LEARNING UNMASKING POLICIES FOR DIFFUSION LANGUAGE MODELS

Metod Jazbec[1]* **Theo X. Olausson**[2]* **Louis Béthune**[3] **Pierre Ablin**[3]
Michael Kirchhof[3] **João Monteiro**[3] **Victor Turrisi**[3] **Jason Ramapuram**[3] **Marco Cuturi**[3]

[1]University of Amsterdam [2]Massachusetts Institute of Technology [3]Apple

ABSTRACT

Diffusion (Large) Language Models (dLLMs) now match the downstream performance of their autoregressive counterparts on many tasks, while holding the promise of being more efficient during inference. One critical design aspect of dLLMs is the *sampling procedure* that selects which tokens to unmask at each diffusion step. Indeed, recent work has found that heuristic strategies such as confidence thresholding improve both sample quality and token throughput compared to random unmasking. However, such heuristics have downsides: they require manual tuning, and we observe that their performance degrades with larger block sizes. In this work, we instead propose to train sampling procedures using reinforcement learning. Specifically, we formalize masked diffusion sampling as a Markov decision process in which the dLLM serves as the environment, and propose a lightweight policy based on a single-layer transformer that maps dLLM token confidences to unmasking decisions. Our experiments show that these trained policies match the performance of heuristics when combined with semi-autoregressive (block) generation, while outperforming them in the full-diffusion setting.

1 INTRODUCTION

Discrete diffusion (Austin et al., 2021a; Hoogeboom et al., 2021; Dieleman et al., 2022; Lou et al., 2024; Shi et al., 2024) has recently emerged as a compelling alternative to the predominant autoregressive (AR) modeling paradigm in large language models (LLMs). Unlike AR models, which generate the next token in a left-to-right fashion (Radford et al., 2018), diffusion LLMs (dLLMs) generate text by learning to reverse a noising process that progressively corrupts token sequences. In particular, masked diffusion models (MDMs; Sahoo et al. 2024; Ou et al. 2025), a subclass of discrete diffusion models that use time-dependent BERT-style (Devlin et al., 2019) masking as the forward noising process, have recently demonstrated impressive performance, with models like LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) matching the performance of similarly sized autoregressive LLMs.

At generation time, MDMs begin with a fully masked sequence and iteratively unmask a fixed number of tokens at randomly sampled positions in each sampling step. As a result, they offer the potential for faster inference, as they can, in principle, generate multiple tokens in parallel using a single model call. Despite this promise, open-source dLLMs have, until recently, lagged behind their AR counterparts in terms of inference efficiency. This changed with Fast-dLLM (Wu et al., 2025), which demonstrated that a dLLM like LLaDA (Nie et al., 2025) can achieve higher token throughput than similarly sized LLaMA models (Dubey et al., 2024) while maintaining competitive performance. A key component of Fast-dLLM lies in its sampling heuristic: instead of unmasking a fixed number of randomly sampled token positions, it proposes to unmask all tokens whose confidences exceed a pre-specified threshold. This success has since inspired the development of increasingly sophisticated sampling heuristics (see Appendix A for a comprehensive overview), further advancing the state of the art. These heuristics can, however, be difficult to tune, and seem to work best when imposing semi-autoregressive (semi-AR) generation, in which small blocks of tokens are unmasked sequentially

*Equal contribution. Work done while MJ and TXO were interns at Apple. Correspondence to m.jazbec@uva.nl, theoxo@mit.edu, cuturi@apple.com.

(Arriola et al., 2025a; Nie et al., 2025). We posit that such limitations are difficult to avoid with heuristics alone, since they are essentially handcrafted solutions to a sequential decision making problem: *What is the optimal order in which to unmask the sequence of tokens, in order to strike a good balance between correctnes and efficiency?*

We address this question by moving beyond heuristics, instead proposing a learning-based approach that leverages a transformer-based unmasking policy trained via reinforcement learning (RL). This approach is motivated by the above observation that unmasking in dLLMs can be viewed as a (Markovian) sequential decision making problem, and follows recent successes in RL for post-training of language models. However, in contrast to recent works that use RL to improve the reasoning abilities of dLLMs (e.g., Zhao et al. 2025b), our goal is not to improve the capacity of the underlying model, but rather to use RL to automate the discovery of adaptive sampling strategies. We therefore treat the underlying dLLM as the *environment* in which to act, not the policy, leaving it unchanged and instead parameterizing a policy as a lightweight stand-alone network (Figure 5). Empirically, we demonstrate that our learned sampling policies match the performance of state-of-the-art heuristic samplers (Wu et al., 2025) in standard semi-AR generation, while also surpassing heuristics in full-diffusion setting. In summary, we make the following contributions:

- We formalize sampling in dLLMs as a Markov decision process (MDP) (Section 3.1), propose a lightweight design for the unmasking policy (Section 3.2), and outline our RL training pipeline based on group relative policy optimization (GRPO; Shao et al. 2024) (Section 3.3).
- Our experiments show that learned policies with our RL framework match the performance of heuristic samplers such as Fast-dLLM (Section 4.1), while also addressing some of the challenges heuristic samplers face outside the semi-AR regime (Section 4.2).
- We study the transferability of our learned samplers across different models and data domains (Section 4.3), and provide detailed insights into stabilizing RL training and policy design (Section D). We also demonstrate that policies exhibit qualitatively different unmasking behavior compared to confidence heuristics (Appendix E).

2 Background

Throughout the paper, we use the notation $[L]$ to represent the set $\{1, \ldots, L\}$. We denote a sequence of tokens as $\boldsymbol{x} = (x^1, \ldots, x^d) \in \mathcal{V}^d$, where d is the sequence length and $\mathcal{V} := [V]$ is the vocabulary. Given our focus on MDMs, we assume the vocabulary includes a special mask token $\boldsymbol{M}$.

2.1 Masked Diffusion Models

We focus on masked diffusion models (MDMs), deferring a more general introduction to discrete diffusion to Appendix I.

Training. MDMs learn to generate data by training a BERT-style (Devlin et al., 2019) masked predictor to reverse the forward noising process. Concretely, given a training sample $\boldsymbol{x}_0 \sim p_{\text{data}}$, the forward process corrupts each token independently by setting it to the mask token with probability proportional to the diffusion timestep $t \in [0, 1]$:

$$p_t(\boldsymbol{x}_t \mid \boldsymbol{x}_0) := \prod_{k=1}^{d} p_t(x_t^k \mid \boldsymbol{x}_0)\,, \text{where } p_t(x_t^k \mid \boldsymbol{x}_0) := \begin{cases} 1-t, & \text{if } x_t^k = x_0^k, \\ t, & \text{if } x_t^k = \boldsymbol{M}, \\ 0, & \text{otherwise.} \end{cases}$$

Recent work has shown that an MDM parametrized by θ can learn the reverse process by maximizing the following evidence lower bound (ELBO) (Ou et al., 2025), $\mathcal{L}(\theta) \leq \mathbb{E}_{\boldsymbol{x}_0 \sim p_{\text{data}}}[\log p_\theta(\boldsymbol{x}_0)]$, denoting $\mathbf{1}[\cdot]$ the indicator function:

$$\mathcal{L}(\theta) := \mathbb{E}_{t \sim U[0,1],\, \boldsymbol{x}_0 \sim p_{\text{data}},\, \boldsymbol{x}_t \sim p_t(\boldsymbol{x}_t \mid \boldsymbol{x}_0)} \left[\frac{1}{t} \sum_{k=1}^{d} \mathbf{1}[x_t^k = \boldsymbol{M}] \, \log p_\theta(x_0^k \mid \boldsymbol{x}_t) \right]. \tag{1}$$

Generation. When generating an answer for a given prompt $\boldsymbol{x}$, MDMs start with a sequence of all-masked tokens $\boldsymbol{y}_T := (\boldsymbol{M}, \ldots, \boldsymbol{M})$, where $L := |\boldsymbol{y}_T|$ is a pre-specified maximum answer length. Then, in each sampling step $t \in [T]$, the MDM outputs token distributions $p_t^k := p_\theta(y_0^k = \cdot \mid \boldsymbol{x}, \boldsymbol{y}_t)$ for all token positions $k \in [L]$ and a sampling strategy decides which subset $\mathcal{U}_t \subseteq \mathcal{M}_t$ of the

still-masked positions $\mathcal{M}_t := \{k \in [L] \mid y_t^k = \boldsymbol{M}\}$ to unmask. A new, partially unmasked/denoised answer $\boldsymbol{y}_{t-1}$ is obtained via

$$y_{t-1}^k := \begin{cases} y \sim p_t^k, & \text{if } k \in \mathcal{U}_t, \\ y_t^k, & \text{otherwise.} \end{cases} \tag{2}$$

The stochasticity of sampling $y \sim p_t^k$ depends on the dLLM temperature τ, with higher temperatures leading to more diverse (but often lower quality) generations. When evaluating dLLMs, it is common to set $\tau = 0$ (Nie et al., 2025; Wu et al., 2025), resembling greedy decoding. The choice of sampling/unmasking strategy is also important, since different choices will produce different unmasking sets $\mathcal{U}_t$, thus affecting both sampling quality and efficiency. This has drawn considerable attention to the development of optimal sampling techniques for dLLMs (see Appendix A).

2.2 Heuristic Samplers

One popular approach to decide which positions $\mathcal{U}_t$ to unmask at each timestep is to construct a heuristic that leverages uncertainty measures derived from the predicted token distributions p_t^k (Wu et al. 2025; Ben-Hamu et al. 2025; *inter alia*). In particular, recent work has shown substantial efficiency gains with heuristic strategies that employ the *confidence* $c_t^k := \max_{v \in \mathcal{V}} p_t^k(v)$ of the underlying dLLM in order to make their decisions. Two representative methods within this space are *high-confidence* unmasking (Chang et al., 2022), in which each forward pass unmasks a fixed number of tokens (K) with the highest confidences,

$$\mathcal{U}_t^K := \left\{ \underset{I \subseteq \mathcal{M}_t,\ |I|=K}{\arg\max} \sum_{k \in I} c_t^k \right\}$$

as well as the confidence-thresholding strategy of Fast-dLLM (Wu et al., 2025), which allows for a variable number of tokens to be unmasked at each step by comparing the confidences to a fixed threshold λ:

$$\mathcal{U}_t^\lambda := \{k \in \mathcal{M}_t \mid c_t^k > \lambda\}.$$

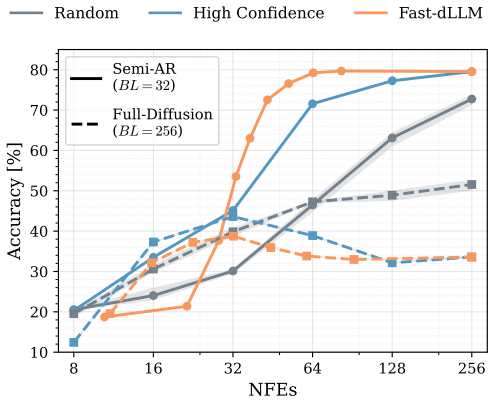


Figure 1: LLaDA-8B-Instruct (Nie et al., 2025) on GSM8k, with semi-AR generation ($BL = 32$; **—**) and without (full-diffusion regime, $BL = L = 256$; **- -**). More datasets and models in Appendix C.1. Generation speed is measured in network function evaluations (NFEs), which corresponds to the number of sampling steps.

Despite the undeniable success of confidence-based samplers, relying on handcrafted solutions comes with drawbacks. Beyond the obvious need for careful design (e.g., selecting an appropriate confidence measure or threshold λ), we also found that these methods often exhibit high sensitivity to the exact sampling configuration. For instance, many heuristics rely on semi-autoregressive (semi-AR) generation (Arriola et al., 2025a; Nie et al., 2025), where diffusion decoding proceeds one "block" of contiguous tokens at a time, with block length BL serving as yet another hyperparameter. As shown in Figure 1, outside this semi-AR regime, the performance of confidence-based heuristics can degrade below that of a random unmasking, and no longer benefit from increased compute (i.e., higher numbers of function evaluations, NFEs). These limitations raise the question of whether effective sampling methods can be *learned* rather than manually tuned.

3 Learning Unmasking Policies

We now introduce our approach to learn sampling in dLLMs via reinforcement learning (RL). We begin by formulating sampling as a Markov decision process (MDP) (Section 3.1), then propose a sampling policy (Section 3.2), and finally provide details on our RL training (Section 3.3).

3.1 dLLM Sampling as a Markov Decision Process

To facilitate the development of RL-based samplers, we describe first the Markov decision process (MDP) (Sutton et al., 1998) for sampling in dLLMs. To stay aligned with the MDM notation in

Section 2.1, we reverse the time in our MDP formulation: $t = T, T-1, \ldots, 1$, with T denoting the maximum horizon (number of sampling steps).

- The *state* $(\boldsymbol{x}, \boldsymbol{y}_t)$ holds a prompt $\boldsymbol{x} \in \mathcal{V}^d$ and the current (partially masked) generation $\boldsymbol{y}_t \in \mathcal{V}^L$. For brevity, we omit $\boldsymbol{x}$ from the state notation unless needed. The initial state has a fully masked generation: $\boldsymbol{y}_T = (\boldsymbol{M}, \ldots, \boldsymbol{M})$.
- An *action* $\boldsymbol{u}_t \in \{0,1\}^L$ is a vector of unmasking decisions indicating which positions have been selected to be unmasked in the next transition step. These actions are chosen according to the policy π_ϕ: $\boldsymbol{u}_t \sim \pi_\phi(\cdot \mid \boldsymbol{y}_t)$ (introduced later in Section 3.2).
- The *transition* $\boldsymbol{y}_{t-1} \sim P(\boldsymbol{y}_{t-1} \mid \boldsymbol{y}_t, \boldsymbol{u}_t; \tau)$ corresponds to standard sampling in dLLMs (cf. Equation (2)), with the action $\boldsymbol{u}_t$ determining which tokens get unmasked: $\mathcal{U}_t^\pi := \{k \in \mathcal{M}_t \mid u_t^k = 1\}$.
- The *reward* $R(\boldsymbol{y}, \boldsymbol{y}_t)$ is provided only at the final generation step, which corresponds to the first timestep where all tokens have been unmasked $\hat{T} := \max\{t \in [T] \mid y_t^k \neq \boldsymbol{M}, \forall k \in [L]\}$. To learn useful samplers, the reward should promote both correctness (i.e., the generated answer $\boldsymbol{y}_{\hat{T}}$ being 'close' to the reference answer $\boldsymbol{y}$) and efficiency (i.e., minimizing the number of steps $T - \hat{T}$). Note that the reward depends on action $\boldsymbol{u}_t$, and therefore on the policy π_ϕ, implicitly through its influence on the generations $\boldsymbol{y}_t$ and the number of steps. We defer our concrete choice of the reward function to Section 3.3.

