Saber: Efficient Sampling with Adaptive Acceleration and Backtracking Enhanced Remasking for Diffusion Language Model in Code Generation

Yihong Dong, Zhaoyu Ma, Xue Jiang, Zhiyuan Fan, Jiaru Qian,
Yongmin Li, Jianha Xiao, Zhi Jin, Ge Li
School of Computer Science, Peking University
{dongyh, mazhaoyu}@stu.pku.edu.cn lige@pku.edu.cn

Abstract

Diffusion language models (DLMs) are emerging as a compelling alternative to the dominant autoregressive paradigm, offering inherent advantages in parallel generation and bidirectional context modeling. However, for the tasks with strict structural constraints such as code generation, DLMs face a critical trade-off between inference speed and output quality, where accelerating generation by reducing sampling steps often leads to catastrophic performance collapse. We find that the fundamental reasons are: 1) the generation difficulty is uneven in the structured sequence decoding steps, making DLM's static acceleration strategy suboptimal; 2) the context of tokens generated by DLM evolves continuously, causing early high-confidence predictions to turn into irreversible errors. In this paper, we introduce efficient **S**ampling with **A**daptive acceleration and **B**acktracking **E**nhanced **R**emasking (i.e., **Saber**), a novel training-free sampling algorithm for DLMs that first achieves both better inference speed and output quality in code generation. Saber dynamically adjusts the number of tokens unmasked per step based on the model's evolving confidence, and utilizes a backtracking mechanism to revert tokens whose confidence drops as new context emerges, with its effectiveness supported by theoretical analysis. Extensive experiments on multiple mainstream code generation benchmarks show that Saber boosts Pass@1 accuracy by an average of 1.9% over mainstream DLM sampling methods, while achieving an average 251.4% inference speedup. By leveraging the inherent advantages of DLMs, our work significantly narrows the performance gap with autoregressive models in code generation.

1 Introduction

Diffusion language models (DLMs) have emerged as a promising non-autoregressive alternative in the field of natural language processing (NLP), with inherent advantages in parallel decoding and bidirectional context modeling through iterative denoising processes (Austin et al., 2021a; Ou et al., 2025; Nie et al., 2025; Ye et al., 2025b). Unlike existing autoregressive models (ARMs) that generate text left-to-right (Radford et al., 2018, 2019; Brown et al., 2020; Touvron et al., 2023), DLMs can simultaneously update multiple token positions by progressively unmasking the generation sequence, enabling global planning and iterative refinement (Ye et al., 2025a; Gong et al., 2025b). This paradigm is especially compelling for structured generation tasks like code generation.

Despite the potential advantages, DLMs still lag behind ARMs in practical performance, especially for code generation tasks. The fundamental bottleneck lies in the crucial speed-quality trade-off. As shown in Figure 1, in code generation tasks, the mainstream DLM sampling strategy can lead to a sharp drop in Pass@1 accuracy (even exceeding 60%), once it increases parallelism to reduce the sampling steps, making the DLMs nearly unusable. This severe trade-off prevents DLMs from realizing their inherent parallel generation advantages in practice, as the computational savings from fewer steps are offset by a significant drop in quality.

We argue that the root cause of this severe trade-off stems from two fundamental challenges inherent to DLM sampling process: 1) This process exhibits non-uniform difficulty. The complexity of correctly predicting a token varies significantly across the task, generation context, and token position. Therefore, static acceleration strategies per step (such as using a fixed token number or confidence threshold) are suboptimal. They are often overly conservative in simple stages, sacrificing speed, while being overly aggressive in complex stages, significantly degrading quality. 2) This process is particularly susceptible to error propagation. Unlike ARMs, which only decide what the next token is, DLMs must decide both where and what

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3623–3642

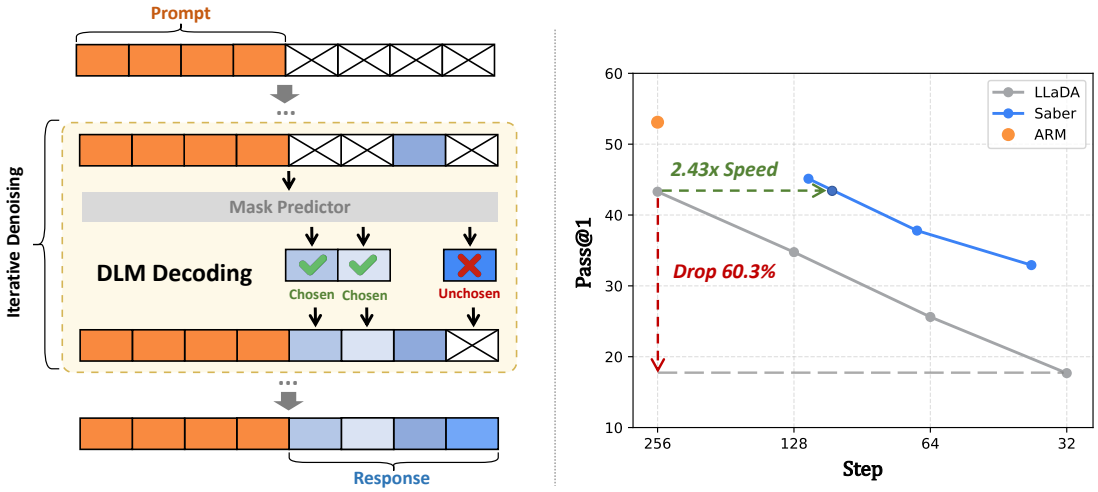

Figure 1: Left: Illustration of DLM Sampling. Right: The trade-off of DLM Sampling between inference speed and output quality on a representative benchmark (HumanEval).

token to generate. An incorrect choice made early in the process, when the contextual information is sparse, becomes permanently "locked in" and cannot be revised. This initial error corrupts the context of all subsequent steps, leading to a cascade of failures from which the model cannot recover.

In this paper, we propose **Saber**, namely efficient **S**ampling with **A**daptive acceleration and **B**acktracking **E**nhanced **R**emasking, a novel training-free sampling algorithm designed to address the two fundamental challenges. Specifically, Saber is built on two key strategies: 1) To address non-uniform difficulty, Saber dynamically adjusts the number of tokens generated in parallel at each step, proceeding cautiously in early, context-poor stages and accelerating as more context is established. 2) To counter error accumulation, Saber introduces a backtracking mechanism. It allows the model to reverse tokens that are identified as likely errors based on newly available context, enabling a self-correction process that improves final output quality. By introducing these two strategies, Saber achieves substantial speedups while enhancing generation quality. We further provide theoretical analysis that validates the effectiveness of the proposed method.

To evaluate the effectiveness and generalizability of Saber, we conduct extensive experiments on multiple mainstream code generation benchmarks. We have the findings from the following aspects: 1) Saber achieves the state-of-the-art performance for DLM sampling in code generation, boosting Pass@1 accuracy by an average improvement of 1.9% over mainstream DLM sampling methods while achieving an average inference speedup of 251.4%. 2) We demonstrate that Saber is a model-agnostic training-free sampling method, which shows effectiveness on various DLMs under different settings and benchmarks with consistent performance gains. 3) Through a comprehensive ablation study and variants experiments, we validate that both adaptive acceleration and backtracking-enhanced remasking are integral to Saber's success. 4) Furthermore, experiments on mathematical reasoning and scientific reasoning benchmarks demonstrate that Saber generalizes beyond code generation tasks. As a result, Saber effectively mitigates the speed-quality trade-off with lower total computational cost, significantly narrowing the performance gap between DLMs and ARMs for code generation.

2 Motivation

Saber is motivated by two key insights from detailed analyses of the DLM sampling process, as shown in Figure 2.

Insight 1: Difficulty Decreases over DLM Generation Process. The task of generating a masked token is not uniformly difficult throughout the DLM generation process. In the initial steps, the context is sparse, consisting mostly of '`[MASK]`' tokens and the initial prompt. In this low-information setting, DLMs are highly uncertain, making token generation challenging. However, as more tokens are generated in subsequent steps, the contextual information available to DLMs increases substantially. This richer context progressively reduces

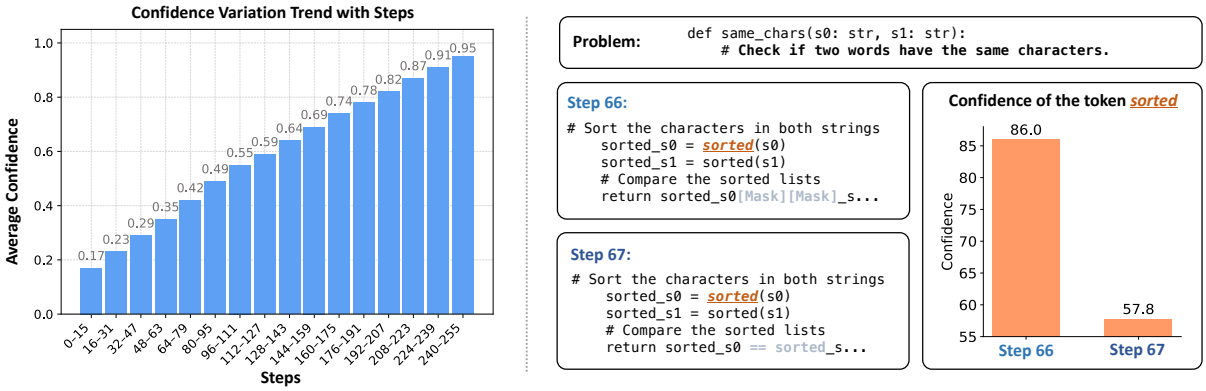

Figure 2: Motivation Example. Left: Average confidence per step. Right: An example of confidence drop for an incorrectly generated token as context evolves during decoding.

