

Scaling Reasoning in Diffusion Large Language Models via Reinforcement Learning

Siyan Zhao^{*1}, Devaansh Gupta^{*1}, Qinqing Zheng², Aditya Grover¹

¹UCLA, ²Meta AI

May 23rd, 2025
NVIDIA



Motivation

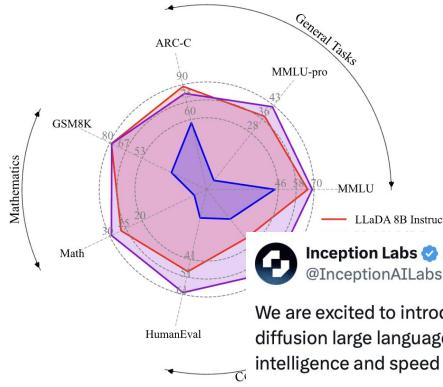
Recently, autoregressive (AR) LLMs benefit from post-training w/ RL

- ❑ E.g. DeepSeek-R1, Kimi K1.5.
- ❑ RL can incentivize/elicit sophisticated reasoning in LLMs.

However, these advances have primarily been limited to autoregressive LLMs that operate through **left-to-right, sequential inference**.

Diffusion Large Language Models (dLLMs) as AR alternatives

TL;DR: We introduce LLaDA, a diffusion model with an unprecedented 8B scale, trained entirely from scratch, rivaling LLaMA3 8B in performance.



We are excited to introduce Mercury, the first commercial-grade diffusion large language model (dLLM)! dLLMs push the frontier of intelligence and speed with parallel, coarse-to-fine text generation.

Two side-by-side code snippets and execution times for generating a sequence of tokens.

Autoregressive LLM (Left-to-right Generation):

```
def auto_regressive_decode(model, start_token, max_length):
    sequence = [start_token]
    while len(sequence) < max_length:
        input = torch.tensor(sequence).unsqueeze(0)
        output = model(input)
        next_token = torch.argmax(output, dim=-1).item()
        sequence.append(next_token)
        if next_token == model.eos_token_id:
            break
    return sequence
```

Inception Diffusion LLM (Coarse-to-fine Generation):

```
def auto_regressive_decode(model, start_token, max_length):
    sequence = [start_token]
    while len(sequence) < max_length:
        input = torch.tensor(sequence).unsqueeze(0)
        output = model(input)
        next_token = torch.argmax(output, dim=-1).item()
        sequence.append(next_token)
        if next_token == model.eos_token_id:
            break
    return sequence
```

Iterations:

- Autoregressive LLM (Left-to-right Generation):** 75 [Completed]
- Inception Diffusion LLM (Coarse-to-fine Generation):** 14 [Completed]

Time:

- Autoregressive LLM (Left-to-right Generation):** 0:05
- Inception Diffusion LLM (Coarse-to-fine Generation):** 0:05

Jiacheng Ye
@JiachengYe15



🚀 Excited to announce Dream 7B (Diffusion reasoning model): the most powerful open diffusion large language model to date.



Google DeepMind 🌐 @GoogleDeepMind · May 20

We've developed **Gemini Diffusion**: our state-of-the-art text diffusion model.



Instead of predicting text directly, it learns to generate outputs by refining noise, step-by-step. This helps it excel at coding and math, where it can iterate over solutions quickly. #GoogleIO

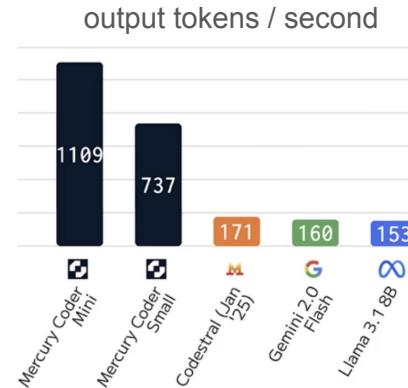
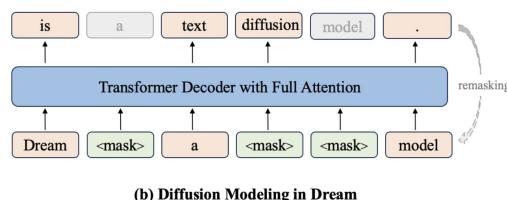
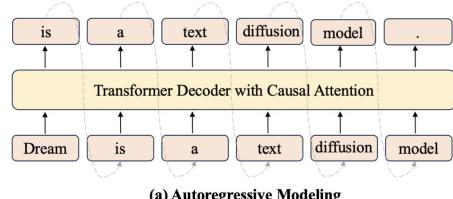


**dLLMs generate through an iterative denoising process,
in a coarse-to-fine manner.**

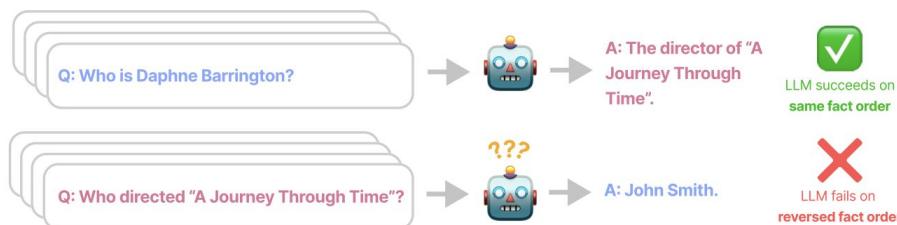


Figure: a generation example of Dream-7B-instruct.

Why dLLMs?



