

Beyond Hard Masks: Progressive Token Evolution for Diffusion Language Models

Linhao Zhong^{1*} Linyu Wu^{2*} Bozhen Fang¹ Tianjian Feng¹ Chenchen Jing^{1,3}
Wen Wang¹ Jiaheng Zhang² Hao Chen¹ Chunhua Shen^{1,3†}
¹Zhejiang University ²National University of Singapore ³Zhejiang University of Technology
zhongzero@zju.edu.cn

Abstract

Diffusion Language Models (DLMs) offer a promising alternative for language modeling by enabling parallel decoding through iterative refinement. However, most DLMs rely on hard binary masking and discrete token assignments, which hinder the revision of early decisions and underutilize intermediate probabilistic representations. In this paper, we propose EvoToken-DLM, a novel diffusion-based language modeling approach that replaces hard binary masks with evolving soft token distributions. EvoToken-DLM enables a progressive transition from masked states to discrete outputs, supporting revisable decoding. To effectively support this evolution, we introduce continuous trajectory supervision, which aligns training objectives with iterative probabilistic updates. Extensive experiments across multiple benchmarks show that EvoToken-DLM consistently achieves superior performance, outperforming strong diffusion-based and masked DLM baselines. Our code is available at <https://github.com/aim-uofa/EvoTokenDLM>.

1 Introduction

Diffusion Language Models (DLMs) (Nie et al., 2025; Zhu et al., 2025; Ye et al., 2025; Zhong et al., 2026) frame language generation as an iterative refinement process, enabling parallel decoding in contrast to the strictly sequential nature of autoregressive models. By replacing causal token-by-token generation with diffusion-based refinement (Ho et al., 2020; Nichol and Dhariwal, 2021; Song et al., 2020), DLMs offer an alternative generation paradigm that improves decoding parallelism.

Most existing DLMs adopt a masked diffusion fashion, commonly referred to as masked diffusion language models (MDLMs) (Nie et al., 2025; Zhu et al., 2025; Ye et al., 2025), in which generation

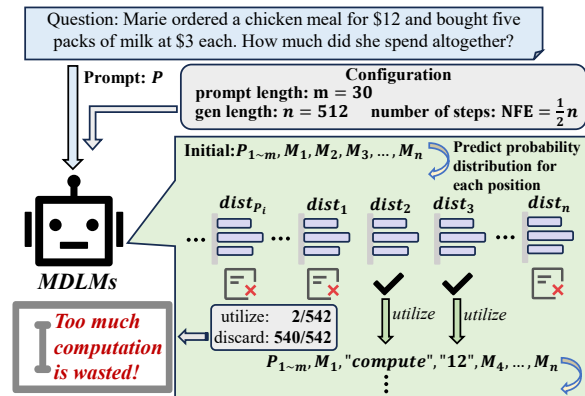


Figure 1: Inefficient utilization of predictions in masked diffusion language models, where distributions are computed for all positions but only a subset are used for decoding. $[M_1, M_2, \dots, M_n]$ denote the initial mask tokens following prompt P , and $dist_i$ represents the predicted probability distribution for the i -th token in the generation sequence. In this example, the total sequence of 542 tokens consists of 30 prompt tokens and 512 generated tokens, while only two positions are updated per step.

is performed by maintaining a partially masked sequence and progressively replacing masked positions with discrete token assignments, enabling the simultaneous decoding of multiple tokens. To further improve the practicality of DLMs, recent work has introduced KV-caching mechanisms (Wu et al., 2025b,a; Ma et al., 2025a; Chen et al., 2025) that reuse hidden states across refinement steps to reduce redundant computation. In parallel, blockwise diffusion models (Wang et al., 2025b; Cheng et al., 2025; Bie et al., 2025) apply diffusion-based generation within local token blocks while preserving autoregressive dependencies across blocks, combining global causal coherence with local parallel efficiency.

However, most MDLMs rely on hard binary masking with discrete token assignments. Once a token is decoded, it is treated as final and excluded from further refinement, resulting in an abrupt tran-

*Equal Contribution.

†Corresponding Author.

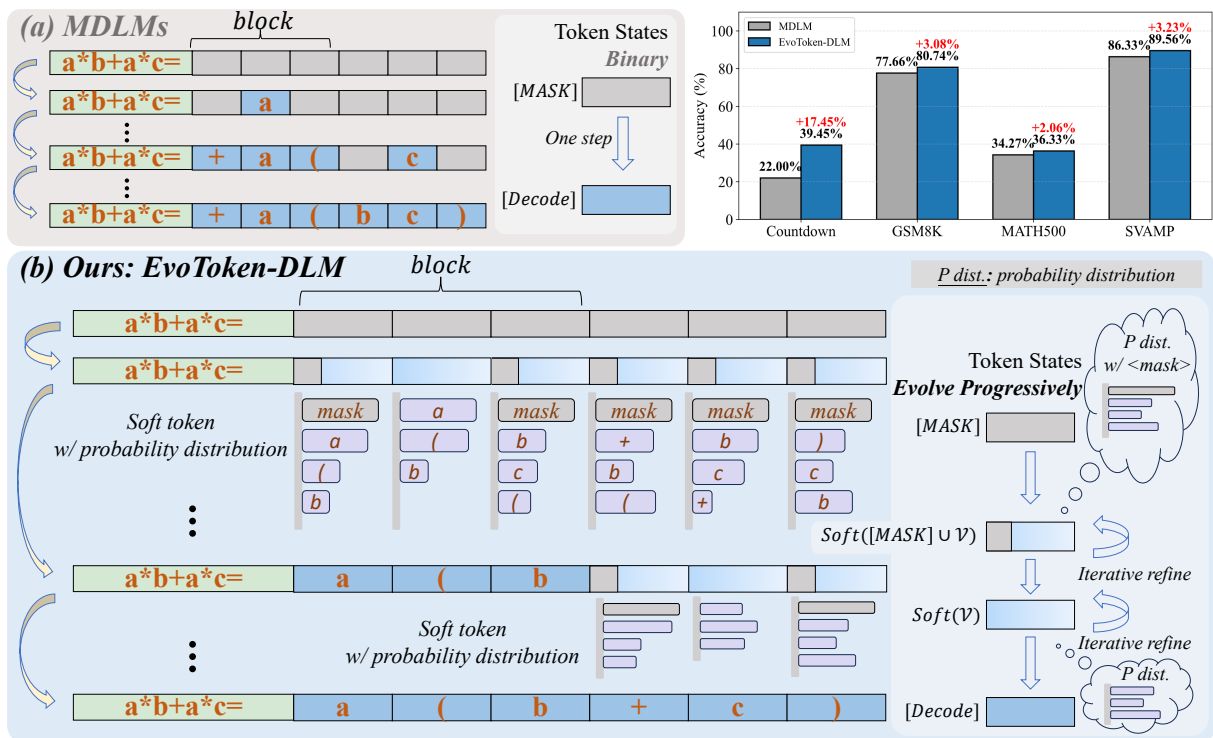


Figure 2: Comparison between MDLMs and EvoToken-DLM. (a) Standard MDLMs employ only two token states, alternating between $\langle \text{mask} \rangle$ and discrete decoded tokens, leading to abrupt mask-to-token transitions. (b) EvoToken-DLM introduces soft tokens represented by probability distributions and four token states, enabling tokens to evolve progressively through iterative refinement. The top-right panel illustrates a quantitative comparison between the two approaches under the same settings based on LLaDA-Instruct-8B.

sition from uncertainty to determinism. This irreversibility limits the model’s ability to revise early decisions and undermines the iterative refinement paradigm of diffusion-based language modeling. In addition, as illustrated in Figure 1, although MDLMs compute token distributions for all positions at each refinement step, only a small subset of positions are updated, while the remaining probabilistic information is discarded.

In this work, we propose **EvoToken-DLM**, a diffusion-based language modeling approach that replaces hard binary masks with evolving soft token distributions. Instead of predicting discrete tokens in a single step, EvoToken-DLM represents each token as a probability distribution over the vocabulary and iteratively refines it throughout the diffusion process. As illustrated in Figure 2, token decoding becomes a progressive and continuous evolution:

$$\begin{aligned} [\text{MASK}] &\rightarrow \text{Soft}([\text{MASK}] \cup \mathcal{V}) \\ &\rightarrow \text{Soft}(\mathcal{V}) \rightarrow [\text{Decode}]. \end{aligned} \quad (1)$$

This evolution gradually transitions tokens from masked uncertainty to mask-aware soft token distributions, then to fully soft token distributions, and

finally to discrete outputs. By allowing token representations to evolve across refinement steps before being finalized, EvoToken-DLM enables smooth and revisable decoding, mitigating premature decisions induced by hard masking.

To support this progressive refinement during training, we introduce **continuous trajectory supervision**, which aligns training objectives with iterative probabilistic updates along the diffusion trajectory. EvoToken-DLM requires no modification to the underlying model architecture and can be readily adapted from existing MDLMs. Moreover, it is fully compatible with KV-caching and naturally extends to blockwise diffusion settings, demonstrating broad applicability. Extensive experiments across multiple benchmarks show that EvoToken-DLM consistently outperforms strong MDLM baselines.

Our main contributions are:

- We propose EvoToken-DLM, a diffusion-based language modeling approach that replaces hard binary masks with evolving soft token distributions, enabling a staged and revisable decoding process throughout diffusion.

- We introduce a continuous trajectory supervision-based training strategy that aligns model optimization with iterative probabilistic token refinement along the diffusion trajectory, effectively supporting progressive token evolution.
- We demonstrate that EvoToken-DLM integrates seamlessly with KV-caching and extends naturally to blockwise diffusion architectures. Extensive experiments across diverse model backbones, datasets, and inference configurations show consistent and robust improvements over strong MDLM baselines, highlighting EvoToken-DLM as a general and effective enhancement.

2 Preliminaries on MDLMs

Masked diffusion language models operate under a masked diffusion paradigm. The generation process consists of two main stages: a forward corruption process and a learned reverse denoising process.

Forward Process. Given an original text sequence $X^0 = (x_1^0, \dots, x_N^0)$ of N tokens, the forward process gradually corrupts it into a noisy sequence X^t over a time schedule $t \in [0, T]$. This corruption is typically achieved by independently replacing each token with a special mask token with probability $\frac{t}{T}$:

$$q(x_i^t | x_i^0) = \begin{cases} 1 - \frac{t}{T}, & \text{if } x_i^t = x_i^0, \\ \frac{t}{T}, & \text{if } x_i^t = \langle \text{mask} \rangle \end{cases}$$

At $t = T$, the sequence X^T becomes fully masked.

Reverse Process. MDLMs learn a parameterized model $p_\theta(X^0 | X^t)$ to reverse the forward process. The model predicts all masked tokens simultaneously at each inference step, enabling high-speed, parallel generation from the fully masked sequence X^T to the original text X^0 .

In practice, at each decoding step, the model selects a subset of masked tokens to finalize based on their predicted confidence rather than decoding all tokens at once. Masked tokens not selected in the current step remain in the mask state and will be decoded in subsequent steps. Furthermore, sequences are partitioned into discrete blocks that are processed in a sequential manner, where the model advances to the next block only upon the complete refinement of all masked tokens within the current one.

