

RFG: Test-Time Scaling for Diffusion Large Language Model Reasoning with Reward-Free Guidance

Tianlang Chen Minkai Xu Jure Leskovec Stefano Ermon

Stanford University

Presenter: Haotian Liu

February 27, 2026

Outline

Introduction

Preliminaries

Method

Experiments

Conclusion

Outline

Introduction

Preliminaries

Method

Experiments

Conclusion

Background & Challenges

► Rise of Diffusion LLMs (dLLMs)

- ▶ dLLMs (e.g., LLaDA, Dream) have shown remarkable progress in math reasoning and coding.
- ▶ Competitive with Autoregressive (AR) models via mask-predict pretraining.
- ▶ **Current Limitation:** Success is mostly limited to pre-training; **test-time computation** is underexplored.

► The Need for Test-Time Computation

- ▶ Mimic human reasoning behavior that thinks longer for more complicated problems to obtain more reliable results.
- ▶ AR models use **Process Reward Models (PRMs)** for step-by-step guidance.

► The Challenge for dLLMs: Difficulty in Applying PRMs to dLLMs

- ▶ **Any-order Sampling:** dLLMs do not generate strictly left-to-right.
- ▶ **Masked States:** Intermediate steps are incomplete sentences with random masks, making it impossible to apply traditional reward models.

Reward-Free Guidance (RFG)

- ▶ **Proposed Framework: Reward-Free Guidance (RFG)**
 - ▶ A principled method to guide dLLM reasoning without explicit process reward models.
 - ▶ Views test-time scaling as a **guided sampling problem**.
- ▶ **Key Innovation: Implicit Parameterization**
 - ▶ Parameterize the reward as the **log-likelihood ratio** between a Policy model (p_θ) and a Reference model (p_{ref}): $r_\theta(x) = \beta \log \frac{p_\theta(x)}{p_{ref}(x)}$
 - ▶ Allows **free decomposition** into step-wise PRMs for each denoising step.
 - ▶ **Analogy:** Resembles *Classifier-Free Guidance (CFG)* used in image diffusion.
- ▶ **Contributions & Results**
 - ▶ **Training-Free:** Works with any off-the-shelf SFT or RL-enhanced checkpoints.
 - ▶ **Performance:** Significant gains on Math and Code benchmarks (up to 9.2% accuracy boost).

Outline

Introduction

Preliminaries

Method

Experiments

Conclusion

Discrete Diffusion Models

► From Continuous to Discrete

- ▶ **Origin:** Diffusion models were originally designed for continuous data (e.g., images) by reversing a Gaussian noising process.
- ▶ **Challenge:** Adding Gaussian noise is ill-defined for discrete text tokens.
- ▶ **Solution:** Discrete Diffusion Models (Austin et al., 2021) use Markov chains to randomize tokens.

► Masked Diffusion Model

- ▶ An effective implementation where "noise" is defined as the special [MASK] token.
- ▶ **Forward Process:** Progressively masks tokens until the sequence is fully masked.
- ▶ **Reverse Process:** Learns to recover original tokens from partially masked sequences.
- ▶ **Significance:** Combines the architecture of Masked Language Modeling (MLM) with the iterative refinement capability of diffusion.

Diffusion Large Language Models (dLLMs)

► Formulation

- **Forward Process:** Masks tokens based on a noise schedule α_t :

$$q(\mathbf{x}_t | \mathbf{x}_0) = \prod_{i=0}^L q(\mathbf{x}_t^{(i)} | \mathbf{x}_0^{(i)}), \quad q(\mathbf{x}_t^{(i)} | \mathbf{x}_0^{(i)}) = \begin{cases} 1 - \alpha_t, & \mathbf{x}_t^{(i)} = [\text{MASK}] \\ \alpha_t, & \mathbf{x}_t^{(i)} = \mathbf{x}_0^{(i)} \end{cases}$$

- **Reverse Process:** Progressively recovers original tokens starting from a fully masked state.

► Advantages of dLLMs

- **Any-order Generation:** Not tied to fixed left-to-right ordering; can unmask any subset of tokens based on context.
- **Parallelized Inference:** Capable of predicting multiple tokens simultaneously.
- **Iterative Refinement:** Allows revisiting and correcting intermediate states (crucial for complex reasoning).
- **Reduced Exposure Bias:** Mitigates error accumulation common in sequential generation.