With the MDP above, finding the optimal sampling becomes a standard RL problem of finding a policy π_ϕ parametrized by ϕ that maximizes the expected reward:

$$\max_\phi \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_{\text{data}},\, \boldsymbol{u}_t \sim \pi_\phi,\, \boldsymbol{y}_t \sim P} \left[\sum_{t=1}^{T} R(\boldsymbol{y}, \boldsymbol{y}_t) \right].$$

3.2 Lightweight Confidence Policy Design

Having outlined the MDP above, we next describe our proposed implementation for the sampling policy π_ϕ (see Appendix B for policy sampling diagram and algorithm).

Confidence-based input. Recall that a state consists of a partially masked token sequence $\boldsymbol{y}_t$. To avoid constructing policies that operate at the token level, and thereby minimize computational overhead, we rely on the vector of token confidences $\boldsymbol{c}_t := (c_t^1, \ldots, c_t^L)$, which are readily available since the token-level predictive distributions p_t^k are computed at each transition step $\boldsymbol{y}_{t-1} \sim P$ anyways. Our choice is further motivated by the aforementioned heuristic methods that primarily operate on confidences (Wu et al., 2025), and also based on our own ablations (see Section D). Specifically, we observed that alternatively relying on dLLM's hidden states required significantly larger policy models without consistently improving performance.

Lightweight transformer. We introduce a small, learnable neural network f_ϕ that maps the vector of confidences $\boldsymbol{c}_t$ to a vector of unmasking scores (logits) $\boldsymbol{b}_t = f_\phi(\boldsymbol{c}_t, \boldsymbol{m}_t, t)$ with $\boldsymbol{b}_t \in \mathbb{R}^L$. We additionally include a binary mask vector $\boldsymbol{m}_t := (m_t^1, \ldots, m_t^L)$, where $m_t^k := \mathbf{1}[k \in \mathcal{M}_t]$ indicates whether position k is still masked, and inform the policy with the time index $t \in [T]$. In practice, f_ϕ is implemented as a lightweight transformer (see Section 4 for more details on the exact policy architecture). Notably, its size is less than 0.01% of the pretrained dLLMs used in our experiments, resulting in negligible computational overhead during sampling (see Figure 7).

Bernoulli likelihood. We sample the unmasking actions via $u_t^k \sim \text{Ber}\big(s_t^k\big)$ with $s_t^k := \sigma(b_t^k)$, where $\text{Ber}(\cdot)$ denotes the Bernoulli distribution and $\sigma(\cdot)$ is the sigmoid function. Conveniently, the policy likelihood $\pi_\phi(\boldsymbol{u}_t) := \prod_{k=1}^L (s_t^k)^{u_t^k} \cdot (1 - s_t^k)^{1-u_t^k}$ is readily available in closed form and does not require any additional approximations (unlike in the case of post-training dLLMs; Zhao et al. 2025b). We also considered an alternative formulation based on the Plackett-Luce model (Luce, 1959; Plackett, 1975) which we call *dynamic Plackett-Luce sampling* (DPLS; see Appendix G), but since our ablations showed comparable performance (Section D), we favor the Bernoulli formulation in our experiments for its simplicity.

At generation, we additionally propose to "temper" the Bernoulli probabilities as $s_t^k := \sigma(b_t^k / \tau_\pi)$. The temperature τ_π thus controls the sharpness of the policy distribution; we found that this can

sometimes serve as a useful test-time knob for trading off accuracy versus efficiency by making the policy more or less "decisive" (Section 4).

To ensure convergence when all sampled actions are zero ($\boldsymbol{u}_t = \mathbf{0}$), we unmask the position with the highest Bernoulli parameter s_t^k (similar to Fast-dLLM). We apply this fallback only at test time, as we found that forcing unmasking during training was prone to reward hacking. Instead, if not all tokens are unmasked within T steps during training, we simply terminate generation and evaluate the answer with some tokens left unmasked. Since such samples tend to get low or no reward, we find that the policies quickly learn not to leave any tokens unmasked without explicit instruction.

3.3 Learning π_ϕ with GRPO

To train our sampling policy, we adopt group relative policy optimization (GRPO) (Shao et al., 2024), a method recently popularized as a simpler and more scalable alternative to earlier policy gradient approaches such as PPO (Schulman et al., 2017). Specifically, for each prompt $\boldsymbol{x} \in \mathcal{D}$, we sample G trajectories of generations $\{\boldsymbol{y}_T^g, \ldots, \boldsymbol{y}_{\hat{T}_g}^g\}_{g=1}^G$ along with their corresponding unmasking decisions $\{\boldsymbol{u}_T^g, \ldots, \boldsymbol{u}_{\hat{T}_g}^g\}_{g=1}^G$. Importantly, we fix the dLLM sampling temperature τ to 0 (i.e., greedy decoding) to ensure that any variation among samples within a group arises solely from different unmasking actions. After computing rewards for each sample in the group, we define the advantage as $A_t^g := R(\boldsymbol{y}, \boldsymbol{y}_{\hat{T}_g}^g) - \frac{1}{G}\sum_{i=1}^G R(\boldsymbol{y}, \boldsymbol{y}_{\hat{T}_i}^i)$, following recent best practices that recommend omitting standard deviation normalization of advantages (Zhao et al., 2025b). Additionally, note that the reward at the final generation step $\hat{T}_g$ for each sample is propagated to all preceding timesteps t to provide a learning signal throughout the entire sampling process. Our final training objective is then

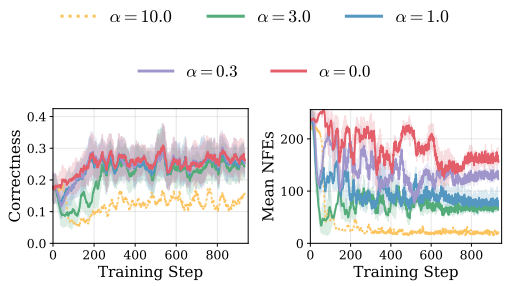


Figure 2: Correctness reward (rolling average, 20 steps) on GSM8k (*left*) and average number of sampling steps (*right*) during training of our policies for various values of α (cf. Equation (3)). Averaged over two random seeds, with shaded areas indicating (min, max); only one seed shown for $\alpha = 10.0$ (···) due to training instability.

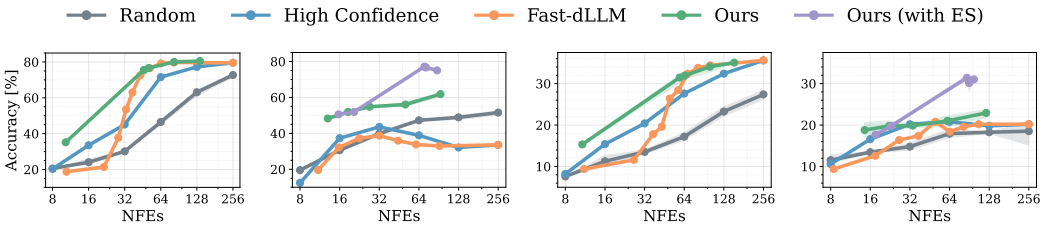
$$\mathcal{J}(\phi) := \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D},\, \{\boldsymbol{y}_{T:\hat{T}_g}^g\}_{g=1}^G \sim P_\theta,\, \{\boldsymbol{u}_{T:\hat{T}_g}^g\}_{g=1}^G \sim \pi_{\phi_{\text{old}}}} \Bigg[\frac{1}{G}\sum_{g=1}^G \frac{1}{T-\hat{T}_g} \sum_{t=\hat{T}_g}^T \min\left\{\rho_t^g \cdot A_t^g,\ \text{clip}(\rho_t^g, 1-\epsilon, 1+\epsilon)\cdot A_t^g\right\}\Bigg]$$

where π_ϕ is the current policy being updated, and $\pi_{\phi_{\text{old}}}$ refers to an earlier version of the policy used to generate the RL rollouts. The likelihood ratio $\rho_t^g := \frac{\pi_\phi(\boldsymbol{u}_t^g)}{\pi_{\phi_{\text{old}}}(\boldsymbol{u}_t^g)}$ serves as the importance sampling correction term, which together with clipping (via ϵ) aims to stabilize the off-policy training. Note that we exclude already-unmasked positions from the policy likelihood computation. Finally, since our policies are trained from scratch, we remove the KL regularization term in our GRPO objective.

Reward As mentioned in Section 3.1, we wish to obtain a policy that yields 'correct' generations while also being fast. To this end, we define our final *multiplicative* reward as

$$R(\boldsymbol{y}, \boldsymbol{y}_t) := \begin{cases} r(\boldsymbol{y}, \boldsymbol{y}_t)\cdot\left(1-\frac{T-t}{T}\right)^\alpha, & \text{if } t = \hat{T}, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $r(\boldsymbol{y}, \boldsymbol{y}_t)$ is a task-specific correctness term (e.g., a binary reward indicating whether the generated mathematical answer is correct). To encourage faster sampling, we incorporate a computational penalty based on the number of steps, $T - \hat{T}$, with $\alpha \geq 0$ serving as a hyperparameter that controls the trade-off between accuracy and speed. While we experimented with an additive penalty of the form $r(\boldsymbol{y}, \boldsymbol{y}_{\hat{T}}) - \alpha\left(\frac{T-\hat{T}}{T}\right)$ (Graves, 2016) we found that this led to problematic reward hacking: when all samples in a group are incorrect, as is the case early on when training from scratch, faster samples may still receive a positive advantage, despite being wrong (cf. Section D).



(a) GSM8k, $BL = 32$ (b) GSM8k, $BL = 256$ (c) MATH, $BL = 32$ (d) MATH, $BL = 256$

Figure 3: Results for LLaDA in semi-AR (Figure 3a & Figure 3c) and full-diffusion (Figure 3b & Figure 3d) generation regimes ($L = 256$). Results for Dream-7B are provided in Figure 8. For our policies we vary $\alpha \in \{10, 3, 1, 0.3, 0\}$ and use $\tau_\pi = 0.5$ for $BL = 32$ and $\tau_\pi = 1$ for $BL = 256$. Expert steering (ES) described in detail in Appendix H. Wall-clock time plots shown in Figure 7.

4 Experiments

We begin by verifying that the learned policies match the performance of confidence-based heuristics (Section 4.1). We then highlight the potential of RL sampling to outperform heuristics in the full-diffusion setting (Section 4.2), and examine the transferability of RL policies across models, datasets, and generation lengths (Section 4.3). Next, we explore different ways to instantiate the MDP and analyze their effects on performance (Section D). Finally, we study the unmasking behavior of RL policies, highlighting their qualitative differences to heuristics (details in Appendix E).

4.1 Learning Effective Sampling via RL

We start by demonstrating that our proposed RL framework yields effective sampling strategies in dLLMs, where effectiveness is defined in terms of both accuracy and speed.

Experiment details. We use LLaDA-8B-Instruct (Nie et al., 2025) as the base dLLM (additional results on Dream-7B-Instruct (Ye et al., 2025) provided in Appendix C.3). We parametrize the policy network f_ϕ as a shallow (single-layer) transformer incorporating adaptive layer normalization for conditioning; hyperparameter details for the architecture are provided in Appendix J and an architecture diagram can be found in Appendix K. We train five different policies, corresponding to $\alpha \in \{10, 3, 1, 0.3, 0\}$. Each policy is trained semi-autoregressively at $BL = 32$ on a single epoch of mixture data, sampled proportionally from the training sets of GSM8k (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021), resulting in roughly 15,000 training samples. Figure 2 shows the training dynamics; as expected, higher α generally gives rise to faster policies.

We then compare the resulting policies on the test sets of both GSM8k and MATH to the confidence-based heuristics introduced in Section 2.2, picking one out of two training seeds based on the final training loss and averaging over three test-time seeds. To obtain Pareto frontiers for the heuristic methods, we use $K \in \{8, 16, 32, 64, 128, 256\}$ for the random baseline and high-confidence unmasking, while for Fast-dLLM we use $\lambda \in \{0.1, 0.2, \ldots, 1.0\}$ throughout. For all methods, we use the standard greedy decoding setting ($\tau = 0$) when generating test answers.

Results for (Short) $BL = 32$, Figure 3a and Figure 3c. Generally, we observe that the performance of the learned policies (—) exceeds those of both the random baseline (—) as well as the high-confidence sampling (—), while matching Fast-dLLM (—). That our learned policies do not surpass Fast-dLLM in the mid-to-high NFEs range may suggest that this heuristic is near-optimal under semi-AR generation. Interestingly, despite very similar quantitative performance, we find that the learned policies exhibit qualitatively somewhat different unmasking behavior compared to Fast-dLLM (see Appendix E). Concretely, the policy unmasks adjacent tokens much less frequently (Figure 19) and allocates compute more uniformly across blocks (Figure 18a).

We note that the effect of α in training (cf. Figure 2) carries over to the test set: policies trained with higher values of α exhibit faster, yet less accurate, sampling. Of particular note is $\alpha = 10.0$, which exhibited greater training instability due to the very sharp slope of the reward. When successfully trained, this yields a very fast policy that outperforms Fast-dLLM in the low-NFE (~ 10) regime on both datasets. Qualitatively, we find evidence that this improved performance is likely due to the

policy learning to slow down in the final block when generating a numerical answer, as shown in Figure 18b (see also Appendix E for more details). This highlights the potential of RL-based policies when optimizing for maximal efficiency.

