Unlocking Coherent Parallel Decoding in Diffusion Language Models

Anonymous ACL submission

Abstract

Masked Diffusion Language Models (MDLMs) enable highly parallel text generation, but parallel token updates often degrade coherence due to conditional independence assumptions during decoding. We introduce **CAPDEC** (**C**orrelation-**A**ware **P**arallel **DEC**oding), a training-free and model-agnostic decoding algorithm that accounts for cross-token dependencies without additional forward passes. CAPDEC formulates parallel decoding as a joint optimization problem balancing token confidence and dependency-aware penalties derived from decoder hidden-state similarity.

Across diverse benchmarks and architectures, CAPDEC consistently improves parallel generation quality while preserving efficiency. On ParallelBench, CAPDEC improves average scores by 11.7%–27.1% over standard decoding baselines. On LiveCodeBench, it improves pass@1 by 16.7%–140.0% and increases syntax-correct generation rates by up to 100.0%, with less than 2% throughput overhead. We release code and evaluation scripts to support future research.

1 Introduction

Autoregressive (AR) language models generate text sequentially, limiting inference-time parallelism and increasing latency for long outputs (Leviathan et al., 2023; Gu et al., 2018). Discrete diffusion language models (dLLMs) address this limitation by iteratively denoising masked tokens in parallel using bidirectional context (Austin et al., 2021; Sahoo et al., 2024). Recent large-scale masked diffusion models demonstrate competitive performance with autoregressive baselines while enabling substantially more parallel decoding (Nie et al., 2025).

The efficiency of masked diffusion language models (MDLMs) relies on a conditional independence assumption: tokens selected in parallel at each denoising step are decoded independently given the current context (Gu et al., 2018; Sahoo et al., 2024). As a result, simultaneously generated positions cannot coordinate their predictions, which often leads to incoherent outputs. For example, when generating the phrase "San Jose" from two masked positions, independent sampling may produce repetitions such as "San San" or "Jose Jose" because both positions assign high probability to the same token (Figure 1). Recent work on ParallelBench (Kang et al., 2025) shows that such independence-induced errors fundamentally limit parallel decoding quality.

In this paper, we introduce **CAPDEC**, a **novel training-free and model-agnostic** decoding algorithm for masked diffusion language models. At each denoising step, CAPDEC selects the set of tokens to commit in parallel by trading off individual token confidence against pairwise dependencies among the candidates, favoring updates that are both confident and mutually compatible. Unlike prior methods that treat parallel positions independently, our approach explicitly accounts for these interactions, and does so without auxiliary models or additional forward passes. Across diverse benchmarks, CAPDEC consistently improves coherence and downstream task performance, narrowing the gap between parallel and autoregressive generation.

2 Related Work

Diffusion Language Models. Diffusion models have recently been extended from continuous domains to discrete text generation (Ho et al., 2020; Song et al., 2021; Li et al., 2022; Austin et al., 2021). In masked diffusion language models (MDLMs), generation proceeds through iterative denoising of masked tokens using bidirectional context (Sahoo et al., 2024; Nie et al., 2025). Unlike autoregressive models, MDLMs enable parallel token prediction, offering potential latency advantages but introducing coherence challenges during