Outline

Introduction

Preliminaries

Method

Experiments

Conclusion

Theoretical Framework of RFG

- **Proposition 3.1.** *Given a diffusion trajectory-level reward that is parameterized as the log-likelihood ratio of two dLLMs, i.e., $r_\theta(\mathbf{x}_{0:T}) := \beta \log \frac{p_\theta(\mathbf{x}_{0:T})}{p_{\text{ref}}(\mathbf{x}_{0:T})}$. Define $Q_\theta^t(\mathbf{x}_{t-1}, \mathbf{x}_{t:T}) := \sum_{i=t}^T \beta \log \frac{p_\theta(\mathbf{x}_{i-1}|\mathbf{x}_{i:T})}{p_{\text{ref}}(\mathbf{x}_{i-1}|\mathbf{x}_{i:T})}$, then we have that Q_θ^t is the expectation of exponential r_θ at step t :*

$$Q_\theta^t(\mathbf{x}_{t-1}, \mathbf{x}_{t:T}) = \beta \log \mathbb{E}_{p_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{x}_{t-1:T})} e^{\frac{1}{\beta} r_\theta(\mathbf{x}_{0:T})}, \quad (1)$$

- Q_θ^t can be viewed as Q function representing the expectation of the overall trajectory reward $r_\theta(\mathbf{x}_{0:T})$ at step t .
- Following the classic setup in RL literature, the step reward r_θ^t can be written as:

$$r_\theta^t(\mathbf{x}_{t-1}|\mathbf{x}_t) = Q_\theta^t - Q_\theta^{t+1} = \beta \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_{t:T})}{p_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_{t:T})} = \beta \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{p_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t)}. \quad (2)$$

Guided Sampling & Connection to CFG

► Reweighted Sampling Formula

- Based on the derived step-reward, the final guided log-probability is:

$$\begin{aligned}\log p^*(\mathbf{x}_{t-1}|\mathbf{x}_t) &= \log p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) e^{\frac{1}{\gamma} r_\theta^t(\mathbf{x}_{t-1}|\mathbf{x}_t)} + C \\ &= (1+w) \log p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) - w \log p_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t) + C\end{aligned}\tag{3}$$

- w controls the **guidance strength** ($w > 0$ emphasizes the improved policy).

► Connection to Classifier-Free Guidance (CFG)

- **CFG (Images):** Uses *Conditional* vs. *Unconditional* models to align generation with prompts.

$$\nabla_{\mathbf{x}} \log p^*(\mathbf{x}|\mathbf{c}) = \nabla_{\mathbf{x}} \log p_{\text{conditional}}(\mathbf{x}|\mathbf{c}) + w \nabla_{\mathbf{x}} \log \frac{p_{\text{conditional}}(\mathbf{x}|\mathbf{c})}{p_{\text{unconditional}}(\mathbf{x})}.$$

- **RFG (Text):** Uses *Policy* vs. *Reference* models to align generation with reasoning logic.
- **Conclusion:** RFG effectively adapts the success of CFG to the domain of logical reasoning.

Implementation & Advantages

Algorithm 1 Sampling with Reward-Free Guidance (RFG)

Require: Reference model p_{ref} , policy model p_{θ} , guidance strength w , query q , answer length L , sampling steps N , denoising strategy \mathcal{S}

```
1: Initialize  $\mathbf{x}_N \leftarrow [\text{MASK}]^L$                                 ▷ Start with a fully masked sequence of length  $L$ 
2: for  $t \leftarrow N$  down to 1 do
3:    $\log \pi_{\text{ref}} \leftarrow \text{Logits}(p_{\text{ref}}(\cdot | \mathbf{x}_t, q))$           ▷ Get logits from reference model
4:    $\log \pi_{\theta} \leftarrow \text{Logits}(p_{\theta}(\cdot | \mathbf{x}_t, q))$           ▷ Get logits from enhanced model
5:    $\log \pi_{\text{RFG}} \leftarrow (1 + w) \log \pi_{\theta} - w \log \pi_{\text{ref}}$           ▷ Combine logits via RFG
6:    $\mathbf{x}_{t-1} \leftarrow \mathcal{S}(\mathbf{x}_t, \log \pi_{\text{RFG}}, t/N)$           ▷ Generate the next state using the guided logits
7: end for
8: return  $\mathbf{x}_0$ 
```

► The Sampling Algorithm

1. **Compute Logits:** Run both models on the current masked sequence.
2. **Combine:** Apply linear combination using guidance weight w .
3. **Unmask:** Use guided logits to determine which tokens to recover.

