exponentially more mass relative to all others in the buffer, and any completion not present in the buffer receives zero gradient signal. As the policy concentrates, future buffers are sampled from an increasingly narrow distribution, so unseen modes remain unseen. Unlike TraFL's learned Z_ϕ, the batch-softmax $\hat{Z}$ degrades to a point estimate over the locked support, removing the correction signal that would otherwise counteract collapse.

E Prompt Templates and Sample Rollouts

Below, we present two representative sampled trajectories, one from the coding domain and one from mathematical reasoning. Each trajectory consists of a task prompt and a completion sampled from the current policy. We use the same prompting format for both training and evaluation: all task instructions and response-format constraints are specified directly in the user prompt, with no separate system prompt.

Rollout Example: System Prompt and Sample Model Output

Rollout Example: User Prompt and Sample Model Output

[User Prompt]
You are a coding expert. You will be given a coding problem to solve.
Solve it step by step. Ensure you wrap the answer in ```` ```python``` ````.

Response Format
<reasoning> Your reasoning here </reasoning>
<answer> ```python ... ``` </answer>

[Task Prompt]
Given a list of subject IDs and a list of trial types, create a function that generates a dataset as a dictionary. Each subject ID is a key, and its value is another dictionary mapping trial types to strings formatted as `Subject {subject_id}, Trial {trial_type}`.

[Model Output]
<reasoning>
To solve this problem: (1) iterate over subject IDs; (2) create a nested dictionary for each subject; (3) map each trial type to the required formatted string; and (4) return the completed dataset.
</reasoning>

<answer>

```python
def generate_dataset(subject_ids, trial_types):
    dataset = {}
    for subject_id in subject_ids:
        dataset[subject_id] = {}
        for trial_type in trial_types:
            dataset[subject_id][trial_type] = (
                f"Subject {subject_id}, Trial {trial_type}"
            )
    return dataset
```

</answer>

Rollout Example: System Prompt and Sample Model Output

Rollout Example: User Prompt and Sample Model Output

[User Prompt]
You are a math expert. You will be given a question to solve.
Solve it step by step. Wrap the final answer in `\boxed{}` inside the <answer> tags..

Response Format
<reasoning> Your reasoning here </reasoning>
<answer> \boxed{...} </answer>

[Task Prompt]
Mark makes custom dog beds. A bed for a Rottweiler takes 8 pounds of stuffing, a bed for a chihuahua takes 2 pounds of stuffing, and a bed for a collie takes the average amount of stuffing between the first two kinds of beds. How many pounds of stuffing does Mark need to make 4 chihuahua beds and 3 collie beds?

[Model Output]
<reasoning>
First, a collie bed uses the average amount of stuffing between a Rottweiler bed and a chihuahua bed, so it needs

$$\frac{8+2}{2} = 5$$

pounds.
Next, 4 chihuahua beds require

$$4 \times 2 = 8$$

pounds, and 3 collie beds require

$$3 \times 5 = 15$$

pounds.
Therefore, the total amount of stuffing needed is

$$8 + 15 = 23.$$

</reasoning>

<answer>

$$\boxed{23}$$

</answer>

F Dataset Filtering and Training-Data Comparison

This appendix gives the filtering procedures used to construct TraFL's coding training set and summarizes how the training data of TraFL relates to the data used by the public baseline checkpoints we compare against.

F.1 Filtering Procedures

AceCode-89K. We use existing CodeLlama-Instruct (Rozière et al., 2024) rollout metadata distributed with AceCode-89K to filter problems by reward uncertainty under the proxy model. We retain problems with substantial reward uncertainty under that proxy, removing both trivially easy problems (uniformly passed) and uniformly failed problems for which binary feedback would not be informative. This yields 11,937 examples.

KodCode-Light-RL-10K. For KodCode, we compute per-example pass-rate statistics from GPT-4o and DeepSeek-R1 (Guo et al., 2025) rollouts. We identify the weaker proxy on average, retain examples with high pass-sequence variance under that weaker proxy, and then keep the lower-pass-rate portion of this diverse subset. This focuses training on harder coding problems while still admitting informative binary feedback. The resulting subset contains 2,695 examples.

Combined coding training set. TraFL's coding model is trained on the concatenation of these two filtered subsets (14,632 examples total). The filtering targets problems with informative binary feedback rather than removing examples on stylistic or domain grounds.

F.2 Training-Data Comparison Across Methods

The public checkpoints of our direct baselines (JustGRPO and ESPO) were trained on different data mixtures than TraFL. Tab. 3 summarizes the training data used by each method we compare against, so that algorithmic and data differences are not conflated when reading the empirical results.