Faster Inference due to decoding multiple tokens at once



Breaks the reversal curse by bidirectional modeling

Controllability: Useful for infilling tasks

Motivation

RL progress has been limited to the AR context

Goal: Explore RL for bidirectional dLLMs

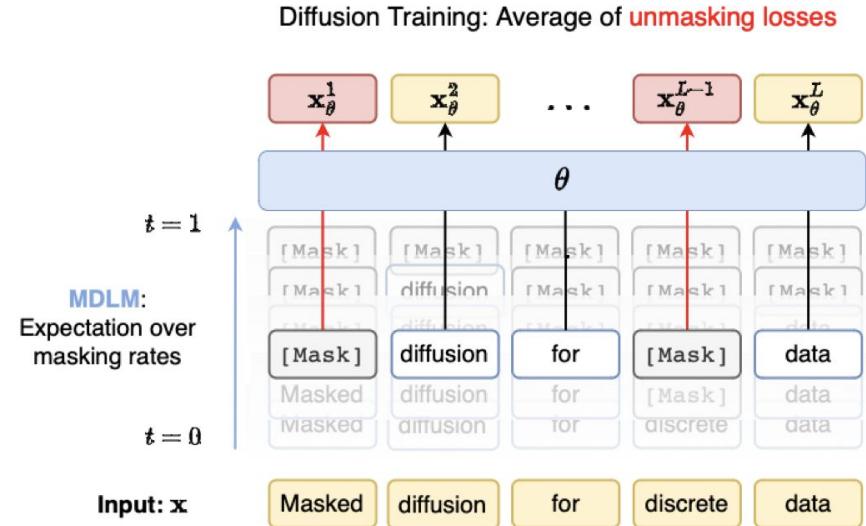
	Model	Recipe
Qwen2.5 7B	AR	SFT+RL
LLaMA3 8B	AR	SFT+RL
LLaDA 8B	Diffusion	SFT
Dream 7B	Diffusion	SFT

Background : Diffusion Language Models

- ❑ Prior works show that **masked dLLMs** consistently perform better than uniform dLLMs and scales better (Lou et al., 2024; Campbell et al., 2024)
- ❑ Recent large dLLMs that scales > 7B, such as **DiffuLLaMA, LLaDA and Dream**, are all based on masked dLLMs
- ❑ We focus on **applying policy gradient RL to masked dLLMs** in this work

Background : Masked dLLMs

- ❑ Forward process: corrupts token sequences with [Mask] (Sahoo et al., 2024; Shi et al., 2024; Austin et al., 2021; Ou et al., 2024; Zheng et al., 2024)
- ❑ At any time t , tokens remain unmasked with probability α_t (noise schedule), which strictly decreases as t increases.
- ❑ When $t = 1$, all tokens become masked.



Background : Masked dLLMs

Training masked dLLMs:

- ❑ Given a specific noise schedule: α_t ; In LLaDA: $\alpha_t = 1 - t$
- ❑ Initialize an unmasking predictor f_θ (e.g. transformer) with bidirectional attention.

Background : Masked dLLMs

Training masked dLLMs:

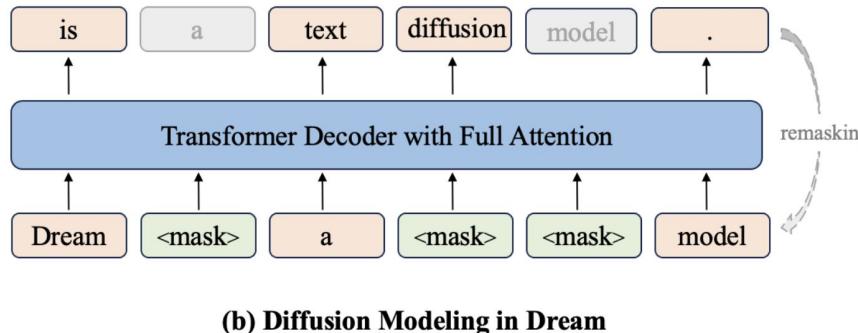
- ❑ Given a specific noise schedule: α_t ; In LLaDA: $\alpha_t = 1 - t$
- ❑ Initialize an unmasking predictor f_θ (e.g. transformer) with bidirectional attention.
- ❑ Sampling $t \in [0, 1]$, masking tokens accordingly, and predicting originals tokens and train with cross-entropy loss:

$$-\mathbb{E}_{t \sim \mathcal{U}[0,1], x_0 \sim p_{\text{data}}, x_t \sim q_{t|0}(x_t|x_0)} \left[\frac{1}{t} \sum_{k=1}^{|x_t|} \mathbf{1}[x_t^k = \text{mask}] \log f_\theta(x_0^k | x_t) \right]$$

Background : Masked dLLMs

Training masked dLLMs: **Adapt from existing AR models.**

- ❑ DiffuLLaMA: adopted from Llama 2 7B
- ❑ Dream: adopted from Qwen2.5 7B
- ❑ Predict **shifted** masked tokens, allowing for initialization with AR models
- ❑ Adapt causal attention mask to bidirectional mask