However, RL policies appear to exhibit less *controllability*, as varying α results in a less smooth traversal of the Pareto frontier than varying the confidence threshold λ. For example, the behavior of the $\alpha = 3.0$ policy is nearly identical to that of $\alpha = 1.0$, despite the reward incorporating a computational penalty that scales exponentially with α. Furthermore, we find that this cannot easily be remedied by using a tighter α grid. In Appendix C.4 we let α range in $\{10.0, 9.0, ...1.0, 0.3, 0.0\}$, but observe that using $\alpha \geq 4.0$ consistently results in converging to a policy either roughly equivalent to that of $\alpha = 3.0$ or to that of $\alpha = 10.0$, with no value of α successfully yielding a policy which interpolates between the two extremes. Besides α, we observe that one can make small changes to the trade-off between compute and performance by varying the policy temperature τ_π at test time; we detail the impact of this parameter in Figure 11.

4.2 Beyond Semi-Autoregressive Decoding

As mentioned in Section 2.2, heuristic samplers often rely on semi-AR generation to achieve good performance. While not typically thought of as problematic in textual domains where strong AR dependencies exist, semi-AR approaches can only partially fulfill the promise of fully parallel generation by restricting the unmasking to the current block. Hence, we train here policies without semi-AR generation, targeting the full-diffusion task ($BL = L = 256$) at both training and test time. We keep all other implementation details identical to those in the previous section.

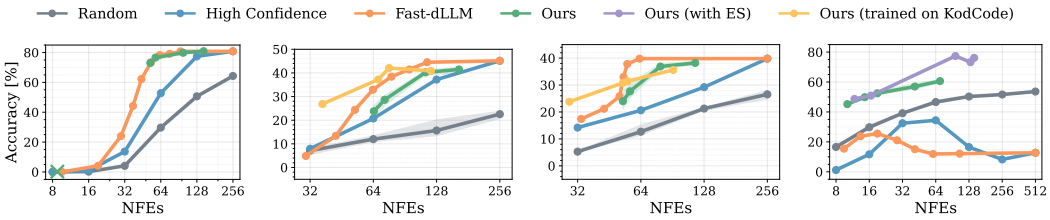
Results for (Long) $BL = 256$, Figure 3b & Figure 3d. Although all sampling strategies experience a performance drop relative to the semi-AR setting (cf. Figure 3a & Figure 3c), we find that the policies produced by RL (—) exhibit the smallest decline and consequently achieve the best overall performance. Furthermore, the results suggest that this methodology is able to produce policies which achieve solid performance even in the low-NFE regime. In particular, the learned policies obtain $\sim 50\%$ accuracy at ~ 12 NFEs on GSM8k (compared to $\leq 30\%$ for the heuristic methods regardless of semi-AR use), highlighting the potential of non-semi-AR sampling for achieving maximal efficiency gains. Note however that, in theory, an RL policy trained with $BL = L$ could learn to emulate semi-AR sampling on its own. Thus, the fact that policies trained with $BL = L$ underperform those trained with $BL = 32$ in the mid-high NFE range (comparing, for example, Figure 3a to Figure 3b) suggests that the policies remain far from optimal.

One hypothesis for why this is the case is that our training procedure is not encouraging sufficient exploration, instead converging to locally optimal policies. To address this, we encourage further exploration by using samples generated via semi-AR sampling from Fast-dLLM. We refer to this approach as *expert steering*, and describe it in more detail in Appendix H. As shown in Figure 3, with expert steering (—) RL is able to discover policies that almost close the performance gap to the best accuracy achieved in the semi-AR setting at mid-to-high NFEs (e.g., $\sim 80\%$ on GSM8k and $\sim 35\%$ on MATH), while mostly retaining their strong performance in the low-NFE regime. However, we do find that expert steering introduces significant instability during training, further reducing the controllability through α, with multiple values of α collapsing to near-identical policies. We leave further investigations into stabilizing expert steering training for future work. Finally, we observe that policy sampling and Fast-dLLM induce markedly different unmasking orders in the full-diffusion setting (see Figure 22 and Appendix E). While Fast-dLLM exhibits reverse (right-to-left) unmasking, the policy with expert steering learns to overcome this behavior and unmasks earlier positions first on average (left-to-right), which likely explains its superior performance.

4.3 Transferability of RL Sampling Policies

We next turn to the question of transferability. A notable advantage of heuristic approaches is that they can be applied *post-hoc* to any model or dataset.This naturally raises the question: to what extent can RL policies trained on a specific model and dataset be reused in other settings?

Model transfer: LLaDA $\rightarrow$ Dream. We begin by investigating the transferability of RL policies across models; note that such transfers are possible because our policies rely solely on token confidences and are therefore agnostic to token embeddings, which would not transfer from one model to another. We thus reuse the policies trained on LLaDA (cf. Section 4.1) and evaluate them on Dream; the results are shown in Figure 4a (see Figure 12 for MATH). Encouragingly, most of



(a) Model transfer: LLaDA → Dream on GSM8k. (b) Domain transfer: math → coding (HumanEval). (c) Domain transfer: math → coding (MBPP). (d) Sequence length transfer: 256 → 512 (GSM8k).

Figure 4: Results for the transferability experiments. Note that in **(a)**, the $\alpha = 10$ policy is represented separately by a (✕) in the lower left to avoid misleading visualization when interpolating to $\alpha = 3$. For results on coding datasets in **(b)** and **(c)**, we omit the low-NFE regime, as all approaches degrade to near-zero performance in this setting.

the LLaDA-trained policies nearly match the performance of Fast-dLLM when evaluated on Dream, and perform very similarly to those trained on Dream directly (Figure 8). The one exception is the $\alpha = 10$ policy (✕), which collapses to Fast-dLLM performance and fails to retain the good low-NFE performance observed with LLaDA (cf. Figure 3a)—suggesting that the steepness of the reward curve (due to high α) causes this policy to overfit to model-specific patterns in LLaDA's confidence.

Domain transfer: Mathematical reasoning to code. Next, we examine how well our policies transfer across different domains. We again reuse the policies from Section 4.1, which were trained on a mixture of mathematical data (GSM8k/MATH), but this time evaluate them on the coding tasks of HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b). As shown in Figure 4b and Figure 4c (—), we find that these policies fail to fully transfer to the coding domains, especially on HumanEval. To investigate whether this drop in performance is due to the lack of domain-relevant training data, we train a new policy on the coding dataset KodCode-RL-10K (Xu et al., 2025) (—). These coding policies narrow the gap to the Fast-dLLM baseline on HumanEval and improve low-NFE performance on MBPP, underscoring the importance of using a diverse data mixture to support generalization across domains.

Sequence length generalization: $L = 256 \to L = 512$. Lastly, we examine whether our policies transfer across different sequence lengths L (not to be confused with the block length BL in semi-AR). Such transfer is possible since we instantiate f_θ with a transformer, rather than, for example, an MLP (which requires fixed-length inputs). We thus take the policies from Section 4.2 and evaluate them at a 2x longer sequence length ($L = 512$). Results are shown in Figure 4d (see Figure 13 for MATH). While the baselines degrade further as the L increases, the policies yield similar performance to before, suggesting that RL policies can transfer effectively across generation lengths without retraining.

5 Conclusion

We introduced a reinforcement learning approach for learning unmasking strategies in diffusion LLMs. Our experiments demonstrated that the learned policies can match or even exceed the performance of recently proposed sampling heuristics, paving the way for the automated discovery of scalable and robust sampling mechanisms.

Limitations and future work. One notable limitation of our approach is that it requires training a separate policy for each α, and that this hyper-parameter sometimes does not yield as much fine-grained control over the policy's behavior as desired (Section 4.1). Thus, an important future direction is to investigate whether the accuracy-speed tradeoff could be controlled through some alternative, ideally test-time, mechanism. We also observe that the optimal policy temperature τ_π varies across generation settings (Figure 11), suggesting the need to explore whether τ_π can be learned jointly with the policy. Beyond overcoming these limitations, promising future extensions are (i) expanding the training mixture to incorporate data from multiple domains; (ii) extending our sampling policies to also support remasking (Wang et al., 2025b; Huang et al., 2025c), and (iii) moving beyond text to learn samplers for multimodal discrete diffusion (Swerdlow et al., 2025; Zhou et al., 2025).

REFERENCES

Daman Arora and Andrea Zanette. Training language models to reason efficiently. *NeurIPS*, 2025.

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

Marianne Arriola, Yair Schiff, Hao Phung, Aaron Gokaslan, and Volodymyr Kuleshov. Encoder-decoder diffusion language models for efficient training and inference. *NeurIPS*, 2025b.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *NeurIPS*, 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:2108.07732*, 2021b.

Iskander Azangulov, Teodora Pandeva, Niranjani Prasad, Javier Zazo, and Sushrut Karmalkar. Parallel sampling from masked diffusion models via conditional independence testing. *arXiv:2510.21961*, 2025.

Parikshit Bansal and Sujay Sanghavi. Enabling approximate joint sampling in diffusion lms. *arXiv:2509.22738*, 2025.

Wenrui Bao, Zhiben Chen, Dan Xu, and Yuzhang Shang. Learning to parallel: Accelerating diffusion large language models via adaptive parallel decoding. *arXiv:2509.25188*, 2025.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *NeurIPS*, 2025.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv:1511.06297*, 2015.

Andrew Campbell, Valentin De Bortoli, Jiaxin Shi, and Arnaud Doucet. Self-speculative masked diffusions. *arXiv:2510.03929*, 2025.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *CVPR*, 2022.

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.

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

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:2110.14168*, 2021.

Muzhi Dai, Chenxu Yang, and Qingyi Si. S-grpo: Early exit via reinforcement learning in reasoning models. *NeurIPS*, 2025.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, et al. Continuous diffusion for categorical data. *arXiv:2211.15089*, 2022.

Yihong Dong, Zhaoyu Ma, Xue Jiang, Zhiyuan Fan, Jiaru Qian, Yongmin Li, Jianha Xiao, Zhi Jin, Rongyu Cao, Binhua Li, et al. Saber: An efficient sampling with adaptive acceleration and backtracking enhanced remasking for diffusion language model. *arXiv:2510.18165*, 2025.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, 2024.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv:2506.20639*, 2025.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv:1603.08983*, 2016.

Gabe Guo and Stefano Ermon. Reviving any-subset autoregressive models with principled parallel sampling and speculative decoding. *arXiv:2504.20456*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Chunsan Hong, Seonho An, Min-Soo Kim, and Jong Chul Ye. Improving discrete diffusion unmasking policies beyond explicit reference policies. *arXiv:2510.05725*, 2025a.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms. *arXiv:2507.18578*, 2025b.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *NeurIPS*, 2021.

Pengcheng Huang, Shuhao Liu, Zhenghao Liu, Yukun Yan, Shuo Wang, Zulong Chen, and Tong Xiao. Pc-sampler: Position-aware calibration of decoding bias in masked diffusion models. *arXiv:2508.13021*, 2025a.

Zemin Huang, Zhiyang Chen, Zijun Wang, Tiancheng Li, and Guo-Jun Qi. Reinforcing the diffusion chain of lateral thought with diffusion language models. *NeurIPS*, 2025b.

Zemin Huang, Yuhang Wang, Zhiyang Chen, and Guo-Jun Qi. Don't settle too early: Self-reflective remasking for diffusion language models. *arXiv preprint arXiv:2509.23653*, 2025c.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. *NeurIPS*, 2025.

Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, and Xu Yang. d^2 cache: Accelerating diffusion-based llms via dual adaptive caching. *arXiv:2509.23094*, 2025.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *ICML*, 2025a.

Seo Hyun Kim, Sunwoo Hong, Hojung Jung, Youngrok Park, and Se-Young Yun. Klass: Kl-guided fast inference in masked diffusion models. *NeurIPS*, 2025b.

Diederik P. Kingma and Ruiqi Gao. Understanding diffusion objectives as the ELBO with simple data augmentation. In *NeurIPS*, 2023.

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

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. Diffusion language models know the answer before decoding. *arXiv:2508.19982*, 2025b.

Nianyi Lin, Jiajie Zhang, Lei Hou, and Juanzi Li. Boundary-guided policy optimization for memory-efficient rl of diffusion large language models. *arXiv:2510.11683*, 2025.

Sulin Liu, Juno Nam, Andrew Campbell, Hannes Stärk, Yilun Xu, Tommi Jaakkola, and Rafael Gómez-Bombarelli. Think while you generate: Discrete diffusion with planned denoising. *ICLR*, 2025.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *ICML*, 2024.

R Duncan Luce. *Individual choice behavior*, volume 4. Wiley New York, 1959.

Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, et al. dinfer: An efficient inference framework for diffusion language models. *arXiv:2510.08666*, 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. *NeurIPS*, 2025.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *ICLR*, 2025.

Robin L Plackett. The analysis of permutations. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 24(2), 1975.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Kevin Rojas, Jiahe Lin, Kashif Rasul, Anderson Schneider, Yuriy Nevmyvaka, Molei Tao, and Wei Deng. Improving reasoning for diffusion language models via group diffusion policy optimization. *arXiv:2510.08554*, 2025.

Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *NeurIPS*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv:2402.03300*, 2024.

Jucheng Shen, Gaurav Sarkar, Yeonju Ro, Sharath Nittur Sridhar, Zhangyang Wang, Aditya Akella, and Souvik Kundu. Improving the throughput of diffusion-based large language models via a training-free confidence-aware calibration. *arXiv:2512.07173*, 2025.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data. In *NeurIPS*, 2024.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv:2508.02193*, 2025.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

Alexander Swerdlow, Mihir Prabhudesai, Siddharth Gandhi, Deepak Pathak, and Katerina Fragkiadaki. Unified multimodal discrete diffusion. *arXiv:2503.20853*, 2025.

Xiaohang Tang, Rares Dolga, Sangwoong Yoon, and Ilija Bogunovic. wd1: Weighted policy optimization for reasoning in diffusion language models. *arXiv:2507.08838*, 2025.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *International conference on pattern recognition (ICPR)*. IEEE, 2016.

Chengyu Wang, Paria Rashidinejad, DiJia Su, Song Jiang, Sid Wang, Siyan Zhao, Cai Zhou, Shannon Zejiang Shen, Feiyu Chen, Tommi Jaakkola, et al. Spg: Sandwiched policy gradient for masked diffusion language models. *arXiv:2510.09541*, 2025a.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. *NeurIPS*, 2025b.

Wen Wang, Bozhen Fang, Chenchen Jing, Yongliang Shen, Yangyi Shen, Qiuyu Wang, Hao Ouyang, Hao Chen, and Chunhua Shen. Time is a feature: Exploiting temporal dynamics in diffusion language models. *arXiv:2508.09138*, 2025c.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, 2018.