DLMs' uncertainty and simplifies the generation of remaining tokens.

As shown in Figure 2(a), the DLMs' average prediction confidence steadily increases as more of the sequence is generated. This observation strongly motivates the need for an adaptive acceleration strategy. An ideal DLM sampler should be cautious when the context is limited and become progressively more aggressive as DLMs' confidence grows. This allows for a more principled approach to acceleration that maximizes speed without prematurely committing to low-confidence tokens.

Insight 2: Dynamic Context of DLM Generated Tokens. A significant difference between DLMs and ARMs is the context of generated tokens. In ARMs, the prefix context for each generated token is fixed. However, in DLMs, the context of generated tokens evolves as '[MASK]' tokens are filled in. Therefore, the DLM's predicted confidence of generated tokens can dramatically change as new information becomes available. For example, a token might be predicted with high confidence based on sparse local context, only to be revealed as a likely error once a more complete global context is established, as depicted in Figure 2(b).

However, traditional DLM sampling methods are irreversible, i.e., once a token is unmasked, the decision is final and cannot be reversed. This makes them highly susceptible to error propagation, where an overconfident early error corrupts the context for all subsequent steps, leading to a cascade of failures. This issue is a primary driver of the catastrophic collapse when attempting parallel decoding, which highlights the necessity of a backtracking remasking mechanism. By allowing DLMs to revise their own predictions, we can mitigate the risk of early error propagation and enable more robust and aggressive parallel generation.

Summary. These two insights reveal a fundamental limitation of current sampling: their static and irreversible design fails to account for the dynamic nature of both generation difficulty and contextual certainty during the DLM sampling process. Therefore, in this paper, we argue that an effective DLM sampler must address these limitations by both adapting its generation speed to the evolving context and being able to revise its own past decisions to mitigate error propagation.

3 Related Work

In this section, we outline the two most relevant directions and associated papers of this work.

3.1 Diffusion Language Models for Code

The current landscape of language models is dominated by the autoregressive paradigm (Radford et al., 2018; Brown et al., 2020; Touvron et al., 2023; Dubey et al., 2024; Guo et al., 2025). However, their strict left-to-right and token-by-token generation process creates a major bottleneck for inference efficiency and inherently limits parallelism (Li et al., 2025). Therefore, a growing body of research on DLMs has emerged (Li et al., 2022a; Austin et al., 2021a; He et al., 2022), which operate through parallel generation and bidirectional context modeling to address the aforementioned constraints. Recently, large-scale DLMs such as Dream (Ye et al., 2025b), DiffuLLaMA (Gong et al., 2025a), and LLaDA (Nie et al., 2025) have demonstrated performance comparable to similar-scale ARMs, making them a highly promising alternative.

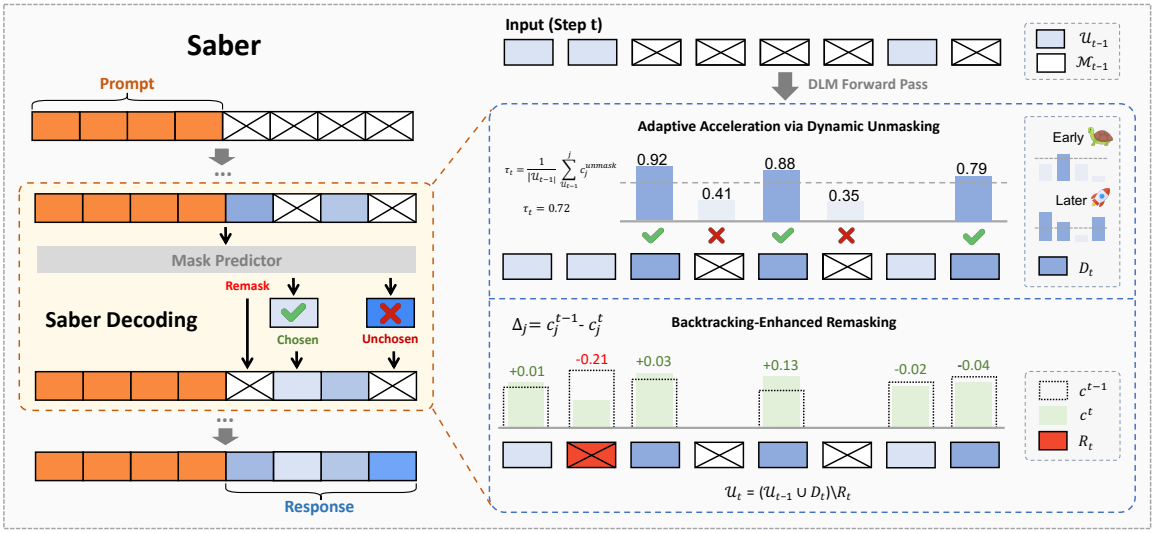

Figure 3: An Overview of Saber in DLM sampling, which consists of two key components, i.e., Adaptive Acceleration via Dynamic Unmasking (AADU) and Backtracking-Enhanced Remasking Mechanism (BERM), during each iterative sampling process.

The inherent capabilities of DLMs in global planning and iterative optimization make them naturally suited for code generation (Gong et al., 2025b; Li et al., 2025). Therefore, the application of DLMs to this domain has become a major research focus (DeepMind, 2025; Gong et al., 2025b; Xie et al., 2025; Khanna et al., 2025). However, these works mainly focus on the training process of DLMs, while Saber is a training-free DLM sampling method and is orthogonal to them.

3.2 Efficient DLM Sampling Methods

The efficiency of DLMs stems from their ability to generate multiple tokens in parallel (Luxembourg et al., 2025; Yu et al., 2025; Hong et al., 2025; Huang et al., 2025). Some studies accelerates this process by setting a fixed threshold, such as Fast-dLLM (Wu et al., 2025), WINO (Hong et al., 2025), and EB-Sampler (Ben-Hamu et al., 2025). However, attempting to unmask multiple tokens in each step degrades the final output quality (Li et al., 2025; Zhang et al., 2025; Wu et al., 2025). Moreover, ReMDM (Wang et al., 2025) proposes a phased sampler that can remask the generated tokens during one of the generation phases. However, the aforementioned methods are less effective on code generation tasks.

To the best of our knowledge, we are the first to combine adaptive acceleration and backtracking enhanced remasking to achieve improvement for both inference speed and output quality in DLM sampling.

4 Saber

In this section, we first provide the preliminaries for DLM sampling (§ 4.1), and then describe the two key components of Saber: Adaptive Acceleration via Dynamic Unmasking (§ 4.2) and Backtracking-Enhanced Remasking Mechanism (§ 4.3). Finally, we provide the overview of Saber (§ 4.4) in DLM sampling, which is also illustrated in Figure 3.

4.1 Preliminaries

Let a token sequence of length L be denoted by $x = (x_1, \ldots, x_L)$, where each token x_i belongs to a vocabulary $\mathcal{V}$. In the diffusion process, we use a special token '[MASK]'. At any denoising step t, the sequence x_t consists of a set of unmasked tokens at indices $\mathcal{U}_t$ and a set of masked tokens at indices $\mathcal{M}_t$. The DLM p_θ, parameterized by θ, takes the partially masked sequence x_t as input and outputs a probability distribution over the vocabulary for each masked position $i \in \mathcal{M}_t$. We define the model's confidence in its top prediction for a masked token i as c_i:

$$c_i := \max_{v \in \mathcal{V}} p_\theta(x_i = v \mid x_t), \tag{1}$$

where the mainstream DLM sampling method is to greedily unmask the single token with the highest confidence at each step. Saber improves upon this by the following two key components.

4.2 Adaptive Acceleration via Dynamic Unmasking

The first component of Saber aims to accelerate inference by unmasking multiple tokens in parallel. Motivated by our observation that the model's prediction difficulty is non-uniform, we introduce a dynamic and adaptive threshold τ_t to determine which tokens to unmask. This threshold is calculated as the average confidence of all previously unmasked tokens:

$$\tau_t = \begin{cases} \frac{1}{|\mathcal{U}_{t-1}|} \sum_{j \in \mathcal{U}_{t-1}} c_j^{\text{unmask}} & \text{if } t > 0, \\ c_{max} & \text{otherwise,} \end{cases} \quad (2)$$

where c_j^{unmask} is the confidence score of token j at the step it was unmasked, and we initialize τ_0 to c_{max} for the initial step.