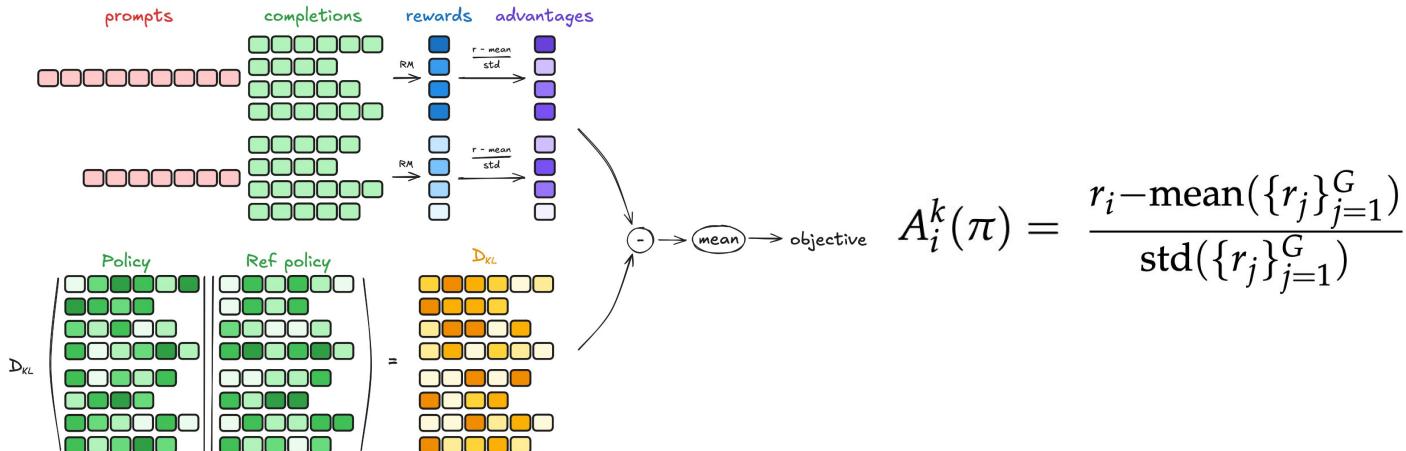


Background : Policy Gradient Methods for LLMs

- ❑ Policy Gradient Methods are widely used to fine-tune LLMs.
 - ❑ PPO requires a state value function to estimate advantages → high compute and memory.

Background : Policy Gradient Methods for LLMs

- ❑ Policy Gradient Methods are widely used to fine-tune LLMs.
 - ❑ PPO requires a state value function to estimate advantages → high compute and memory.
- ❑ Group Relative Preference Optimization (GRPO)
 - ❑ GRPO avoids learning a value model.
 - ❑ For a query, sample G responses $\{o_1, o_2, \dots, o_G\}$ from policy.
 - ❑ Assign advantage to all tokens in responses using:



Background : Policy Gradient Methods for LLMs

- ❑ Policy Gradient Methods are widely used to fine-tune LLMs.
 - ❑ PPO requires a state value function to estimate advantages → high compute and memory.
- ❑ Group Relative Preference Optimization (GRPO)
 - ❑ GRPO avoids learning a value model.
 - ❑ For a query, sample G responses $\{o_1, o_2, \dots, o_G\}$ from policy.
 - ❑ Assign advantage to all tokens in responses using:

$$A_i^k(\pi) = \frac{r_i - \text{mean}(\{r_j\}_{j=1}^G)}{\text{std}(\{r_j\}_{j=1}^G)}$$

- ❑ GRPO objective: clipped ratio + KL penalty

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)]$$

$$\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \min \left[\frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})} \hat{A}_{i,t}, \text{clip} \left(\frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}, 1 - \varepsilon, 1 + \varepsilon \right) \hat{A}_{i,t} \right] - \beta \mathbb{D}_{KL} [\pi_\theta || \pi_{ref}] \right\}$$

Log-Likelihood Computation

- ❑ Policy gradient methods maximize the expected return by computing the gradient of the expected reward wrt the log-prob of actions taken under the policy.
- ❑ GRPO objective requires computing the likelihood at both the **token level** and the **sequence level**.

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)]$$

$$\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \min \left[\frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})} \hat{A}_{i,t}, \text{clip} \left(\frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}, 1 - \varepsilon, 1 + \varepsilon \right) \hat{A}_{i,t} \right] - \beta \mathbb{D}_{KL} [\pi_\theta || \pi_{ref}] \right\}$$

Log-Likelihood Computation

- ❑ AR LLMs directly model the exact per-token log-probabilities with CE loss
- ❑ The sequence-level log-probability can be easily computed through the chain rule using one forward pass:

$$\log \pi_{\text{AR}}(o|q) = \sum_{k=1}^{|o|} \log \pi_{\text{AR}}(o^k|q, o^{<k})$$

- ❑ However, **dLLMs do not adhere to sequential factorization** of the sequence log-probability - they optimize an **ELBO**
- ❑ The exact marginal likelihood is **intractable**

Monte Carlo Estimation utilizes excessive memory

Algorithm 3 Conditional Log-likelihood Evaluation of LLaDA

Require: mask predictor p_θ , prompt p_0 , response r_0 , the number of Monte Carlo estimations n_{mc}

```
1: log_likelihood = 0
2: for  $i \leftarrow 1$  to  $n_{mc}$  do n_mc = 128 => large computational graph with hundreds of forward passes
3:    $l \sim \{1, 2, \dots, L\}$  #  $L$  is the sequence length of  $r_0$ 
4:   Obtain  $r_l$  by uniformly sampling  $l$  tokens from  $r_0$  without replacement for masking
5:    $\log_{likeli} = \log_{likeli} + \frac{L}{l} \sum_{i=1}^L \mathbf{1}[r_l^i = M] \log p_\theta(r_0^i | p_0, r_l)$ 
6: end for
7:  $\log_{likeli} = \log_{likeli} / n_{mc}$ 
8: Return  $\log_{likeli}$ 
```

Sequence-Level Log Likelihood with Mean Field Decomposition

- ❑ **Recall:** dLLMs treat the token sequence as a whole and its sequence-level log-probability lacks the AR decomposition.
- ❑ To efficiently estimate it, we use a simple mean-field decomposition:

$$\log \pi_{\theta}(o|q) \text{ by } \sum_{k=1}^{|o|} \log \pi_{\theta}(o^k|q)$$

- ❑ Approximate a complicated joint distribution over many variables with a product of simpler, independent distributions.