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

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv:2509.06949*, 2025e.

Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Dongrui Liu, and Linfeng Zhang. Accelerating diffusion large language models with slowfast: The three golden principles. *arXiv:2506.10848*, 2025.

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

Shutong Wu and Jiawei Zhang. Free draft-and-verification: Toward lossless parallel decoding for diffusion large language models. *arXiv:2510.00294*, 2025.

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

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

Jingyang Yi, Jiazheng Wang, and Sida Li. Shorterbetter: Guiding reasoning models to find optimal inference length for efficient reasoning. *NeurIPS*, 2025.

Runpeng Yu, Xinyin Ma, and Xinchao Wang. Dimple: Discrete diffusion multimodal large language model with parallel decoding. *arXiv:2505.16990*, 2025.

Anthony Zhan. Principled and tractable rl for reasoning with diffusion language models. *arXiv:2510.04019*, 2025.

Hanyang Zhao, Dawen Liang, Wenpin Tang, David Yao, and Nathan Kallus. Diffpo: Training diffusion llms to reason fast and furious via reinforcement learning. *arXiv:2510.02212*, 2025a.

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

Renping Zhou, Zanlin Ni, Tianyi Chen, Zeyu Liu, Yang Yue, Yulin Wang, Yuxuan Wang, Jingshu Liu, and Gao Huang. Co-grpo: Co-optimized group relative policy optimization for masked diffusion model. *arXiv:2512.22288*, 2025.

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. *arXiv:2510.08233*, 2025.

APPENDIX

The appendix is organized as follows:

- In Appendix A, we describe related work.
- In Appendix B, we provide a diagram and an algorithm for our proposed policy sampling.
- In Appendix C, we present additional plots to supplement the experiments from the main paper:
  - In C.1, we replicate Figure 1 across models (LLaDA, Dream) and datasets (GSM8k, MATH).
  - In C.2, we replicate Figure 3 using latency (wall-clock time) as the efficiency measure.
  - In C.3, we replicate Figure 3 for policies trained on Dream.
  - In C.4, we show results for the same setting as Figure 3a and Figure 3c but with a denser α-grid. We also show the spread of policies across different training seeds.
  - In C.5, we plot the impact of varying the policy temperature τ_π.
  - In C.6, we provide model transfer results for the MATH dataset.
  - In C.7, we provide generation length (L) transfer results for the MATH dataset.
- In Appendix D, we conduct ablation experiments for our proposed policy design.
- In Appendix E, we perform a qualitative analysis of our learned sampling policies.
- In Appendix F, we visualize token generation trajectories.
- In Appendix G, we describe dynamic Plackett-Luce sampling as an alternative to Bernoulli sampling.
- In Appendix H, we detail our *expert steering* approach.
- In Appendix I, we discuss an extended background on discrete diffusion models.
- In Appendix J, we provide implementation and hyperparameters details.
- In Appendix K, we visualize the architecture of our sampling policy.

A Related work

Heuristic Samplers for dLLMs. Sampling in diffusion LLMs (dLLMs) (Nie et al., 2025; Ye et al., 2025) has recently attracted significant attention, with much of the work in this area proposing heuristic approaches to improve the decoding process in a training-free manner. Throughout this paper, we focus on Fast-dLLM (Wu et al., 2025) as a representative heuristic method, as it popularized confidence-thresholded sampling in dLLMs and demonstrated its crucial role in enabling faster inference in dLLMs compared to autoregressive models. Many recently proposed heuristics can be viewed as either reinterpretations or extensions of such confidence thresholding strategies (Ben-Hamu et al., 2025; Wei et al., 2025; Hong et al., 2025b; Li et al., 2025b; Yu et al., 2025; Kim et al., 2025b; Shen et al., 2025). Other notable heuristic approaches explore incorporating spatial (Huang et al., 2025a) or temporal information (Wang et al., 2025c), alternative confidence measures (Kim et al., 2025a), or more explicit modeling of token dependencies (Azangulov et al., 2025). In this work, we aim to complement ongoing research on sampling heuristics by investigating whether effective sampling strategies can be learned directly via reinforcement learning. Note that beyond improving the efficiency of unmasking in dLLMs, heuristics have also been proposed for remasking (Hong et al., 2025b; Dong et al., 2025) and for dynamically adjusting the generation length (Li et al., 2025a), which we do not target in this work.

Reinforcement Learning Post-Training for dLLMs. Early work on post-training diffusion LLMs via reinforcement learning includes d1 (Zhao et al., 2025b), which introduces a variant of GRPO tailored to dLLMs, and DiffuCoder (Gong et al., 2025), which focuses on enhancing the coding abilities of LLaDA-style models using RL. Most recent extensions aim to improve the quality of policy gradient estimators (Tang et al., 2025; Wang et al., 2025a; Lin et al., 2025; Rojas et al., 2025; Zhu et al., 2025; Wang et al., 2025e; Zhan, 2025), and have demonstrated promising results in further enhancing the reasoning capabilities of dLLMs. The key distinction from our approach is that these methods use a fixed sampling strategy (e.g., high-confidence sampling), and the policy corresponds to the dLLM itself (or a LoRA-augmented version for efficiency). Closest to our work are DCOLT (Huang et al., 2025b), which trains a separate unmasking module (with a fixed number of unmasked tokens in each step) in addition to updating the base model via RL, and DiFFPO (Zhao et al., 2025a), which jointly learns an unmasking confidence threshold while updating the base model through RL. Differently to both these works, our primary goal is not in improving the reasoning abilities of the underlying dLLM; hence we keep the underlying dLLM fixed and focus solely on training a standalone policy with the aim of learning fast and adaptive sampling while preserving the performance of the base model. Another related concurrent work is Hong et al. (2025a) where an unmasking policy for dLLMs trained via GRPO is proposed, similar to our approach. However, their policy unmasks a fixed number of tokens at each step (similar to DCOLT), whereas ours dynamically adapts the number of tokens to unmask via our Bernoulli formulation. Finally, Seed Diffusion (Song et al., 2025) identifies computation-aware reinforcement learning as a key ingredient for achieving competitive efficiency in closed-source coding dLLMs.

Orthogonal efforts to accelerate dLLMs. Besides the aforementioned heuristic-based sampling innovations, other efforts to improve inference efficiency in dLLMs (Ma et al., 2025) include KV caching (Jiang et al., 2025; Wu et al., 2025), (variants of) speculative decoding (Israel et al., 2025; Campbell et al., 2025; Guo & Ermon, 2025; Wu & Zhang, 2025), training separate decoder modules during pretraining (Liu et al., 2025; Arriola et al., 2025b), and diffusion forcing (Wang et al., 2025d), among others. Concurrent work explores distilling faster generation patterns directly into the base model (Chen et al., 2025), or training a (small) separate unmasking network on top of the pretrained dLLM (Bansal & Sanghavi, 2025; Bao et al., 2025). Crucially, unlike our approach, where this training is done via RL, these works train unmasking modules by distilling generation trajectories from the base model. As a result, we expect that such approaches may still inherit limitations observed in dLLM sampling, such as difficulties in the non semi-AR regime (cf. Section 2.2).

RL for Adaptive Compute Since our sampling policies result in a variable number of sampling steps ($T - \hat{T}$) per input, our work connects to the broader literature on adaptive computation (Graves, 2016; Teerapittayanon et al., 2016). Notable examples of using RL to learn input-dependent compute policies include conditional computation with stochastic gating (Bengio et al., 2015), dynamic block skipping in residual networks (Wang et al., 2018), and RL-based early-exit decisions for long chain-

of-thought reasoning (Dai et al., 2025). To the best of our knowledge, our work is the first to explore learning adaptive policies using RL for the task of sampling in dLLMs. The closest related concurrent work is DiFFPO (Zhao et al., 2025a); however, unlike our approach–where the policy is learned end-to-end–DiFFPO learns to predict adaptive thresholds, which are then used in the same fashion as the fixed thresholds employed by Fast-dLLM (Wu et al., 2025). Concurrently, there is also a growing body of work that uses RL to introduce adaptivity into reasoning chains of autoregressive LLMs (Arora & Zanette, 2025; Yi et al., 2025).

B POLICY SAMPLING DIAGRAM AND ALGORITHM

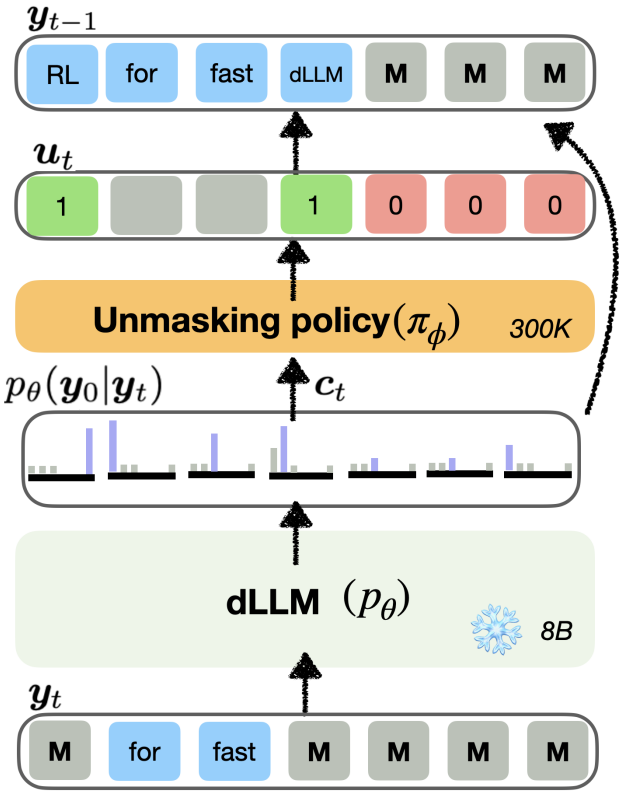


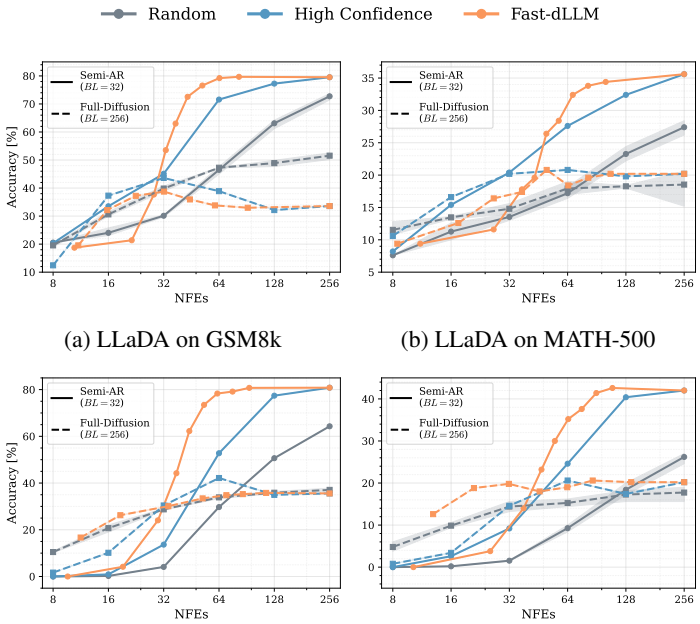
Figure 5: Diagram showing how our policies are used on top of a pretrained dLLM to unmask tokens and generate text.

Algorithm 1 Policy Sampling (full-diffusion setting $BL = L$)

Input: Prompt $\boldsymbol{x} \in \mathcal{V}^d$, maximum generation length L, maximum diffusion steps T, dLLM p_θ, sampling policy π_ϕ (via f_ϕ), policy temperature τ_π
Output: Generated sequence $\boldsymbol{y}_{\hat{T}}$, number of sampling steps
$\boldsymbol{y}_T \leftarrow (\boldsymbol{M}, \ldots, \boldsymbol{M})$ ▷ Fully masked initial sequence
$\mathcal{M}_T \leftarrow \{1, \ldots, L\}$ ▷ Set of still-masked positions
for $t = T, T-1, \ldots, 1$ **do**
  if $\mathcal{M}_t = \emptyset$ **then**
    break ▷ Stop generating when all unmasked
  end if
  $p_t^k \leftarrow p_\theta(\cdot \mid \boldsymbol{x}, \boldsymbol{y}_t), \forall k \in [L]$ ▷ Token distributions
  $c_t^k \leftarrow \max_{v \in \mathcal{V}} p_t^k(v), \forall k \in [L]$ ▷ Token confidences
  $\boldsymbol{c}_t \leftarrow (c_t^1, \ldots, c_t^L)$
  $m_t^k \leftarrow \mathbf{1}[k \in \mathcal{M}_t], \forall k \in [L]$
  $\boldsymbol{b}_t \leftarrow f_\phi(\boldsymbol{c}_t, \boldsymbol{m}_t, t)$ ▷ Policy logits
  $s_t^k \leftarrow \sigma(b_t^k / \tau_\pi), \forall k \in \mathcal{M}_t$
  $u_t^k \sim \mathrm{Ber}(s_t^k), \forall k \in \mathcal{M}_t$ ▷ Unmasking decisions
  $u_t^k \leftarrow 0, \forall k \notin \mathcal{M}_t$ ▷ Ignore already-unmasked
  if $\sum_{k=1}^{L} u_t^k = 0$ **then**
    $u_t^{k^*} \leftarrow 1, k^* = \mathrm{argmax}_{k \in \mathcal{M}_t} s_t^k$ ▷ Argmax fallback
  end if
  $\mathcal{U}_t^\pi \leftarrow \{k \in \mathcal{M}_t \mid u_t^k = 1\}$
  $y_{t-1}^k \sim p_t^k, \forall k \in \mathcal{U}_t^\pi$ ▷ Sample tokens
  $y_{t-1}^k \leftarrow y_t^k, \forall k \notin \mathcal{U}_t^\pi$
  $\mathcal{M}_{t-1} \leftarrow \mathcal{M}_t \setminus \mathcal{U}_t^\pi$
end for
$\hat{T} \leftarrow t$
return $\boldsymbol{y}_{\hat{T}}, T - \hat{T}$

C Additional Results

C.1 Figure 1 replicated for {LLaDA-8B-Instruct, Dream-7B-Instruct} × {GSM8k, MATH-500}

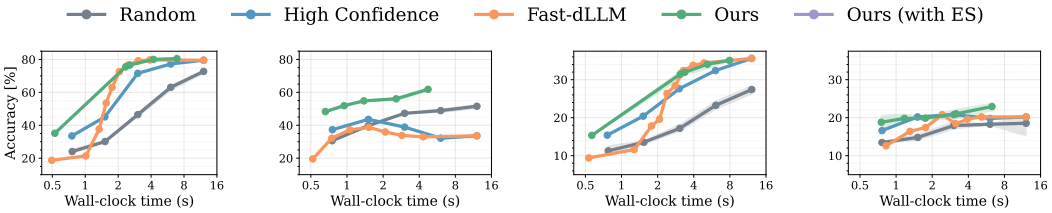


(a) LLaDA on GSM8k (b) LLaDA on MATH-500

(c) Dream on GSM8k (d) Dream on MATH-500

Figure 6: Performance comparison with ($BL = 32$; —) and without ($BL = 256$; - -) semi-AR generation. The same trend observed in Figure 1 holds across all models and datasets: confidence-based heuristics perform well under semi-AR generation but degrade significantly without it.