Two aspects of this comparison are worth noting. First, all methods follow the same per-task convention for math, training one checkpoint on GSM8K and a separate checkpoint on MATH, so the math comparisons are matched in training-data scale and source. Second, the coding training set used by TraFL (14,632 examples) is smaller than the coding mixtures used to produce the public JustGRPO and ESPO coding checkpoints, so coding comparisons are conservative with respect to TraFL: any algorithmic advantage we report on coding holds despite TraFL being trained on less data than the baselines we compare against. None of these training-data differences affect the held-out

Table 3: Training-data summary across the methods we compare against. All methods train separate task-specific checkpoints for GSM8K and for MATH (reported separately throughout the paper). For coding, the public JUSTGRPO and ESPO checkpoints we evaluate were trained on substantially larger coding mixtures than TraFL.

Method	GSM8K-trained	MATH-trained	Code training data
JUSTGRPO (Ni et al., 2026)	GSM8K (7,473)	MATH (7,500)	AceCode-89K (21,000 hard samples)
ESPO (Ou et al., 2026)	GSM8K (7,473)	MATH (7,500)	AceCode-89K (21,000 hard samples)
TraFL (Ours)	GSM8K (7,473)	MATH (7,500)	AceCode-89K filtered (11,937) + KodCode filtered (2,695)

Table 4: Single-sample accuracy (%) on reasoning and coding benchmarks. Results are reported for two maximum completion lengths, 256 and 512. Results marked with † are quoted directly from the original papers; these entries are included for context rather than as fully controlled direct comparisons.

	GSM8K		MATH-500		HumanEval		MBPP		
Method	256	512	256	512	256	512	256	512	Avg.
D1† (Zhao et al., 2025)	81.1	82.1	38.6	40.2	32.9	37.8	44.7	42.8	50.03
DMPO† (Zhu et al., 2026b)	82.41	85.22	38.20	42.80	–	–	–	–	–

evaluations on Minerva Math and LiveCodeBench, where the conclusions reported in the main text are drawn under matched decoding settings on benchmarks that no method was trained on directly.

G ADDITIONAL BASELINE COMPARISON DETAILS

In our main experiments, we directly compare against JUSTGRPO (Ni et al., 2026) and ESPO (Ou et al., 2026), since these were the strongest closely related diffusion-LM post-training baselines with usable public checkpoints. This enables a controlled comparison in which all directly evaluated methods are run under the same decoding protocol and evaluation pipeline.

For completeness, Tab. 4 also reports published results for D1 (Zhao et al., 2025) and DMPO (Zhu et al., 2026b), copied from their respective papers. These methods are closely related to our setting, but their public checkpoints were not available at the time of our experiments. We therefore include their numbers only as contextual reference points, rather than as fully controlled comparisons. The table marks such quoted results with †.

H LLM-AS-A-JUDGE DIVERSITY EVALUATION: FULL PROTOCOL AND RESULTS

We use an LLM-as-a-judge protocol with GPT-5-4 on MATH-500 (IID to our math post-training distribution) and LiveCodeBench (OOD to our coding post-training distribution). For each problem, the judge compares two sets of 16 responses (D_A, D_B) generated under matched decoding (max length 256, $T = 0.6$), with answer-set order randomized to provide the most diverse set (A or B), or tie, $\mathcal{S}_{LLM} : (D_A, D_B) \to \{A_{wins}, B_{wins}, Tie\}$. We run pairwise comparisons between TraFL and each baseline. The full judge prompt is in App. I.

The judge evaluates diversity in the underlying solution approach, not surface form. An answer set is preferred only when it contains a broader range of substantively different methods, such as different equation setups, proof strategies, or algorithmic decompositions. The prompt asks the judge to ignore wording, formatting, variable names, verbosity, and minor errors that do not change the core approach. We pre-specify three one-sided sign tests (Conover, 1999) (p-value$\leq$0.05) with hypotheses ($\mathbf{H_{alt}}$) of the form (a) $\mathrm{TraFL}_{wins} > \mathrm{Base}_{wins}$, (b) $\mathrm{ESPO}_{wins} > \mathrm{Base}_{wins}$,and (c) $\mathrm{TraFL}_{wins} > \mathrm{ESPO}_{wins}$ for each benchmark.

Table 5: **LLM-as-a-judge diversity evaluation.** TraFL shows the strongest and most consistent diversity gains over correct solutions, where it is preferred over both the base model and ESPO on MATH-500 and LiveCodeBench. ESPO is not significantly more diverse than the base model on either benchmark. "For"/"Against" are the fractions of judged problems favoring each side; p-values are from one-sided exact sign tests over non-tied comparisons.