Sequence-Level Log Likelihood with Mean Field Decomposition

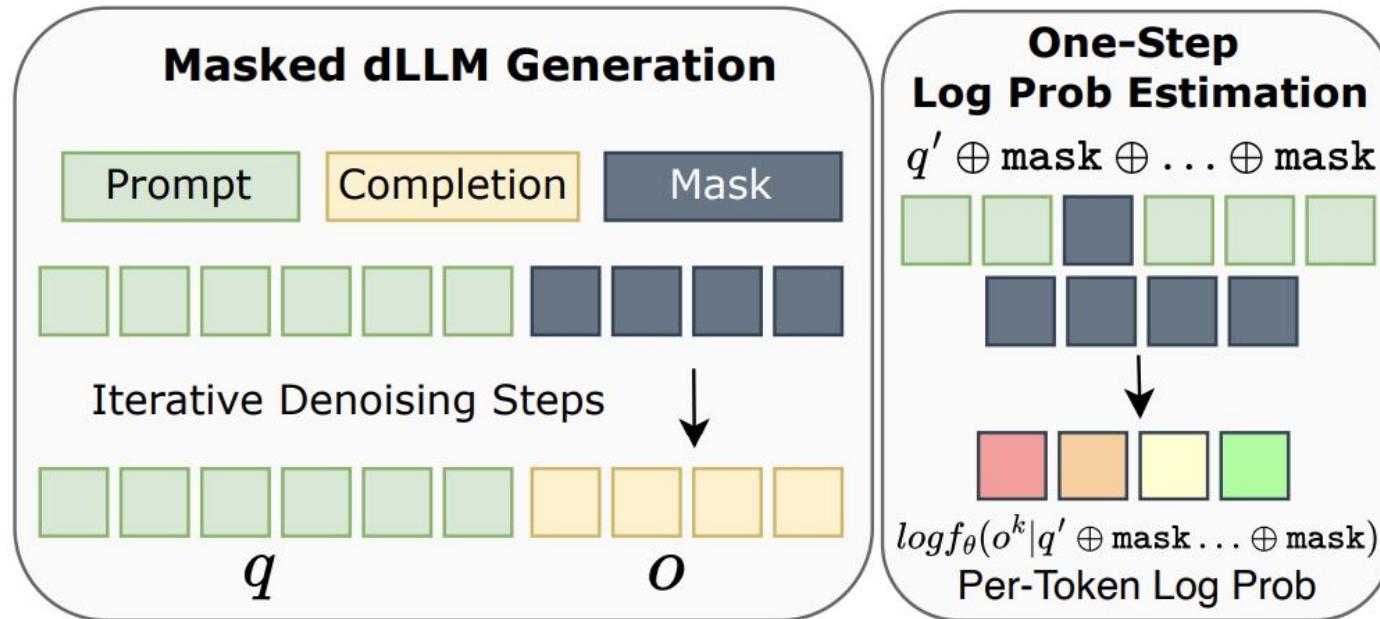
- ❑ The per-token log-probability are also expensive to compute since the decoding process invokes the unmasking predictor f_θ multiple times
- ❑ To efficiently estimate it, we use a simple one-step estimator:
 - ❑ We perturb q where every token is randomly masked out, resulting in q' .

$$\log f_\theta(o^k | q' \oplus \text{mask} \dots \oplus \text{mask})$$

- ❑ We then do one-step unmasking to obtain and use it as an estimation of

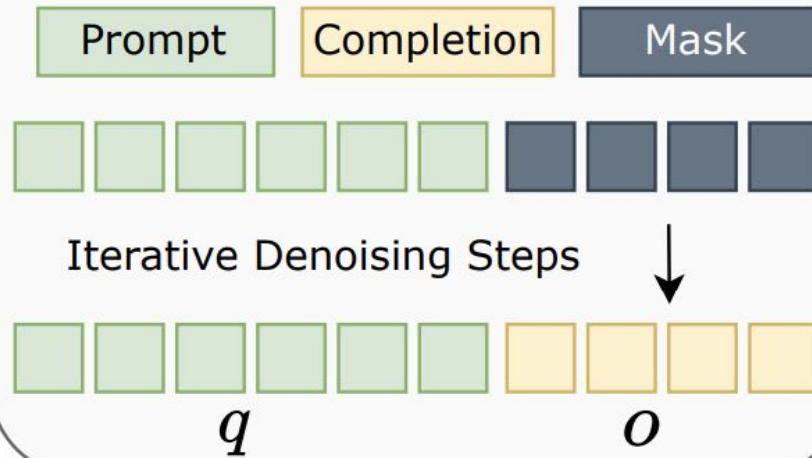
$$\log \pi_\theta(o^k | q), 1 \leq k \leq |o|.$$

Efficient One-Step Log-Prob Estimation



Efficient One-Step Log-Prob Estimation

Masked dLLM Generation



One-Step Log Prob Estimation

$$q' \oplus \text{mask} \oplus \dots \oplus \text{mask}$$



Why apply
random masking
to prompt?

Generation Reuse during Policy Gradient Optimization

- ❑ As other online methods, GRPO is bottlenecked by generation time
- ❑ A key approach to make GRPO more sample-efficient is to reuse generated samples during optimization. Conventionally used with PPO