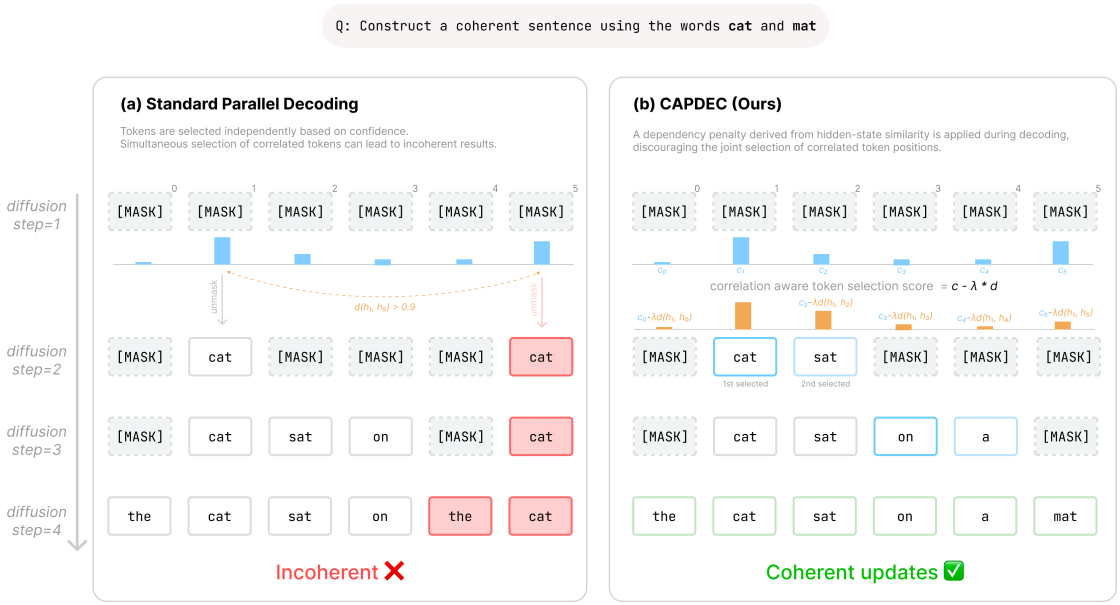

Figure 1: **CAPDEC mitigates incoherence in parallel decoding.** Left (a): Standard parallel decoding selects multiple tokens independently based on confidence, which can result in inconsistent predictions (e.g., selecting "cat" twice). Right (b): CAPDEC introduces a dependency-aware penalty based on hidden state similarity, reducing the chance of jointly selecting semantically correlated positions. This leads to more coherent and fluent generations during diffusion decoding.

parallel decoding.

Recent work demonstrates that diffusion language models can scale to billions of parameters and achieve competitive performance in both natural language and code generation (Nie et al., 2025; Xie et al., 2025). Additional architectural and inference improvements, including block-wise decoding and mixture-of-experts approaches, further improve scalability and efficiency (Bie et al., 2025).

Sampling and decoding in MDLMs. A large body of work focuses on improving inference-time decoding without retraining the model. Existing approaches primarily rely on confidence-based token selection (Chang et al., 2022; Sahoo et al., 2024), uncertainty-aware heuristics, remasking strategies (Wang et al., 2025), or adaptive scheduling methods such as *EB-Sampler* (Ben-Hamu et al., 2025). These methods improve the quality–efficiency trade-off of parallel decoding, but generally treat token decisions independently and do not explicitly model dependencies between simultaneously generated positions.

Parallel generation and token consistency. Recent studies investigate the degradation caused by aggressive parallel decoding and introduce benchmarks for measuring the trade-off between generation quality and parallelism (Kang et al., 2025). Our work builds on this line of research by explicitly modeling cross-token interactions during decoding, enabling more coherent parallel generation without additional training or forward passes.

3 Coherent Parallel Decoding

In this section, we present **CAPDEC**, a decoding method designed to mitigate coherence degradation in parallel generation. We first review the basics of Masked Diffusion Language Models, then motivate the need for modeling cross-token dependencies, and finally formulate token selection as a constrained optimization problem.

3.1 Preliminaries of MDLMs

We follow the standard *masked* (absorbing-state) discrete diffusion formulation (Austin et al., 2021; Lou et al., 2023; Sahoo et al., 2024; Nie et al., 2025). A clean sequence $x_0 = (x_0^{(1)}, \ldots, x_0^{(L)}) \in \mathcal{V}^L$ is progressively corrupted by replacing tokens with a mask symbol $[\mathtt{M}]$, and a denoiser p_θ is trained to predict a categorical distribution over $\mathcal{V}$ at every masked position, conditioned on the partially observed sequence. At inference, generation proceeds over T denoising steps. Let $x_s \in (\mathcal{V} \cup \{[\mathtt{M}]\})^L$ be the partially masked sequence at step s, and let $\mathcal{M}_s = \{i : x_s^{(i)} = [\mathtt{M}]\}$ be the set of currently masked positions. A common decoding pattern is

Table 1: Main evaluation results using **4 tokens per diffusion step** across all models and benchmarks. CAPDEC consistently improves generation quality on all testbeds compared to vanilla confidence-based decoding, while incurring negligible inference overhead. The ΔSpeed column reports the relative change in decoding throughput (TPS) with respect to vanilla decoding. All models are evaluated using *vanilla decoding*, except **LLaDA2.0**, which uses *blockwise decoding* due to its training regime and block-structured attention mask.