3 From Discrete to Continuous: A Continuous Relaxation Perspective

Continuous Relaxation. Let $\mathcal{V} = \{1, \dots, V\}$ denote the vocabulary of size V . We define the discrete token space as the set of one-hot vectors $\mathcal{X} = \{\delta_1, \dots, \delta_V\} \subset \{0, 1\}^V$. Associated with the vocabulary is an embedding matrix $\mathbf{U} \in \mathbb{R}^{V \times D}$. The embedding function maps a token index i to a continuous vector $\mathbf{e} = \mathbf{U}_i$. We denote the continuous embedding space as the convex hull of the token embeddings: $\mathcal{E} = \text{Conv}(\mathbf{U}) \subset \mathbb{R}^D$. A soft token is any vector $\tilde{\mathbf{e}} \in \mathcal{E}$ that can be expressed as $\tilde{\mathbf{e}} = \mathbf{U}^\top \mathbf{p}$, where $\mathbf{p} \in \Delta^{V-1}$ lies on the probability simplex. This formulation relaxes the categorical selection into a continuous domain.

Iterative Refinement in Continuous Domain.

Unlike standard MDLMs which predict $p_\theta(X^0 | X^T)$ iteratively over the discrete vocabulary, our method models the reverse process as an iterative refinement loop in the continuous domain \mathcal{E} . Specifically, let X^T denote the masked input sequence and \mathbf{E}^T be its corresponding embeddings, where each element of \mathbf{E}^T belongs to \mathcal{E} . We introduce auxiliary token states \mathbf{Z}^T to enable continuous token evolution. The refinement process is governed by a transition function Φ , which recursively updates both the continuous embeddings and the token states: $(\mathbf{E}^{t-1}, \mathbf{Z}^{t-1}) = \Phi(\mathbf{E}^{\geq t}, \mathbf{Z}^{\geq t})$. Through successive applications of Φ , the model progressively purifies the noisy input until it reaches the terminal $\mathbf{E}^0, \mathbf{Z}^0$. Finally, \mathbf{E}^0 is mapped back to the discrete domain to produce the output sequence X^0 .

4 EvoToken-DLM

4.1 Progressive Inference with EvoToken-DLM

We formally define the progressive inference procedure of EvoToken-DLM as follows. Given a prompt P , the objective is to generate a response of length N . The output is partitioned into $M = N/B$ discrete blocks, each of size B . The sequence X is constructed by concatenating the prompt P with N tokens, denoted as $X = (P, x_1, x_2, \dots, x_N)$, where each token x_i is characterized by a pair (e_i, z_i) , comprising continuous embeddings e_i and a token state z_i . Initially, all target positions are initialized as mask tokens, where $z_i = [\text{MASK}]$ for all $i \in \{1, \dots, N\}$, and the corresponding embedding sequence is represented

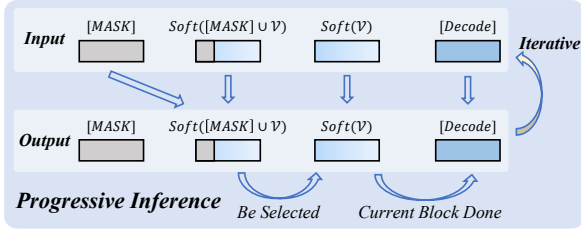


Figure 3: Progressive step-wise token update with block-wise decoding in EvoToken-DLM.

as $\mathbf{E} = (e_P, e_1^{\langle \text{mask} \rangle}, \dots, e_N^{\langle \text{mask} \rangle})$. During the evolution process, each token x_i transitions through a state space consisting of four distinct stages:

[MASK], $\text{Soft}([\text{MASK}] \cup \mathcal{V})$, $\text{Soft}(\mathcal{V})$, [Decode],

where \mathcal{V} is the vocabulary.

Token Prediction. At each inference step, we input embeddings \mathbf{E} into the model to obtain a predicted distribution $\{p_i^c\}_{c=1}^{|\mathcal{V}|}$ over the vocabulary for each position i . We retain the top- K probabilities and renormalize them to obtain $\{\hat{p}_i^c\}_{c=1}^K$, along with their corresponding tokens $\{\hat{v}_i^c\}_{c=1}^K \subseteq \mathcal{V}$. *Soft embeddings* are then computed as:

$$\begin{aligned} e_i^{\text{dist}} &= \sum_{c=1}^K \hat{p}_i^c \cdot e^{\hat{v}_i^c}, \\ e_i^{\text{dist+M}} &= \alpha e_i^{\langle \text{mask} \rangle} + (1 - \alpha) e_i^{\text{dist}}, \end{aligned} \quad (2)$$

where $\alpha \in [0, 1]$ controls the mixing ratio of the mask embedding.

Embedding Assignment by Token State. For token x_i , its newly generated embeddings at the current step is assigned based on its current state:

$$e_i = \begin{cases} e_i^{\langle \text{mask} \rangle}, & z_i = [\text{MASK}] \\ e_i^{\text{dist+M}}, & z_i = \text{Soft}([\text{MASK}] \cup \mathcal{V}) \\ e_i^{\text{dist}}, & z_i = \text{Soft}(\mathcal{V}) \\ e^{v_i}, & z_i = [\text{Decode}] \end{cases} \quad (3)$$

where v_i is selected as the token in the vocabulary with the highest confidence among all historical predictions made after x_i enters the $\text{Soft}(\mathcal{V})$ state.

Step-wise Token Update. By default, tokens in the [MASK] state transition to $\text{Soft}([\text{MASK}] \cup \mathcal{V})$, whereas tokens already in the $\text{Soft}([\text{MASK}] \cup \mathcal{V})$, $\text{Soft}(\mathcal{V})$, or [Decode] states retain their current state. At each step, a subset of tokens currently in the [MASK] or $\text{Soft}([\text{MASK}] \cup \mathcal{V})$ states in the current block is selected to transition to the $\text{Soft}(\mathcal{V})$ state.

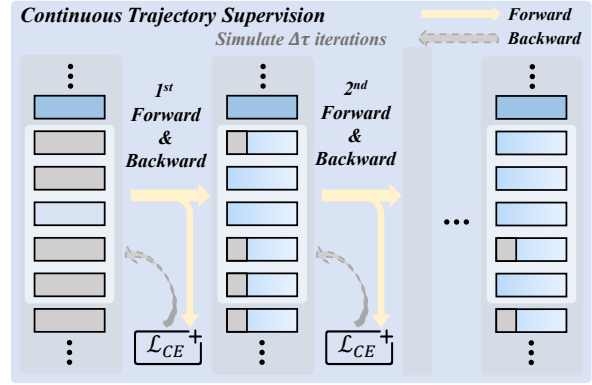


Figure 4: Continuous trajectory supervision by performing $\Delta\tau$ consecutive refinement steps during training and applying supervision at each step, aligning the training objective with the inference process.

Let S denote the set of these selected tokens. The complete update rule is formalized as:

$$z_i \leftarrow \begin{cases} \text{Soft}([\text{MASK}] \cup \mathcal{V}), & z_i \in \{\text{Soft}([\text{MASK}] \cup \mathcal{V}), \\ & [\text{MASK}]\} \text{ and } x_i \notin S \\ \text{Soft}(\mathcal{V}), & x_i \in S \text{ or } z_i = \text{Soft}(\mathcal{V}) \\ [\text{Decode}], & z_i = [\text{Decode}] \end{cases} \quad (4)$$

Blockwise Decoding. Let \mathcal{B} denote the set of tokens in the current block. Once all tokens in \mathcal{B} reach the $\text{Soft}(\mathcal{V})$ state, they are simultaneously converted to the [Decode] state:

$$\begin{aligned} z_i &\leftarrow [\text{Decode}], \quad \forall x_i \in \mathcal{B}, \\ &\text{if all tokens } x_j \in \mathcal{B} \text{ are in the } \text{Soft}(\mathcal{V}) \text{ state.} \end{aligned} \quad (5)$$

As illustrated in Figure 3, combining step-wise token update with blockwise decoding, EvoToken-DLM allows each token to gradually refine its representation from [MASK] to final [Decode] through progressive token evolution. The detailed algorithm for progressive inference is provided in Appendix B.1.

4.2 Continuous Trajectory Supervision

Unlike conventional masked diffusion frameworks, EvoToken-DLM employs a progressive evolution mechanism. In this approach, the current states and embeddings of the tokens are conditioned on the cumulative history of the preceding refinements. This temporal dependency renders standard single-step denoising objectives inapplicable, necessitating a specialized training paradigm that models the trajectory of token evolution. We propose continuous trajectory supervision, a training strategy that aligns model optimization with iterative probabilistic token refinement along the diffusion trajectory \mathcal{T} , as illustrated in Figure 4. This approach ensures consistency from training to inference.

Initialization and Masking Strategy. Given a sequence comprising a prompt and a target response, we sample a contiguous segment of length L from the response as the current training block. To align with the blockwise inference procedure, tokens preceding this block are set to the ground truth, while tokens after this block are replaced with [MASK]. Within the selected block, we randomly mask a subset of tokens to initialize the state $X^{(0)}$.

Trajectory Unrolling. Starting from $X^{(0)}$, we simulate $\Delta\tau$ consecutive refinement steps to sample the trajectory:

$$X^{(i)}, \mathcal{L}^{(i)} = \text{Model}(X^{(i-1)}), \quad \forall i = 1, \dots, \Delta\tau, \quad (6)$$

where each forward pass produces probability distributions, updated continuous embeddings, and updated token states according to the progressive inference rules described in Section 4.1.

Cumulative Trajectory Loss. At each step i , we compute a supervised loss $\mathcal{L}^{(i)}$ against the ground-truth tokens within the current block. Rather than backpropagating only through the final step, we perform a backward pass for every forward step:

$$\nabla_{\theta} \mathcal{L}^{(i)}, \quad i = 1, \dots, \Delta\tau. \quad (7)$$

By explicitly simulating the progressive refinement during training, continuous trajectory supervision aligns the learning objective with the inference behavior of EvoToken-DLM. The detailed algorithm for continuous trajectory supervision is provided in Appendix B.2, and an ablation on the trajectory length $\Delta\tau$ is reported in Appendix E.3.

4.3 Extension to Blockwise Diffusion

EvoToken-DLM naturally extends to blockwise diffusion by partitioning the sequence into consecutive blocks. Within each block, tokens undergo full progressive refinement before the generation moves to the next, preserving the global autoregressive structure while enabling intra-block parallelism. For training, we adapt continuous trajectory supervision to this setting. Following existing frameworks (Wang et al., 2025b; Cheng et al., 2025; Bie et al., 2025), we exploit block-level causal dependencies to enable independent, parallel training of blocks. Within each block, the continuous trajectory supervision procedure simulates $\Delta\tau$ refinement steps for supervision.

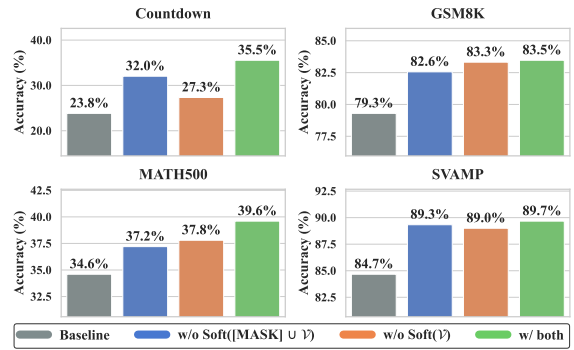


Figure 5: Ablation study on the presence of intermediate refinement states in EvoToken-DLM.

