Adaptive Multi-Step Lookahead Decoding for Diffusion Language Models

Anonymous ACL submission

Abstract

Masked diffusion language models (DLMs) enable parallel text generation by iteratively refining masked tokens, offering a promising alternative to autoregressive decoding. Recent lookahead-based decoding methods improve the accuracy–efficiency trade-off by exploring future decoding states before committing token updates. However, existing approaches mainly rely on shallow one-step lookahead, which optimizes immediate information gain but can be suboptimal for longer-horizon decoding trajectories. Meanwhile, we find that a naive extension for deeper lookahead is also ineffective, as fixed-depth rollout introduces additional computation and cannot adapt to heterogeneous intermediate decoding states. Thus, in this work, we propose AdaLook, an adaptive lookahead framework for DLM decoding. AdaLook dynamically determines whether to continue rollout based on candidate-score variance and further enables branch expansion when intermediate rollout states require additional exploration. This design avoids unnecessary deep rollout while allowing the decoder to re-trigger lookahead from informative intermediate states. Experiments on various benchmarks and models demonstrate that AdaLook achieves a better accuracy–decoding steps trade-off than existing one-step lookahead decoding methods.

1 Introduction

Masked diffusion language models (DLMs) have recently emerged as an alternative to traditional autoregressive (AR) models for text generation. Unlike AR models, which generate tokens sequentially in a token-by-token manner, DLMs operate by iteratively refining a sequence of masked tokens, enabling parallel decoding across multiple positions (Sahoo et al., 2024; Nie et al., 2026; Ye et al., 2025). This property offers a fundamental advantage over AR decoding: By relaxing the strict left-to-right dependency, DLMs can potentially improve generation efficiency and enable more flexible inference strategies.

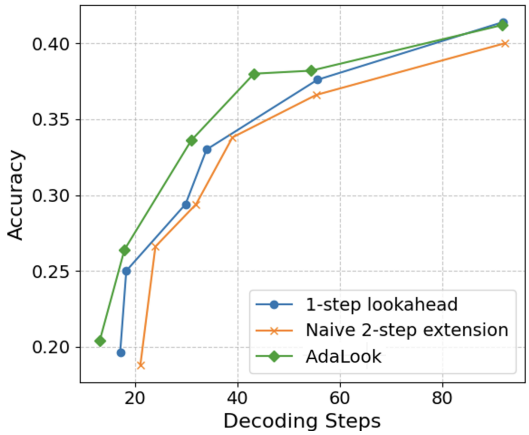

Figure 1: Accuracy vs Decoding Steps (evaluated on MATH500 with LLaDA-8B-Instruct)

While the diffusion framework supports parallel token updates in principle, its denoising-based training objective does not prescribe a specific decoding order or token selection strategy, leaving a large design space for inference-time decoding that can substantially affect generation quality and efficiency. To better exploit the parallel nature of DLMs, a growing line of work focuses on designing efficient decoding strategies. Existing methods have evolved from fixed-step parallel decoding (Nie et al., 2026) to more adaptive confidence-aware decoding frameworks (Yu et al., 2025; Wu et al., 2025; Wei et al., 2025).

While most methods rely on greedy token selection based on current confidence estimates, Fu et al. (2025) propose a lookahead-based strategic exploration mechanism that evaluates candidate token updates by their downstream decoding benefit. Specifically, this method performs a lookahead search over future decoding states when the model exhibits low confidence over the remaining masked positions. It aims to identify informative tokens whose commitment leads to a chain of subsequent high-confidence predictions, thereby

enabling more tokens to be decoded per step and reducing the total number of decoding steps required.

Since the method is designed from an immediate information-gain perspective, it primarily evaluates whether a tentative token commitment can make more masked positions confident in the next decoding state. This naturally leads to a one-step lookahead design. However, from a longer-horizon perspective, such local exploration can be suboptimal: the token that maximizes immediate confidence gain may not lead to the best future trajectory, causing the search to be trapped in a locally favorable but globally suboptimal commitment.

To bridge this gap, we focus on developing a multi-step lookahead framework that evaluates candidate token updates over longer decoding horizons rather than based on immediate one-step evidence alone. A straightforward approach is to perform deeper rollouts over future decoding trajectories. However, this extension introduces additional technical challenges. According to our initial experiments shown in Figure 1, naively increasing lookahead depth does not improve the trade-off between generation quality and decoding steps.