Model	ParallelBench	HumanEval		LiveCodeBench		GSM8K	ΔSpeed
	Score	pass@1	SyntaxRate	pass@1	SyntaxRate	Score	%
LLaDA-8B	49	0.17	0.82	0.10	0.75	0.32	
+ CAPDEC	60	0.31	0.88	0.24	0.93	0.61	-1.1%
LLaDA2-Flash	48	0.38	0.58	0.54	0.70	0.23	
+ CAPDEC	61	0.47	0.65	0.63	0.83	0.42	-1.5%
LLaDA-MoE	60	0.23	0.94	0.24	0.92	0.27	
+ CAPDEC	67	0.54	0.98	0.38	0.90	0.52	-1.3%
DreamCoder	48	0.35	0.88	0.20	0.33	0.08	
+ CAPDEC	59	0.59	0.95	0.32	0.66	0.33	-2.0%

confidence-based scheduling (Chang et al., 2022; Sahoo et al., 2024): for each $i \in \mathcal{M}_s$, compute the marginal confidence $c_i = \max_{v \in \mathcal{V}} p_\theta(x_0^{(i)} = v \mid x_s)$, and commit the n_s positions with the highest c_i in parallel. This baseline implicitly treats committed positions as conditionally independent – the assumption CAPDEC relaxes.

3.2 Cross-Token Dependence in Parallel Decoding

Standard MDLM decoding factorizes the joint over tokens committed in the same step into a product of marginals, ignoring their interactions. As a result, individually plausible tokens can be mutually incompatible.

ParallelBench (Kang et al., 2025) formalizes this cost. If the target sequence is partitioned into T disjoint sets $\{S_i\}_{i=1}^T$ decoded in parallel across steps, the minimum achievable KL divergence between any factorized model and the data distribution is lower-bounded by the sum of conditional total correlations:

$$\min_\theta \; D_{\mathrm{KL}}\big(P_{\mathrm{data}}(Y \mid X) \,\|\, P_\theta(Y \mid X)\big) \;\geq \sum_{i=1}^{T} \mathbb{E}_{S_{<i} \sim P_{\mathrm{data}}}\big[C(S_i \mid X, S_{<i})\big], \tag{1}$$

where $C(\cdot)$ denotes the *Conditional Total Correlation* of the tokens in S_i. The bound is irreducible: no amount of additional capacity or training data can eliminate it without reducing parallelism. Reducing the total correlation among tokens committed in the same step is therefore the only lever available at decoding time.

Computing $C(S_i \mid X, S_{<i})$ exactly is intractable, and prior work has approximated it with a separately trained model (Huang et al., 2022). We instead adopt the weaker goal of *moving in the direction the bound prescribes*: discouraging the joint selection of strongly dependent positions, using a tractable proxy computed from a single forward pass.

3.3 Combinatorial Formulation

To mitigate the independence assumption, we reformulate the token selection process as a joint optimization problem. Instead of selecting tokens solely by individual confidence, we seek a subset of tokens that are both individually confident and mutually compatible.

Setup. Fix a diffusion step s and let $x_s \in (\mathcal{V} \cup \{[\mathtt{M}]\})^L$ be the current partially masked sequence. Denote the set of masked (candidate) positions by

$$\mathcal{M}_s \;:=\; \{\, i \in \{1, \ldots, L\} \;:\; x_s^{(i)} = [\mathtt{M}] \,\}.$$

For each $i \in \mathcal{M}_s$, the model produces a predictive distribution $p_\theta(\cdot \mid x_s)$. Let c_i be the per-position confidence score as defined in Section 3.1. We introduce a pairwise dependency (correlation) score $d_{ij} \in [0, 1]$ for $i \neq j$, which quantifies the conflict or mutual dependence between positions i and j. The specific instantiation of d_{ij} is discussed in Section 4.

Let n_s be the number of positions to commit at step s, and let $\lambda \geq 0$ be a trade-off parameter. Introduce binary decision variables $z_i \in \{0, 1\}$ for $i \in \mathcal{M}_s$, where $z_i = 1$ indicates that position i is selected for an update.

Objective. We select exactly n_s positions by solving the fixed-cardinality quadratic binary opti-

mization problem:

$$\max_{z} \quad \sum_{i \in \mathcal{M}_s} c_i\, z_i \;-\; \lambda \sum_{\substack{i,j \in \mathcal{M}_s \\ i<j}} d_{ij}\, z_i z_j \tag{2}$$
$$\text{s.t.} \quad \sum_{i \in \mathcal{M}_s} z_i \;=\; n_s, \qquad z_i \in \{0,1\} \;\; \forall i \in \mathcal{M}_s.$$

This objective mirrors the *Maximal Marginal Relevance* (MMR) criterion widely used in information retrieval to balance relevance with redundancy (Carbonell and Goldstein, 1998).