5 Experiments

5.1 Experimental Setup

We employ LLaDA-Instruct-8B (Nie et al., 2025) as our primary backbone for fine-tuning. To evaluate cross-model consistency, we also apply our method to LLaDA-1.5 (Zhu et al., 2025), Dream-Instruct-7B (Ye et al., 2025) and D2F-LLaDA (Wang et al., 2025b), the last of which serves as the base model for our blockwise diffusion experiments. For fine-tuning, we utilize the S1K dataset (Muennighoff et al., 2025) and train the pretrained model for a default duration of 10k steps using continuous trajectory supervision. Evaluations are performed across several mathematical and reasoning benchmarks, including Countdown (Pan et al., 2025), GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2023), and SVAMP (Patel et al., 2021). More details are presented in Appendix C.

5.2 Evaluation Results

Main Performance Comparison. Table 1 compares EvoToken-DLM against the original LLaDA-Instruct-8B and the FT-baseline across multiple reasoning benchmarks. EvoToken-DLM predominantly surpasses both baselines, exhibiting substantial performance leaps under varying configurations. Specifically, at $\frac{NFE}{Gen Len} = 1$, our method yields average accuracy gains of 17.45% on Countdown, 3.08% on GSM8K, 2.06% on MATH500, and 3.23% on SVAMP compared to the original model. These results underscore the superiority of our soft token evolution framework in enhancing reasoning capabilities and generation quality. Additional results with different block sizes are presented in Appendix E.5, and qualitative comparisons are presented in Appendix E.7. Appendix E.2

Table 1: Performance comparison on the Countdown, GSM8K, MATH500 and SVAMP datasets across various generation lengths and NFEs based on LLaDA-Instruct-8B. EvoToken-DLM is initialized from LLaDA-Instruct-8B and fine-tuned for 10k steps using continuous trajectory supervision. Comparisons are conducted against both the baseline model and the sft-baseline.

$\frac{NFE}{Gen Len}$	Method	Countdown				GSM8K				MATH500				SVAMP			
		128	256	512	Avg.	128	256	512	Avg.	128	256	512	Avg.	128	256	512	Avg.
1	Baseline	21.48	23.83	20.70	22.00	70.20	79.30	83.47	77.66	28.80	34.60	39.40	34.27	88.33	84.67	86.00	86.33
	FT-Base (10k FT)	33.20	21.48	19.53	24.74	71.04	82.11	82.56	78.57	26.60	36.20	40.40	34.40	87.67	89.67	89.67	89.00
	EvoToken (10k FT)	39.84	35.55	42.97	39.45	74.30	83.47	84.46	80.74	28.40	39.60	41.00	36.33	89.00	89.67	90.00	89.56
		+18.36	+11.72	+22.27	+17.45	+4.10	+4.17	+0.99	+3.08	-0.40	+5.00	+1.60	+2.06	+0.67	+5.00	+4.00	+3.23
$\frac{1}{2}$	Baseline	26.17	16.41	16.80	19.79	67.55	77.63	79.83	75.00	26.60	32.20	33.20	30.67	86.00	86.67	84.00	85.56
	FT-Base (10k FT)	28.12	16.80	16.41	20.44	63.91	78.62	79.00	73.84	22.60	31.20	34.00	29.27	85.00	87.00	89.33	87.11
	EvoToken (10k FT)	34.77	30.08	30.08	31.64	73.54	82.03	81.80	79.12	29.20	36.40	37.40	34.33	89.33	92.33	89.67	90.44
		+8.60	+13.67	+13.28	+11.85	+5.99	+4.40	+1.97	+4.12	+2.60	+4.20	+4.20	+3.66	+3.33	+5.66	+5.67	+4.88
$\frac{1}{4}$	Baseline	17.19	15.62	16.41	16.41	59.14	68.23	66.57	64.65	23.40	26.60	29.60	26.53	81.00	77.33	75.00	77.78
	FT-Base (10k FT)	14.06	13.67	9.77	12.50	49.05	62.17	61.87	57.70	16.20	19.60	23.20	19.67	66.67	75.33	72.00	71.33
	EvoToken (10k FT)	23.05	16.02	12.11	17.06	64.82	75.74	72.33	70.96	23.60	31.00	31.20	28.60	78.33	83.33	81.33	81.00
		+5.86	+0.40	-4.30	+0.65	+5.68	+7.51	+5.76	+6.31	+0.20	+4.40	+1.60	+2.07	-2.67	+6.00	+6.33	+3.22

shows that a fixed $\alpha = 0.7$ already performs strongly across almost all settings.

Importance of Intermediate States. Figure 5 illustrates the ablation study on the presence of intermediate refinement states. The performance drop observed when removing these states confirms that the gradual transition from mask to soft-token is essential for the model to iteratively refine its predictions.

Qualitative Visualization. We provide a qualitative visualization of the inference process in Figure 6. By tracing the evolution of a selected subsequence, we observe how initial uncertain tokens progressively converge into precise and coherent results. This visualization confirms that EvoToken-DLM effectively implements a progressive refinement mechanism, allowing the model to iteratively calibrate its predictions within the diffusion framework. Additional qualitative visualization results of the inference process are presented in Appendix E.6.

Compatibility with KV-Caching. To further demonstrate the practical efficiency of EvoToken-DLM, we integrate it with the KV-caching mechanism as proposed in Fast-dLLM (Wu et al., 2025b). This integration is essential to ensure that our adaptive token evolution does not interfere with the accelerated inference pipelines of DLMs. As reported in Table 2, we evaluate the performance of EvoToken-DLM equipped with KV-caching against the baseline on Countdown. The results indicate that our method consistently maintains superior performance with KV-caching across var-

Table 2: Performance comparison on Countdown with KV-caching. EvoToken integrates seamlessly with KV-caching mechanism.

Gen Len	$\frac{NFE}{Gen Len}$	Baseline+Cache	EvoToken+Cache
128	1	21.88	36.33 (+14.45)
	1/2	23.05	28.52 (+5.47)
	1/4	7.42	18.75 (+11.33)
256	1	21.48	28.52 (+7.04)
	1/2	19.92	25.00 (+5.08)
	1/4	9.77	12.50 (+2.73)

ious computational budgets, proving its seamless integration with the KV-caching mechanism.

Robustness across Thresholds. We further adopt another parallel generation strategy using the confidence threshold proposed in Fast-dLLM (Wu et al., 2025b) to replace the fixed NFE setting. This allows for a more flexible allocation of computational resources during inference. We analyze the sensitivity of our method to different thresholds on the MATH500 dataset, with KV-caching enabled. As illustrated in Figure 7, EvoToken consistently outperforms the binary masking baseline given the same average token budget per step. These results demonstrate the superior adaptability of EvoToken-DLM. Detailed threshold-based trade-offs for different α are reported in Appendix E.2.

Generalization Across Models. We evaluate the transferability of our approach by applying continuous trajectory supervision to the Dream-Instruct-7B pretrained base. As shown in Figure 8, the improvements observed in the primary model consistently generalize to the alternative backbone. This con-

<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	Step 0
In 0.56	the 0.73	first 0.79	'' 0.83	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 1
Lily 0.34	runs 0.14	'' 0.11	4 0.11	4 0.34	hours 0.33	'' 0.36	2 0.21	1 0.29	'' 0.23	1 0.19	2 0.22	kilometers 0.21	'' 0.21	hour 0.22	'' 0.21	'' 0.21	Step 2
The 0.10	can 0.12	run 0.09	1 0.05	1 0.09	2 0.09	kilometers 0.08	runs 0.17	1 0.11	'' 0.18	2 0.19	1 0.14	'' 0.15	per 0.15	hour 0.17	4 0.16	hour 0.13	Step 3
In 0.86	the 0.99	first 0.99	'' 0.97	4 0.97	hours 0.99	'' 0.96	<mask> 0.50	runs 0.80	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 4
For 0.12	runs 0.01	'' 0.01	\$ 0.02	hours 0.02	\$ 0.01	at 0.03	Lily 0.39	can 0.16	'' 0.29	1 0.40	2 0.32	2 0.20	\ 0.22	'' 0.20	'' 0.28	'' 0.28	
During 0.02	The 0.00	the 0.00	part 0.01	of 0.01	4 0.01	of 0.01	she 0.09	covers 0.04	\$ 0.15	'' 0.06	1 0.19	\ 0.19	times 0.17	times 0.18	\ 0.12	\ 0.12	
In	the	first	''	4	hours	,	Lily	runs	\$	1	2	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	
												\ 0.32	times 0.28	'' 0.25	'' 0.24	'' 0.24	
												km 0.10	per 0.12	\ 0.17	\ 0.15	\ 0.15	
												2 0.08	\ 0.10	hour 0.07	4 0.11	4 0.11	
In	the	first	''	4	hours	,	Lily	runs	\$	1	2	\ 0.89	times 0.76	'' 0.86	<mask> 0.50	<mask> 0.50	
												km 0.07	per 0.16	'' 0.07	4 0.38	4 0.38	
												kilometers 0.04	'' 0.08	hour 0.07	= 0.08	= 0.08	
														1 0.04	1 0.04	1 0.04	

Figure 6: An illustrative example of EvoToken-DLM during inference, showing intermediate refinement states for a selected subsequence across successive steps. The block size is set to 12, and the refinement process for the first 16 output tokens is visualized. For each position, only the top $K = 3$ most probable tokens are retained.

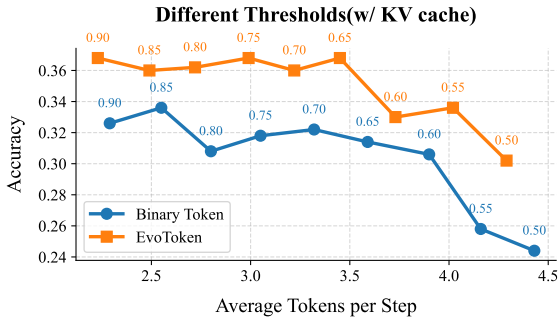


Figure 7: Comparison between EvoToken and binary masking baseline on MATH500 with KV-caching and different confidence thresholds. EvoToken consistently achieves higher accuracy than baseline across various thresholds under the same average tokens per step.

sistency underscores that EvoToken-DLM serves as a general enhancement for diffusion language models. Additional results based on LLaDA-1.5 are presented in Appendix E.1.

Extension to Blockwise Diffusion. To further validate the versatility of EvoToken, we extend our method to the blockwise diffusion framework, specifically using D2F-LLaDA as the base model. As illustrated in Figure 9, EvoToken significantly outperforms the binary masking baseline, proving its robustness and adaptability.

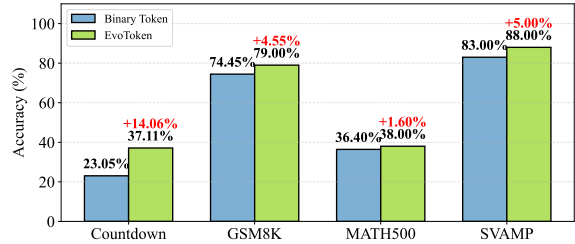


Figure 8: Comparison between EvoToken and the binary masking baseline based on another pretrained model Dream-Instruct-7B. We apply continuous trajectory supervision and evaluate performance on various datasets.