This dynamic threshold naturally encourages a cautious-to-aggressive decoding trajectory. In early steps, when the context is sparse and average confidence is low, τ_t is low, allowing only the most certain tokens to be unmasked. As more high-confidence tokens are generated, τ_t rises, permitting more aggressive parallel unmasking in later, more context-rich stages. Using the threshold τ_t, we identify a set of candidate tokens to be drafted, $\mathcal{D}_t$, which includes all masked tokens whose current confidence exceeds τ_t:

$$\mathcal{D}_t = \{i \in \mathcal{M}_{t-1} \mid c_i \geq \tau_t\}, \quad (3)$$

where the tokens are provisionally unmasked with their most likely prediction.

4.3 Backtracking-Enhanced Remasking Mechanism

The second component of Saber introduces a backtracking mechanism to correct for potential errors made during the aggressive generation in the previous stage. This step is crucial for preventing the error propagation that causes performance collapse.

Unlike methods that use a fixed threshold, Saber's backtracking mechanism first determines the number of tokens to revise, μ_t, based on how aggressively it generated tokens in the current step:

$$\mu_t = \max(1, \lfloor |\mathcal{D}_t|/\mu \rfloor), \quad (4)$$

where $|\mathcal{D}_t|$ is the size of the newly unmasked set and μ is a hyperparameter. This ensures that we revise at least one token while limiting the revision to a small fraction of the current step's output to maintain speed.

Algorithm 1 Pseudocode of Saber in each step.

Input: Sequence x_{t-1}, DLM p_θ, unmasked indices $\mathcal{U}_{t-1}$, unmasked confidences c^{unmask}
Output: Updated sequence x_t
// S1: Adaptive Acceleration
Compute confidences c_i for all $i \in \mathcal{M}_{t-1}$ using $p_\theta(\cdot \mid x_{t-1})$.
if $t > 0$ **then**
    $\tau_t \leftarrow \frac{1}{|\mathcal{U}_{t-1}|} \sum_{j \in \mathcal{U}_{t-1}} c_j^{\text{unmask}}$.
else
    $\tau_t \leftarrow c_{max}$.
end if
$\mathcal{D}_t \leftarrow \{i \in \mathcal{M}_{t-1} \mid c_i > \tau_t\}$.
Create candidate sequence x'_t by unmasking tokens in $\mathcal{D}_t$.
// S2: Backtracking-Enhanced Remasking
$\mu_t \leftarrow \max(1, \lfloor |\mathcal{D}_t|/\mu \rfloor)$.
Re-evaluate confidences c_j^t for all $j \in \mathcal{U}_{t-1}$ using the new context $p_\theta(\cdot \mid x'_t)$.
Initialize an empty set for confidence drops Δ.
for each token $j \in \mathcal{U}_{t-1}$ **do**
    $\Delta_j \leftarrow c_j^{t-1} - c_j^t$. ▷ Calculate drop
    Add (j, Δ_j) to Δ.
end for
$\mathcal{R}_t \leftarrow$ indices of the μ_t tokens from Δ with the largest drop.
Create final sequence x_t by re-masking tokens at indices $\mathcal{R}_t$ in x'_t.
Update c^{unmask} by removing confidences for $j \in \mathcal{R}_t$ and adding confidences for $i \in \mathcal{D}_t$.
$\mathcal{U}_t \leftarrow (\mathcal{U}_{t-1} \cup \mathcal{D}_t) \setminus \mathcal{R}_t$.
return $x_t, \mathcal{U}_t, c^{\text{unmask}}$.

Then, we identify which tokens to revise by focusing on those previously unmasked tokens that are most inconsistent with the newly available context. For each existing token $j \in \mathcal{U}_{t-1}$, we compute its confidence drop, Δ_j, defined as the difference between the unmasked confidences of (t-1)-th time c_j^{t-1}, and its re-evaluated confidence at the current step c_j^t:

$$\Delta_j = c_j^{t-1} - c_j^t, \quad (5)$$

where a large Δ_j indicates that the model's confidence in its earlier prediction has significantly weakened. We then identify the set of tokens to be reversed, $\mathcal{R}_t \subseteq \mathcal{U}_{t-1}$, by selecting the μ_t tokens that exhibit the largest confidence drop. These are the tokens the model has the most regret about, and they are reverted to '`[MASK]`' to be reconsidered in future steps with a richer context.

4.4 Overall Procedure of Saber

At the conclusion of each step t, the final set of unmasked tokens is updated by integrating the outcomes of both the adaptive acceleration and backtracking stages:

$$\mathcal{U}_t = (\mathcal{U}_{t-1} \cup \mathcal{D}_t) \setminus \mathcal{R}_t. \tag{6}$$

By combining adaptive acceleration with an efficient backtracking mechanism, Saber can decode aggressively while pruning the most probable errors, thus achieving a superior balance between inference speed and generation quality. The pseudocode of Saber in each DLM sampling step is summarized in Algorithm 1.

5 Theoretical Analysis

To provide analytical support for the Saber algorithm and explain why it mitigates the failure of standard DLMs in code generation, we analyze the DLM decoding process by modeling the evolution of errors across sampling steps.

Error Accumulation in the Standard DLMs. Given the strict structural dependencies of programming languages, early generation errors severely contaminate the bidirectional context, irreversibly amplifying the model's prediction uncertainty for subsequent tokens. Let e_t be the random variable representing the number of structural errors rigidly locked in the unmasked context at step t, and let $\mathrm{err}(\mathcal{D}_t) = \sum_{k \in \mathcal{D}_t} \mathbb{I}(\mathrm{err}_k)$ denote the number of errors within the newly unmasked candidate set $\mathcal{D}_t$. In traditional static parallel DLM sampling, the error evolution is purely monotonic:

$$\mathbb{E}[e_t \mid x_{t-1}] = \mathbb{E}[e_{t-1} \mid x_{t-1}] + \mathbb{E}[\mathrm{err}(\mathcal{D}_t) \mid x_{t-1}]. \tag{7}$$

Saber addresses this through two mechanisms: Adaptive Acceleration via Dynamic Unmasking (AADU) and the Backtracking-Enhanced Remasking Mechanism (BERM).

AADU (Bounding New Errors). By utilizing a dynamic threshold τ_t to conservatively constrain the unmasking candidate set $\mathcal{D}_t$, AADU establishes a dynamic fault-tolerance boundary. Assuming the DLM's subjective confidence serves as a proxy for token accuracy with an allowable calibration slack (formalized in Appendix A.1), we establish the following bound.

Lemma 5.1 (Step-wise Error Injection Bounding)**.** *Given the context x_{t-1} and a model overconfidence slack $\delta_{calib} \geq 0$, the conditional expected number of newly injected errors is bounded by:*

$$\mathbb{E}[\mathrm{err}(\mathcal{D}_t) \mid x_{t-1}] \leq |\mathcal{D}_t|(1 - \tau_t + \delta_{calib}). \tag{8}$$

While Lemma 5.1 holds for any threshold, our specific choice of using the historical average of unmasked tokens for τ_t serves as an effective empirical way to track the model's evolving certainty, safely expanding $|\mathcal{D}_t|$ only when the context becomes reliable.

BERM (Pruning Contextual Conflicts). While AADU bounds the injection of *new* errors, BERM is designed to mitigate the strict monotonic accumulation of *past* errors, as shown in Eq. (7). It evaluates the step-wise confidence drop (i.e., Δ_j) of previously generated tokens to identify contextual conflicts (see Appendix A.1).

Proposition 5.2 (Degradation Pruning). By selecting the μ_t tokens with the largest confidence drop to form the rollback set $\mathcal{R}_t$, BERM selects the subset of historical context that minimizes the upper bound of the total step-wise reliability degradation within the retained context for a given pruning budget μ_t.

System Evolution under Saber. By combining AADU and BERM, Saber aims to transform the monotonically increasing error accumulation into a more controlled process. Parallel unmasking inherently introduces a contextual collision penalty bounded by ϵ (Assumption 3), stemming from ignored conditional dependencies. Because ϵ scales quadratically with the unmasked set size $|\mathcal{D}_t|$, it could easily overwhelm a linear pruning budget. Here, the synergy of Saber is critical: AADU bounds $|\mathcal{D}_t|$ to prevent ϵ from exploding, while BERM actively offsets the remaining structural errors. Relying on assumptions regarding local reliability preservation (Assumption 2), as derived in Appendix A.2, by dynamically bounding the new errors (via AADU) and actively pruning the most conflicting tokens (via BERM), the evolution of Saber's expected error bound can be formulated as:

$$\mathbb{E}[e_t^{\text{Saber}} \mid x_t] \leq \mathbb{E}[e_{t-1} \mid x_{t-1}] - \underbrace{\sum_{i \in \mathcal{R}_t} \mathbb{P}(\mathrm{err}_i \mid x_{t-1})}_{\text{BERM Pruning}} + \underbrace{\sum_{k \in \mathcal{D}_t} \mathbb{P}(\mathrm{err}_k \mid x_{t-1})}_{\text{AADU Bounding}} + \epsilon.$$

Table 1: Comparison of Saber and existing DLM sampling methods, where the **bold** indicates the best performance in this column while the underline indicates the second-best performance, and ET means the Pass@1 performance on its extended test case version.