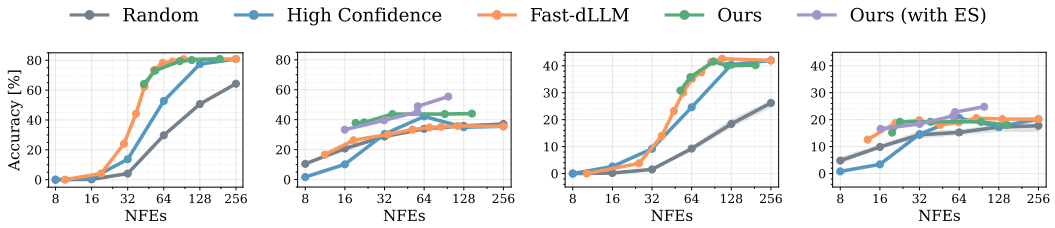
C.2 Figure 3 replicated using wall-clock time as the efficiency measure



(a) LLaDA GSM8k (b) LLaDA GSM8k (c) LLaDA MATH (d) LLaDA MATH

Figure 7: Figure 3 reproduced using wall-clock time (in seconds) as the efficiency metric instead of NFEs. The difference in our policy curves (—) when measured in wall-clock time versus NFEs (see Figure 3) is minimal to non-existent, demonstrating the negligible computational overhead of our policy. This low overhead is primarily due to the small size of the unmasking model relative to the base dLLM (300K vs. 8B parameters in the case of LLaDA). All experiments run on A100 GPUs.

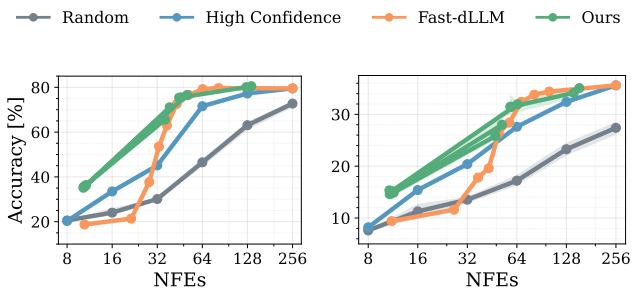
C.3 FIGURE 3 REPLICATED FOR DREAM-7B-INSTRUCT



(a) GSM8k, $BL = 32$ (b) GSM8k, $BL = 256$ (c) MATH-500, $BL = 32$ (d) MATH-500, $BL = 256$

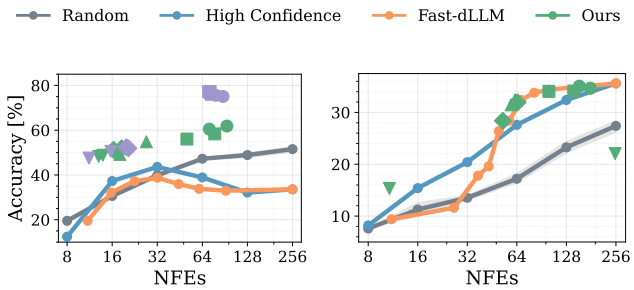
Figure 8: Results for Dream in semi-AR (Figure 8a & Figure 8c) and full-diffusion (Figure 8b & Figure 8d) generation regimes. For the policies we vary $\alpha \in \{10, 3, 1, 0.3, 0\}$ and use $\tau_\pi = 0.5$ for $BL = 32$ and $\tau_\pi = 1$ for $BL = 256$.

C.4 CONTROLLABILITY OF LEARNING UNMASKING POLICIES WITH RL (VIA α)



(a) GSM8k, $BL = 32$ (b) MATH-500, $BL = 32$

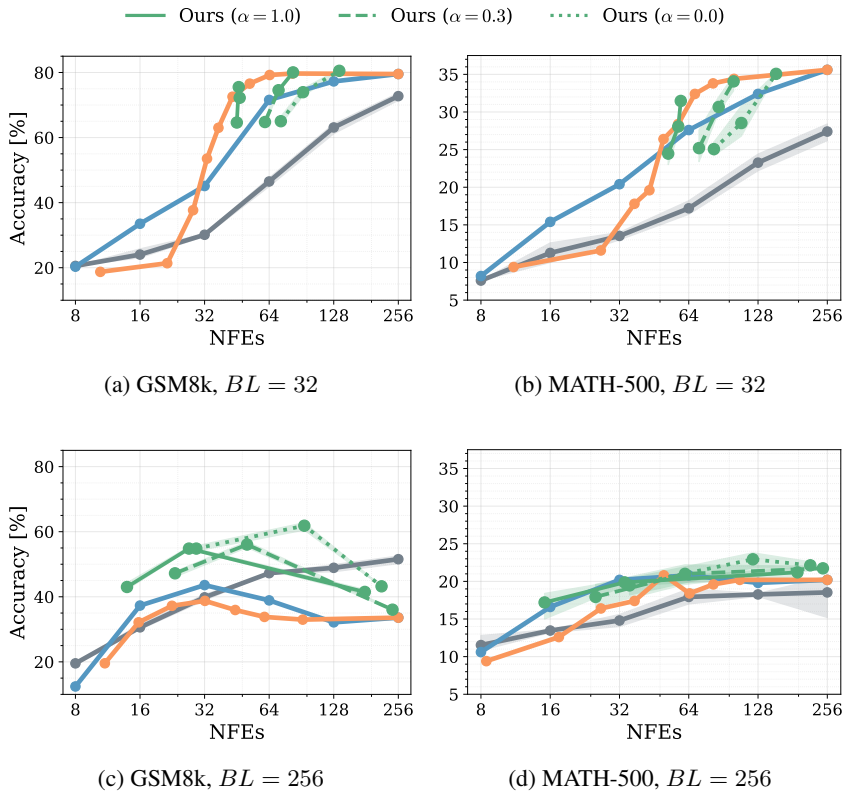
Figure 9: $BL = 32$ results for LLaDA with a denser regularization grid, $\alpha \in \{10.0, 9.0, \ldots, 1.0, 0.3, 0.0\}$. Single training seed due to cost; error bars show (min, max) over three test-time seeds. Note that for $\alpha \geq 4.0$, the change in NFEs is not monotonic; different values lead to convergence either close to the $\alpha = 3.0$ policy, or to that of $\alpha = 10.0$. Points for policy results are connected by α to emphasize the non-monotone behavior.



(a) GSM8k, $BL = 256$ (b) MATH-500, $BL = 32$

Figure 10: $BL = 256$ (GSM8k) and $BL = 32$ (MATH-500) results for LLaDA with all training seeds scattered. Even for a fixed value of α, the resulting policy can vary in accuracy and speed due to the randomness of the training procedure. Marker shape denotes the value of α.

C.5 IMPACT OF POLICY TEMPERATURE τ_π



(a) GSM8k, $BL = 32$

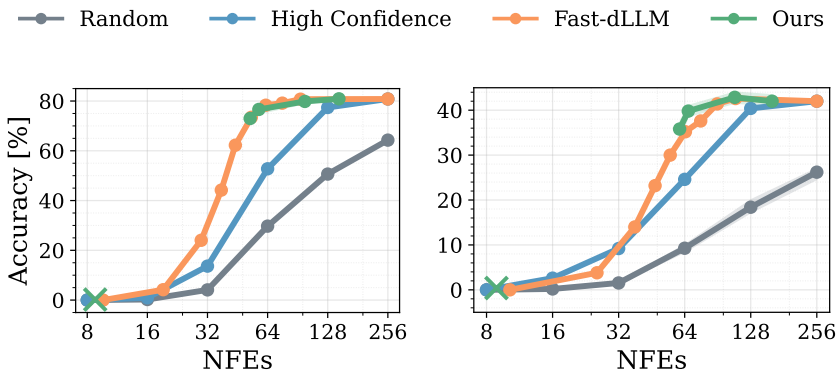
(b) MATH-500, $BL = 32$

(c) GSM8k, $BL = 256$

(d) MATH-500, $BL = 256$

Figure 11: We study the effect of changing the policy temperature τ_π (cf. Section 3.2). For each $\alpha \in \{3, 0.3, 0\}$, we construct a corresponding test-time Pareto frontier by varying $\tau_\pi \in \{1.5, 1.0, 0.5\}$. Interestingly, in some cases—such as $\alpha = 0$ with $BL = 32$—adjusting τ_π enables an effective trade-off between compute and performance. Moreover, we find that $\tau_\pi = 0.5$ is optimal in the semi-AR, while $\tau_\pi = 1$ performs best in the full-diffusion ($BL = L = 256$) setting.

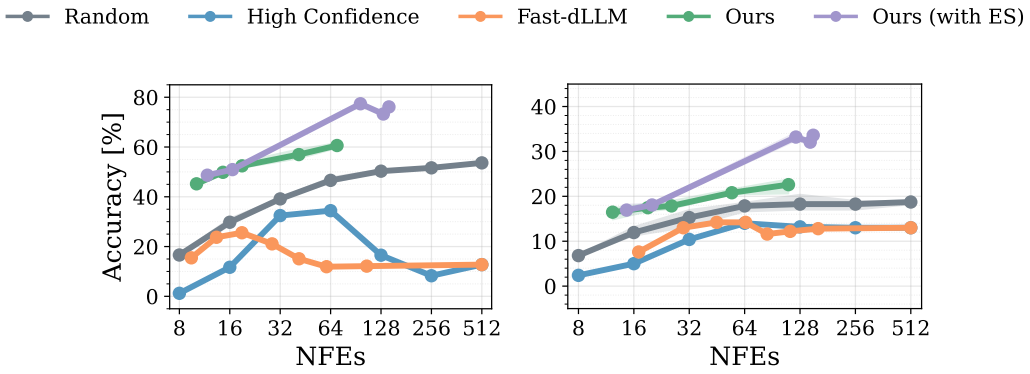
C.6 MODEL TRANSFER RESULTS



(a) GSM8k, $BL = 32$ (b) MATH-500, $BL = 32$

Figure 12: Model transfer results. We use policies trained on LLaDA and evaluate them on Dream with $\tau_\pi = 0.5$. Encouragingly, transferred policies give comparable results to training Dream specific policies (cf. Figure 8a & Figure 8c). Note that the $\alpha = 10$ policy is represented separately by a (✕) in the lower left to avoid misleading visualization when interpolating to $\alpha = 3$.

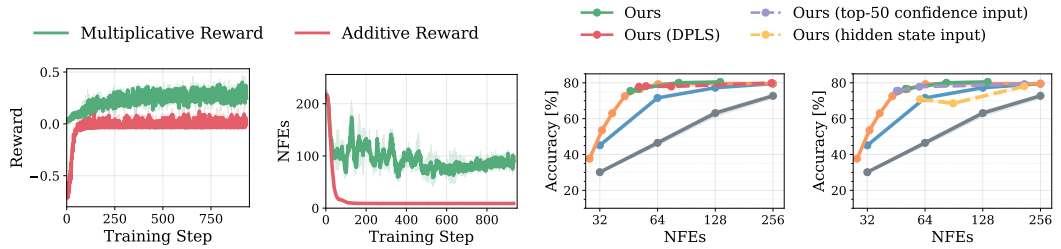
C.7 SEQUENCE-LENGTH TRANSFERABILITY



(a) GSM8k (b) MATH-500

Figure 13: $BL = L = 256$-trained policies from Section 4.2 evaluated with a 2x longer sequence length ($BL = L = 512$) with $\tau_\pi = 1$. Note that the learned policies yield almost identical performance, while the heuristic methods degrade further compared to $L = 256$ (cf. Figure 3b & Figure 3d). Both for LLaDA-8B-Instruct.

D Exploring the Design Space of π_ϕ



(a) Training reward for LLaDA with additive vs. multiplicative reward function (both $\alpha = 1.0$).

(b) Mean NFEs when training LLaDA with additive vs. multiplicative reward (both $\alpha = 1.0$).

(c) Bernoulli vs DPLS sampling. Both for LLaDA on GSM8k.

(d) Bernoulli policies with varying inputs. All for LLaDA on GSM8k.

Figure 14: Ablations for our proposed RL framework.

Additive vs. multiplicative rewards. We begin by ablating the structure of the reward function, comparing our proposed multiplicative combination of correctness and computational terms (cf. Equation (3)) with an additive alternative: $r(\boldsymbol{y}, \boldsymbol{y}_{\hat{T}}) - \alpha\left(\frac{T-\hat{T}}{T}\right)$. As shown in Figure 14a, while both exhibit increasing reward as training progresses, we observe that the additive formulation is much more prone to 'reward hacking', where it collapses to a very-fast-but-often-wrong policy. This is illustrated in Figure 14b: training with the additive reward results in a policy that unmasks everything at once, leading the number of sampling steps to collapse to the minimum possible for all inputs. As discussed in Section 3.3, we attribute this issue to the fact that incorrect but fast samples may still receive a positive advantage under the additive reward. In contrast, we find that the multiplicative reward effectively mitigates this issue by assigning a positive advantage only if the generation is correct, resulting in more stable and predictable training behavior.