Inference Efficiency. Figure 10 illustrates that EvoToken-DLM introduces only negligible latency compared to standard MDLMs, with the marginal overhead stemming primarily from the element-wise addition of token embeddings during the refinement process. Such minimal overhead, alongside substantial improvements, makes EvoToken-DLM highly practical for real-world deployment. A direct comparison of training-time and peak-memory overhead is provided in Appendix E.4.

Impact of Top- K Filtering. In Figure 11, we analyze the sensitivity of the model to different top- K settings during the refinement process. EvoToken-DLM shows robust performance across a wide range of K values, consistently outperforming the baseline. Additional detailed ablation study on the

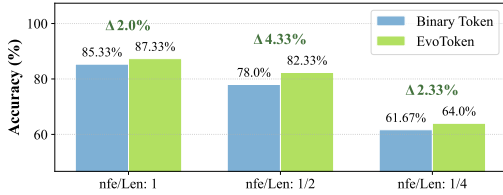


Figure 9: Comparison between EvoToken and the binary masking baseline based on blockwise diffusion model D2F-LLaDA. We apply continuous trajectory supervision and evaluate performance on SVAMP.

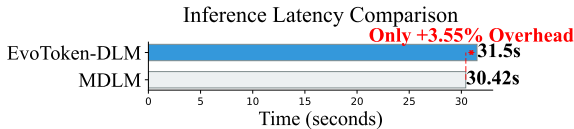


Figure 10: EvoToken-DLM exhibits competitive inference efficiency, introducing minimal latency penalties relative to standard MDLM architectures.

mixing ratio α is provided in Appendix E.2.

More analyses regarding rapid adaptation from pre-trained MDLMs are presented in Appendix D.

6 Related Work

6.1 MDLMs

Masked Diffusion Language Models (MDLMs) adapt the diffusion paradigm (Ho et al., 2020; Podell et al., 2023; Rombach et al., 2022; Song and Ermon, 2019; Song et al., 2020) to discrete text generation. Building on foundational work in noise scheduling and objectives (Austin et al., 2021; Sahoo et al., 2024; Bie et al., 2025; Yang et al., 2025; Nie et al., 2025; Wu et al., 2025c; Wang et al., 2025a; Zhong et al., 2026), recent large-scale models like LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) have shown that MDLMs can match autoregressive baselines in complex reasoning. Despite their potential, the iterative denoising process remains computationally expensive. Current research addresses this through two main efficiency frontiers: developing specialized caching mechanisms and architecting blockwise generative processes.

KV-Cache Optimization for MDLMs. Standard KV caching is incompatible with bidirectional MDLMs, often necessitating recomputation per step. Fast-dLLM (Wu et al., 2025b,a) mitigates this via block-wise approximate caching, while others (Ma et al., 2025b; Shen et al., 2025; Bao et al., 2025) refine generation coherence. Furthermore, dKV-Cache (Ma et al., 2025a) and dLLM-

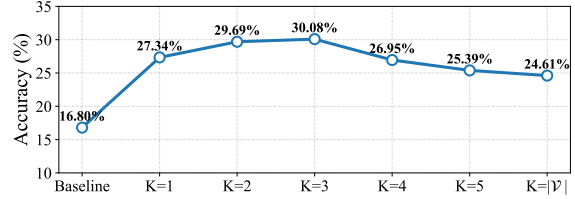


Figure 11: Performance comparison between the baseline and EvoToken-DLM with different top- K settings on Countdown.

Cache (Liu et al., 2025c) utilize selective token updates, and Sparse-dLLM (Song et al., 2025) applies dynamic eviction to reduce long-context memory overhead.

Blockwise Diffusion Language Models. Blockwise MDLMs (Han et al., 2023; Arriola et al., 2025; Zhao et al., 2025; Liu et al., 2025a,b; Cheng et al., 2025; Wang et al., 2025b) hybridize AR global ordering with intra-block diffusion to support KV-caching. To eliminate serial bottlenecks, Wang et al. (2025b) introduces D2F, which enables decoding future blocks from noisy intermediate states.

6.2 Latent Reasoning

Reasoning in Continuous Space. To enhance Chain-of-Thought expressivity, recent works transition from discrete tokens to latent spaces. Hao et al. (2024) leverage transformer hidden states, while Xu et al. (2025a,b) and Zhang et al. (2025); Zhuang et al. (2025) utilize projection modules or probability-weighted embeddings.

Latent Reasoning with DLMs. To mitigate re-masking information loss, recent DLMs integrate continuous semantics. Hersche et al. (2025) propose Soft Masking via dynamic embedding blends, while Zheng et al. (2025) employ dual discrete-continuous diffusion. Additionally, Kang et al. (2025) utilize VAE-based latent spaces to refine reasoning trajectories. HDLM (Zhou et al., 2025a) further structures this process through a hierarchical vocabulary for coarse-to-fine refinement, whereas CCDD (Zhou et al., 2025b) co-evolves continuous and discrete modalities in a joint space to balance expressivity with trainability.

7 Conclusion

In this paper, we presented EvoToken-DLM, a novel diffusion language modeling approach that replaces rigid binary masks with evolving soft token distributions. This shift enables a progres-

sive decoding process, overcoming the limitations of irreversible discrete assignments in traditional MDLMs. By introducing continuous trajectory supervision, we effectively align the training objective with iterative probabilistic refinement. Extensive experiments demonstrate that EvoToken-DLM consistently outperforms strong baselines while remaining fully compatible with KV-caching and blockwise architectures.

Limitations

While our approach enables rapid adaptation from pretrained MDLMs to EvoToken-DLM via lightweight supervised fine-tuning, it faces training challenges when applied to models initialized with autoregressive (AR) priors. The inherent discrepancy between unidirectional AR pretraining and our iterative bidirectional refinement process leads to increased training difficulty and slower convergence for AR-based backbones. We provide a detailed comparative analysis in Appendix D.

References

- Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*.
- Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. 2021. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993.
- Wenrui Bao, Zhiben Chen, Dan Xu, and Yuzhang Shang. 2025. Learning to parallel: Accelerating diffusion large language models via learnable parallel decoding. *arXiv preprint arXiv:2509.25188*.
- Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, and 1 others. 2025. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*.
- Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. 2025. dparallel: Learnable parallel decoding for dlms. *arXiv preprint arXiv:2509.26488*.
- Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and 1 others. 2025. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. 2023. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11575–11596.
- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*.
- Michael Hersche, Samuel Moor-Smith, Thomas Hofmann, and Abbas Rahimi. 2025. Soft-masked diffusion language models. *arXiv preprint arXiv:2510.17206*.
- Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851.
- Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Nicklas Majamäki, Navdeep Jaitly, Yi-An Ma, and Lianhui Qin. 2025. Ladir: Latent diffusion enhances llms for text reasoning. *arXiv preprint arXiv:2510.08558*.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*.
- Xiaoran Liu, Yuerong Song, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2025a. Longllada: Unlocking long context capabilities in diffusion llms. *arXiv preprint arXiv:2506.14429*.
- Yangzhou Liu, Yue Cao, Hao Li, Gen Luo, Zhe Chen, Weiyun Wang, Xiaobo Liang, Biqing Qi, Lijun Wu, Changyao Tian, and 1 others. 2025b. Sequential diffusion language models. *arXiv preprint arXiv:2509.24007*.
- Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. 2025c. dlm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*.
- Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025a. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*.

- Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, and 1 others. 2025b. dinfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv:2510.08666*.
- Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332.
- Alexander Quinn Nichol and Prafulla Dhariwal. 2021. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR.
- Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. Large language diffusion models. *arXiv preprint arXiv:2502.09992*.
- Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. 2025. Tinyzero. <https://github.com/Jiayi-Pan/TinyZero>. Accessed: 2025-01-24.
- Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.
- Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. 2023. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*.
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.
- Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184.
- Jucheng Shen, Gaurav Sarkar, Yeonju Ro, Sharath Nitur Sridhar, Zhangyang Wang, Aditya Akella, and Souvik Kundu. 2025. Improving the throughput of diffusion-based large language models via a training-free confidence-aware calibration. *arXiv preprint arXiv:2512.07173*.
- Jiaming Song, Chenlin Meng, and Stefano Ermon. 2020. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*.
- Yang Song and Stefano Ermon. 2019. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32.
- Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2025. Sparse-dllm: Accelerating diffusion llms with dynamic cache eviction. *arXiv preprint arXiv:2508.02558*.
- Wen Wang, Bozhen Fang, Chenchen Jing, Yongliang Shen, Yangyi Shen, Qiuyu Wang, Hao Ouyang, Hao Chen, and Chunhua Shen. 2025a. Time is a feature: Exploiting temporal dynamics in diffusion language models. *arXiv preprint arXiv:2508.09138*.
- Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. 2025b. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. 2025a. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328*.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. 2025b. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*.
- Linyu Wu, Linhao Zhong, Wenjie Qu, Yuexin Li, Yue Liu, Shengfang Zhai, Chunhua Shen, and Jiaheng Zhang. 2025c. Dmark: Order-agnostic watermarking for diffusion large language models. *arXiv preprint arXiv:2510.02902*.
- Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 2025a. Softcot: Soft chain-of-thought for efficient reasoning with llms. *arXiv preprint arXiv:2502.12134*.
- Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 2025b. SoftCoT++: Test-time scaling with soft chain-of-thought reasoning. *arXiv preprint arXiv:2505.11484*.
- Ling Yang, Ye Tian, Bowen Li, Xinchun Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. 2025. Mmada: Multimodal large diffusion language models. *arXiv preprint arXiv:2505.15809*.
- Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*.
- Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. 2025. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. *arXiv preprint arXiv:2505.15778*.

- Siyao Zhao, Devansh Gupta, Qinqing Zheng, and Aditya Grover. 2025. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*.
- Huangjie Zheng, Shansan Gong, Ruixiang Zhang, Tianrong Chen, Jiatao Gu, Mingyuan Zhou, Navdeep Jaitly, and Yizhe Zhang. 2025. Continuously augmented discrete diffusion model for categorical generative modeling. *arXiv preprint arXiv:2510.01329*.
- Linhao Zhong, Linyu Wu, Wen Wang, Yuling Xi, Chenchen Jing, Jiaheng Zhang, Hao Chen, and Chunhua Shen. 2026. Efficient self-evaluation for diffusion language models via sequence regeneration. *arXiv preprint arXiv:2603.02760*.
- Cai Zhou, Chenyu Wang, Dinghuai Zhang, Shangyuan Tong, Yifei Wang, Stephen Bates, and Tommi Jaakkola. 2025a. Next semantic scale prediction via hierarchical diffusion language models. *arXiv preprint arXiv:2510.08632*.
- Cai Zhou, Chenxiao Yang, Yi Hu, Chenyu Wang, Chubin Zhang, Muhan Zhang, Lester Mackey, Tommi Jaakkola, Stephen Bates, and Dinghuai Zhang. 2025b. Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner. *arXiv preprint arXiv:2510.03206*.
- Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and 1 others. 2025. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*.
- Yufan Zhuang, Liyuan Liu, Chandan Singh, Jingbo Shang, and Jianfeng Gao. 2025. Text generation beyond discrete token sampling. *arXiv preprint arXiv:2505.14827*.