Benchmark	H_{alt}	**All samples**					**Correct solutions**				
		For	**Against**	**Tie**	p	**Sig.**	**For**	**Against**	**Tie**	p	**Sig.**
MATH-500	TraFL> Base	21.8	15.4	62.8	2.3×10^{-2}	✓	12.4	5.1	82.5	6.0×10^{-3}	✓
	ESPO > Base	18.2	17.6	64.2	8.8×10^{-1}	✗	7.5	10.0	82.6	3.9×10^{-1}	✗
	TraFL> ESPO	21.8	14.2	64.0	5.7×10^{-3}	✓	14.3	5.6	80.1	1.0×10^{-3}	✓
LiveCodeBench	TraFL> Base	20.9	23.0	56.1	3.9×10^{-1}	✗	31.2	19.3	49.6	4.1×10^{-5}	✓
	ESPO > Base	14.5	44.2	41.3	1.0×10^{0}	✗	31.1	34.0	34.9	7.9×10^{-1}	✗
	TraFL> ESPO	45.9	12.2	41.9	$< 10^{-4}$	✓	38.3	26.4	35.3	7.8×10^{-4}	✓

Tab. 5 separates two notions of diversity. The all-samples setting measures the diversity of the model's sampled behavior overall, including both correct and incorrect responses. Under this view, TraFL shows a modest but significant gain over the base model on MATH-500, indicating that it explores a broader set of reasoning attempts in the math setting; the signal is not reliable on LiveCodeBench. We do not find support for ESPO_{wins}>Base_{wins}. Against ESPO, TraFL is preferred on both: 21.8% vs. 14.2% on MATH-500 ($p = 0.0057$) and 45.9% vs. 12.2% on LiveCodeBench ($p < 10^{-4}$).

The correct-solution setting asks the more targeted question, and is the one most directly tied to the trajectory-locking analysis. We filter each side to its correct samples and keep a comparison only when both sides contain at least one correct solution. Relative to the base model, TraFL is preferred on both MATH-500 ($p = 0.006$) and LiveCodeBench ($p = 4.1 \times 10^{-5}$). Relative to ESPO, the same holds: 14.3% vs. 5.6% on MATH-500 ($p = 0.001$) and 38.3% vs. 26.4% on LiveCodeBench ($p = 7.8 \times 10^{-4}$).

I LLM-AS-A-JUDGE DIVERSITY PROMPT

We use the following prompt for the LLM-as-a-judge diversity evaluation. The prompt is designed to isolate differences in solution strategy, rather than correctness or surface-level variation. For each comparison, `{problem}`, `{set_a}`, and `{set_b}` are replaced with the problem statement and the two sampled answer sets being compared.

LLM-as-a-Judge Prompt for Diversity Evaluation

[Judge Role]
You are an expert judge evaluating diversity among LLM-generated answers to the same problem.
Your goal is to assess approach diversity, not correctness or answer quality.

[Problem]
`{problem}`

[Set A]
`{set_a}`

[Set B]
`{set_b}`

[Definition of Distinct Approach]
An answer represents a distinct approach only if it uses a recognizably different core method, such as a different algorithm, data structure, mathematical reduction, equation setup, case analysis, proof strategy, or reasoning path.

[Do Not Count as Diversity]
– different wording, formatting, verbosity, or variable names
– different final answers from the same method
– minor bugs in the same method
– random guesses or incoherent reasoning
– extra explanation that does not change the core method

[Instructions]
1. Identify the distinct approach types in each set.
2. Compare the number and substance of these approach types.
3. Prefer TIE if the difference is mostly superficial or unclear.

[Response Format]
Return exactly this JSON:

```
{{
  "set_a_approaches": ["short names of distinct approaches"],
  "set_b_approaches": ["short names of distinct approaches"],
  "winner": "A" | "B" | "TIE",
  "confidence": "low" | "medium" | "high",
  "reason": "one concise sentence"
}}
```

J Training Compute

The post-training experiments can be reproduced on any 8 card modern parallel computing unit (NPUs, TPUs, GPUs) with at least 80 GB memory. Each math post-training experiment takes approximately 10 hours, and the code post-training experiment takes approximately 18 hours.

K Dataset and Model Licenses

We use publicly released datasets, benchmarks, and models under their respective licenses. GSM8K (Cobbe et al., 2021) is released under the MIT License. MATH (Lightman et al., 2023) is released under the MIT License. HumanEval (Chen et al., 2021) is released under the MIT License. MBPP (Austin et al., 2021) is released under CC BY 4.0. AceCode-89K (Zeng et al., 2025) is released under the MIT License. KodCode-Light-RL-10K (Xu et al., 2025) is released under CC BY-NC 4.0 and is used only for non-commercial research. Minerva Math (Lewkowycz et al., 2022) is listed under the MIT License in the public release we use. LiveCodeBench (Jain et al., 2025) release v5 is released under the Creative Commons license family and is used only for evaluation. LLaDA-8B-Instruct (Nie et al., 2025) is released under the MIT License. We cite all original dataset and model sources and follow their usage protocols.