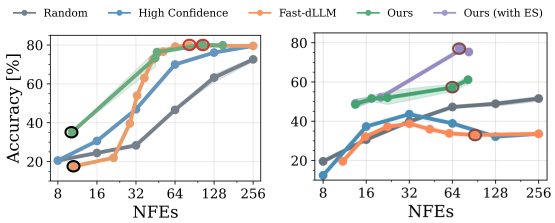
Policy likelihood. Recall from Section 3.2 that we model the unmasking probability for each position using a Bernoulli distribution. This has the advantage of admitting very efficient inference and likelihood calculations, but relies on the network f_ϕ to implicitly embed relationships between different tokens in the scores $\boldsymbol{b}_t$, and runs the risk of producing $\mathcal{U}_t^\pi = \emptyset$ in case $\boldsymbol{u}_t = \mathbf{0}$. Here, we therefore investigate a more involved sampling procedure, which we call *dynamic Plackett-Luce sampling* (DPLS; detailed description in Appendix G) which is guaranteed to unmask at least one position in each step and which combines the scores $\boldsymbol{b}_t$ through a softmax. We retrain the policies from Section 4.1 using DPLS; the resulting downstream accuracy on GSM8k is then shown in Figure 14c. We observe that both methods achieve very similar performance, aligning closely with the Fast-dLLM, which might provide additional support for the hypothesis that this frontier is optimal in the semi-AR setting. Furthermore, DPLS policies appear to show slightly better controllability via α (as indicated by a larger spread of policies trained with varying α).

Confidence-based policy input. Finally, we revisit our design choice of relying solely on the maximum token confidence values c_t^k as input to the sampling policy (cf. Section 3.2). We first train and evaluate a set of policies which do not take only the highest confidence per position c_t^k as input, but rather the top 50 highest values, thus giving the model more detailed information about the token predictive distributions p_t^k and potentially allowing it to design its own confidences measures for effective sampling. The results are presented in Figure 14d. Somewhat surprisingly, using only the maximum confidence per position (—) performs slightly better than using the top-50 confidences (—), which suggests that alternative uncertainty measures are unlikely to yield performance gains over the simple confidences c_t^k.

Additionally, we explore parametrizing the policy as an additional classification head on top of LLaDA's final hidden state $\boldsymbol{h}_t^k \in \mathbb{R}^H$. While this results in a significantly larger policy ($300M$ parameters), it offers the potential advantage of incorporating token-level semantic information. However, in practice we observe that hidden state-based policies perform worse (see - - in Figure 14d) and exhibit less stable training dynamics than the confidence-based policies. These results suggest that the unembedding matrix $W \in \mathbb{R}^{H \times V}$, which maps hidden states to token logits, plays a vital role in enabling effective policy decisions via confidence signals.

E WHAT ARE RL UNMASKING POLICIES ACTUALLY DOING?

To better understand the behaviour of our learned policies, we conduct a qualitative analysis comparing generation/unmasking trajectories produced by our RL policies against a confidence heuristic like Fast-dLLM (Wu et al., 2025). We consider three settings that span different points on the accuracy-efficiency Pareto frontier and different generation regimes (see Figure 15):



(a) $BL = 32$ (b) $BL = 256$

Figure 15: Results for LLaDA in semi-AR ($BL = 32$; figure replicated from Figure 3a) and full-diffusion ($BL = 256$; figure replicated from Figure 3b) generation regimes. We highlight the policies used in our qualitative analysis of unmasking orders: ◯ for **BL32-slow**, ◯ for **BL32-fast**, and ◯ for **BL256**.

- **BL32-slow** (◯): Semi-AR ($BL = 32$) with Fast-dLLM ($\lambda = 0.9$) vs. Policy ($\alpha = 0.3$)
- **BL32-fast** (◯): Semi-AR ($BL = 32$) with Fast-dLLM ($\lambda = 0.1$) vs. Policy ($\alpha = 10$)
- **BL256** (◯): Full-diffusion ($BL = L = 256$) with Fast-dLLM ($\lambda = 0.9$) vs. Policy ($\alpha = 0.3$) vs. Policy with expert steering ($\alpha = 0.3$)

Given a prompt $\boldsymbol{x}$ and generated answer $\hat{\boldsymbol{y}}$ (with a maximum answer length and maximum number of steps to set to L and T, respectively), the *unmasking order*[1] is defined as $\boldsymbol{o} \in [T]^L$, where $o_k := \min\{t \mid y_t^k \neq \boldsymbol{M}\}$ denotes the timestep when the token at position k is first unmasked. For presentation clarity, we reverse the time axis relative to the main paper, so time flows as $t = 1, 2, \ldots, T$. For all experiments, we use $N = 100$ GSM8k test samples with $L = T = 256$.

We describe main insights for each setting next.

E.1 BL32-SLOW

In this setting, our policy achieves performance nearly identical to Fast-dLLM in both accuracy and efficiency (see ◯ in Figure 15a). Despite these similarities, we find that the learned policy adopts a somewhat different sampling strategy compared to plain confidence thresholding. This is visually apparent in Figure 17, where we observe that, although both strategies predominantly follow a left-to-right generation pattern (due to semi-AR regime), the policy's unmasking order lines (—) display more "wiggliness" within each block. This is due to the fact that, as shown in Figure 19, the policy is far less likely to unmask adjacent tokens simultaneously—doing so for only around $\sim 20\%$ of tokens, in contrast to Fast-dLLM, which does so for $\sim 65\%$ on average. This suggests that the policy learns a more dispersed unmasking strategy, favoring token commitments at scattered positions within each block rather than in contiguous chunks. Such a scattered approach may enable the model to gather more diverse contextual information before finalizing neighboring tokens, potentially mitigating errors that arise from parallel generation. The more scattered generation pattern within each block is also evident in the token generation trajectories shown in Figure 23.

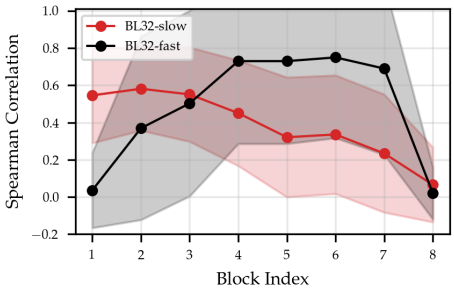


Figure 16: Spearman rank correlation between the unmasking orders of Fast-dLLM and our RL policies per block for semi-AR policies considered here. Note that for certain blocks, the unmasking orders exhibit zero correlation: first and last block for BL32-fast(◯), and last block for BL32-slow (◯).

The two methods also differ substantially in how they allocate compute across blocks. As shown in Figure 18a, Fast-dLLM assigns a larger number of NFEs to earlier blocks, allocating $\sim$17 NFEs to the first block and gradually decreasing to just $\sim$4 NFEs in the final block. In contrast, our learned

[1]Note that we consider partial orders here, as multiple tokens can be unmasked simultaneously.

policy distributes compute almost uniformly across blocks, using $\sim$10 NFEs per block. This likely also contributes to the decreasing Spearman rank correlation of unmasking orders between the policy and Fast-dLLM sampling across blocks (see Figure 16).

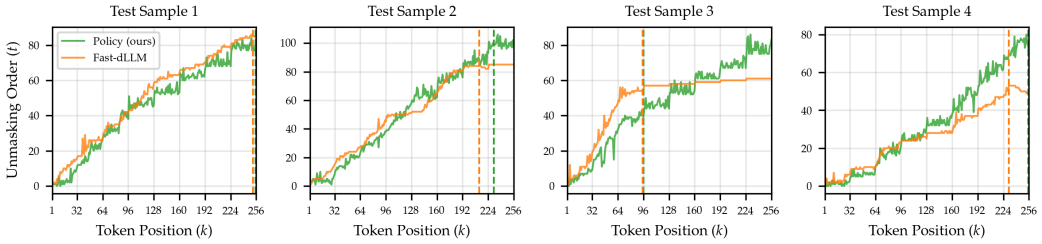
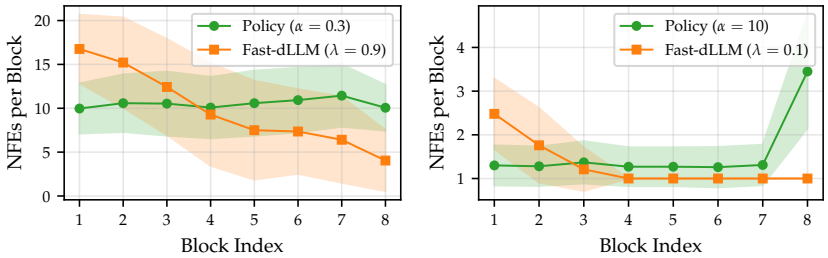


Figure 17: **BL32-slow**: unmasking orders for policy sampling ($\boldsymbol{o}_{\alpha=0.3}$) and Fast-dLLM ($\boldsymbol{o}_{\lambda=0.9}$) for four GSM8k test samples under semi-AR generation ($BL = 32, L = 256$). Vertical dashed lines indicate the position where the model outputs the end-of-sequence (*EOS*) token. Interestingly, Fast-dLLM appears more effective at accelerating generation after producing the *EOS* token, whereas policy sampling tends to sometimes waste compute by not unmasking all padding tokens (within the current block) after EOS in parallel, see the third test sample.



(a) **BL32-slow** (○) (b) **BL32-fast** (○)

Figure 18: Average number of sampling steps (NFEs) per block for LLaDA on GSM8k with semi-AR generation (same setting as in Figure 3a). *Left* we show NFEs per block for 'slow' variants of Fast-dLLM and policy sampling ($\sim$ 75 NFEs), while in the *right* plot we show 'fast' variants ($\sim$ 10 NFEs). Fast-dLLM exhibits a pattern of allocating more compute to earlier blocks. Our policy sampling, by contrast, distributes compute more uniformly across blocks, except in the fast ($\alpha = 10$) policy, where most compute is expended in the final block while generating numerical answers. To see where evaluated policies here are located on the pareto frontier, see Figure 15a. More details on qualitative differences between Fast-dLLM and policy sampling are provided in Appendix E.

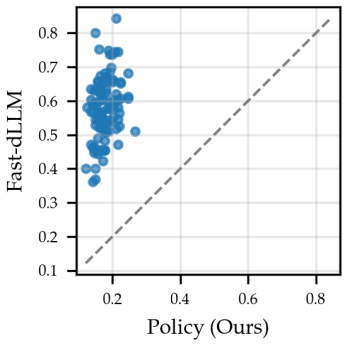


Figure 19: **BL32-slow**: for each of the $N = 100$ samples, we report the frequency of tokens whose right-adjacent token was unmasked at the same sampling step under Fast-dLLM versus our policy sampling. Interestingly, policy sampling unmasks adjacent tokens much less frequently than Fast-dLLM.

E.2 BL32-FAST

Here both strategies achieve very fast but low-accuracy sampling (see ◯ in Figure 15a). However, policy sampling significantly outperforms Fast-dLLM, achieving approximately $\sim 38\%$ accuracy compared to $\sim 18\%$. Examining the unmasking orders in Figure 20 and the average NFEs per block in Figure 18b, we observe that policy sampling tends to slow down in the final block, typically generating the numerical answer last (see also the token generation trajectory in Figure 25 and how the tokens of the numerical answer are generated last under policy sampling). This behavior may help explain its improved performance over Fast-dLLM and shows the promise of RL for discovering new sampling techniques.

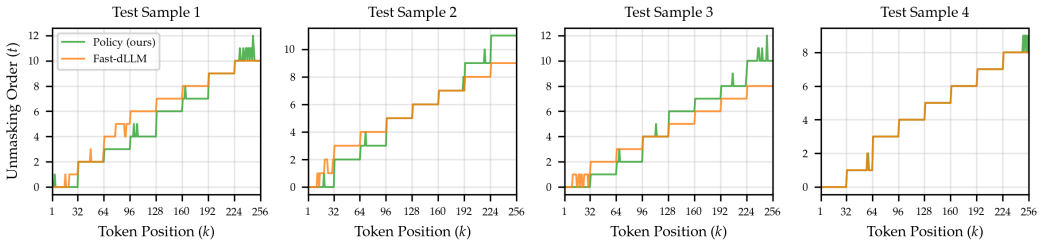


Figure 20: **BL32-fast**: unmasking orders for policy sampling ($\boldsymbol{o}_{\alpha=10}$) and Fast-dLLM ($\boldsymbol{o}_{\lambda=0.1}$) for four GSM8k test samples under semi-AR generation ($BL = 32, L = 256$). Under low-NFE constraints, both strategies often predict all tokens within a block simultaneously (as indicated by flat unmasking-order lines). In the final block, when generating the numerical answer, the policy slows down.

E.3 BL256

Lastly, we consider the full-diffusion setting ($BL = L = 256$). While all three strategies achieve a comparable number of NFEs (see ◯ in Figure 15b), their accuracy differs significantly. The fact that these strategies implement notably different sampling procedures is further evidenced by the low values of Spearman rank correlations 0.26 ± 0.21 when comparing Fast-dLLM orders with policy sampling, and -0.24 ± 0.51 when comparing Fast-dLLM with policy trained with expert steering. These differences are also reflected in the divergent unmasking orders shown in Figure 21 (also refer to Figure 22 for a plot of average unmasking order across samples).

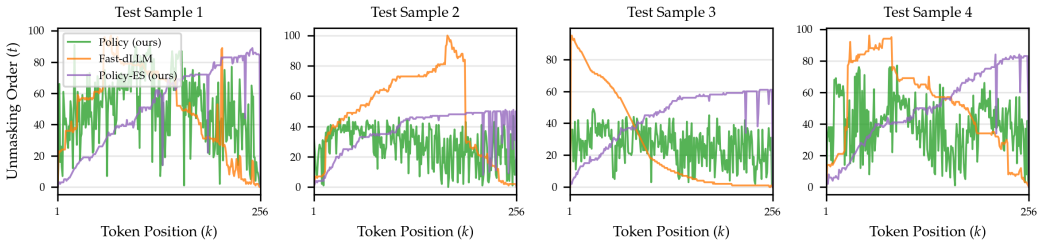


Figure 21: **BL256:** unmasking orders for policy sampling ($\boldsymbol{o}_{\alpha} = 0.3$) and Fast-dLLM ($\boldsymbol{o}_{\lambda=0.9}$) for four GSM8k test samples under full-diffusion generation ($BL = L = 256$).