Method	HumanEval				MBPP				LiveCodeBench		
	Pass@1 ↑	ET ↑	Step ↓	Time ↓	Pass@1 ↑	ET ↑	Step ↓	Time ↓	Pass@1 ↑	Step ↓	Time ↓
Standard DLM Sampling											
Random	14.63	12.80	256	1:29:40	22.95	18.26	256	2:51:28	0	256	4:09:49
Entropy	41.46	34.15	256	1:30:22	42.15	31.14	256	2:56:42	4.00	256	4:30:31
Confidence	43.29	35.79	256	2:11:52	42.86	31.38	256	3:12:08	9.75	256	5:59:07
Efficient DLM Sampling											
Confidence (p=2)	34.76	28.66	128	51:13	40.75	28.57	128	1:35:13	9.25	128	2:57:16
SAR (p=2)	35.98	29.27	128	1:33:00	40.05	27.86	128	1:36:05	9.50	128	2:57:17
Fast-dLLM	39.63	34.15	256	59:40	44.03	30.44	256	2:30:24	8.75	256	2:33:29
Fast-dLLM (+parallel)	39.63	33.54	**96.24**	**25:25**	39.34	27.63	**73.13**	**43:18**	2.30	96.28	**43:22**
ReMDM	20.73	18.29	128	1:26:50	31.62	22.48	128	1:28:51	3.30	128	2:50:23
WINO	40.24	31.71	100.12	57:10	43.09	31.38	88.49	1:44:51	9.25	**77.43**	2:40:30
Saber	**45.12**	**35.98**	118.92	**41:55**	**44.73**	**33.02**	110.96	1:33:33	**11.00**	122.47	2:33:17

Superiority over Traditional Bounds. In traditional static parallel DLM sampling, the expected error accumulates monotonically without any pruning mechanism. In contrast, Saber disrupts this monotonic accumulation, which can achieve a lower theoretical error bound, $\mathbb{E}[e_t^{\text{Saber}} \mid x_t] < \mathbb{E}[e_t^{\text{Traditional}} \mid x_t]$. We empirically observe and validate this penalty-offsetting condition in our ablation studies (Section 6). The detailed mathematical formulation of the traditional bound and the comparative proof are provided in Appendix A.3.

Summary. In strong structural constraints tasks, such as code generation, the underlying structural dependencies are highly non-linear. A single early token error catastrophically cascades, causing the entire structure to collapse. Therefore, keeping the sequence entirely error-free ($e_t = 0$) throughout decoding is paramount. By tightly bounding early error injection and actively pruning degraded tokens to offset parallel collision penalties, Saber effectively preserves the code's structural integrity. We note that this analytical framework operates at the per-step level. The end-to-end error dynamics throughout the generation trajectory are validated by our subsequent experiments.

6 Experimental Results

In this section, we present a comprehensive empirical evaluation of Saber. We first compare its performance and efficiency against a wide range of existing DLM sampling methods on multiple code generation benchmarks (§6.1). Next, we demonstrate the model-agnostic nature of Saber by applying it to various state-of-the-art DLMs (§6.2). Finally, we conduct a detailed ablation study to dissect the individual contributions of our proposed components (§6.3) and provide the discussion of Saber (§6.4). The detailed description of experiment setups can be found in Appendix J.

6.1 Main Results

Table 1 presents the main results of our comparison on the HumanEval, MBPP, HumanEval-ET and MBPP-ET, and LiveCodeBench datasets. The findings clearly demonstrate that Saber sets a new state-of-the-art for DLM sampling in code generation, achieving the highest Pass@1 scores across all benchmarks while simultaneously delivering substantial improvements in inference speed.

Saber Effectively Mitigates the Speed-Quality Trade-off. Compared to standard DLM sampling strategies (Random, Entropy, Confidence), Saber delivers vastly superior performance. For instance, on HumanEval, Saber improves the Pass@1 score from 43.3% (Confidence) to 45.1% while reducing the inference time by nearly 70% (from over 2 hours to just 41 minutes). This result directly refutes the notion that acceleration must come at the cost of quality. While naively increasing parallelism by generating more tokens per step (e.g., Confidence p=2) leads to a significant performance drop (from 43.3% to 34.8%), Saber's intelligent sampling process successfully avoids this collapse.

Saber Outperforms State-of-the-Art Efficient Samplers. When compared to recent efficient sampling methods, Saber establishes a new Pareto frontier for the speed-quality trade-off. WINO, a strong baseline, achieves impressive speed by minimizing decoding steps. However, Saber is even faster

in terms of time on most benchmarks, indicating a more efficient computation per step. For example, on HumanEval, Saber is over 25% faster than WINO while also achieving a ∼5% higher Pass@1 score. This superior performance is attributed to our backtracking mechanism, which provides a safety net for the adaptive acceleration, allowing for aggressive parallelization without sacrificing accuracy. Similarly, while Fast-dLLM shows competitive results on MBPP, Saber matches its quality while being nearly 40% faster. On LiveCodeBench, a benchmark designed to be robust against contamination, Saber also achieves the state-of-the-art performance, demonstrating its strong generalization capabilities.

Overall, these results confirm that Saber successfully breaks the existing speed-quality compromise in DLM sampling for code generation.

6.2 Generalizability Across Different DLMs

To validate the model-agnostic claim of Saber, we apply it to three distinct open-source DLMs, i.e., `LLaDA-8B-Instruct` (Nie et al., 2025), `Dream-v0-Instruct-7B` (Ye et al., 2025b), and `DiffuCoder-7B-cpGRPO` (Gong et al., 2025b). We compare the performance of Saber against the standard confidence-based sampler for each DLM on the HumanEval benchmark.

Table 2: Effectiveness of Saber compared to mainstream DLM sampling method based on different DLMs.

	Pass@1 ↑	Steps ↓	Time ↓
LLaDA-8B-Instruct			
Confidence (p=1)	0.4329	256	2:11:52
Saber	**0.4512**	**118.92**	**41:55**
Dream-v0-Instruct-7B			
Confidence (p=1)	0.2805	256	1:16:15
Saber	**0.2927**	**156.68**	**46:39**
DiffuCoder-7B-cpGRPO			
Confidence (p=1)	0.5671	256	1:12:47
Saber	**0.5732**	**140.34**	**37:08**

As shown in Table 2, Saber consistently improves both accuracy and efficiency across all tested models, demonstrating that its benefits are not tied to a specific architecture or training process. For each model, Saber delivers a higher Pass@1 score while simultaneously reducing the number of decoding steps and the total inference time. For instance, on `Dream-v0-Instruct-7B`, Saber boosts Pass@1 and cuts inference time by nearly 40%. On `DiffuCoder-7B`, a model specifically optimized for code, Saber further enhances its performance while reducing the inference time by nearly half.

To evaluate the generalizability of Saber from more dimensions, we conduct experiments on different settings, domains, and benchmarks in Appendix D, Appendix E, and Appendix H, respectively. Notably, beyond code generation, Saber also demonstrates consistent improvements on mathematical reasoning and scientific reasoning tasks (See Appendix E). This robust performance across different model families, settings, domains, and benchmarks validates that Saber addresses fundamental challenges in DLM sampling, making it a general, plug-and-play enhancement.

6.3 Ablation Study

To understand the individual contributions of the two core components of Saber, i.e., Adaptive Acceleration via Dynamic Unmasking and Backtracking-Enhanced Remasking Mechanism, we conduct a thorough ablation study on the HumanEval dataset. The results are presented in Table 3.

Adaptive Acceleration is the Primary Driver of Efficiency. When we remove the Adaptive Acceleration via Dynamic Unmasking, the sampler relies solely on the backtracking mechanism. While the Pass@1 score remains high at 44.5%, the number of decoding steps reverts to the baseline 256, and the inference time increases dramatically to over 90 minutes. This clearly demonstrates that the adaptive acceleration component is the main source of Saber's speedup.

Table 3: Ablation study of different components in our proposed method.

Method	Pass@1 ↑	Steps ↓	Time ↓
Ours	**0.4512**	118.92	41:55
w/o Adaptive Accelerate	0.4451	256	1:32:33
w/o Backtracking Remask	0.3523	**65.67**	**28:30**
w/o both	0.3476	128	51:13
Δ confidence from init.	0.4207	121.46	42:32

Backtracking is Essential for High Quality. Conversely, when we remove Backtracking-Enhanced Remasking Mechanism, the sampler becomes a purely aggressive adaptive accelerator. This variant is extremely fast, finishing in under 30 minutes with only 65.67 steps on average. However, this speed comes at a steep price: the Pass@1 score drops significantly from 45.1% to 35.23%. This result highlights that aggressive parallelization without a corrective mechanism is prone to error

propagation, confirming that the backtracking stage is crucial for maintaining high generation quality.