Greedy Approach. Directly maximizing Eq. 2 is NP-hard (Pardalos and Jha, 1992). However, greedy strategies are known to perform effectively for diversity-promoting objectives in practice (Borodin et al., 2012). We construct the update set incrementally: starting from $\mathcal{U}_s^{(0)} = \emptyset$, at iteration t we add the position whose inclusion yields the largest marginal improvement relative to the positions already selected:

$$\Delta\Big(i \mid \mathcal{U}_s^{(t)}\Big) = c_i \;-\; \lambda \sum_{j \in \mathcal{U}_s^{(t)}} d_{ij}, \qquad i \in \mathcal{M}_s \setminus \mathcal{U}_s^{(t)}. \tag{3}$$

This continues until $|\mathcal{U}_s^{(t)}| = n_s$, requiring only $O(n_s \cdot |\mathcal{M}_s|)$ operations. The full procedure is summarized in Algorithm 1.

Algorithm 1 CAPDEC: Correlation-Aware Parallel Decoding

Input: Masked seq. x_s, indices $\mathcal{M}_s$, count n_s, penalty λ.
Data: Logits P and hidden states H from forward pass.
Output: Updated sequence x_{s+1}.
1. Pre-compute Metrics: $\forall i, j \in \mathcal{M}_s$ compute:
$v_i \leftarrow \arg\max_v P_i(v)$; $c_i \leftarrow P_i(v_i)$; $d_{ij} \leftarrow |\cos(h_i, h_j)|$
2. Greedy Selection: Initialize $\mathcal{U} \leftarrow \emptyset$
for $t = 1$ **to** n_s **do**
  Select candidate maximizing marginal gain (Eq. 3):
  $k^\star \leftarrow \arg\max_{k \in \mathcal{M}_s \setminus \mathcal{U}} \Big(c_k - \lambda \sum_{j \in \mathcal{U}} d_{kj}\Big)$
  $\mathcal{U} \leftarrow \mathcal{U} \cup \{k^\star\}$
end for
3. Update: $x_{s+1} \leftarrow x_s$; $x_{s+1}^{(i)} \leftarrow v_i \;\forall i \in \mathcal{U}$
return x_{s+1}

4 Algorithmic Design

The formulation in Section 3.3 leaves three design choices unspecified: (i) which signal to use as the pairwise dependency score d_{ij} (§4.1), (ii) whether the resulting penalty actually fires on incoherent updates (§4.3), and (iii) how to schedule the penalty within a single decoding step (§4.4). This section addresses each in turn.

4.1 Selecting the Dependency Proxy

A good proxy for d_{ij} must (a) be computable from the single forward pass that CAPDEC already performs and (b) track the true cross-token dependence identified by the ParallelBench bound. We evaluate three candidates against a direct, but expensive, measurement of that dependence.

Ground-truth dependency target. For two masked positions i, j with top-1 predictions t_i, t_j, we define the *symmetric KL dependency* as the average shift in the predictive distribution at one position when the other is committed:

$$d_{\text{true}}(i,j) = \tfrac{1}{2}\big[D_{\mathrm{KL}}\big(p_i \,\|\, p_i^{j \to t_j}\big) + D_{\mathrm{KL}}\big(p_j \,\|\, p_j^{i \to t_i}\big)\big], \tag{4}$$

where $p_i = p_\theta(\cdot \mid x_s)_i$ and $p_i^{j \to t_j}$ denotes the same distribution after unmasking j to t_j. This signal requires two conditioned forward passes per pair and is used only as ground truth, not at runtime.

Candidate proxies and setup. We compare three signals available from the same forward pass that produces the token logits: (i) *self-attention*, the layer- and head-averaged attention weight between positions i and j, symmetrized as $\frac{1}{2}(A_{ij} + A_{ji})$; (ii) *logit cosine similarity*, the absolute cosine similarity between the predictive logit vectors at i and j; and (iii) *hidden-state cosine similarity*, the absolute cosine similarity between the final decoder hidden states at i and j. Absolute values are used since we aim to detect the magnitude of coupling, not its sign. To evaluate each proxy, we feed prompts from each benchmark to the model with a masked generation suffix appended, run a single forward pass, and form candidate pairs from the most confident mask positions – the same positions a confidence-based scheduler would actually consider committing together. For every such pair, we record the three proxy values alongside d_{true} from Eq. 4, and report Spearman (rank) and Pearson (linear) cor-

Table 2: Average correlation with the symmetric KL dependency target (Eq. 4).

Model	Spearman $\rho \uparrow$			Pearson $r \uparrow$		
	Attn.	Logit	Hidden	Attn.	Logit	Hidden
DreamCoder	0.379	0.467	**0.664**	0.255	0.287	**0.543**
LLaDA-MoE	0.377	0.246	**0.597**	0.413	0.151	**0.579**

Table 3: ParallelBench score when each proxy is plugged into Algorithm 1.