► Key Advantages

- **Model-Agnostic:** Can be applied to any dLLM architecture.
- **Plug-and-Play:** Acts as a test-time booster for already trained models.

Outline

Introduction

Preliminaries

Method

Experiments

Conclusion

Experimental Setup

- ▶ **Tasks & Benchmarks**
 - ▶ **Mathematical Reasoning:** GSM8K, MATH-500.
 - ▶ **Code Generation:** HumanEval, MBPP.
- ▶ **Model Configurations**
 - ▶ **Reference (p_{ref}):** LLaDA-Base / Dream-Base.
 - ▶ **Policy (p_θ):**
 - ▶ *Instruction Tuned:* LLaDA-Instruct, Dream-Instruct.
 - ▶ *RL-Enhanced:* d1-LLaDA (Math), LLaDA-1.5 (DPO), DiffuCoder (Code).
- ▶ **Baselines**
 - ▶ **Original Policy:** Direct inference using the post-trained model.
 - ▶ **Naive Ensemble:** Averaging logits from the policy and reference models.
 - ▶ *Note: The Naive Ensemble matches the exact compute budget of RFG, serving as a controlled ablation.*
- ▶ **Evaluation Protocol**
 - ▶ **Setting:** Zero-shot setting (to assess intrinsic reasoning).
 - ▶ **Metrics:** Accuracy for math; pass@1 for code.

Main Results

Table 1: Performance on GSM8K and MATH-500. We report the accuracy of the original model, the naïve ensemble baseline, and RFG, along with the relative gain over the original model.

Model	GSM8K				MATH-500			
	Original	Ensemble	RFG	Rel. Gain	Original	Ensemble	RFG	Rel. Gain
Instruction Fine-tuning								
LLaDA 8B Instruct	79.8	68.0	81.3	+1.5	38.4	33.6	39.6	+1.2
Dream 7B Instruct	80.9	68.3	82.1	+1.2	43.6	30.6	46.4	+2.8
Reinforcement Learning								
d1-LLaDA	82.5	68.5	84.7	+2.2	38.8	34.8	41.6	+2.8
LLaDA 1.5	81.6	70.9	82.1	+0.5	42.4	40.6	44.4	+2.0

Table 2: Performance on HumanEval and MBPP. We report the pass@1 accuracy of the original model, the naïve ensemble baseline, and RFG, along with the relative gain over the original model.

Method	HumanEval				MBPP			
	Original	Ensemble	RFG	Rel. Gain	Original	Ensemble	RFG	Rel. Gain
Instruction Fine-tuning								
LLaDA 8B Instruct	45.1	40.9	47.6	+2.5	49.8	49.8	50.6	+0.8
Dream 7B Instruct	62.2	58.5	65.2	+3.0	64.6	69.6	70.4	+5.8
Reinforcement Learning								
diffuCoder	69.5	73.2	78.7	+9.2	72.8	70.8	74.3	+1.5

- ▶ The substantial margin over the *Naive Ensemble* confirms that the gain stems from the principled **RFG formulation**, not merely from adding a reference model or computational scaling.
- ▶ RFG effectively guides generation toward accurate reasoning traces using off-the-shelf checkpoints, without requiring additional training or external reward models.

Qualitative Analysis

Query (GSM8K)

Christina is planning a birthday party and needs .75 gift bags per invited guest, because 1/4 of attendees don't show up. She invited 16 friends. Gift bags are \$2 each. How much will she spend?

Original (LLaDA 1.5)

To determine how much Christina will spend on gift bags, we need to follow these steps:

1. Calculate the number of friends who don't show up.
2. Determine the number of gift bags needed.
3. Calculate the total cost of the gift bags.

First, let's find out how many friends don't show up. Christina invited 16 friends, and 1/4 of them don't show up:

$$\frac{1}{4} \times 16 = 4$$

So, 4 friends don't show up.