Synergy of Components. Saber achieves the best of both worlds, i.e., a high Pass@1 score of 45.1% and a fast inference time of ∼41 minutes, which also shows that the two components are synergistic. The adaptive acceleration allows for aggressive sampling, while the backtracking mechanism provides the necessary safety net to prune errors, enabling a combination of speed and accuracy that neither component can achieve alone. We also validate our dynamic thresholding strategy by replacing it with the average threshold of init generation of tokens (Δ confidence from init.). This results in a lower Pass@1 score of 42.1%, confirming the benefits of an adaptive approach that adjusts to the evolving context.

6.4 Qualitative Analysis

Error Type Analysis. We conduct an error type analysis and categorize the generation failures into three distinct types: Syntax Errors, Compilation/Runtime Errors, and Semantic Errors. As shown in Table 4, Saber consistently outperforms the standard baseline across all error categories. Notably, Saber significantly reduces Syntax Errors by 66.7% and Compilation/Runtime Errors by 21.5%. This quantitative evidence robustly supports our theoretical assertion that the backtracking-enhanced remasking mechanism effectively mitigates structural and syntactical error propagation during parallel decoding.

Table 4: Error type analysis.

Error Type	Baseline	Saber	$\downarrow \Delta$
Syntax Error	3	**1**	66.7%
Compilation/Runtime Error	14	**11**	21.5%
Semantic Error	92	**73**	20.7%

Case Study. Figure 4 presents a side-by-side comparison of code generated by the default LLaDA sampler and Saber on two problems from the HumanEval benchmark. These examples highlight how Saber's ability to self-correct prevents the kind of logical failures that plague standard irreversible samplers.

In Problem 1, the default sampler produces code, which is syntactically plausible but logically nonsensical. In contrast, Saber generates the correct, standard nested loop structure. This suggests that the iterative refinement process, guided by backtracking, helps enforce logical and structural coherence, which is paramount in code generation. In Problem 2, the default sampler fundamentally misunderstands the problem's constraints. Saber, however, correctly decomposes the problem into its core logical components: checking the array's length and verifying the occurrence count of the maximum element. This ability to correctly construct multi-step, constraint-based logic is a direct benefit of the backtracking mechanism. We hypothesize that the model may initially draft a simpler, incorrect solution, which is then revised in subsequent steps as the evolving context makes the error more apparent, leading to the robust final code.

7 Conclusion

In this paper, we addressed the critical speed-quality trade-off for DLM sampling in code generation and introduced Saber, a novel, training-free sampling algorithm for DLMs that combines both adaptive acceleration via dynamic unmasking and backtracking-enhanced remasking mechanism. Our extensive experiments indicate that Saber substantially outperforms existing DLM sampling methods with great generalizability, significantly narrowing the performance gap with autoregressive models in code generation. Moreover, results on mathematical reasoning and scientific reasoning benchmarks suggest that Saber generalizes beyond code generation. We leave a broader exploration across diverse domains and tasks as future work.

8 Limitation

Our work has the following two main limitations.

First, Saber requires slightly more computational resources than direct sampling in a DLM sampling step. Specifically, Saber requires $O(N)$ CPU-based operations (where N is the sequence length) to execute the dynamic unmasking and backtracking remasking logic. However, compared to the massive GPU-intensive matrix multiplications required by DLM forward passes, it is marginal and acceptable. Furthermore, since Saber significantly reduces the total number of denoising iterations, the overall computational cost is actually lower than that of standard sampling. Detailed discussion can be found in Appendix C.

Second, we only explore the choice of hyperparameters within reasonable ranges, considering the trade-off between performance and speed, as

shown in Table 8 and the right of Figure 1. Further optimization of hyperparameters could yield additional improvements.

9 Acknowledgments

This research is supported by the National Natural Science Foundation of China under Grant No. 62192733, 62192730, 62192731, the National Key R&D Program under Grant No. 2023YFB4503801, and the Beijing Major Science and Technology Project under Contract No. Z251100008425005.

References

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021a. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems 34, NeurIPS 2021*, pages 17981–17993.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021b. Program synthesis with large language models. *ArXiv*, abs/2108.07732.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. 2025. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *ArXiv*, abs/2505.24857.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33, NeurIPS 2020*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021a. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021b. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374.

DeepMind. 2025. Gemini diffusion.

Yihong Dong, Jiazheng Ding, Xue Jiang, Ge Li, Zhuo Li, and Zhi Jin. 2025a. Codescore: Evaluating code generation by learning code execution. *ACM Trans. Softw. Eng. Methodol.*, 34(3):77:1–77:22.

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-collaboration code generation via ChatGPT. *ACM Trans. Softw. Eng. Methodol.*, 33(7):189:1–189:38.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025b. A survey on code generation with llm-based agents. *CoRR*, abs/2508.00083.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony S. Hartshorn, Aobo Yang, and et al. 2024. The llama 3 herd of models. *ArXiv*, abs/2407.21783.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, Hao Peng, and Lingpeng Kong. 2025a. Scaling diffusion language models via adaptation from autoregressive models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*.

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

Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, and 1 others. 2017. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. In *ACL (1)*, pages 7212–7225. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *CoRR*, abs/2401.14196.

Zhengfu He, Tianxiang Sun, Kuan Wang, Xuanjing Huang, and Xipeng Qiu. 2022. Diffusionbert: Improving generative masked language models with diffusion models. In *Annual Meeting of the Association for Computational Linguistics, ACL 2023*.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and

Jiangchao Yao. 2025. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms. *ArXiv*, abs/2507.18578.

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

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

Xue Jiang, Yihong Dong, Yongding Tao, Huanyu Liu, Zhi Jin, and Ge Li. 2025. ROCODE: integrating backtracking mechanism and program analysis in large language models for code generation. In *ICSE*, pages 334–346. IEEE.

Xue Jiang, Yihong Dong, Lecheng Wang, Qiwei Shang, and Ge Li. 2024. Self-planning code generation with large language models. *ACM Trans. Softw. Eng. Methodol.*, 33(7):182:1–182:30.

Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. 2025. Mercury: Ultra-fast language models based on diffusion. *ArXiv*, abs/2506.17298.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, and 48 others. 2023. Starcoder: may the source be with you! *Trans. Mach. Learn. Res.*, 2023.

Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. 2025. A survey on diffusion language models. *ArXiv*, abs/2508.10875.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori Hashimoto. 2022a. Diffusion-lm improves controllable text generation. In *Advances in Neural Information Processing Systems 35, NeurIPS 2022*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022b. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, Fumin Wang, and Andrew W. Senior. 2016. Latent predictor networks for code generation. In *ACL (1)*. The Association for Computer Linguistics.

Omer Luxembourg, Haim H. Permuter, and Eliya Nachmani. 2025. Plan for speed - dilated scheduling for masked diffusion language models. *ArXiv*, abs/2506.19037.

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*, abs/2502.09992.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. 2025. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, and 1 others. 2018. Improving language understanding by generative pre-training. San Francisco, CA, USA.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Veselin Raychev, Martin T. Vechev, and Eran Yahav. 2014. Code completion with statistical language models. In *PLDI*, pages 419–428. ACM.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, and 6 others. 2023. Code llama: Open foundation models for code. *ArXiv*, abs/2308.12950.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aur'elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. Remasking discrete diffusion models with inference-time scaling. *ArXiv*, abs/2503.00307.

Yue Wang, Weishi Wang, Shafiq R. Joty, and teven C. H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *EMNLP (1)*, pages 8696–8708.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent abilities of large language models. *ArXiv*, abs/2206.07682.

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

Zhihui Xie, Jiacheng Ye, Lin Zheng, Jiahui Gao, Jingwei Dong, Zirui Wu, Xueliang Zhao, Shansan Gong, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025. Dream-coder 7b: An open diffusion language model for code.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025a. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*.

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

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

Lingzhe Zhang, Liancheng Fang, Chiming Duan, Minghua He, Leyi Pan, Pei Xiao, Shiyu Huang, Yunpeng Zhai, Xuming Hu, Philip S. Yu, and Aiwei Liu. 2025. A survey on parallel text generation: From parallel decoding to diffusion language models. *ArXiv*, abs/2508.08712.

A Extended Analytical Formulation and Proofs

A.1 Formal Assumptions and Definitions

*Throughout our analysis, $\mathbb{P}(err_i \mid x)$ denotes the **true objective error probability** at position i under the reference ground truth distribution.*

Assumption 1 (Tight Confidence Calibration with Slack)**.** *We assume the DLM's predicted confidence score provides a tight two-sided bound on the true error probability evaluated under any arbitrary conditioning context x, subject to a calibration slack $\delta_{calib} \geq 0$. Specifically, the conditional probability of a prediction error at position i satisfies:*

$$|\,\mathbb{P}(\mathrm{err}_i \mid x) - (1 - c_i^x)| \leq \delta_{calib}, \tag{9}$$

where c_i^x is the predicted confidence given context x, i.e., $p_\theta(x_i = y_i^* \mid x)$. (for simplicity in our formulations, we use c_i^t to denote the confidence given context x_t, and c_i^{t-1} given x_{t-1}).