Dependency Proxy d_{ij}	Avg. Score
Self-attention	51
Logit cosine similarity	54
Hidden-state cosine similarity	**66**

relations between each proxy and d_{true} aggregated across pairs. Evaluation spans two masked diffusion models (Dream-Coder-7B, LLaDA-MoE-7B-A1B) and four generation domains covering math reasoning (GSM8K, MATH500), code (HumanEval), and open-ended dialogue (MT-Bench).

Results. Table 2 shows that hidden-state cosine similarity achieves the strongest correlation with the symmetric KL dependency target for both model families and both correlation metrics. The attention and logit proxies are less stable across models: logit similarity is competitive for Dream-Coder under Spearman correlation, but degrades on LLaDA-MoE, while hidden-state similarity remains consistently strongest.

We therefore instantiate the dependency score in CAPDEC as

$$d_{ij} = \left| \frac{\mathbf{h}_i \cdot \mathbf{h}_j}{\|\mathbf{h}_i\| \, \|\mathbf{h}_j\|} \right|, \tag{5}$$

the absolute cosine similarity of the final decoder hidden states, and use this definition throughout the remainder of the paper.

4.2 Downstream Validation

The probe in §4.1 establishes that hidden-state similarity is the strongest *predictor* of true dependence. We next verify that this advantage transfers to actual decoding quality: integrating each candidate proxy into the greedy selection of Algorithm 1 and evaluating on ParallelBench (Dream-Coder-Instruct-7B, k=4). Table 3 shows that hidden-state similarity is also the strongest signal end-to-end, lifting the average score from 51 (self-attention) and 54 (logit similarity) to 66. The two layers of evidence agree: the proxy that best tracks the symmetric KL dependency is also the proxy that yields the most coherent generations.

4.3 Trace-Level Validation

We additionally verify that the chosen proxy fires *on the right positions* during decoding, by inspecting traces on the *Shuffle* task of ParallelBench, which is explicitly designed to expose conditional-independence failures. For each generation from Dream-Coder-Instruct-7B with $n_s = 4$, we approximate the intended target as the valid permutation of input items with the lowest edit distance to the output, and classify each generated token as *correct* or *erroneous* relative to this target. We then compute, at each diffusion step, the mean CAPDEC penalty $\sum_{j \in \mathcal{U}} d_{ij}$ separately over the two classes of tokens.

As shown in Figure 2, erroneous tokens receive systematically higher penalties than correct tokens throughout decoding. The proxy therefore not only correlates with true dependence in aggregate (§4.1) and improves downstream quality (§4.2), but also acts on the specific positions responsible for incoherent updates.

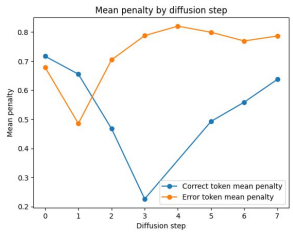

Figure 2: **Penalty fires on incoherent tokens.** Mean CAPDEC penalty for correct vs. erroneous tokens across diffusion steps on Shuffle.

4.4 The ξ Schedule.

While the formulation in Section 3.3 treats λ (equals 1 by default) as a fixed trade-off parameter, we find that decaying the penalty weight within a single decoding step further stabilizes selection under high parallelism. Concretely, at greedy iteration t of Algorithm 1 (line 9), we scale the penalty weight by $\xi^{\max(t-2,\,0)}$, so the marginal-gain criterion in Eq. 3 becomes

$$\Delta\Big(i \mid \mathcal{U}_s^{(t)}\Big) = c_i - \lambda \cdot \xi^{\max(t-2,\,0)} \sum_{j \in \mathcal{U}_s^{(t)}} d_{ij}, \tag{6}$$

where t indexes the current selection within the step (starting from 1), c_i is the token confidence,

and d_{ij} is the hidden-state cosine similarity. The motivation is to prevent the accumulated pairwise penalty from dominating the confidence term as the selected set $\mathcal{U}$ grows. For $k = 4$ (Tables 1 and 8) we use the constant schedule $\xi = 1$; the decay is applied only under aggressive parallelism ($k \geq 8$) and confidence-threshold decoding (Tables 6 and 7).

Sensitivity Analysis. Table 4 reports ParallelBench scores across $\xi \in \{0.3, 0.5, 0.7, 0.9\}$ for LLaDA-8B and DreamCoder at $k = 4$ and $k = 8$. CAPDEC consistently outperforms naive confidence-based decoding for every tested value, indicating robustness to this hyperparameter, with $\xi = 0.7$ achieving the best balance between confidence and dependency penalization across models and parallelism levels.