Appendix

A Appendix Overview

- **Appendix B: More Methodological Details**
 - Appendix B.1: EvoToken Algorithm for Progressive Inference
 - Appendix B.2: EvoToken Algorithm for Continuous Trajectory Supervision
- **Appendix C: More Implementation Details**
 - Appendix C.1: Detailed Descriptions of Training Dataset and Evaluation Benchmarks
 - Appendix C.2: Training Configurations for Continuous Trajectory Supervision
 - Appendix C.3: Inference and Evaluation Setup
- **Appendix D: More Analyses**
 - Appendix D.1: Paradigm Consistency: Why MDLMs Rapidly Adapt to EvoToken-DLM
 - Appendix D.2: Adaptation Hurdles for AR Backbones: Causal Prior Mismatch
- **Appendix E: More Experimental Results**
 - Appendix E.1: Additional Results Based on LLaDA-1.5
 - Appendix E.2: Ablation Study on the Mixing Ratio α
 - Appendix E.3: Ablation on Training Trajectory Length $\Delta\tau$
 - Appendix E.4: Training Overhead Analysis
 - Appendix E.5: Additional Results with Different Block Sizes
 - Appendix E.6: Additional Inference Examples of EvoToken-DLM
 - Appendix E.7: Qualitative Comparisons for EvoToken-DLM and MDLMs

B More Methodological Details

B.2 EvoToken Algorithm for Continuous Trajectory Supervision

B.1 EvoToken Algorithm for Progressive Inference

As presented in Algorithm S1, the core of the inference process lies in the management of the token states \mathbf{Z} and continuous embeddings \mathbf{E} . Each token position starts in the [MASK] state. For each block \mathcal{B}_b , the model performs multiple forward passes to refine the soft embeddings. In each step, tokens in a subset S are promoted from mask or soft-mask states to pure soft states. To ensure the stability of the final output, we track the historical high-confidence predictions v_i for each position since it entered the soft state. Once all tokens in the current block are in pure soft states, they are transitioned to the [Decode] state, and their hard embeddings are used as context for the next block. This mechanism effectively facilitates a progressive refinement process for each token.

Algorithm S2 presents the continuous trajectory supervision procedure. The core design philosophy is to bridge the bias between the static training of traditional MDLMs and the iterative nature of EvoToken inference. We introduce a training-time simulation of the inference trajectory by partitioning the target block \mathcal{B} into S_{soft} and S_{mask} , and we initialize S_{soft} positions with their corresponding ground-truth (GT) embeddings e^{y_k} . This multi-step refinement loop, repeated for $\Delta\tau$ steps within each training iteration, ensures that the model parameters θ are optimized not just for single-step recovery, but for the evolutionary path. This continuous optimization allows the model to effectively learn the progressive refinement over successive iterations.

Algorithm S1 Progressive Inference with EvoToken-DLM

Require: Prompt P , target length N , block size B , mixing ratio α , filtering threshold K

Ensure: Decoded sequence $V = (v_1, \dots, v_N)$

```
1: Initialize states  $\mathbf{Z} = (z_1, \dots, z_N)$  where  $z_i \leftarrow [\text{MASK}]$ ,  $\forall i \in \{1, \dots, N\}$ 
2: Initialize embeddings  $\mathbf{E} = (e_P, e_1, \dots, e_N)$  where  $e_i \leftarrow e^{\langle \text{mask} \rangle}$ ,  $\forall i \in \{1, \dots, N\}$ 
3: for block  $b = 1 \rightarrow N/B$  do
4:    $\mathcal{B}_b \leftarrow$  indices of the current block
5:   while  $\exists i \in \mathcal{B}_b$  s.t.  $z_i \neq [\text{Decode}]$  do
6:      $\{v_i^c, p_i^c\}_{c=1}^{|\mathcal{V}|} \leftarrow \text{Model}(\mathbf{E})$ 
7:      $\{\hat{v}_i^c, \hat{p}_i^c\}_{c=1}^K \leftarrow \text{Normalize}(\text{TopK}(\{v_i^c, p_i^c\}_{c=1}^{|\mathcal{V}|}, K))$ 
8:     for  $i \in \{1, \dots, N\}$  do
9:        $e_i^{\text{dist}} \leftarrow \sum_{c=1}^K \hat{p}_i^c \cdot e^{\hat{v}_i^c}$ 
10:       $e_i^{\text{dist}+M} \leftarrow \alpha e^{\langle \text{mask} \rangle} + (1 - \alpha)e_i^{\text{dist}}$ 
11:    end for
12:    Select subset  $S \subseteq \{i \in \mathcal{B}_b \mid z_i \in \{[\text{MASK}], \text{Soft}([\text{MASK}] \cup \mathcal{V})\}\}$ 
13:     $z_i \leftarrow \text{Soft}(\mathcal{V})$ ,  $\forall i \in S$ 
14:     $z_i \leftarrow \text{Soft}([\text{MASK}] \cup \mathcal{V})$ ,  $\forall i \notin S$  s.t.  $z_i = [\text{MASK}]$ 
15:    if  $\forall i \in \mathcal{B}_b, z_i = \text{Soft}(\mathcal{V})$  then
16:       $z_i \leftarrow [\text{Decode}]$ ,  $\forall i \in \mathcal{B}_b$ 
17:      Identify  $v_i$  as the highest-confidence token since  $z_i$  transitioned to  $\text{Soft}(\mathcal{V})$ ,  $\forall i \in \mathcal{B}_b$ 
18:    end if
19:    for  $i \in \{1, \dots, N\}$  do
20:      
$$e_i \leftarrow \begin{cases} e^{\langle \text{mask} \rangle} & \text{if } z_i = [\text{MASK}] \\ e_i^{\text{dist}+M} & \text{if } z_i = \text{Soft}([\text{MASK}] \cup \mathcal{V}) \\ e_i^{\text{dist}} & \text{if } z_i = \text{Soft}(\mathcal{V}) \\ e^{v_i} & \text{if } z_i = [\text{Decode}] \end{cases}$$

21:    end for
22:  end while
23: end for
24: return  $V = (v_1, \dots, v_N)$ 
```

C More Implementation Details

C.1 Detailed Descriptions of Training Dataset and Evaluation Benchmarks

We utilize a combination of high-quality instruction-tuning data and diverse mathematical benchmarks.

Training Dataset.

- **S1K** (Muennighoff et al., 2025): A high-quality dataset featuring 1,000 diverse and challenging problems, each accompanied by distilled reasoning traces and solutions to facilitate complex chain-of-thought reasoning.

Evaluation Benchmarks. We evaluate the performance of our model across the following four benchmarks, covering a spectrum of arithmetic and logical difficulty.

- **Countdown** (Pan et al., 2025): A combinatorial arithmetic task that requires models to reach a target value using a specific set of numbers and basic operators.
- **GSM8K** (Cobbe et al., 2021): A collection of 8.5K grade-school math problems requiring 2–8 steps of multi-step arithmetic reasoning.
- **MATH500** (Lightman et al., 2023): A subset of 500 challenging high-school competition-level problems selected from the MATH dataset.
- **SVAMP** (Patel et al., 2021): A benchmark of 1K elementary math word problems designed to test model robustness against linguistic variations in narratives.

Algorithm S2 Continuous Trajectory Supervision for EvoToken-DLM

Require: Training dataset \mathcal{D} , refinement steps T , learning rate η , total iterations N_{iter}

Ensure: Optimized parameters θ

```
1: for iteration  $n = 1 \rightarrow N_{iter}$  do
2:   Sample training pair  $(X, Y) \sim \mathcal{D}$ 
3:   Sample a target block  $\mathcal{B} \subseteq Y$ 
4:   Partition  $\mathcal{B}$  into subset  $S_{soft}$  and  $S_{mask}$ 
5:   Initialize states  $\mathbf{Z}^{(0)}$ :
6:      $z_i \leftarrow [\text{Decode}], \quad \forall i < \mathcal{B}$ 
7:      $z_i \leftarrow \text{Soft}(\mathcal{V}), \quad \forall i \in S_{soft}$ 
8:      $z_i \leftarrow [\text{MASK}], \quad \forall i \in S_{mask} \cup \{i > \mathcal{B}\}$ 
9:   Initial embeddings  $\mathbf{E}^{(0)}$ :
10:     $e_X^{(0)} \leftarrow e^{\{X\}}$ 
11:     $e_i^{(0)} \leftarrow e^{y_i}, \quad \forall i \in S_{soft} \cup \{i < \mathcal{B}\}$ 
12:     $e_i^{(0)} \leftarrow e^{\langle \text{mask} \rangle}, \quad \forall i \in S_{mask} \cup \{i > \mathcal{B}\}$ 
13:   for step  $i = 1 \rightarrow \Delta\tau$  do
14:      $P^{(i)} \leftarrow \text{Model}_\theta(\mathbf{E}^{(i-1)})$ 
15:      $\mathcal{L}^{(i)} \leftarrow \text{CrossEntropy}(P_{\mathcal{B}}^{(i)}, Y_{\mathcal{B}})$ 
16:     Update states  $\mathbf{Z}^{(i)}$ , embeddings  $\mathbf{E}^{(i)}$  and decode tokens  $V^{(i)}$  as in Algorithm S1
17:      $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}^{(i)}$ 
18:   end for
19: end for
20: return  $\theta$ 
```

C.2 Training Configurations for Continuous Trajectory Supervision

To fine-tune the model under the progressive token evolution mechanism, we employ the following training configurations. The model is trained on the S1K dataset (Muennighoff et al., 2025) for a total of 10k steps.

- **LoRA Configuration:** The LoRA adapter is applied to the query, key, and value projections. We set the rank $r = 128$, LoRA alpha $\alpha = 256$, and a dropout rate of 0.05, with no bias parameters tuned.
- **Optimization Settings:** We use a learning rate of $1e - 5$ with a total batch size of 8.
- **Sequence Handling:** The maximum sequence length is truncated at 1,024 tokens.
- **Continuous Simulation:** Following our proposed framework, the number of continuous simulation steps $\Delta\tau$ is set to 4. For blockwise processing, the current block size is fixed at 512. During training, the number of transition tokens $|S|$ is dynamically sampled from the set $\{1, 2, 4, 8\}$ to enhance the model’s robustness across varying generation densities. The

mixing ratio is stochastically sampled from a uniform distribution $\mathcal{U}(0.5, 1.0)$.

C.3 Inference and Evaluation Setup

To ensure a fair and reproducible evaluation, we standardize our inference parameters across all datasets. We employ a decoding temperature of 0.5 and fix the random seed to 42 to eliminate stochastic variance. For the proposed refinement mechanism, we perform a grid search for the hyperparameter α within the candidate set $\{0.5, 0.6, 0.7, 0.8, 0.9\}$ and report the performance associated with the optimal α for each setting. For evaluations based on Dream-Instruct-7B (Ye et al., 2025), we set the default generation length to 256 with 128 NFEs. For evaluations based on D2F-LLaDA (Wang et al., 2025b), we set the default maximum generation length to 512.