Algorithm 1 Iterative Group Relative Policy Optimization

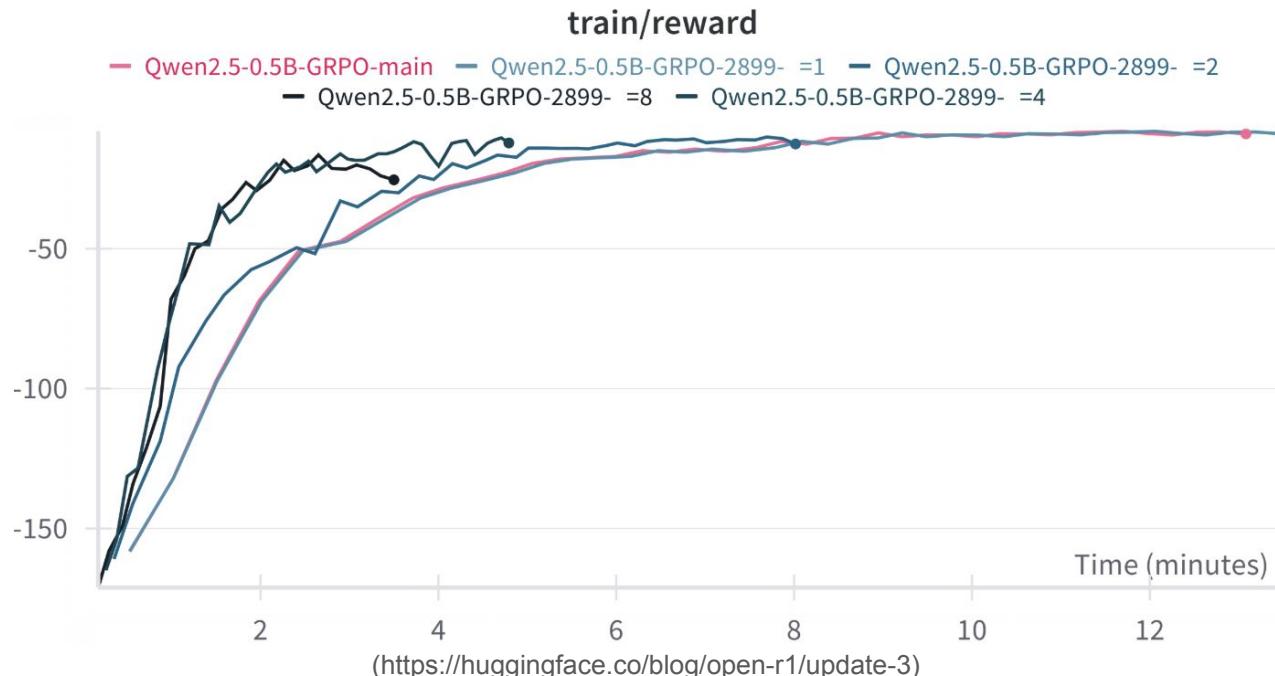
Input initial policy model $\pi_{\theta_{\text{init}}}$; reward models r_φ ; task prompts \mathcal{D} ; hyperparameters ε, β, μ

- 1: policy model $\pi_\theta \leftarrow \pi_{\theta_{\text{init}}}$
- 2: **for** iteration = 1, ..., I **do**
- 3: reference model $\pi_{ref} \leftarrow \pi_\theta$
- 4: **for** step = 1, ..., M **do**
- 5: Sample a batch \mathcal{D}_b from \mathcal{D}
- 6: Update the old policy model $\pi_{\theta_{old}} \leftarrow \pi_\theta$
- 7: Sample G outputs $\{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(\cdot | q)$ for each question $q \in \mathcal{D}_b$
- 8: Compute rewards $\{r_i\}_{i=1}^G$ for each sampled output o_i by running r_φ
- 9: Compute $\hat{A}_{i,t}$ for the t -th token of o_i through group relative advantage estimation.
- 10: **for** GRPO iteration = 1, ..., μ **do**
- 11: Update the policy model π_θ by maximizing the GRPO objective (Equation 21)
- 12: Update r_φ through continuous training using a replay mechanism.

Output π_θ

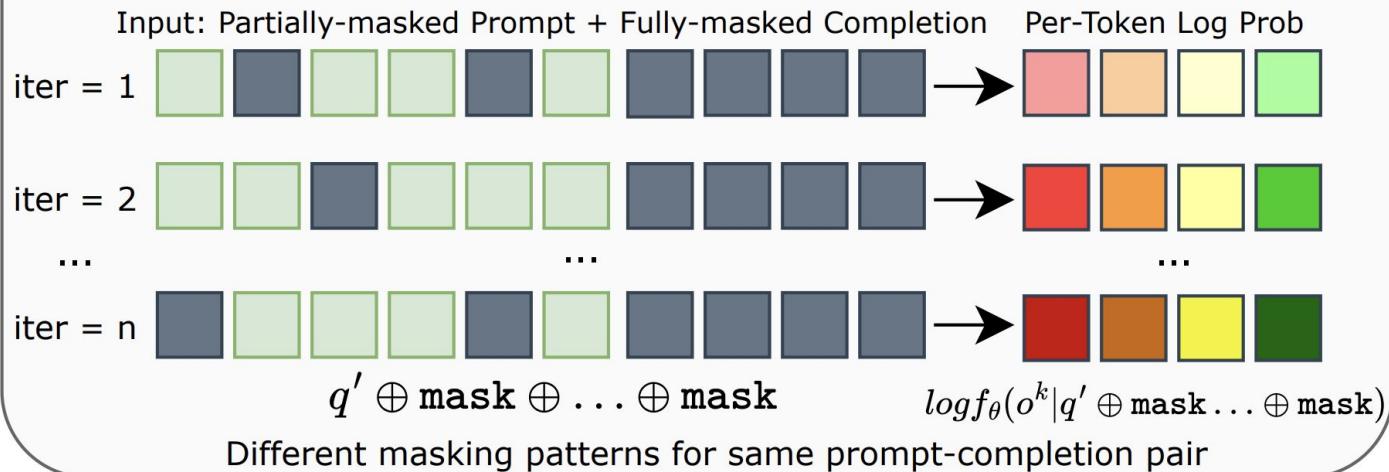
Generation Reuse during Policy Gradient Optimization