Starting with Fast-dLLM (—), we observe in Figure 21 that it often generates tokens in a "reverse" order (right-to-left), see also the token trajectories in Figure 28). This behavior likely stems from the fact that LLaDA did not exclude padding tokens from the loss computation during SFT (Nie et al., 2025) to preserve the dLLM's ability to generate sequences of varying (effective) lengths. Consequently, the model overfits to padding tokens, which confidence-based methods like Fast-dLLM then mistakenly prioritize, leading to their early generation. This helps explain the poor performance of confidence-based strategies observed in Figure 1, highlighting the fragility of heuristic-driven approaches.

Looking at the unmasking orders of policy sampling (—), the policy appears to implement a near-random unmasking strategy, as there is little evident spatial structure in the unmasking order. This is encouraging, as it suggests that the policy can learn to disregard corrupted signals—such as inflated

confidence for padding tokens—without explicit supervision, leading to more robust performance across different generation settings (semi-AR vs full-diffusion). Lastly, we observe that the policy with expert steering (—) generates in an AR fashion (left-to-right). This behavior likely accounts for its superior performance in the full-diffusion setting. It indicates that, during full-diffusion training, the policy is able to pick up the accuracy benefits of AR generation from its expert demonstrations—drawn from the semi-AR regime (Appendix H)—while still learning to generate significantly faster than a purely AR strategy that unmaskes one token at a time.

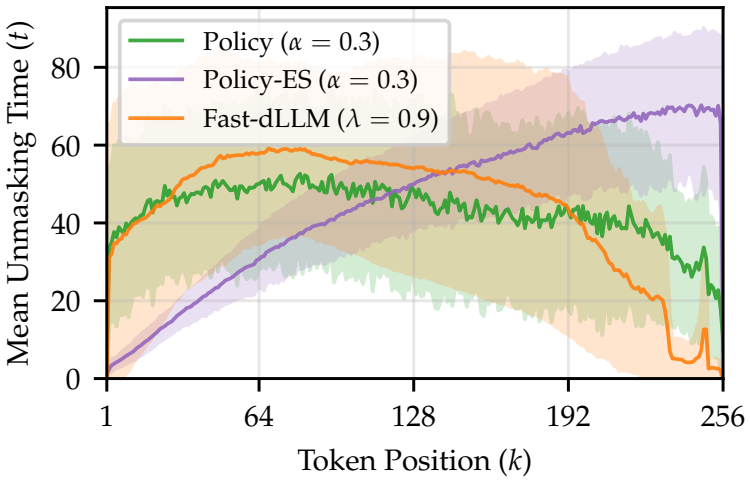


Figure 22: Mean unmasking time for each token position ($L = 256$), averaged over $N = 100$ samples, for LLaDA on GSM8k under full-diffusion generation (same setting as in Figure 3b). For visualization purposes, time is shown in reverse. Fast-dLLM (—) exhibits somewhat reverse (i.e., right-to-left) generation due to LLaDA's corrupted confidence on padding tokens (see Appendix E). Encouragingly, the expert steering policy (—) learns to overcome this issue and recovers left-to-right generation (observe how tokens at earlier positions are generated first on average). The locations of the evaluated policies on the Pareto frontier are shown in Figure 15b (see ○).

F GENERATION TRAJECTORIES VISUALIZATIONS

Here we visualize token generation trajectories for the various sampling strategies considered in our work. Specifically, for a randomly selected GSM8k test sample, we display the generated token at each of the $L = 256$ positions, along with its unmasking time (shown in the bottom right corner of each cell, starting from $t = 0$). For visual clarity, token cells are color-coded according to their unmasking time, with blue indicating earlier and red indicating later unmasking. Note that for semi-AR generations, color-coding is done within each block separately.

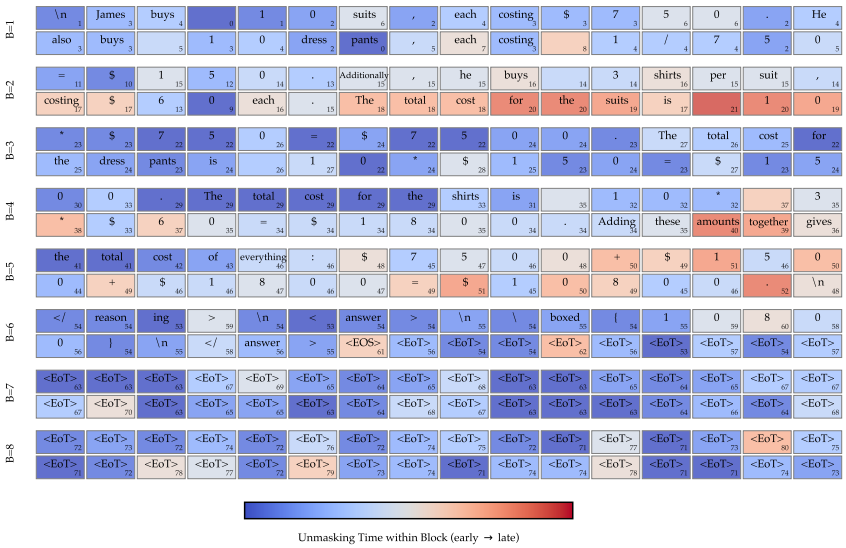


Figure 23: **BL32-slow**: generation trajectory for policy sampling ($\alpha = 0.3$, semi-AR).

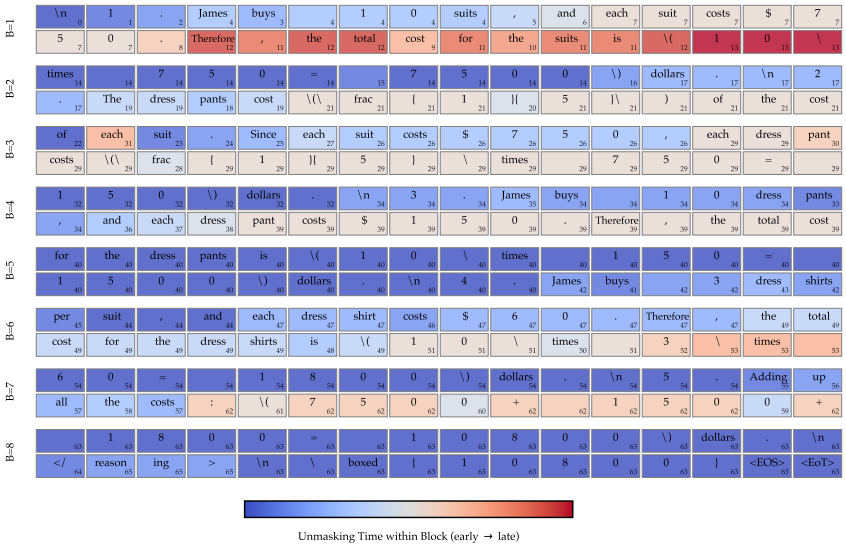


Figure 24: **BL32-slow**: generation trajectory for Fast-dLLM sampling ($\lambda = 0.9$, semi-AR).

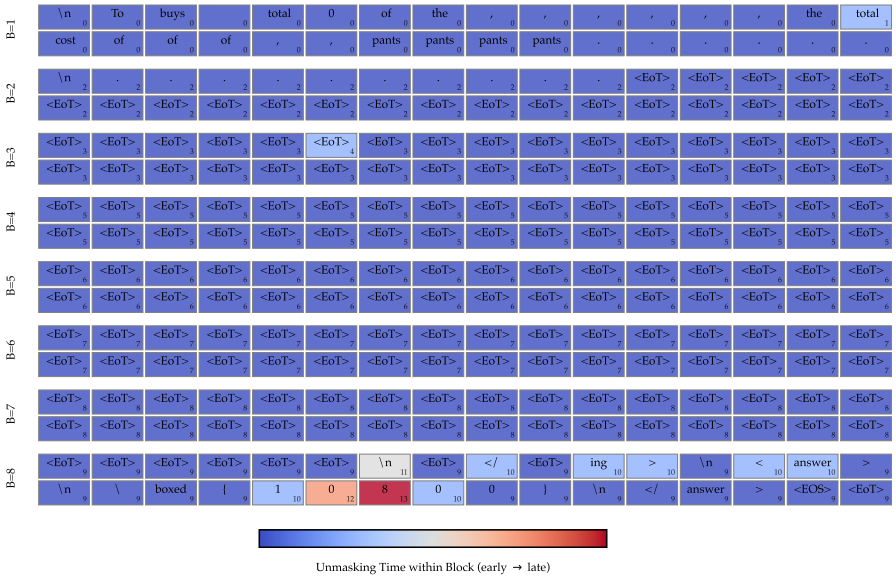


Figure 25: **BL32-fast**: generation trajectory for policy sampling ($\alpha = 10$, semi-AR).

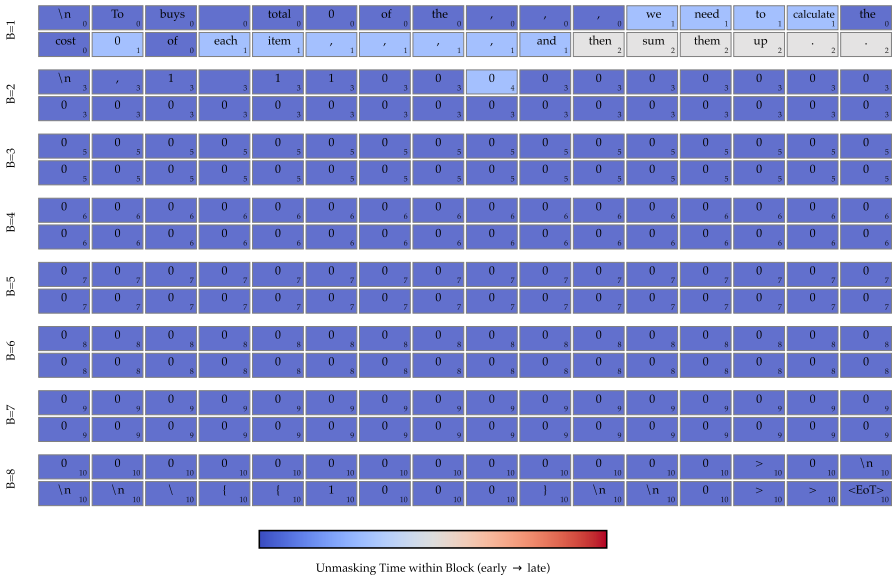


Figure 26: **BL32-fast**: generation trajectory for Fast-dLLM sampling ($\lambda = 0.1$, semi-AR).

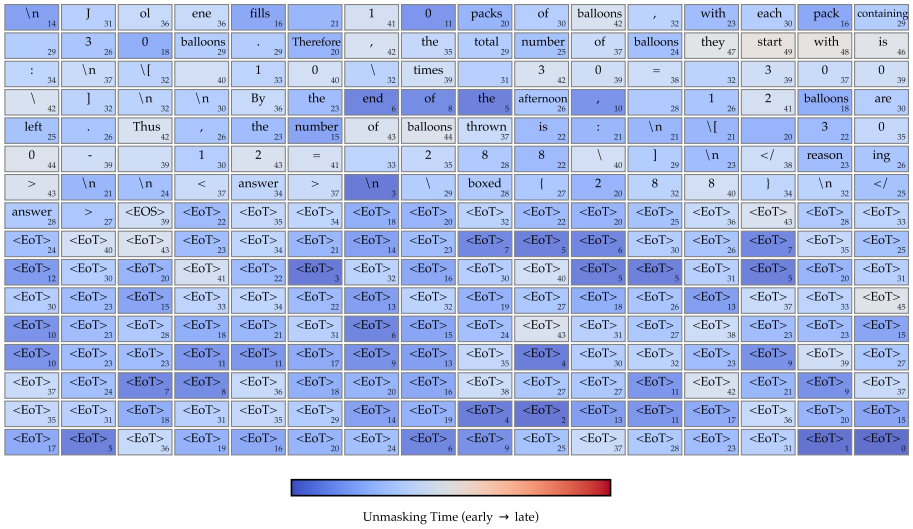


Figure 27: **BL256**: generation trajectory for policy sampling ($\alpha = 0.3$, full-diffusion).

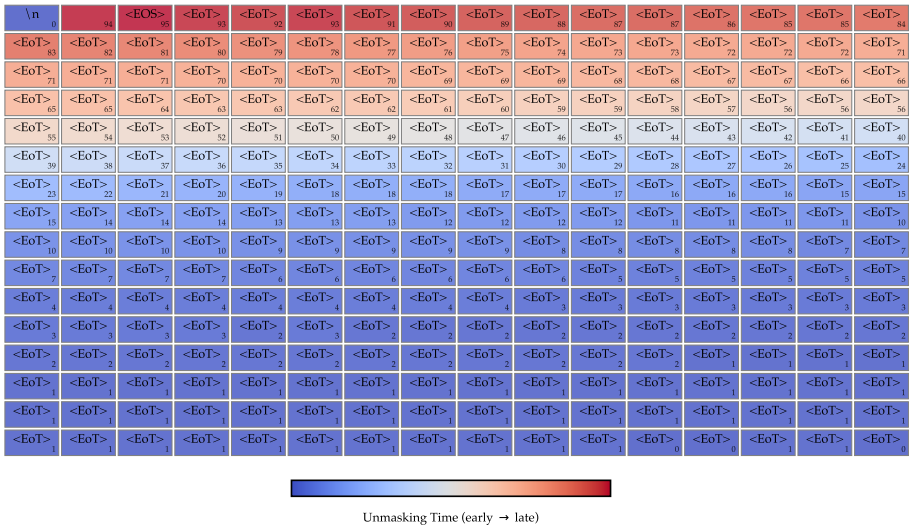


Figure 28: **BL256**: generation trajectory for Fast-dLLM sampling ($\lambda = 0.9$, full-diffusion).