Table 4: Sensitivity of CAPDEC to the decay multiplier ξ on ParallelBench. The *naive* row is vanilla confidence-based decoding.

ξ	**LLaDA-8B**		**DreamCoder**	
	k=4	k=8	k=4	k=8
naive	49	33	48	40
0.3	55	35	52	40
0.5	57	37	55	41
0.7	**59**	**41**	**57**	**43**
0.9	60	37	58	40

5 Experiments

In this section, we empirically validate the effectiveness of **CAPDEC** in mitigating coherence degradation during parallel generation. We conduct a comprehensive evaluation across code generation, mathematical reasoning, and constrained text manipulation tasks. We also report the changes in inference speed to quantify the latency impact of our method.

5.1 Experimental Setup

Models. We evaluate our method on a diverse set of Masked Diffusion Language Models (MDLMs) to demonstrate model-agnostic applicability. Specifically, we utilize: LLaDA-8B (Nie et al., 2025), LLaDA-MoE (Bie et al., 2025), LLaDA2-Flash, and DreamCoder-Instruct-7B (Xie et al., 2025). All experiments for 7B/8B models were conducted on a single NVIDIA A100-SXM4 (80GB) GPU. Experiments for LLaDA2-Flash were distributed across 4 NVIDIA A100 GPUs due to memory constraints.

Benchmarks. We evaluate CAPDEC on benchmarks targeting coherence-sensitive parallel generation, code synthesis, and reasoning.

ParallelBench (Kang et al., 2025) evaluates the trade-off between parallelism and generation quality in diffusion language models across 17 tasks spanning list manipulation, text generation, and global constraint satisfaction. We report per-task scores and macro-average results on the standard 0–100 scale.

HumanEval (Chen et al., 2021) contains 164 Python programming problems. We report **pass@1** and **Syntax Correct Rate**, which captures structural consistency under parallel decoding.

LiveCodeBench (Jain et al., 2024) is a contamination-free competitive programming benchmark collected after most model training cutoffs. We evaluate the *Easy* subset using **pass@1** and Syntax Correct Rate, since current diffusion models rarely produce executable solutions on harder subsets.

GSM8K (Cobbe et al., 2021) evaluates multi-step mathematical reasoning on grade-school math word problems.

5.2 Baselines

We compare **CAPDEC** against standard MDLM decoding strategies and recent inference-time samplers. The primary baseline is confidence-based decoding (Chang et al., 2022; Sahoo et al., 2024), which commits the top-k positions by marginal confidence ($k = 4$ by default), implicitly assuming independence among parallel updates. We additionally evaluate confidence-threshold decoding, which demasks all positions above a threshold τ, and blockwise decoding (Bie et al., 2025), applying CAPDEC within fixed-size blocks to test long-context compatibility. We also compare with ReMDM (Wang et al., 2025), using the recommended max-capped schedule with $\eta_{\text{cap}} = 0.04$, and EB-Sampler (Ben-Hamu et al., 2025), reporting two operating points, $\gamma = 0.05$ and $\gamma = 0.7$, to cover its quality–speed frontier.

5.3 Results

Table 1 reports the main evaluation results on ParallelBench, HumanEval, LiveCodeBench, and GSM8K. All experiments are conducted using a fixed parallelism of 4 tokens per diffusion step across all models.

Across all evaluated masked diffusion language models, CAPDEC consistently outperforms vanilla

Table 5: Comparison of CAPDEC with prior dependency- and efficiency-aware samplers on ParallelBench (PB) and HumanEval (HE), namely ReMDM (Wang et al., 2025) and EB-Sampler (Ben-Hamu et al., 2025). All methods use the same pretrained model; CAPDEC and the confidence-based baseline use a fixed budget of $k = 4$ tokens per step, while EB-Sampler adaptively determines how many tokens to unmask at each step. TPS denotes decoding throughput in tokens-per-second, reported separately for each benchmark since sequence lengths differ. Dashes indicate configurations outside the operating range of the method (EB at γ=0.05 collapses to near-sequential decoding on PB; EB at γ=0.7 produces too few syntactically valid programs on HE to report meaningful pass@1).