D More Analyses

D.1 Paradigm Consistency: Why MDLMs Rapidly Adapt to EvoToken-DLM

We begin with a brief overview of the training process for MDLMs. Formally, we characterize the model distribution via a diffusion process consisting of a forward and a reverse process. The forward

k	2	4	6	8
rank $\leq k$	61.2%	76.8%	91.5%	94.6%

Table S1: Probability of the ground-truth token appearing in the Top- k predictions when substituting an input token in the noisy sequence with a random token. The high hit rates demonstrate the model’s robust capability to infer the correct output based solely on contextual information.

process $q(X^t|X^0)$ gradually corrupts the initial sequence X^0 by independently masking tokens with a time-dependent probability $t \in [0, T]$, resulting in a partially masked sequence X^t where each token is replaced by a special mask symbol with probability t and remains unchanged with probability $1 - \frac{t}{T}$. At $t = T$, the sequence becomes fully masked. Conversely, the reverse process learns to recover the original data distribution by iteratively predicting the masked tokens in X^t as t transitions from T to 0. The training objective is defined as follows:

$$\mathcal{L}(\theta) \triangleq -\mathbb{E}_{t, X^0, X^t} \left[\frac{T}{t} \sum_{i=1}^N \mathbf{1}[x_i^t = \langle \text{mask} \rangle] \log p_{\theta}(x_i^0 | X^t) \right], \quad (\text{S1})$$

where N denotes the sequence length.

Under this training paradigm, the model gradually develops the capability to infer the potential probability distribution of each position based on its surrounding context. Notably, although the supervision is explicitly applied only to the mask tokens, the inherent generalization of the model enables it to support inference at all positions, even those not marked by a mask.

We further substantiate this observation through empirical analysis, as shown in Table S1. By intentionally replacing a token at a specific position with a random token in a noisy sequence, we observe the model’s ability to predict the ground-truth (GT) token at that location. Our statistical findings indicate that despite the random substitution, the GT tokens remain consistently concentrated among the top-ranked candidates in the model’s output distribution. This suggests that MDLMs possess a robust, context-driven predictive mechanism that transcends the specific masking patterns encountered during training.

The inherent capability of MDLMs to continuously predict tokens based on context aligns remarkably well with the core objective of EvoToken, which aims to refine its own tokens in real-

time. This alignment facilitates a seamless transition from standard MDLMs to the EvoToken-DLM framework. Leveraging this pre-trained inductive bias, the model requires only minimal supervised fine-tuning to adapt to the EvoToken paradigm. Essentially, the foundational training of MDLMs serves as a robust prior, enabling the model to internalize the evolutionary refinement process with high training efficiency.

D.2 Adaptation Hurdles for AR Backbones: Causal Prior Mismatch

In contrast to the seamless adaptation of MDLMs, backbones pre-trained under the autoregressive (AR) paradigm face significant challenges when transitioning to the EvoToken framework. This difficulty stems from a fundamental mismatch between the AR prior and the iterative refinement nature of EvoToken.

The primary objective of AR training is to predict the next token conditioned solely on preceding context, enforced by a unidirectional causal attention mask. This intrinsic constraint prevents the model from performing bi-directional information aggregation, which is essential for iteratively updating and refining existing tokens based on full context. While some recent works attempt to bridge this gap by fine-tuning AR models into diffusion-like or blockwise models (e.g., SDAR (Cheng et al., 2025)), these models often remain heavily tethered to their original causal priors. Consequently, adapting such models to the EvoToken paradigm necessitates substantial training resources to override the deep-seated unidirectional bias. Due to these significant computational overheads, we do not conduct further exploration on these AR-based variants.

E More Experimental Results

E.1 Additional Results Based on LLaDA-1.5

To verify the generalizability of EvoToken-DLM across different model versions, we extend our framework to the LLaDA-1.5 pretrained model via continuous trajectory supervision. Figure S2 illustrates the performance gains across multiple benchmarks. The results confirm that the improvements provided by our progressive evolution mechanism are generalizable. EvoToken-DLM consistently enhances reasoning precision compared to the binary masking baseline, even based on another pretrained model.

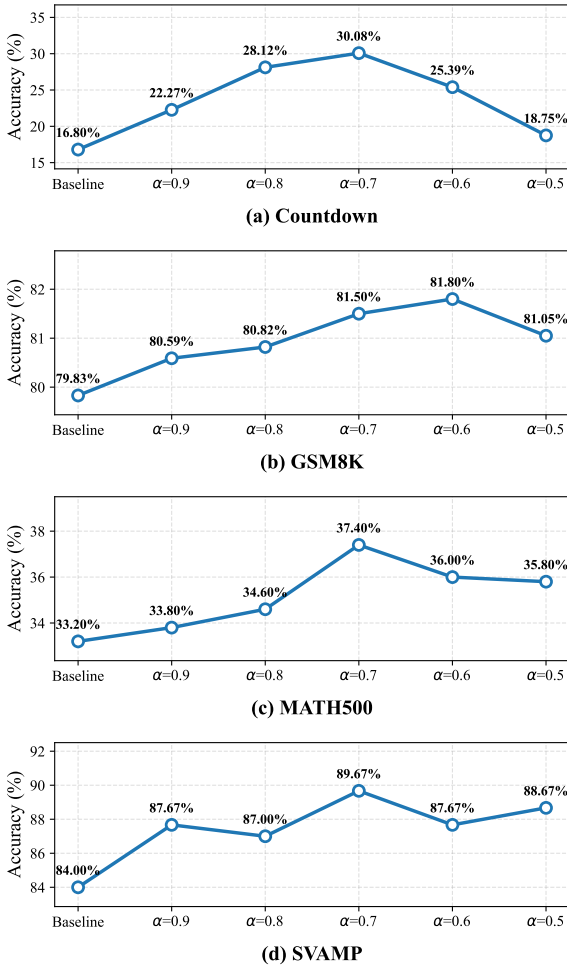


Figure S1: Performance comparison between the baseline and EvoToken-DLM with different α settings on various datasets.

E.2 Ablation Study on the Mixing Ratio α

We investigate the sensitivity of EvoToken-DLM to the mixing ratio α in Figure S1. The results demonstrate that our framework maintains remarkably stable performance across the $\alpha \in [0.5, 0.9]$ range on all four evaluation datasets, highlighting its substantial algorithmic robustness. EvoToken-DLM effectively captures the essential refinement signals even under varying fusion intensities. The fact that performance remains consistent across diverse benchmarks, ranging from simple arithmetic to complex reasoning, underscores the robustness of the mechanism.

To examine whether EvoToken-DLM relies on per-setting tuning, we additionally fix $\alpha = 0.7$ across all datasets and decoding budgets in Table S2. Even under this single default choice, EvoToken-DLM still surpasses the baseline in 15 out of 16 settings, indicating that $\alpha = 0.7$ is a robust and practical default.

Table S2: Comparison between the baseline and EvoToken-DLM using a fixed mixing ratio $\alpha = 0.7$ across datasets and decoding budgets based on LLaDA-Instruct-8B.

Dataset	Gen Len	$\frac{NFE}{Gen Len}$	Baseline	EvoToken ($\alpha = 0.7$)
Countdown	256	1	23.83	35.55
Countdown	256	1/2	16.41	30.08
Countdown	512	1	20.70	42.97
Countdown	512	1/2	16.80	30.08
GSM8K	256	1	79.30	83.47
GSM8K	256	1/2	77.63	81.88
GSM8K	512	1	83.47	84.23
GSM8K	512	1/2	79.83	81.50
MATH500	256	1	34.60	37.60
MATH500	256	1/2	32.20	35.80
MATH500	512	1	39.40	39.20
MATH500	512	1/2	33.20	37.40
SVAMP	256	1	84.67	89.67
SVAMP	256	1/2	86.67	92.33
SVAMP	512	1	86.00	89.00
SVAMP	512	1/2	84.00	89.67

Table S3: Accuracy and average finalized tokens per step under different mixing ratios α and confidence thresholds on MATH500. Higher average tokens per step indicate faster convergence.

Threshold	Metric	$\alpha = 0.5$	$\alpha = 0.6$	$\alpha = 0.7$	$\alpha = 0.8$	$\alpha = 0.9$
0.9	Accuracy (%)	34.60	36.80	36.00	34.00	36.80
	Avg. Tokens/Step	2.42	2.30	2.21	2.14	2.07
0.8	Accuracy (%)	35.20	36.20	34.00	35.20	34.20
	Avg. Tokens/Step	2.94	2.77	2.70	2.58	2.59
0.7	Accuracy (%)	34.40	36.00	34.40	34.00	34.20
	Avg. Tokens/Step	3.43	3.29	3.20	3.09	3.08
0.6	Accuracy (%)	32.40	33.00	33.00	29.80	32.60
	Avg. Tokens/Step	3.92	3.79	3.73	3.62	3.57
0.5	Accuracy (%)	30.20	28.60	29.00	27.60	29.40
	Avg. Tokens/Step	4.44	4.41	4.30	4.17	4.13

Table S3 further studies the interaction between α and threshold-based decoding on MATH500. Smaller α values decode slightly more tokens per step, and this smooth trade-off indicates that the method is not overly sensitive to the exact choice of α .

E.3 Ablation on Training Trajectory Length $\Delta\tau$

We compare the default multi-step supervision ($\Delta\tau = 4$) against a single-step variant ($\Delta\tau = 1$) in Table S4. Reducing the simulated trajectory to a single step only slightly lowers the overall average from 59.22 to 58.62, showing that the model preserves strong revisability even with shorter training trajectories. Nevertheless, the multi-step setting remains slightly more reliable overall.

E.4 Training Overhead Analysis

Table S5 compares the wall-clock training cost of FT-Base and EvoToken with continuous trajectory supervision for the same 10k-step run on one H200.

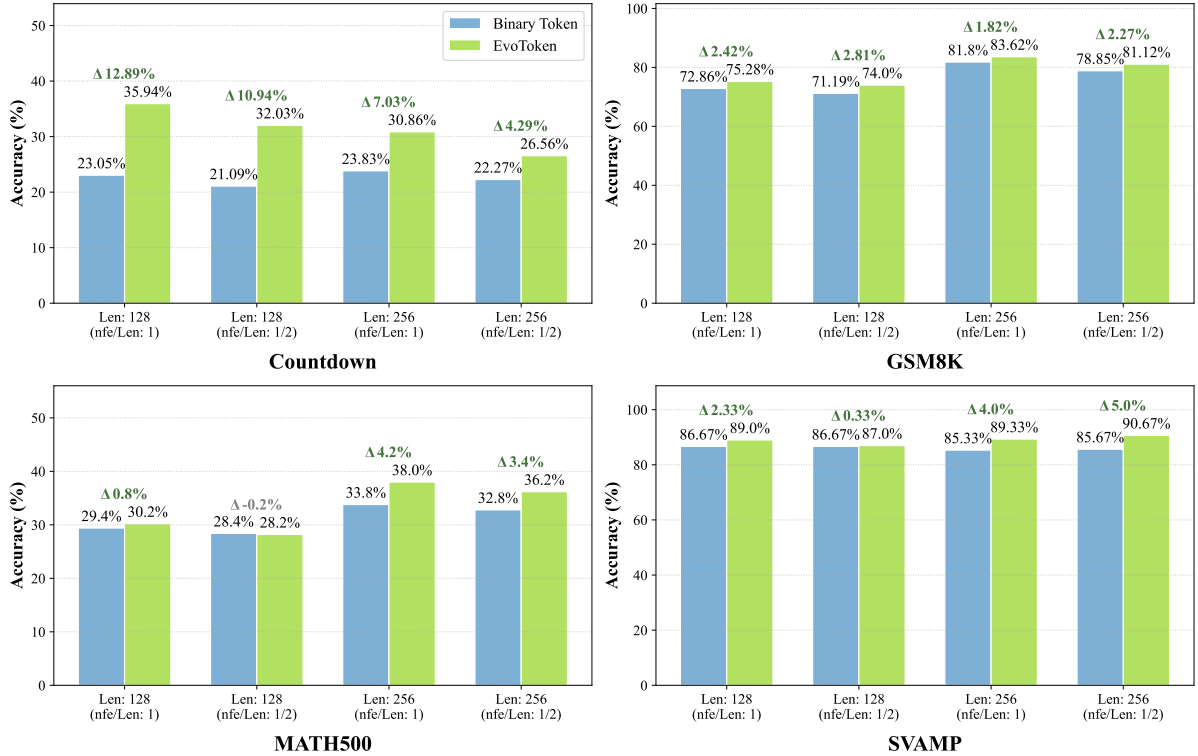


Figure S2: Additional comparison of EvoToken and the binary masking baseline based on another pretrained model LLaDA-1.5. We apply continuous trajectory supervision and evaluate performance on various benchmarks.