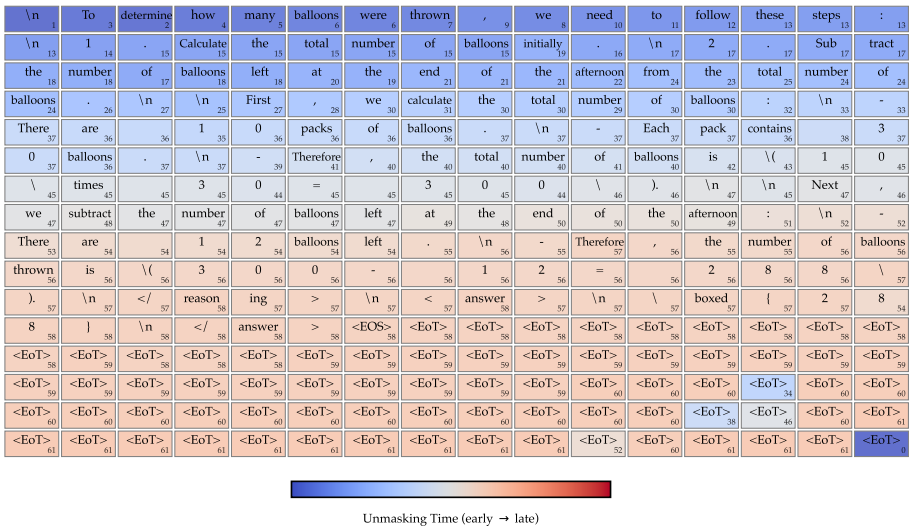


Figure 29: **BL256**: generation trajectory for policy sampling ($\alpha = 0.3$, expert steering, full-diffusion).

G DYNAMIC PLACKETT-LUCE SAMPLING

We detail here the dynamic Plackett-Luce sampling (DPLS) strategy as an alternative to the Bernoulli sampling (cf. Section 3.2) used in the main experiments of this paper. The name reflects the connection the Plackett-Luce (PL) model proposed by Luce (1959) and Plackett (1975), with one key difference: the number of selected items is not fixed but can vary freely between 1 and L.

Formally, the Plackett-Luce model operates as follows. Let $\boldsymbol{b}_t = (b_t^1, \ldots, b_t^L)$ denote the unmasking logits (corresponding to the output of policy network f_ϕ, same as in Section 3.2). Interpreting these scores as unnormalized utilities associated with choosing each token for unmasking, under the PL model the likelihood of any particular *permutation* $\boldsymbol{\sigma} := (\sigma_1, \ldots, \sigma_L)$ of the tokens is given by

$$P(\boldsymbol{\sigma} \mid \boldsymbol{b}_t) = \prod_{l \in [L]} \frac{\exp(b_t^{\sigma_l})}{\sum_{j \geq l}^{L} \exp(b_t^{\sigma_j})} \,.$$

Informally, this corresponds to sampling all indices without replacement, where at each step the probability of choosing an item is proportional to its (exponentiated) utility.

The Plackett-Luce model can be easily adapted to model sampling of a fixed-length *ordered* set $\mathcal{U}_t \subseteq [L]$ where $|\mathcal{U}_t| = K$. This is because the probability of any particular partial permutation is equal to the marginalization over all permutations which complete it. Conretely, let $\Sigma(\mathcal{U}_t)$ be the set of permutations which begin with the sequence $\mathcal{U}_t$, then it follows that

$$P(\mathcal{U}_t \mid \boldsymbol{b}_t) = \sum_{\boldsymbol{\sigma} \in \Sigma(\mathcal{U}_t)} \underbrace{\prod_{l \in [L]} \frac{\exp(b_t^{\sigma_l})}{\sum_{j \geq l}^{L} \exp(b_t^{\sigma_j})}}_{P(\boldsymbol{\sigma} \mid \boldsymbol{b}_t)} = \prod_{l \in \mathcal{U}_t} \frac{\exp\left(b_t^{\sigma_i}\right)}{\sum_{j \in \mathcal{U}_t^C \cup \mathcal{U}_t^{\geq l}} \exp\left(b_t^{\sigma_j}\right)}$$

where $\mathcal{U}_t^C$ denotes the complement of $\mathcal{U}_t$, and $\mathcal{U}_t^{\geq l}$ denotes the indices which are in $\mathcal{U}_t$ at or after position l.

To extend the PL model to a *variable-length* unmasking sequence, we introduce a special *STOP* token with fixed utility $b_{\text{STOP}} = 0$, and proceed as follows:

1. Sample a single token $l \in [L]$ to unmask
$$l \sim \text{softmax}(\boldsymbol{b}_t / \tau_\pi)$$
and initialize $\mathcal{U}_t^\pi = \{l\}$.
2. Sample another token l' from the *renormalized distribution*
$$l' \sim \text{softmax}([\boldsymbol{b}_t^{\setminus \mathcal{U}}; b_{\text{STOP}}] / \tau_\pi)$$
where $[\boldsymbol{b}_t^{\setminus \mathcal{U}}; b_{\text{STOP}}]$ denotes $\boldsymbol{b}_t$ concatenated with b_{STOP} and with the logits of all previously selected indices $l \in \mathcal{U}_t^\pi$ set to $-\infty$.
3. Add l' to $\mathcal{U}_t^\pi$.
4. If l' was not *STOP*, repeat from 2.

As written above, this algorithm is not GPU-friendly due to the dynamic computation graph it implies. However, it can be implemented in an efficient manner by treating the loop in steps 2-4 as a Gumbel-argsort and simply masking out all actions which get sampled after *STOP*. Once the samples have been obtained, their likelihoods can be efficiently computed in the same manner as in the fixed-length case above by marginalizing over all actions which occur after *STOP*.

H EXPERT STEERING

As mentioned in Section 4.2, we find that naïvely training confidence policies directly on full-diffusion generation ($BL = L$) task yields policies which beat out the heuristic methods, but underperform the ones trained in the semi-AR setting (cf. Figure 3).

Since semi-AR decoding lies within the function class representable by the policy π_ϕ—that is, there is some ϕ such that π_ϕ approximates the semi-AR setting—we hypothesize that our failure to obtain similar policies is due to the vanishingly small probability of encountering autoregressive-like rollouts by chance. More concretely, imagine trying to train a policy on a task for which purely AR generation is substantially more effective than any other decoding strategy. In such a scenario, starting from a randomly initialized policy, the probability of observing a rollout resembling a purely AR sequence is approximately $1/L! \approx 0$ (where L is the generation length), making it extremely unlikely to sample such rollouts during RL training.

To try to eschew this problem, we devise the *Expert Steering* (ES) training strategy as follows. Formally, letting π_ϕ denote the sampling policy to be learned, we simply replace it *at train time only* with the mixture

$$\pi_\phi^{ES}(\cdot \mid \boldsymbol{y}_t) = \frac{G}{G+E}\pi_\phi(\cdot \mid \boldsymbol{y}_t) + \frac{E}{G+E}\sum_{e=1}^{E}\delta_e(\cdot \mid \boldsymbol{y}_t)$$

where G is the GRPO group size, E is the number of experts to mimic, and each Dirac distribution δ_e represents a chosen deterministic "expert policy" (e.g., a heuristic method). Then in the outer loop of GRPO, instead of sampling a group $\boldsymbol{g} \sim \pi_\phi$ of size G from π_ϕ, we simply sample an augmented group $\boldsymbol{g}' \sim \pi_\phi^{ES}$ of size $G + E$. In the inner loop of GRPO we proceed as normal, making sure to use π_ϕ^{ES} when calculating the likelihood ratios to avoid the training instabilities that would otherwise arise (as samples from the Diracs may have likelihood ≈ 0 under π_ϕ)

In practice, for our experiments in Section 4.2 we use a single deterministic expert δ_e corresponding to Fast-dLLM with $\lambda = 0.9$ and $BL = 32$, and force exactly one draw ($E = 1$) from this heuristic per group. The effect is that if the policy is *worse* than this heuristic, that single sample will tend to have positive advantage, causing the policy to be biased toward it. On the other hand, if the policy is *better* than the heuristic, it will have negative advantage, causing it to be made *less* likely. Thus, expert steering encourages exploration towards the expert—in this case, Fast-dLLM—while still allowing the policy to go beyond it, thanks to the group relative loss in GRPO.

I EXTENDED BACKGROUND

Masked vs. uniform discrete diffusion. The D3PM framework (Austin et al., 2021a) characterizes forward corruption via a Markov kernel Q_t. Two canonical choices are (i) *masked* (absorbing-state) diffusion, which maps every token toward a distinguished mask state $\boldsymbol{M}$,

$$Q_t^{\text{mask}} \;=\; (1-\alpha_t)I \;+\; \alpha_t\, \mathbf{1}\boldsymbol{e}_m^\top,$$

with $\alpha_t \in [0,1]$, $\boldsymbol{e}_m$ denotes a one-hot vector (with dimension V) for $\boldsymbol{M}$ token, and $\mathbf{1}$ represents a vector of all 1s of size V, and (ii) *uniform-state* diffusion, which replaces tokens uniformly at random,

$$Q_t^{\text{uni}} \;=\; (1-\alpha_t)I \;+\; \frac{\alpha_t}{V}\,\mathbf{1}\mathbf{1}^\top.$$

These differ in their limiting behavior and reverse constraints: Q_t^{mask} has a point-mass stationary distribution at $\boldsymbol{M}$ and induces a monotonic forward trajectory in the mask count (reverse steps deterministically move from $\boldsymbol{M} \to$ token), whereas Q_t^{uni} has a uniform stationary distribution and permits token$\leftrightarrow$ token substitutions in both directions. The absorbing choice recovers the familiar MDM objective (cf. Equation (1)) under a particular reweighting (Ou et al., 2025) and thus ties masked LMs to discrete diffusion (Austin et al., 2021a).

Continuous-time discrete diffusion. Recent work derives continuous-time limits (CTMC) for discrete diffusion with absorbing corruption and shows that the training objective is a *weighted time integral of token-wise cross-entropy*, with weight proportional to a signal-to-noise ratio term (e.g., $\alpha_t/(1-\alpha_t)$). This yields a simple, schedule-invariant objective, clarifies forward–reverse consistency, and improves optimization and sampling without changing the prediction target (Shi et al., 2024; Sahoo et al., 2024). Conceptually, this places masked diffusion on the same ODE/SDE footing as continuous-state diffusion (Kingma & Gao, 2023) while retaining native discrete token inference.

Continuous-input diffusion for categorical data. A complementary approach to discrete-state diffusion is to keep the diffusion process continuous in both time and input by operating on token embeddings. For example in Dieleman et al. (2022), an orthogonal line embeds tokens into $\mathbb{R}^d$ and applies fully continuous diffusion in *both* time and input space. Training can be done with cross-entropy via score interpolation, and sampling uses ODE/SDE solvers with tools like classifier-free guidance. This preserves the continuous machinery but introduces an embedding decoding interface and relaxes exact discreteness during the trajectory.

Summary of differences.

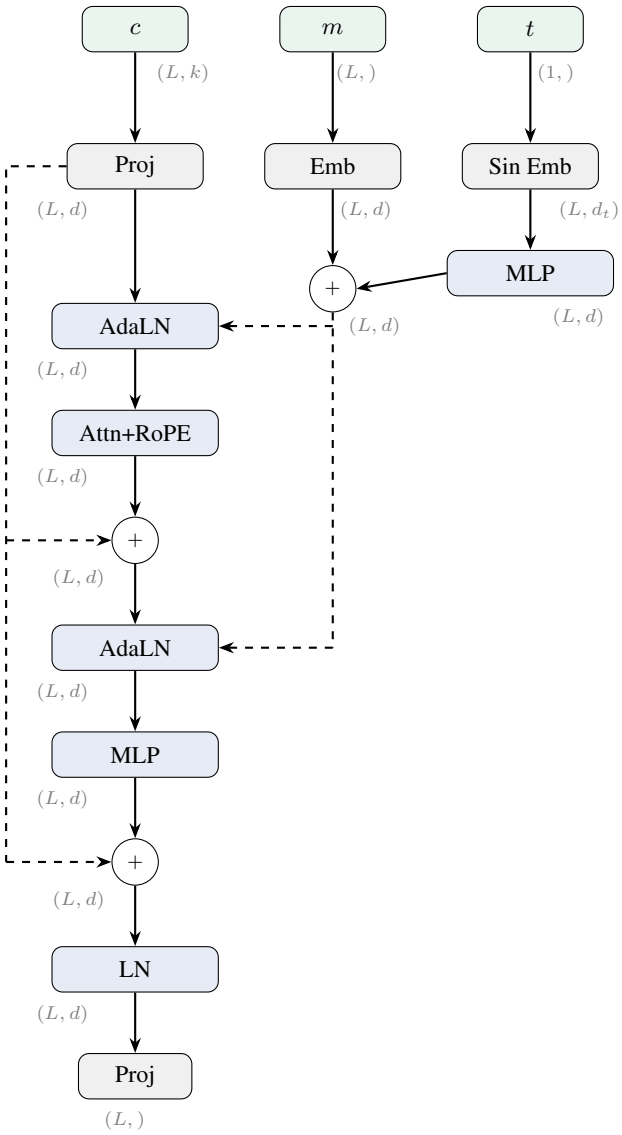
- **Masked (absorbing)**: monotonic reverse dynamics ($\boldsymbol{M} \to$ token only); objective reduces to weighted MDM; pairs naturally with unmask-only sampling policies used in this work.
- **Uniform-state**: non-monotone reverse dynamics (token $\leftrightarrow$ token); encourages broader exploration but typically requires remasking/substitution moves and would thus complicate policy design.
- **Continuous-input**: inherits continuous samplers and guidance; trades exact discreteness for an embedding path and decoding step.

We adopt the absorbing formulation in this work.

J TRAINING AND POLICY NETWORK CONFIGURATION

Table 1: Training and policy configuration for our main experiments.

Category	**Parameter**	**Value**
Training	Learning rate	3e-5
	LR scheduler	Cosine
	Warmup steps	100
	Effective batch size	16
	Weight decay	0.1
	Max gradient norm	0.2
	Clipping factor (ϵ)	0.5 (0.2 for expert steering)
GRPO	Training data	GSM8k and MATH training set mixture
	Num train epochs	1
	Group size	8
	β (KL penalty)	0.0
	Correctness reward $r(\boldsymbol{y}, \boldsymbol{y}_t)$	We use d1 (Zhao et al., 2025b) rewards.
Policy Network	Number of transformer blocks	1
	Hidden dimension	128
	Feedforward dimension	512
	Number of heads	2
	Time embedding dim	128
	Total parameter count	300K

K POLICY ARCHITECTURE DIAGRAM

c = confidence m = mask t = timestep k = top-k d = hidden_dim