- ❑ In practice, scaling mu too large will impact learning, due to overfitting.
- ❑ In TRL's open-r1 update, they suggested using mu=2 or 4 for Qwen2.5



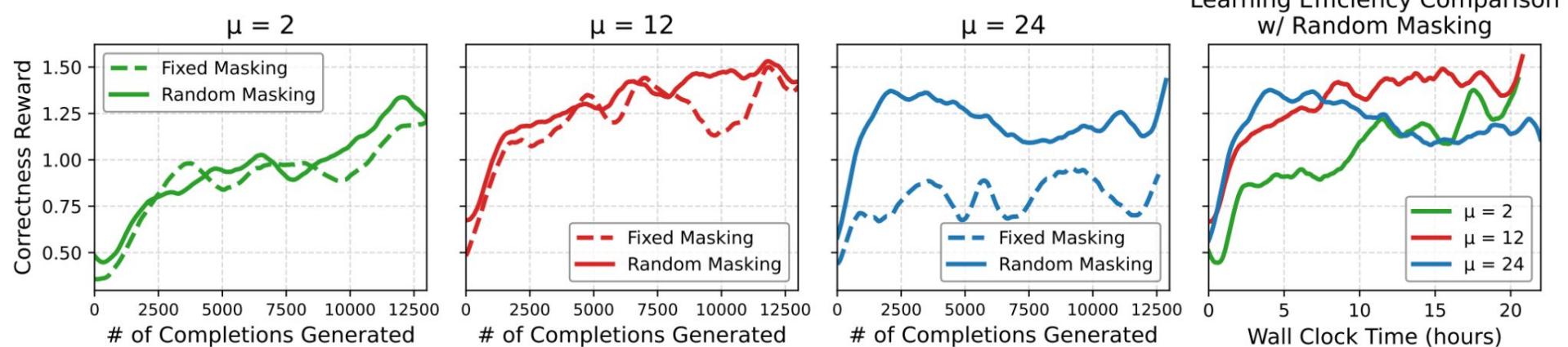
Random Prompt Masking Mitigates Overfitting

Random Prompt Masking for Each Gradient Update Iteration



Applying different random masks to prompt, creating diverse input of the **same prompt + completion pair** at each iteration

Random Prompt Masking enables scaling to larger μ



d1: Framework to Scale dLLM Reasoning

Stage 1: Instilling reasoning behavior through SFT.

SFT on high-quality reasoning dataset.

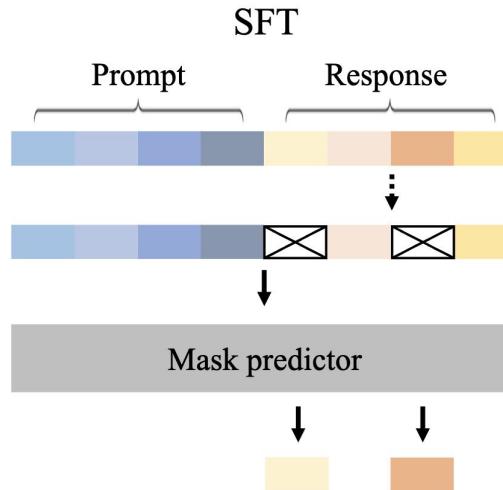
Stage 2: Reinforcing good reasoning traces via RL.

Apply diffu-GRPO on the SFT model.

We instantiate d1 using LLaDA-8B-Instruct as our base model

Stage 1 : Masked SFT on Reasoning Traces from s1K

- ❑ Dataset: s1K (Muennighoff et al., 2025)
 - ❑ 1,000 high-quality reasoning questions
 - ❑ Detailed step-by-step problem-solving traces
 - ❑ Includes verification and backtracking mechanisms



Stage 2 : diffu-GRPO on 6 reasoning tasks

- ❑ Train on each tasks individually:
 - ❑ Math reasoning: GSM8K, MATH500
 - ❑ Logical reasoning: Countdown, 4x4 Sudoku Puzzles
 - ❑ Code: HumanEval, MBPP
- ❑ Reward Functions:
 - ❑ Rule-Based reward.
 - ❑ Correctness
 - ❑ Format
 - ❑ Fraction of passed unit tests for code

Qualitative Example

Masked SFT on reasoning traces from s1k instill LLaDA with self-correction and backtracking behaviors

Question: Jennifer's dog has 8 puppies 3 of which have spots. Brandon's dog has 12 puppies 4 of which have spots. What percentage of all the puppies have spots?

<reasoning> To determine the percentage of all the puppies that have spots, we need to follow these steps:

1. Calculate the total number of puppies from both dogs.
2. Calculate the total number of puppies with spots from both dogs.
3. Determine the percentage of puppies with spots.

First, let's find the total number of puppies from both dogs:

- Jennifer's dog has 8 puppies.
- Brandon's dog has 12 puppies.
- Total number of puppies = $8 + 12 = 20$.

Next, let's find the total number of puppies with spots from both dogs:

- Jennifer's dog has 3 puppies with spots.
- Brandon's dog has 4 puppies with spots.
- Total number of puppies with spots = $3 + 4 = 7$.

Now, we need to determine the percentage of puppies with spots. The formula for percentage is:

$$\text{Percentage} = \left(\frac{\text{Number of puppies with spots}}{\text{Total number of puppies}} \right) \times 100$$

Substituting the values we have:

$$\text{Percentage} = \left(\frac{7}{20} \right) \times 100$$

To simplify the fraction, we find the greatest common divisor (GCD) of 7 and 20, which is 1.

$$\frac{7}{20} = 7 \div 20$$

However, it seems there was a mistake in the calculation. Let's recheck the steps.

1. Total number of puppies = $8 + 12 = 20$.
2. Total number of puppies with spots = $3 + 4 = 7$.
3. Percentage of puppies with spots = $\left(\frac{7}{20} \right) \times 100$.