While modern neural networks consistently exhibit context- and position-dependent calibration issues, explicitly bounding the deviation globally via δ_{calib} serves as a simplifying mathematical assumption. In practice, assuming equality holds in expectation, variations in δ_{calib} are implicitly absorbed by our conservative dynamic thresholding strategy.

Definition 1 (Step-wise Reliability Degradation)**.** For a historical token $j \in \mathcal{U}_{t-1}$, the degradation in reliability between the previous step $t-1$ and the current step t is quantified by its step-wise confidence drop, defined as:

$$\Delta_j = c_j^{t-1} - c_j^t. \tag{10}$$

A larger Δ_j indicates that the model's certainty regarding the previously generated token j has been significantly weakened by the newly established context x_t.

Assumption 2 (Local Reliability Preservation via Conflict Resolution)**.** *A high confidence drop (Δ_j) indicates that a previously generated token j is highly incompatible with the newly decoded context $\mathcal{D}_t$. We assume that by removing the μ_t most conflicting tokens ($\mathcal{R}_t$), BERM acts as a localized coordinate descent step. Driven by the Maximum A Posteriori (MAP) objective of the masked diffusion model, we assume this targeted pruning resolves the largest contextual contradictions, ensuring the expected reliability of the remaining retained context ($\mathcal{U}_{t-1} \setminus \mathcal{R}_t$) does not severely deteriorate:*

$$\sum_{j \in \mathcal{U}_{t-1} \setminus \mathcal{R}_t} \mathbb{P}(\mathrm{err}_j \mid x_t) \leq \sum_{j \in \mathcal{U}_{t-1} \setminus \mathcal{R}_t} \mathbb{P}(\mathrm{err}_j \mid x_{t-1}). \tag{11}$$

While masking tokens can theoretically disrupt long-range syntactic dependencies in non-autoregressive generation, we treat this targeted pruning as a first-order approximation that primarily resolves severe local structural conflicts without catastrophically destabilizing the broader context.

Assumption 3 (Bounded Contextual Collision in Parallel Unmasking)**.** *Unlike purely autoregressive decoding, unmasking multiple tokens ($\mathcal{D}_t$) simultaneously introduces joint dependencies. We encapsulate this structural collision penalty as ϵ, which is upper-bounded by a quadratic function of the unmasked set size:*

$$\epsilon \leq C \cdot |\mathcal{D}_t|^2, \tag{12}$$

where $C > 0$ is a constant representing the maximum pairwise mutual information penalty. In fully autoregressive decoding, the joint probability incorporates full dependencies: $p(x_A, x_B) = p(x_A)p(x_B \mid x_A)$. Parallel unmasking simplifies this by assuming local conditional independence: $p(x_A, x_B) \approx p(x_A)p(x_B)$. Since a set of size $|\mathcal{D}_t|$ contains $\binom{|\mathcal{D}_t|}{2}$ interacting pairs, the structural collision approximation error naturally scales quadratically. This highlights the necessity of AADU: without dynamically bounding $|\mathcal{D}_t|$, the quadratic growth of ϵ would inevitably exceed the BERM's linear pruning capacity.

A.2 Detailed System Evolution Derivation

In sequence generation, the error probability of any token is strongly conditioned on the current context. When transitioning from step $t-1$ to t, the context evolves from x_{t-1} to x_t based on the updated unmasked set $\mathcal{U}_t = (\mathcal{U}_{t-1} \setminus \mathcal{R}_t) \cup \mathcal{D}_t$. Let $e_t^{\text{Saber}} = \sum_{m \in \mathcal{U}_t} \mathbb{I}(\text{err}_m)$ denote the total number of errors in the unmasked set at step t.

By taking the conditional expectation with respect to the *updated* context x_t, applying the contextual preservation (Assumption 2), and incorporating the parallel collision penalty (Assumption 3), the upper bound of the system error at the end of step t is derived as:

$$\begin{aligned}
\mathbb{E}[e_t^{\text{Saber}} \mid x_t] &= \sum_{j \in \mathcal{U}_{t-1} \setminus \mathcal{R}_t} \mathbb{P}(\text{err}_j \mid x_t) + \sum_{k \in \mathcal{D}_t} \mathbb{P}(\text{err}_k \mid x_t) \\
&\leq \sum_{j \in \mathcal{U}_{t-1} \setminus \mathcal{R}_t} \mathbb{P}(\text{err}_j \mid x_{t-1}) + \sum_{k \in \mathcal{D}_t} \mathbb{P}(\text{err}_k \mid x_{t-1}) + \epsilon \\
&= \mathbb{E}[e_{t-1} \mid x_{t-1}] - \sum_{i \in \mathcal{R}_t} \mathbb{P}(\text{err}_i \mid x_{t-1}) + \sum_{k \in \mathcal{D}_t} \mathbb{P}(\text{err}_k \mid x_{t-1}) + \epsilon. \qquad (13)
\end{aligned}$$

Notice that the transition to the final equality holds by the linearity of expectation, where the total expected error at $t-1$ encapsulates the sum of probabilities. The third term $\sum_{k \in \mathcal{D}_t} \mathbb{P}(\text{err}_k \mid x_{t-1})$ is bounded by $|\mathcal{D}_t|(1 - \tau_t + \delta_{\text{calib}})$ per Lemma 5.1, while the pruning term is designed to offset the ϵ collision penalty.

A.3 Comparison with Traditional Bounds

In traditional static parallel DLM sampling, the expected error accumulates monotonically without any pruning mechanism. Given a candidate set $\mathcal{D}_t$, the error bound at step t is:

$$\mathbb{E}[e_t^{\text{Traditional}} \mid x_t] \leq \mathbb{E}[e_{t-1} \mid x_{t-1}] + |\mathcal{D}_t|(1 - \tau_{\text{static}} + \delta_{\text{calib}}) + \epsilon, \qquad (14)$$

where ϵ is the corresponding collision penalty, and τ_{static} denotes the fixed confidence threshold used by standard parallel decoding baselines.

In contrast, Saber disrupts this monotonicity. By dynamically bounding the new errors (via AADU) and actively pruning the most conflicting tokens (via BERM), Saber's expected error satisfies:

$$\mathbb{E}[e_t^{\text{Saber}} \mid x_t] \leq \mathbb{E}[e_{t-1} \mid x_{t-1}] - \sum_{i \in \mathcal{R}_t} \mathbb{P}(\text{err}_i \mid x_{t-1}) + \sum_{k \in \mathcal{D}_t} \mathbb{P}(\text{err}_k \mid x_{t-1}) + \epsilon. \qquad (15)$$

Because the pruning term explicitly offsets the collision penalty ϵ, and the dynamic thresholding restricts the injection of new errors, under the condition that $\sum_{i \in \mathcal{R}_t} \mathbb{P}(\text{err}_i \mid x_{t-1}) > |\mathcal{D}_t|(\tau_{\text{static}} - \tau_t)$, Saber can achieve a lower error bound $\mathbb{E}[e_t^{\text{Saber}} \mid x_t] < \mathbb{E}[e_t^{\text{Traditional}} \mid x_t]$.

A.4 Proofs

Proof of Lemma 5.1

Proof. Given the context x_{t-1}, the candidate set $\mathcal{D}_t$ and the threshold τ_t are deterministic. By the design of AADU, for every token index $i \in \mathcal{D}_t$, $c_i^t \geq \tau_t$ holds.

Based on Assumption 1 (specifically the upper bound property), the conditional probability of an error at position i is bounded:

$$\mathbb{P}(\text{err}_i \mid x_{t-1}) \leq 1 - c_i^t + \delta_{\text{calib}}.$$

By the linearity of expectation, the conditional expected number of errors drafted within $\mathcal{D}_t$ is:

$$\mathbb{E}[\text{err}(\mathcal{D}_t) \mid x_{t-1}] = \sum_{i \in \mathcal{D}_t} \mathbb{P}(\text{err}_i \mid x_{t-1}).$$

Since $c_i^t \geq \tau_t \implies 1 - c_i^t \leq 1 - \tau_t$ for all $i \in \mathcal{D}_t$, we can bound the expectation:

$$\mathbb{E}[\text{err}(\mathcal{D}_t) \mid x_{t-1}] \leq \sum_{i \in \mathcal{D}_t} (1 - c_i^t + \delta_{\text{calib}}) \leq \sum_{i \in \mathcal{D}_t} (1 - \tau_t + \delta_{\text{calib}}) = |\mathcal{D}_t|(1 - \tau_t + \delta_{\text{calib}}).$$

This establishes the conditional adaptive step-wise error injection bound, concluding the proof. □