This limitation mainly arises from two aspects. **(L1)** Deeper lookahead introduces additional computational overhead, while the optimal rollout depth can vary significantly across different examples within the same dataset and decoding stages within the same example. As a result, a uniformly fixed rollout depth may introduce additional computation without yielding proportional performance gains, ultimately degrading the overall quality-efficiency trade-off. **(L2)** Naive multi-step rollout follows a fixed forward trajectory for each hypothesis, with no mechanism to assess whether intermediate rollout states require further branching. This prevents the decoder from dynamically branching into alternative decoding paths when intermediate rollout states remain uncertain.

To address the limitations, we propose **AdaLook** (**Ada**ptive **Look**ahead Decoding), a multi-step lookahead framework that dynamically determines the rollout depth based on the decoding state while enabling branch expansion during lookahead. To address **(L1)**, before each additional rollout step, AdaLook determines whether further rollout is necessary based on the variance of candidate scores computed from frontier confidence. Only when the variance exceeds a predefined threshold does the decoder continue expanding future decoding trajectories. To address **(L2)**, after each rollout step, each branch is re-evaluated to determine whether additional expansion is still required. The decoder then jointly considers the expansion status of all candidate branches: branches that no longer require further expansion are prioritized. If all branches require further expansion, the highest-scoring branch is selected and its decoding state is passed back as the starting point for the next round of lookahead exploration. This adaptive process avoids unnecessary deep rollout while enabling flexible re-triggering of lookahead at intermediate stages, allowing the decoder to discover and commit to more informative decoding trajectories.

As shown in Figure 1, AdaLook achieves a better accuracy–decoding steps trade-off than the existing 1-step lookahead decoding method. Extensive experiments further demonstrate that this advantage consistently generalizes across different datasets and model backbones.

2 Related Works

Diffusion Language Models. Diffusion models have achieved remarkable success in continuous domains such as image and audio generation (Ho et al., 2020; Song et al., 2020; Kong et al., 2020), which has motivated efforts to extend the diffusion paradigm to discrete text. Early work on discrete diffusion, notably D3PM (Austin et al., 2021), established a general framework for defining noising processes over categorical variables, including absorbing-state masking corruption. Subsequent works leverage this masking-based formulation to develop masked diffusion language models (MDLMs) that generate text via iterative unmasking (Sahoo et al., 2024; Shi et al., 2024; Ou et al., 2025). Building on these foundations, recent large-scale MDLMs have reached performance competitive with autoregressive counterparts: LLaDA (Nie et al., 2026) trains an 8B model from scratch with bidirectional attention, Dream (Ye et al., 2025) develops a diffusion-based LLM initializing from a pretrained autoregressive model, and LLaDA 2.0 (Bie et al., 2025) further scales to 100B parameters with a Mixture-of-Experts architecture.

Decoding Strategies Methods for MDLMs. To fully leverage the parallel nature of MDLMs, a growing line of work focuses on designing efficient decoding strategies. Initial attempts focus on fixed-step parallel decoding (Nie et al., 2026; Ye et al., 2025), while subsequent works introduce more adaptive confidence-aware decoding strategies, including fixed-threshold and dynamic thresh-

old decoding (Yu et al., 2025; Wu et al., 2025). Other studies further improve decoding efficiency through multi-stage decoding strategies (Wei et al., 2025), confidence calibration (Huang et al., 2025), entropy-bounded unmasking (Ben-Hamu et al., 2026), and KV Caching techniques (Wu et al., 2025; Liu et al., 2025). More recently, lookahead-based decoding methods introduce inference-time search by exploring future decoding trajectories before committing updates (Lee et al., 2025; Fu et al., 2025). However, as discussed in Section 1, existing methods mainly rely on shallow one-step lookahead, which limits their ability to perform long-horizon planning during decoding. Notably, our work mainly builds on the efficiency-oriented lookahead framework of Fu et al. (2025), which targets a better accuracy–decoding steps trade-off. In contrast, Lee et al. (2025) uses lookahead primarily to enhance generation performance, with less emphasis on the decoding efficiency.