Model	**Decoding**	**ParallelBench**		**HumanEval**		
		Score	TPS	pass@1	SyntaxRate	TPS
LLaDA-8B	conf-based (baseline)	49	130	0.17	0.82	61
	ReMDM	49	118	0.18	0.87	59
	EB-Sampler (γ=0.05)	—	—	0.12	0.99	45
	EB-Sampler (γ=0.7)	63	128	—	—	—
	CAPDEC (ours)	60	125	0.31	0.88	60
DreamCoder-7B	conf-based (baseline)	48	129	0.35	0.88	55
	ReMDM	55	125	0.37	0.88	54
	EB-Sampler (γ=0.05)	—	—	0.39	0.96	83
	EB-Sampler (γ=0.7)	55	106	—	—	—
	CAPDEC (ours)	59	124	0.59	0.95	55

confidence-based decoding. The most pronounced improvements are observed on ParallelBench, where CAPDEC improves the average score by 7–13 points, indicating substantially improved robustness to coherence degradation under parallel decoding.

On code generation benchmarks, CAPDEC yields significant gains in both functional correctness and syntactic validity. In particular, CAPDEC improves `pass@1` on LiveCodeBench by up to $2.5\times$ relative to vanilla decoding, while also substantially increasing the syntax-correct rate. Similar trends are observed on HumanEval, where CAPDEC consistently improves both metrics across all models.

CAPDEC also improves performance on GSM8K for all evaluated models, suggesting that dependency-aware token selection benefits not only syntactic coherence but also multi-step reasoning. Importantly, these quality improvements are achieved with negligible inference overhead, with decoding throughput changes within 2% relative to vanilla decoding.

Robustness to Increased Parallelism. Table 6 analyzes the impact of increasing the number of tokens generated per diffusion step. While vanilla decoding exhibits a sharp degradation in quality as parallelism increases, CAPDEC maintains substantially higher performance across all settings.

In particular, when increasing parallelism from $k = 2$ to $k = 8$, CAPDEC consistently reduces the quality drop on ParallelBench for both LLaDA-8B and DreamCoder. At $k = 4$ and $k = 8$, CAPDEC improves average scores by up to $10 - 11$ points compared to vanilla decoding, demonstrating improved stability under aggressive multi-token updates. These results indicate that explicitly accounting for cross-token dependencies significantly mitigates the failure modes induced by the conditional independence assumption in parallel decoding.

A complementary view on robustness is provided by confidence-threshold decoding, where at each diffusion step only tokens whose confidence exceeds a predefined threshold are demasked, resulting in a variable number of tokens updated in parallel. Table 7 reports results under this adaptive parallelism regime.

While vanilla decoding remains unstable across thresholds, CAPDEC consistently achieves higher ParallelBench scores for all confidence levels. These results indicate that dependency-aware token selection is particularly effective when the degree of parallelism varies dynamically during inference.

Comparison with recent samplers. Table 5 compares CAPDEC with ReMDM and EB-Sampler. This comparison should be interpreted with an important distinction in mind: EB-Sampler adaptively changes both which tokens and how many tokens are unmasked at each diffusion step, whereas CAPDEC keeps the parallelism budget fixed ($k = 4$) and only modifies the position-selection rule. Thus, EB-Sampler can reduce the number of committed tokens in high-uncertainty states and increase it when more predictions are reliable, while CAPDEC tests whether dependency-aware selection improves coherence under the same fixed to-

Table 6: ParallelBench average score and decoding throughput (TPS) versus the number of tokens generated per step (k). CAPDEC maintains higher quality under increased parallelism. Results use the decreasing ξ schedule ($\xi = 0.7$).

Model	$k = 1$		$k = 2$		$k = 4$		$k = 8$	
	Score	TPS	Score	TPS	Score	TPS	Score	TPS
LLaDA-8B	70	33	59	63	49	130	33	248
+ CAPDEC	70	33	68	64	59	124	41	240
DreamCoder	74	33	61	64	48	129	40	254
+ CAPDEC	74	33	67	64	57	123	43	242

Table 7: ParallelBench results at different demasking confidence thresholds on **LLaDA2.0-Flash**. CAPDEC reduces quality degradation. LLaDA2.0 is used since it natively supports confidence-threshold decoding. All runs use a decreasing ξ schedule ($\xi = 0.7$).

Model	Confidence Threshold			
	1.0	0.9	0.75	Δ Speed
LLaDA2-Flash	54	55	54	
+ CAPDEC	64	63	60	-2.1%

ken budget. Under this stricter setting, CAPDEC remains competitive on ParallelBench and gives consistent gains in HumanEval pass@1, suggesting that dependency-aware selection is complementary to adaptive token-count scheduling.

Table 8 provides a fine-grained analysis on representative ParallelBench tasks. CAPDEC yields large absolute gains on *Shuffle*, a task explicitly designed to expose conditional-independence failures, improving scores by $+35$–56 points across all models. Substantial improvements are also observed on Words-to-Sentence (W2S), indicating improved compositional and grammatical consistency under parallel decoding. On Puzzles, which require global logical consistency, CAPDEC improves or preserves performance across models, suggesting that dependency-aware selection does not trade off logical constraints for syntactic gains.