Table S4: Comparison for different training trajectory lengths $\Delta\tau$ based on LLaDA-Instruct-8B. The default setting uses $\Delta\tau = 4$, while $\Delta\tau = 1$ reduces supervision to a single simulated refinement step.

Dataset	Gen Len	$\frac{NFE}{Gen Len}$	Baseline	EvoToken ($\Delta\tau = 1$)	EvoToken ($\Delta\tau = 4$)
Countdown	128	1	21.48	40.23	39.84
Countdown	128	1/2	26.17	33.59	34.77
Countdown	256	1	23.83	33.20	35.55
Countdown	256	1/2	16.41	32.03	30.08
GSM8K	128	1	70.20	73.46	74.30
GSM8K	128	1/2	67.55	71.95	73.54
GSM8K	256	1	79.30	83.09	83.47
GSM8K	256	1/2	77.63	79.61	82.03
MATH500	128	1	28.80	32.00	28.40
MATH500	128	1/2	26.60	28.60	29.20
MATH500	256	1	34.60	39.00	39.60
MATH500	256	1/2	32.20	35.20	36.40
SVAMP	128	1	88.33	89.00	89.00
SVAMP	128	1/2	86.00	89.00	89.33
SVAMP	256	1	84.67	89.67	89.67
SVAMP	256	1/2	86.67	88.33	92.33
Average			53.15	58.62	59.22

EvoToken requires comparable training time and peak GPU memory, confirming that the proposed supervision incurs negligible overhead in practice.

E.5 Additional Results with Different Block Sizes

To further evaluate the robustness and scalability of EvoToken-DLM, we conduct experiments across a range of block sizes $\in \{8, 16, 32, 64\}$. As shown in Table S6, our method consistently outperforms both the original LLaDA-Instruct-8B baseline and the FT-baseline across all tested configurations and

Table S5: Wall-clock training time and peak GPU memory usage on LLaDA-Instruct-8B using one H200.

Method	Training Time	Peak GPU Memory
FT-Base	11.0h \times 1 H200	22.1G
EvoToken	10.7h \times 1 H200	22.7G

datasets.

The experimental results highlight several key observations:

- **Universal Performance Gains:** EvoToken-DLM achieves significant accuracy improvements regardless of the block size. For instance, in the Countdown task, we observe absolute gains ranging from +10.55% to +13.67% over the baseline, demonstrating that our approach effectively enhances the model’s reasoning capabilities across various inference granularities.
- **Robustness to Block Size Variations:** EvoToken-DLM demonstrates high resilience to changes in block size, yielding stable and superior results across all tested discretization settings. This suggests that the learned token evolution patterns are agnostic to specific

Table S6: Performance comparison on the Countdown, GSM8K, MATH500 and SVAMP datasets across various block sizes based on LLaDA-Instruct-8B. EvoToken-DLM is initialized from LLaDA-Instruct-8B and fine-tuned for 10k steps using continuous trajectory supervision. Comparisons are conducted against both the baseline model and the sft-baseline. All experiments are conducted with a generation length of 256 and an NFE of 128.

Block Size	Method	Countdown	GSM8K	MATH500	SVAMP
8	Baseline	12.11%	74.30%	32.00%	85.00%
	FT-Base	17.97%	76.50%	31.60%	87.33%
	EvoToken	22.66% +10.55%	79.00% +4.70%	34.60% +2.60%	88.00% +3.00%
16	Baseline	15.23%	75.21%	32.60%	84.33%
	FT-Base	16.80%	79.08%	32.00%	88.67%
	EvoToken	26.95% +11.72%	80.59% +5.38%	37.20% +4.60%	90.00% +5.67%
32	Baseline	16.41%	77.63%	32.20%	86.67%
	FT-Base	16.80%	78.62%	31.20%	87.00%
	EvoToken	30.08% +13.67%	82.03% +4.40%	36.40% +4.20%	92.33% +5.66%
64	Baseline	19.92%	76.42%	33.80%	86.67%
	FT-Base	19.14%	78.01%	34.60%	87.33%
	EvoToken	31.64% +11.72%	83.40% +6.98%	38.00% +4.20%	91.00% +4.33%

block partitions.

- **Consistency across Tasks:** The superiority of our method is consistently maintained across diverse benchmarks, from symbolic reasoning (Countdown) to complex mathematical problem-solving (GSM8K, MATH500), further validating the generalizability of our approach in various downstream scenarios.

Overall, these results underscore that EvoToken-DLM is not finely tuned for a specific inference setting but rather provides a fundamental enhancement to the underlying diffusion generation process.

E.6 Additional Inference Examples of EvoToken-DLM

We provide additional detailed visualizations of the internal refinement process to demonstrate how EvoToken-DLM iteratively refines intermediate states. As shown in Figure S3 and Figure S4, for the prompt involving arithmetic, the model progressively clarifies the soft token representations. The visualization highlights how uncertain embeddings at early simulation steps are refined into sharp, symbolically correct tokens as the step increases.

<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	Step 0
Let 0.46	<mask> 0.50	<mask> 0.50	the 0.54	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 1
If 0.34	the 0.25	original 0.27	price 0.33	be 0.25	the 0.29	the 0.17	the 0.27	.0.23	.0.23	.0.22	.0.26	.0.18	5 0.18	5 0.20	5 0.22	Step 1
The 0.19	\$ 0.14	price 0.12	5 0.13	original 0.14	price 0.16	book 0.17	of 0.12	.0.15	.0.16	.0.19	.0.13	5 0.17	5 0.17	5 0.16	5 0.15	Step 1
	's 0.11	be 0.11		the 0.10	5 0.05	of 0.16	5 0.11	5 0.12	5 0.11	5 0.09	The 0.11	5 0.15	the 0.16	2 0.15	2 0.13	Step 1
Let 0.54	<mask> 0.50	<mask> 0.50	price 0.53	be 0.76	the 0.75	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 2
If 0.30	\$ 0.23	original 0.31	the 0.29	original 0.12	price 0.18	original 0.38	price 0.27	the 0.19	the 0.18	.0.24	.0.26	.0.26	5 0.17	5 0.18	5 0.20	Step 2
A 0.16	the 0.22	P 0.14	\$ 0.18	of 0.12	5 0.07	price 0.07	the 0.12	of 0.17	.0.17	.0.17	book 0.14	.0.17	The 0.14	2 0.16	2 0.15	Step 2
	's 0.05	x 0.06				2 0.05	of 0.12	.0.15	.0.15	book 0.16	The 0.12	original 0.07	5 0.10	' 0.16	' 0.15	Step 2
Let 0.99	\$ 0.95	<mask> 0.50	\$ 0.92	be 0.98	the 0.99	original 0.98	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 3
If 0.01	(0.03	P 0.34	price 0.04	of 0.01	\$ 0.00	price 0.02	price 0.48	of 0.34	of 0.39	book 0.48	.0.48	.0.26	The 0.23	2 0.20	5 0.22	Step 3
We 0.00	the 0.02	x 0.12) 0.04	price 0.01	price 0.00	\$ 0.00	of 0.02	.0.16	The 0.07	The 0.01	2 0.01	The 0.01	5 0.15	% 0.15	% 0.14	Step 3
		p 0.04					\$ 0.00	the 0.01	' 0.04	' 0.01	' 0.01	' 0.01	A 0.06	Kyle 0.12	' 0.15	Step 3
Let 0.99	\$ 1.00	<mask> 0.50	\$ 1.00	be 1.00	the 0.99	original 1.00	price 1.00	of 0.93	the 0.98	book 0.99	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 4
If 0.00	the 0.00	P 0.28	\$ 0.00	is 0.00	\$ 0.02	price 0.01	of 0.01	of 0.01	The 0.04	\$ 0.01	.0.49	The 0.30	the 0.18	price 0.20	5 0.22	Step 4
Let 0.00	5 0.00	x 0.14	P 0.00	= 0.00	= 0.00	the 0.00	original 0.00	.0.16	The 0.09	\$ 0.03	paid 0.02	' 0.14	' 0.16	2 0.16	2 0.13	Step 4
		p 0.08						.0.00	' 0.03		the 0.02	Kyle 0.09	a 0.16	paid 0.16	is 0.13	Step 4
Let 1.00	\$ 1.00	<mask> 0.50	\$ 1.00	be 1.00	the 0.99	original 1.00	price 1.00	of 0.93	the 0.98	book 0.99	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 5
If 0.00	the 0.00	P 0.28	\$ 0.00	is 0.00	\$ 0.02	price 0.00	of 0.00	of 0.07	The 0.04	\$ 0.01	.0.49	The 0.30	the 0.18	price 0.20	5 0.22	Step 5
Let 0.00	(0.00	x 0.15	P 0.00	represent 0.00	= 0.00	the 0.00	the 0.00	.0.00	We 0.00	price 0.00	.0.01	' 0.12	' 0.17	2 0.16	a 0.14	Step 5
		p 0.07									is 0.00	A 0.09	price 0.16	paid 0.14	paid 0.13	Step 5
Let	\$	P	\$	be	the	original	price	of	the	book	.	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 6
												The 0.29	the 0.18	price 0.20	5 0.21	Step 6
												' 0.13	price 0.17	paid 0.16	a 0.15	Step 6
												After 0.07	' 0.16	2 0.15	is 0.14	Step 6
Let	\$	P	\$	be	the	original	price	of	the	book	.	The 0.57	<mask> 0.50	price 0.47	<mask> 0.50	Step 7
												' 0.24	price 0.19	paid 0.28	5 0.21	Step 7
												Kyle 0.19	the 0.18	2 0.25	is 0.15	Step 7
												' 0.14	' 0.14	' 0.14	a 0.14	Step 7

Figure S3: Additional inference example with a block size of 12, showing intermediate refinement states for a selected subsequence across successive steps. We showcase the refinement states for the first 16 output tokens based on the prompt: "Kyle bought last year's best-selling book for \$19.50. This is with a 25% discount from the original price. What was the original price of the book?".