Proof of Proposition 5.2

Proof. By Assumption 1 (tight two-sided calibration), the conditional probability bounds for any historical token $j \in \mathcal{U}_{t-1}$ satisfy $\mathbb{P}(\text{err}_j \mid x_t) \leq 1 - c_j^t + \delta_{\text{calib}}$ and $\mathbb{P}(\text{err}_j \mid x_{t-1}) \geq 1 - c_j^{t-1} - \delta_{\text{calib}}$. Thus, the relative conditional error growth for token j is bounded by:

$$\mathbb{P}(\text{err}_j \mid x_t) - \mathbb{P}(\text{err}_j \mid x_{t-1}) \leq (1 - c_j^t) - (1 - c_j^{t-1}) + 2\delta_{\text{calib}} = \Delta_j + 2\delta_{\text{calib}}.$$

Therefore, Δ_j acts as a valid proxy for the upper bound of the step-wise error degradation. The total upper bound of reliability degradation of the *retained* context $\mathcal{U}_{t-1}^{\text{retain}} = \mathcal{U}_{t-1} \setminus \mathcal{R}_t$ is proportional to:

$$\sum_{j \in \mathcal{U}_{t-1}} \Delta_j - \sum_{i \in \mathcal{R}_t} \Delta_i.$$

BERM partitions the historical context into the rollback set $\mathcal{R}_t$ (size μ_t) and the retained set $\mathcal{U}_{t-1}^{\text{retain}}$. By algorithm design, BERM selects $\mathcal{R}_t$ such that the elements with the largest confidence drop (Δ_i) are chosen. This guarantees that for any alternative pruning set $\mathcal{R}' \subseteq \mathcal{U}_{t-1}$ of size μ_t:

$$\sum_{i \in \mathcal{R}_t} \Delta_i \geq \sum_{k \in \mathcal{R}'} \Delta_k.$$

Because BERM explicitly maximizes the subtracted term $\sum_{i \in \mathcal{R}_t} \Delta_i$ via its greedy selection strategy, it minimizes the proxy upper bound of the remaining total error degradation within the retained context sequence given the pruning budget μ_t. This completes the argument under the stated assumptions. □

B Case Study

Figure 4 presents a comparison of code generated by the base model LLaDA and Saber on two HumanEval problems. In Task I, LLaDA fails to enumerate all subarrays correctly. It uses a single loop with an incorrect range derived from the distinct element count, producing both logically flawed and syntactically invalid code. Saber instead generates the correct nested loop structure that systematically checks every subarray. In Task II, LLaDA misinterprets the "Good array" definition. Its length check relies on an incorrect relationship, and its boundary check after sorting is logically impossible for a valid sorted array. Saber correctly decomposes the problem into verifying the length condition (*n=m+1n = m+1 n=m+1*) and the occurrence constraint (*count(m)=2count(m) = 2 count(m)=2*). These examples illustrate how Saber's backtracking mechanism helps avoid the kind of early structural errors that the irreversible base sampler cannot recover from.

C Computational Overhead Analysis

In this section, we provide a detailed complexity breakdown and a comparative analysis of Saber's operations. Saber introduces two primary sources of per-step overhead, both of which are strictly bounded by the sequence length N:

- **Adaptive Acceleration via Dynamic Unmasking:** The process involves calculating a dynamic threshold (by averaging the confidence probabilities of previously unmasked tokens) and scanning masked positions to select candidates. These operations require only a single sequential pass, resulting in a time complexity of $\mathcal{O}(N)$.

- **Backtracking-Enhanced Remasking Mechanism:** Identifying the lowest-confidence tokens to re-mask involves calculating confidence differences and finding the bottom-μ_t elements within the candidate set. This selection process can be executed efficiently in $\mathcal{O}(N)$ time using linear-time selection algorithms.

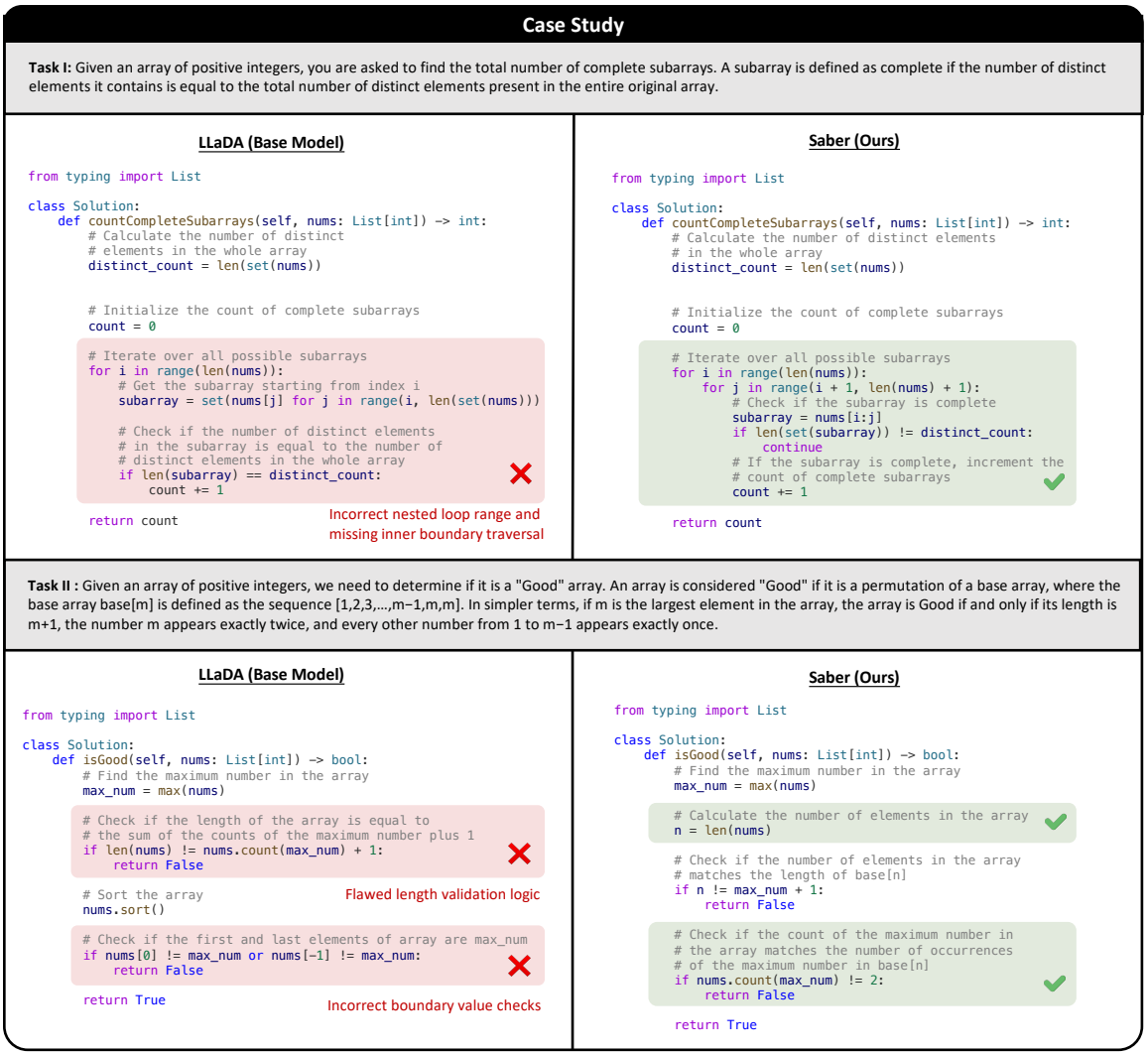

Figure 4: Case study.

Crucially, these linear $\mathcal{O}(N)$ algorithmic operations are remarkably lightweight and are executed as CPU-based logic. In stark contrast, a single forward pass of a Discrete Diffusion Language Model (DLM) requires massive, heavily parameterized matrix multiplications on the GPU—such as self-attention mechanisms and feed-forward networks—which represent billions of operations and dominate the actual inference latency.

Empirically, our runtime breakdown confirms this theoretical analysis. The algorithmic overhead of Saber does not noticeably inflate the per-step execution time. Instead, because Saber's adaptive and self-correcting mechanisms effectively reduce the total number of required denoising iterations, the marginal per-step CPU overhead is overwhelmingly offset by the reduction in expensive GPU forward passes. Therefore, the overall computational cost is lower than that of standard sampling. As demonstrated in our main results (Table 1), Saber achieves an average overall inference speedup of 251.4% under the condition of boosting Pass@1 accuracy by an average of 1.9%.

As a result, **the computational cost introduced by Saber in each step is entirely marginal, but its total computational cost is instead lower**, so it's highly acceptable for real-world code generation deployments.

D Evaluation with Different Configurations on other base DLMs

In our main experiments, standard decoding settings (e.g., temperature $= 0$) were adopted to maintain consistency with previous DLM sampling literature. To comprehensively evaluate performance under dif-

ferent configurations, we reproduced the experiments following the best settings of Dream and DiffuCoder, where we directly followed the settings reported in their original paper for Dream and we conducted a grid search to determine the optimal settings for DiffuCoder (since its original paper did not report). As shown in Table 5, Saber demonstrates stable and consistent improvements under these specific settings, validating the effectiveness of our approach across different configurations.