Next, we calculate the number of friends who actually show up:

$$16 - 4 = 12$$

Since Christina needs 0.75 gift bags per invited guest, the number of gift bags needed is:

$$0.75 \times 12 = 9$$

Finally, we calculate the total cost of the gift bags. Each gift bag costs \$2, so the total cost is:

$$9 \times 2 = 18$$

Therefore, the amount Christina will spend is:

18

RFG (LLaDA 1.5)

To determine how much Christina will spend on gift bags, we need to follow these steps:

1. Calculate the number of friends who don't show up.
2. Determine the number of gift bags needed based on the actual attendees.
3. Calculate the total cost of the gift bags.

First, we calculate the number of friends who don't show up. Since 1/4 of the attendees don't show up and she invited 16 friends:

$$\frac{1}{4} \times 16 = 4$$

So, 4 friends don't show up.

Next, we determine the number of gift bags needed based on the actual attendees. Christina needs 0.75 gift bags per invited guest, so for 16 invited guests:

$$0.75 \times 16 = 12$$

So, she needs 12 gift bags.

Finally, we calculate the total cost of the gift bags. Each gift bag costs \$2, so:

$$12 \times 2 = 24$$

Therefore, the total amount Christina spends is:

24

Query (MBPP)

Write a python function to find the number of numbers with an odd value when rotating a binary string the given number of times. Your code should pass the test:

```
assert odd_Equivalent("011001", 6) == 3
***
```

Original (Dream Instruct)

```
def odd_Equivalent(binary_string, rotations):
    n = len(binary_string)
    odd_count = 0
    for i in range(rotations):
        binary_string = binary_string[1:] + binary_string[0]
        odd_count += sum(int(bit) for bit in binary_string) % 2
    return odd_count
```

RFG (Dream Instruct)

```
def odd_Equivalent(binary, n):
    count = 0
    for i in range(n):
        binary = binary[1:] + binary[0]
        if int(binary, 2) % 2 == 1:
            count += 1
    return count
```

- ▶ **Math Reasoning:** RFG produces more coherent multi-step derivations, avoiding hallucinations and inconsistent conclusions.
- ▶ **Code Generation:** Generated code is not only syntactically correct but also more robust, reducing common errors like missing edge conditions or incomplete logic.

Sensitivity to Guidance Strength

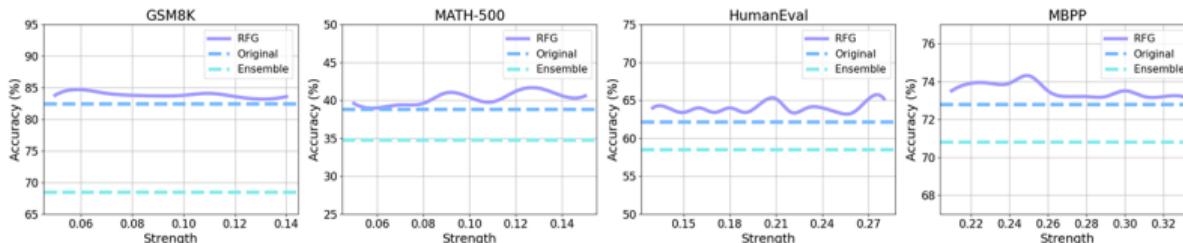


Figure 3: Accuracy of RFG under varying guidance strength w across four benchmarks: GSM8K and MATH-500 for mathematical reasoning using d1-LLaDA, and HumanEval and MBPP for code generation using Dream-Instruct and DiffuCoder, respectively. We observe that RFG consistently improves performance over a broad range of guidance strength.

- ▶ Conduct a sensitivity analysis by varying the guidance strength w .
- ▶ **Finding:** Performance is not confined to a narrow peak; instead, it exhibits a **wide plateau** of strong performance across a broad range of w values.

Outline

Introduction

Preliminaries

Method

Experiments

Conclusion

Conclusion

- ▶ Introduced **Reward-Free Guidance (RFG)**, a novel and principled framework to enhance dLLM reasoning at test time.
- ▶ **Core Innovation:** Guides the denoising process **without** requiring an explicitly trained process reward model (PRM).
- ▶ **Mechanism:** Elegantly bridges trajectory-level rewards with step-wise guidance via the **log-likelihood ratio** of Policy and Reference models.
- ▶ **Empirical:** Validated effectiveness on challenging benchmarks (Math & Code), outperforming baselines.

Thank you!