Table 9 evaluates CAPDEC under both vanilla and blockwise decoding. CAPDEC consistently improves performance in both regimes, with gains persisting under blockwise generation where parallel decoding is applied within restricted context. Notably, the improvements are strongest for LLaDA2.0-Flash, indicating that dependency-aware decoding remains effective and compatible with block-structured generation.

Overall, these results confirm that CAPDEC improves robustness to parallel decoding independently of considered underlying generation setups.

Table 8: ParallelBench results by task category. CAPDEC improves performance across syntactic, compositional and logical reasoning tasks. Scores for Words-to-Sentence (W2S) and Puzzles represent the mean accuracy aggregated over multiple subtasks.

Model	Shuffle	W2S	Puzzles
LLaDA-8B	24	43	15
+ CAPDEC	82	58	32
DreamCoder	5	17	0
+ CAPDEC	57	51	0
LLaDA2-Flash	40	27	11
+ CAPDEC	76	61	27
LLaDA-MoE	25	37	62
+ CAPDEC	78	66	66

Table 9: Evaluation under vanilla and blockwise generation. CAPDEC improves both regimes.

Model	Setup	PB	LCB	
		Avg. Score	pass@1	Syntax
DreamCoder	Vanilla	48	0.20	0.33
+ CAPDEC	Vanilla	59	0.32	0.66
DreamCoder	Blockwise	45	0.18	0.46
+ CAPDEC	Blockwise	53	0.24	0.68

6 Conclusion

We identify conditional independence in parallel decoding as a key source of coherence degradation in masked diffusion language models. To address this issue, we introduce **CAPDEC**, a training-free and model-agnostic decoding strategy that accounts for cross-token dependencies during inference without additional forward passes.

By combining token quality and dependency-aware selection, CAPDEC consistently improves coherence and downstream performance across language and code generation benchmarks while preserving the efficiency advantages of parallel diffusion decoding. Compared with recent dependency- and efficiency-aware samplers, CAPDEC achieves a stronger quality–throughput trade-off by directly targeting cross-token interactions.

Future work may extend dependency-aware decoding beyond position selection to reasoning over full token assignments and richer dependency structures.

Limitations

CAPDEC is a decoding-time method and therefore depends on the quality of the underlying masked diffusion language model. It does not correct errors that are already strongly preferred by the model dis-

tribution, and its effectiveness may vary with the choice of the dependency weight λ, the decoding schedule, and the model architecture. Our dependency metric uses hidden-state cosine similarity as a tractable proxy rather than directly estimating conditional total correlation, so it should be interpreted as a practical signal for reducing harmful parallel updates rather than a faithful measurement of true token dependence. While the greedy selection step has complexity $O(n_s \cdot |\mathcal{M}_s|)$, in the long-context regime CAPDEC operates within block-wise decoding (Table 9), so $|\mathcal{M}_s|$ is bounded by the block size rather than the full sequence length and the overhead remains independent of total context length. Finally, our code-generation evaluation on LiveCodeBench focuses on the Easy subset because current diffusion language models rarely produce executable solutions on harder subsets, leaving broader evaluation on more capable future diffusion models for future work.

References

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.

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

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*.

Allan Borodin, Hsiu-Chin Lee, and Yuli Ye. 2012. Max-sum diversification, monotone submodular function maximization and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 155–166. ACM.

Jaime Carbonell and Jade Goldstein. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. 2022. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11315–11325.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, H. P. de O. Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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*.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations (ICLR)*.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*.

Fei Huang, Tianhua Tao, Hao Zhou, Lei Li, and Minlie Huang. 2022. On the learning of non-autoregressive transformers. In *International conference on machine learning*, pages 9356–9376. PMLR.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjeng Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination-free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, and 1 others. 2025. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B Hashimoto. 2022. Diffusion-lm: Improves controllability and diversity in text generation. In *NeurIPS*.

Aaron Lou, Chenlin Meng, and Stefano Ermon. 2023. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*.

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*.

Panos M. Pardalos and Somesh Jha. 1992. Complexity of uniqueness and local search in quadratic 0–1 programming. *Operations Research Letters*, 11(2):119–123.

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.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*.

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

Zhihui Xie, Jiacheng Ye, Lin Zheng, Jiahui Gao, Jingwei Dong, Zirui Wu, Xueliang Zhao, Shansan Gong, Xin Jiang, Zhenguo Li, and 1 others. 2025. Dreamcoder 7b: An open diffusion language model for code. *arXiv preprint arXiv:2509.01142*.