Let's calculate the percentage correctly:

$$\text{Percentage} = \left(\frac{7}{20} \right) \times 100 = \frac{7 \times 100}{20} = \frac{700}{20} = 35$$

The calculation answer. < /reasoning><answer> 35 < /answer>

Results

Table 1: **Model performance on GSM8K, MATH500, Countdown, and Sudoku Benchmarks:** All models are evaluated with 0-shot prompting, where the generation sequence length varies from 128 to 512. **Green values** indicate best performance and **blue values** indicate second-best performance in each column. The results demonstrate that **d1-LLaDA** consistently outperforms all other models, applying *diffu*-GRPO consistently improves the starting checkpoint, and *diffu*-GRPO alone shows better performance than SFT.

Model / Seq Len	GSM8K (0-shot)			MATH500 (0-shot)			Countdown (0-shot)			Sudoku (0-shot)		
	128	256	512	128	256	512	128	256	512	128	256	512
LLaDA-8B-Instruct	68.7	76.7	78.2	26.0	32.4	36.2	20.7	19.5	16.0	11.7	6.7	5.5
+ SFT	66.5	78.8	81.1	26.2	32.6	34.8	20.3	14.5	23.8	16.5	8.5	4.6
+ diffu-GRPO	72.6	79.8	81.9	33.2	37.2	39.2	33.2	31.3	37.1	18.4	12.9	11.0
+ SFT + diffu-GRPO (d1-LLaDA)	73.2	81.1	82.1	33.8	38.6	40.2	34.8	32.0	42.2	22.1	16.7	9.5

- ❑ diffu-GRPO consistently outperforms both base LLaDA and SFT
- ❑ diffu-GRPO consistently improve over their initialization checkpoint
- ❑ d1 recipe yields the highest gains
- ❑ diffu-GRPO improves reasoning beyond training sequence length (trained with fixed 256 seq len generation)

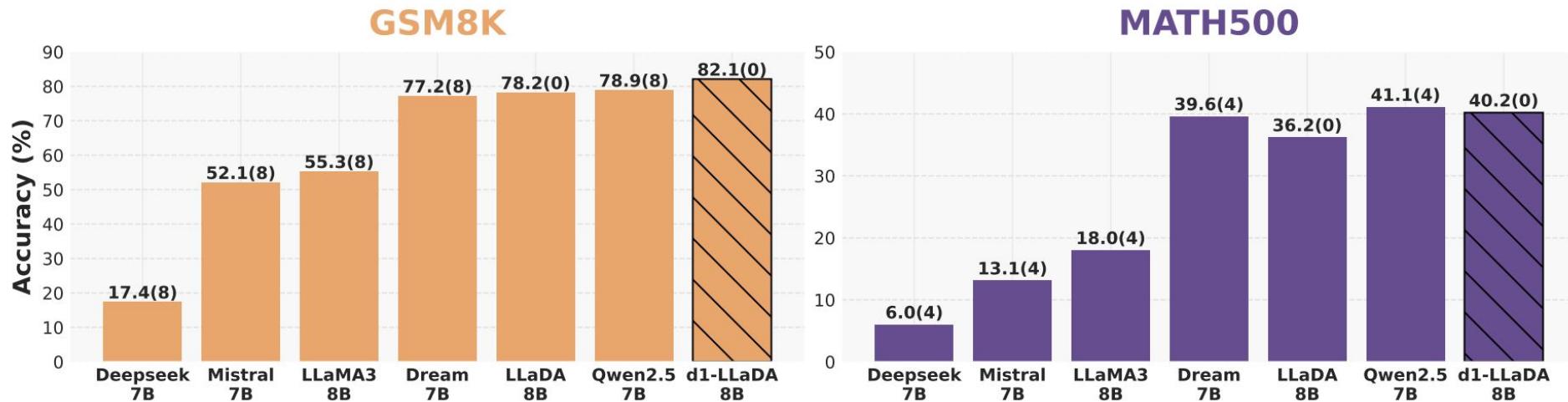
Results : Code

Table 3: **Effectiveness of *diffu*-GRPO on Coding Benchmarks:** Evaluated with and w/o *diffu*-GRPO on HumanEval and MBPP. *diffu*-GRPO consistently improves over initialization checkpoint.

Model / Seq Len	HumanEval			MBPP		
	128	256	512	128	256	512
LLaDA-8B-Instruct	19.5	31.7	33.5	33.1	38.1	41.6
+ <i>diffu</i> -GRPO	24.4	37.8	32.3	40.9	43.2	41.6
Δ (<i>diffu</i> -GRPO gain)	+4.9	+6.1	-1.2	+7.8	+5.1	+0.0
LLaDA-8B-Instruct + SFT (s1k)	11.6	17.7	11.0	25.7	26.5	25.7
+ <i>diffu</i> -GRPO	21.3	27.4	22.0	38.5	32.3	29.2
Δ (<i>diffu</i> -GRPO gain)	+9.7	+9.7	+11.0	+12.8	+5.8	+3.5