Table 5: Effectiveness of Saber compared to the baseline under the optimal settings of Dream and DiffuCoder on the HumanEval benchmark.

Base Model	Method	HumanEval ↑	HumanEval-ET ↑	Step ↓	Time ↓
Dream	Baseline	0.5749	0.4756	256	1:18:12
	Ours	**0.6037**	**0.5183**	**141.04**	**40:49**
DiffuCoder	Baseline	0.6829	0.5937	256	1:19:56
	Ours	**0.6890**	**0.6098**	**168.02**	**53:19**

E Generalizability Across Different Domains

To further evaluate the generalizability of Saber beyond code generation tasks, we extended our experiments to mathematical reasoning and general question-answering domains. Specifically, we evaluated our method against the default DLM sampling strategy on the MATH-500 and ARC-Challenge benchmarks. As shown in Table 6, Saber consistently demonstrates superior performance and efficiency across both datasets. On MATH-500, Saber improves the Pass@1 score from 32.4% to 34.0% while reducing the total inference time by nearly 50% and cutting the average decoding steps to 146. Similarly, on ARC-Challenge, Saber boosts accuracy from 53.51% to 56.19% with an even more substantial reduction in steps (from 256 to 129.66) and inference time. These results strongly confirm that the core mechanisms of Saber are not limited to tasks with strict structural constraints like code. Instead, Saber serves as a highly generalized, task-agnostic efficient sampling strategy for diffusion language models across diverse application scenarios.

Table 6: Performance comparison on MATH-500 and ARC-Challenge benchmarks.

Method	MATH-500			ARC-Challenge		
	Pass@1 ↑	Step ↓	Time ↓	Pass@1 ↑	Step ↓	Time ↓
Baseline	0.3240	256	3:01:02	0.5351	256	1:49:51
Saber	**0.3400**	**146.00**	**1:31:15**	**0.5619**	**129.66**	**0:49:02**

F Performance in Long-Context Scenarios

To explore the sampling efficiency and correction behavior of diffusion language models under longer input conditions, we evaluated Saber in long-context scenarios on tasks with input lengths between 512 and 4096 tokens. Although long-context generation presents a significant challenge for DLMs, Saber demonstrates remarkable robustness, achieving more than three times the Pass rate of the baseline method, as shown in Table 7.

Table 7: Performance comparison in long-context scenarios (input lengths between 512 and 4096 tokens).

Method	Performance	↑ Δ
Baseline	1.4%	-
Saber (Ours)	**4.3%**	**+207.2%**

G Hyperparameter Impact.

We conduct a hyperparameter impact analysis of the backtracking ratio μ on the HumanEval benchmark to evaluate the robustness of our approach. As shown in Table 8, Saber demonstrates strong robustness: the Pass@1 performance remains consistently high across a wide range of μ values (from $1/8$ to $1/2$). Crucially, all these configurations significantly outperform the baseline setting where no backtracking is applied ($\mu = 0$, yielding a Pass@1 of 0.3523). This confirms that while the exact choice of μ offers a flexible trade-off between inference speed and accuracy, the backtracking mechanism itself provides a stable and substantial quality improvement regardless of minor hyperparameter variations.

Table 8: Hyperparameter sensitivity analysis of the backtracking ratio μ.

μ	Pass@1 ↑	Steps ↓	Time ↓
1/2	0.4512	158.40	59:12
1/4	0.4451	128.55	48:54
1/8	0.4512	118.92	41:55
1/16	0.3963	109.93	39:59
0 (w/o Backtracking-Enhanced Remasking)	0.3523	65.67	28:30

H Extended Results of Generalizability across Different DLMs

To further validate the generalizability of Saber, we conducted broader evaluation on the MBPP and MBPP-ET datasets in our default setting. As detailed in Table 9, Saber achieves consistent improvements in Pass@1 scores, alongside significant reductions in decoding steps and inference time, across LLaDA, Dream, and DiffuCoder.

Table 9: Broader testing of Saber on MBPP and MBPP-ET benchmarks across different DLMs.

Base Model	Method	MBPP ↑	MBPP-ET ↑	Step ↓	Time ↓
LLaDA	Baseline	0.4286	0.3138	256	3:12:08
	Ours	**0.4473**	**0.3302**	**110.96**	**1:33:33**
Dream	Baseline	0.6182	0.4637	256	2:38:03
	Ours	**0.6206**	**0.4660**	**131.48**	**1:26:16**
DiffuCoder	Baseline	0.5152	0.3677	256	2:42:28
	Ours	**0.5691**	**0.4192**	**125.65**	**1:20:51**

I Extended Related Work

I.1 Code Generation

Since the advent of artificial intelligence in the 1950s, code generation has been considered the Holy Grail of computer science research (Gulwani et al., 2017). With the rapid expansion of codebases and the increasing capacity of deep learning models, using deep learning for program generation has shown great potential and practicality (Raychev et al., 2014; Ling et al., 2016; Dong et al., 2024, 2025b; Jiang et al., 2024, 2025). In recent years, the rise of pre-training techniques has brought new momentum to the field of code generation. For example, studies like CodeT5 (Wang et al., 2021) and UniXcoder (Guo et al., 2022) pre-train models for code generation tasks. With the continual increase in model parameters, researchers have discovered emergent phenomena in LLMs, leading to new breakthroughs . Against this backdrop, LLMs such as AlphaCode (Li et al., 2022b), Codex (Chen et al., 2021a), Starcoder (Li et al., 2023), CodeLlama (Rozière et al., 2023), and DeepSeek Coder (Guo et al., 2024) have emerged.

J Detailed Experimental Setup

In this section, we present the setups of our experiments below.

J.1 Datasets

We conduct experiments on five code generation datasets to demonstrate the effectiveness of Saber, including HumanEval (Chen et al., 2021b), MBPP (Austin et al., 2021b), HumanEval-ET and MBPP-ET (Dong et al., 2025a), and LiveCodeBench (Jain et al., 2025). For all datasets, tasks are presented in a zero-shot format.

- **HumanEval** is a widely used benchmark for evaluating LLMs' ability to generate correct Python functions from docstrings.
- **MBPP** (Mostly Basic Python Problems) consists of small-to-medium Python programming tasks designed to test basic algorithmic reasoning.
- **LiveCodeBench** is a contamination-aware benchmark that continuously collects new programming problems from contest platforms (LeetCode, AtCoder, Codeforces) and focuses beyond simple code generation to broader code reasoning capabilities.
- **HumanEval-ET and MBPP-ET** are extended versions of the original HumanEval and MBPP. They augment each task with over 100 additional test cases and include edge-case tests, which enhance the reliability of the evaluation.

J.2 Baselines

We conduct a comprehensive evaluation of Saber against established baseline decoding methods for DLMs. The results confirm that Saber achieves superior performance, effectively validating its effectiveness.

- **Standard DLM sampling (Default)**:In this mode, DLM generates responses by continuously decoding over a predetermined full output length. The decoding methods include confidence-based, entropy-based, and random approaches.
- **Efficient DLM sampling Methods**, including: **Semi-autoregressive (SAR) (Nie et al., 2025)**: This strategy decodes in blocks from left to right. It thus combines aspects of autoregressive order with diffusion's simultaneous updates. Within each block, tokens are decoded based on confidence; **parallelism increase (p)**: increasing the number of tokens per sampling; **WINO** (Hong et al., 2025): using a fixed threshold for acceleration; **Fast-dLLM** (Wu et al., 2025): using cache for acceleration; and **ReMDM** (Wang et al., 2025): using a fixed threshold for remasking.
- **Parallelism Increase (p)**, **Semi-autoregressive (SAR)** (Nie et al., 2025), **WINO (Hong et al., 2025)**, **Fast-dLLM (Wu et al., 2025)**, and **ReMDM** (Wang et al., 2025) are recently proposed efficient DLM sampling methods.

J.3 Metric

Our evaluation employs **Pass@1** as the primary metric. It is calculated as the percentage of problems for which the generated code passes all test cases with a single attempt. The formula is as follows:

$$\text{pass@1} = \frac{1}{|N|} \sum_{i=1}^{|N|} \mathbb{I}(\text{Passed}(\text{Generation}_i))$$

where $|N|$ is the total number of problems, and the indicator function $\mathbb{I}(\cdot)$ is 1 if the single generation for a given problem passes all its test cases, and 0 otherwise.

In addition to performance, we also measure the **Step** (i.e., average generation steps per sample) and **Time** (i.e., total generation time).

J.4 Implementation Details

In this paper, we employ the LLaDA-8B-Instruct (Nie et al., 2025) as the base model for our experiments. The default temperature for all baselines is set at 0 for consistency. In the fixed-length decoding scenario, we set the generation length to 256 tokens. For the semi-autoregressive approach, the block length was configured to 128. All other efficient DLM sampling methods follow the same configuration as their original paper. To mitigate the instability of model sampling, we report the average results of five trials in the experiments. All experiments were conducted on a workstation equipped with 8 NVIDIA A6000 GPUs (48GB each) and 1TB RAM, with a single GPU for each experiment.