<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	<mask>	Step 0
<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	girls 0.78	<mask> 0.50	boys 0.72	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 1
To 0.41	determine 0.35	how 0.43	many 0.45	more 0.45	more 0.03	than 0.40	of 0.04	the 0.24	school 0.26	has 0.26	,	,	we 0.23	need 0.30	to 0.30	Step 2	
The 0.05	find 0.09	the 0.04	how 0.03	many 0.03	the 0.02	girls 0.06	than 0.03	are 0.17	in 0.20	the 0.20	school 0.20	we 0.21	need 0.23	to 0.16	need 0.12	Step 3	
Step 0.04	,	I 0.03	number 0.02	of 0.02		the 0.04		there 0.09	the 0.04	school 0.04	has 0.07	the 0.04	follow 0.05	the 0.03	the 0.08	Step 4	
<mask> 0.50	<mask> 0.50	<mask> 0.50	many 1.00	more 1.00	girls 1.00	<mask> 0.50	boys 1.00	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 5
To 0.49	determine 0.48	how 0.49	number 0.00	of 0.00	boys 0.00	than 0.49	the 0.00	the 0.27	school 0.32	has 0.28	,	we 0.26	we 0.32	need 0.15	to 0.12	subtract 0.19	Step 6
I 0.00	find 0.02	the 0.00	how 0.00	many 0.00	of 0.00	the 0.00	girls 0.00	are 0.17	in 0.17	the 0.22	school 0.23	we 0.21	need 0.15	to 0.12	subtract 0.19	Step 7	
Step 0.00	,	number 0.00				girls 0.00		there 0.06	are 0.02	in 0.00	the 0.01	school 0.01	we 0.03	we 0.02	need 0.10	Step 8	
To 0.00	<mask> 0.50	<mask> 0.50	many 1.00	more 1.00	girls 1.00	than 1.00	boys 1.00	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 9
find 0.00	determine 0.47	how 0.50	number 0.00	many 0.00	boys 0.00	the 0.00	the 0.00	the 0.31	school 0.32	has 0.31	,	we 0.26	we 0.28	need 0.38	to 0.30	Step 10	
I 0.00	find 0.03	the 0.00	of 0.00	of 0.00	number 0.00	girls 0.00	girls 0.00	are 0.15	in 0.17	the 0.19	school 0.21	need 0.23	to 0.19	subtract 0.11	subtract 0.16	Step 11	
	out 0.00	number 0.00						there 0.05	are 0.01	school 0.00	the 0.01	school 0.01	we 0.03	we 0.02	need 0.05	Step 12	
To	determine	how	many	more	girls	than	boys	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	Step 13
								the 0.34	school 0.34	has 0.32	,	we 0.29	we 0.27	need 0.28	to 0.41	subtract 0.35	Step 14
								are 0.12	in 0.16	the 0.17	school 0.20	need 0.21	to 0.19	to 0.08	subtract 0.13	Step 15	
								there 0.04	are 0.00	school 0.00	the 0.01	school 0.01	we 0.03	we 0.01	the 0.03	Step 16	
To	determine	how	many	more	girls	than	boys	the 0.78	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	<mask> 0.50	need 0.82	<mask> 0.50	<mask> 0.50	Step 17
								are 0.19	school 0.38	has 0.34	,	we 0.29	we 0.28	to 0.25	to 0.16	to 0.36	Step 18
								there 0.05	in 0.12	the 0.16	school 0.20	need 0.21	need 0.22	we 0.02	subtract 0.12	need 0.03	Step 19
								are 0.00	school 0.00	has 0.01	school 0.01	0.03					Step 20
To	determine	how	many	more	girls	than	boys	are 0.78	<mask> 0.50	<mask> 0.50	<mask> 0.50	,	we 0.78	need 0.82	<mask> 0.50	<mask> 0.50	Step 21
								the 0.01	in 0.49	the 0.48	school 0.48	we 0.02	you 0.01	to 0.01	to 0.48	<mask> 0.50	Step 22
								there 0.01	school 0.01	has 0.01	,	0.00	need 0.00	can 0.01	subtract 0.01	<mask> 0.50	Step 23
								are 0.00	a 0.00	has 0.00					perform 0.00	<mask> 0.50	Step 24
To	determine	how	many	more	girls	than	boys	are 0.99	<mask> 0.50	the 1.00	school 1.00	,	we 1.00	need 1.00	to 0.50	<mask> 0.50	Step 25
								there 0.01	in 0.50	in 0.00	number 0.00	of 0.00	you 0.00	you 0.00	can 0.00	<mask> 0.50	Step 26
								than 0.00	at 0.00	than 0.00	the 0.00	,	We 0.00	we 0.00	the 0.00	<mask> 0.50	Step 27
								there 0.00								<mask> 0.50	Step 28
To	determine	how	many	more	girls	than	boys	are	in	the	school	,	we	need	to		Step 29

Figure S4: Additional inference example with a block size of 8, showing intermediate refinement states for a selected subsequence across successive steps. We showcase the refinement states for the first 16 output tokens based on the prompt: "In a school there are 569 girls and 236 boys. How many more girls than boys does the school have?".

E.7 Qualitative Comparisons for EvoToken-DLM and MDLMs

Example 1

Question: Mattison is 60 inches tall. His two-story house is 7 times as high as he is. The floor to the ceiling on the first floor is thrice as high as he is. How high is the roof from the second floor's floor in inches?

GT: 240.0

MDLMs: To determine the height of the roof from the second floor's floor, we need to follow these steps:

1. Calculate the height of Mattison's two-story house.
2. Calculate the height of the floor to the ceiling on the first floor.
3. Determine the height of the second floor from the ground.
4. Calculate the height of the roof from the second floor's floor.

1. The height of Mattison's two-story house is 7 times his height.

$$\text{Height of the house} = 7 \times 60 = 420 \text{ inches}$$

2. The height of the floor to the ceiling on the first floor is 3 times his height.

$$\text{Height of floor to ceiling} = 3 \times 60 = 180 \text{ inches}$$

3. The height of the second floor from the ground is the height of the first floor minus the height of the first floor.

$$\text{Height of second floor} = 420 - 180 = 240 \text{ inches}$$

4. The height of the roof from the second floor's floor is the height of the second floor minus the height of the first floor.

$$\text{Height of roof} = 240 - 180 = 60 \text{ inches}$$

<answer> </answer>

EvoToken-DLM (OURS): Solution:

Let H_1 be the height of the first floor.

Let H_2 be the height of the second floor.

Let H be the height of the house.

We are given: Mattison's height, 60 inches. The height of the house is 7 times Mattison's height, so $H = 7 * 60 = 420$ inches. The floor to the ceiling on the first floor is 3 times Mattison's height, so $H_1 = 3 * 60 = 180$ inches.

The height of the house is the sum of the height of the first floor and the height of the second floor:

$$H = H_1 + H_2$$

$$420 = 180 + H_2$$

$$H_2 = 420 - 180$$

$$H_2 = 240 \text{ inches}$$

The height of the roof from the second floor's floor is the height of the second floor, which is 240 inches.

Final Answer: The final answer is </reasoning>

<answer> </answer>

Example 2

Question: A class of 30 students recently took a test. If 20 students scored 80, 8 students scored 90, and 2 students scored 100, then what was the class average (the mean) on this test?

GT: 84

MDLMs: To find the class average (the mean) on the test, we need to calculate the total score of all students and then divide by the total number of students.

1. Calculate the total score for the students who scored 80:

$$20 \times 80 = 1600$$

2. Calculate the total score for the students who scored 90:

$$8 \times 90 = 720$$

3. Calculate the total score for the students who scored 100:

$$2 \times 100 = 200$$

4. Add these totals together to get the total score:

$$1600 + 720 + 200 = 2520$$

5. Divide the total score by the total number of students to find the mean:

$$\frac{2520}{30} = 84$$

</, reasoning>

<answer> 84 </answer><leot_id>

EvoToken-DLM (OURS): Solution:

To find the average score, we need the total sum of all scores and then divide by the number of students.

20 students scoring 80 : $20 * 80 = 1600$

8 students scoring 90 : $8 * 90 = 720$

2 students scoring 100 : $2 * 100 = 200$

Total sum of scores = $1600 + 720 + 200 = 2520$

Number of students = 30

Average = Total sum of scores / Number of students = $2520 / 30 = 84$

Final Answer: The final answer is 84 </reasoning>

<answer> 84 </answer>

Example 3

Question: Using only the provided numbers, create an arithmetic expression that evaluates to exactly the provided target number. You may use the operations +, -, *, and / as needed, but each number must be used exactly once. Numbers: [92, 43, 58]; Target: 77.

GT: $92 - 58 + 43$

MDLMs: To create an arithmetic expression that evaluates to 77 using the numbers 92, 43, and 58, we need to find a combination of operations (+, -, *, /) that will result in 77. Let's start by considering the different operations:

- Addition and Subtraction:** $- 92 + 43 = 135$ - $92 - 43 = 49$ - $92 + 58 = 150$ - $92 - 58 = 34$
- Multiplication:** $- 92 \times 43 = 3956$ - $92 \times 58 = 5376$ - $43 \times 58 = 2454$
- Division:** $- 92 \div 43 \approx 2.097$ - $92 \div 58 \approx 1.587$ - $43 \div 58 \approx 0.732$

We need to find a combination of these operations that results in 77. Let's try different combinations:

- Multiplication and:** $996 - 43 = 953$ $953 - 58 = 895$ $895 - 43 = 852$ $852 - 58 = 794$ $794 - 43 = 751$ $751 - 58 = 693$ $693 - 43 = 650$ $650 - 58 = 592$ $592 - 43 = 549$ $549 - 58 = 77$

Thus, the expression $592 - 43 - 58 = 77$

</answer> $592 - 43 - 58$ </answer>

EvoToken-DLM (OURS): We need to find an arithmetic expression using the numbers 92, 43, and 58 that evaluates to 77. We operations allowed are +, -, *, and / Each number must be used exactly once.

Let's try different combinations of operations:

- Subtraction:** $92 - 43 = 49$, $49 - 58 = -9$, $-9 + 77 = 68$ (not target)
- Multiplication:** $92 * 43 = 3916$, $3916 - 58 = 3858$, $3858 - 77 = 3981$ (not target)
- Division:** $92 / 43 = 2$, $2 * 58 = 116$, $116 - 77 = 39$ (not target)

Let's try a different approach:

the. $92 - 43 = 49$, $49 - 58 = -9$ (not target)

the. $92 / 43 = 2$, $92 - 58 = 34$ (not target)

the. $92 - 58 = 34$, $34 + 43 = 77$ (target)

The correct expression is $92 - 58 + 43$. </reasoning>

<answer> $92 - 58 + 43$ </answer>

Example 4

Question: *Faye was placing her pencils into rows with 22 pencils in each row. She had 6 packs of pencils each one having 14 pencils. How many pencils does she have?*

GT: 84.0

MDLMs: To determine the total number of pencils Faye has, we need to add the number of pencils in the rows to the number of pencils in the packs. First, we calculate the total number of pencils in the rows by Since there are 22 pencils in each row and Faye has multiple rows, we don't need to exact number of rows for this calculation. Next, we calculate the total number of pencils in the packs by multiplying the number of packs by the number of pencils in each pack. Faye has 6 packs, and each pack contains 14 pencils, so the total number of pencils in the packs is $6 \times 14 = 84$. Finally, we add the number of pencils in the rows to the number of pencils in the packs to get the total number of pencils. </reasoning>

<answer> 116 </answer>

EvoToken-DLM (OURS): We need to determine the total number of pencils Faye has. We know that she has 6 packs with 14 pencils each.

number of pencils = number of packs * number of pencils per pack number of pencils = $6 * 14 = 84$

Faye is placing her pencils into rows with 22 pencils in each row. We need to find out how many rows she can make with 84 pencils.

number of rows = number of pencils / number of pencils per row number of rows = $84 / 22 = 4$

However the question asks for the total number of pencils, which is 84.

Final Answer: The final answer is 84 </reasoning>

<answer> 84 </answer>