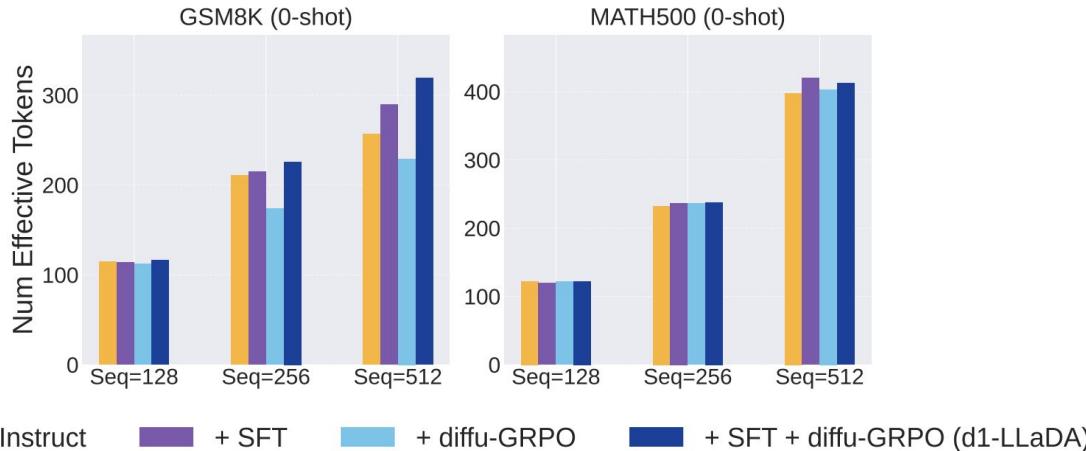
- ❑ Trained on a single code dataset
- ❑ *diffu*-GRPO consistently outperforms over the initialization checkpoint
- ❑ LLaDA + SFT < LLaDA, since the s1k dataset does not contain any datapoints with code

SOTA dLLMs vs similar-sized AR models

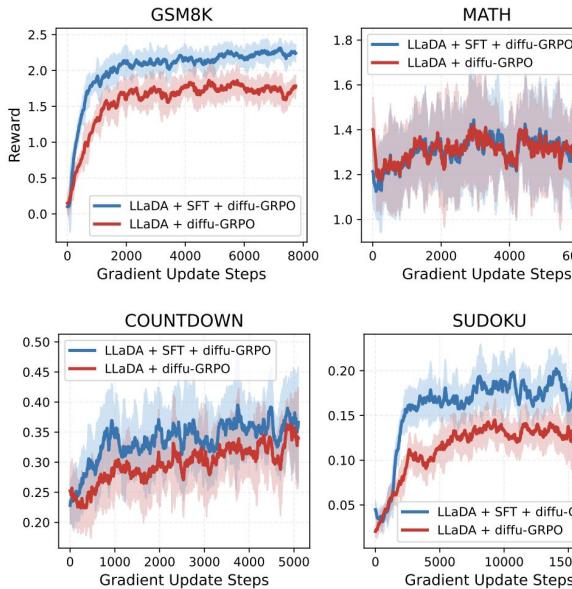


Effective Tokens Usage

- ❑ Effective tokens (# of non-padding, non-EOS tokens) grows with predefined sequence length.
- ❑ Applying SFT increase effective tokens.
- ❑ Applying RL alone decreases effective tokens.



Unified RL Training



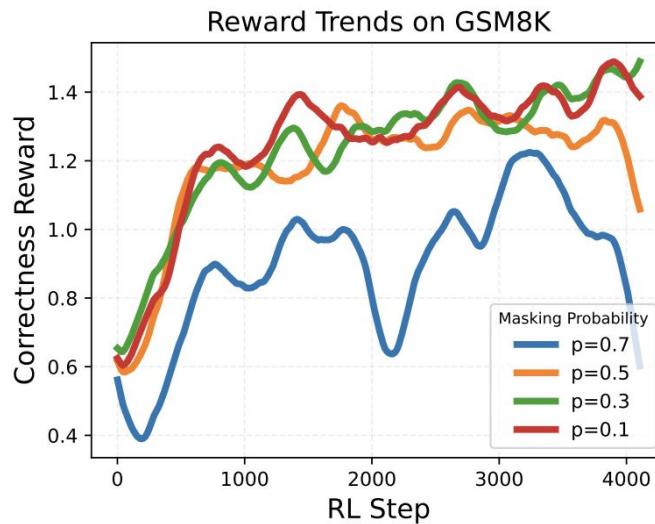
=>
Multi-task
training

Table 2: Unified Model Performance Across Reasoning Tasks: For *diffu*-GRPO and d1-LLaDA variants, a single model was trained on the combined dataset of GSM8K, MATH500, Countdown, and Sudoku. **Green** and **blue values** indicate the best and second-best performance.

Model / Seq Len	GSM8K		MATH500		Countdown		Sudoku	
	128	256	128	256	128	256	128	256
LLaDA-8B-Instruct	68.7	76.7	26.0	32.4	20.7	19.5	11.7	6.7
+ SFT (s1k)	66.5	78.8	26.2	32.6	20.3	14.5	16.5	8.5
+ combined <i>diffu</i> -GRPO	72.4	78.2	30.2	36.6	27.7	19.5	22.9	15.7
combined d1-LLaDA	75.1	81.1	29.8	35.4	30.1	32.8	21.9	15.4

Ablation on Prompt Masking Rate

Lower masking probabilities (0.1, 0.3) show more stable and higher performance, while higher masking probabilities (0.5, 0.7) demonstrate increased instability particularly in later training stages.



A better
masking
strategy?

Limitations and Future Directions

- ❑ **Fixed Generation Length:** LLaDA requires predetermined output length, limiting the model's ability to discover optimal reasoning paths, hindering emergence of advanced reasoning strategies.
 - ❑ Future Direction: Apply diffu-GRPO on Block Diffusion to enable flexible-length generation.
 - ❑ Extend LLaDA to support flexible generation length.

Limitations and Future Directions

❑ SFT Limitation:

- ❑ LLaDA needs to take loss on the PAD tokens to learn to terminate its generation. The amount of padding depends on the batch size, which will affect the performance.
- ❑ We observe that training on truncated sequences will lead LLaDA to generate unfinished response.
- ❑ Future Direction: a more robust SFT algorithm.

Thank You!



[Code](#)



[Paper](#)



[Website](#)