3 Preliminary

Our multi-step lookahead framework is built upon the Explore-then-Exploit (ETE) decoding strategy introduced by Fu et al. (2025). In this section, we introduce the two components of ETE: *Fast Block Diffusion Sampling* and *Confidence-based Lookahead Mechanism*.

3.1 Fast Block Diffusion Sampling

Block diffusion decoding generates the sequence progressively in a block-by-block manner, where a sequence of length n is partitioned into L decoding blocks with block size n_b. Within each block, the decoder performs iterative refinement before proceeding to the next block (Nie et al., 2026).

Fast Block Diffusion Sampling improves decoding efficiency by assigning a fixed budget of N decoding steps to each block and moving forward to the next block once the budget is exhausted, rather than waiting for complete convergence within the current block. Since earlier blocks may still contain masked tokens when the current block begins decoding, the decoder commits high-confidence tokens across all unlocked blocks simultaneously at each step, thereby increasing the number of tokens unmasked per decoding step compared to decoding within a single block alone.

3.2 Confidence-Based Lookahead Mechanism

The core idea of the confidence-based lookahead mechanism is to selectively perform lookahead exploration when the model's confidence over remaining masked positions is low. It uses a 1-step forward rollout to identify which token updates are most likely to unlock subsequent high-confidence predictions. The confidence here refers to the model's predicted probability for its most likely token at each masked position. The exploration procedure can be divided into three components: exploration triggering, candidate construction, and hypothesis selection.

1. Exploration triggering. The decoder activates lookahead exploration when the average confidence over the current decoding frontier $\mathcal{F}$ falls below a threshold γ, while the number of remaining masked positions exceeds a minimum N_e. The frontier $\mathcal{F}$ is defined as the set of masked positions up to the midpoint of the current block.

2. Candidate construction. According to the candidate selection strategy of Fu et al. (2025), once exploration is triggered, the method identifies a set of informative candidate positions from the masked positions:

$$\mathcal{H} = \mathrm{Topk}_{i \in \mathcal{M}_t} \left(- \left| c_t^i(\mathbf{x}_t) - c^{\mathrm{info}} \right| + \beta \cdot (i - (b_t - 1)n_b) \right),$$

where $\mathbf{x}_t$ denotes the partially decoded sequence at decoding step t, with masked positions $\mathcal{M}_t$. $c_t^i(\mathbf{x}_t)$ denotes the confidence at position i, b_t is the index of the current decoding block at step t, n_b is the block size, and β is a hyper-parameter balancing the two terms. The first term selects positions whose confidence is close to the target exploration level c^{info}, which captures tokens that are uncertain but potentially informative. Following the empirical findings of Fu et al. (2025), we set $c^{\mathrm{info}} = 0.2$, as tokens around this confidence level are shown to be more informative for triggering more downstream high-confidence predictions. The second term uses $i - (b_t - 1)n_b$, the relative position of token i within the current block, to assign slightly larger scores to later positions when $\beta > 0$, thereby encouraging the decoding frontier to move forward.

3. Hypothesis selection. For each candidate position $j \in \mathcal{H}$, the decoder commits position j and performs a one-step forward pass to obtain the resulting state $x_{t+1;j}$. Each hypothesis is scored as:

$$s(j) = \alpha \cdot \log c_t^j(\mathbf{x}_t) + \log \sum_{i \in \hat{S}_{t+1}(j)} c_{t+1}^i(\mathbf{x}_{t+1;j}))\mathbf{1}\left(c_t^i(\mathbf{x}_{t+1;j}) \geq C\right).$$

$\hat{S}_{t+1}(j)$ denotes the predicted token positions after rollout, C is the confidence threshold, and α is a

regularization parameter balancing the two terms. Intuitively, the scoring function jointly considers both the confidence of the explored token itself and the amount of downstream high-confidence predictions induced after rollout. The candidate with the highest score is selected as the committed decoding trajectory for subsequent inference.

4 Adaptive Multi-Step Lookahead

As in Section 1, one-step lookahead remains suboptimal due to its limited rollout horizon, while naively extending rollout depth fails to improve the quality-efficiency trade-off. To more flexibly determine rollout depth and better handle intermediate rollout states that require further exploration, we propose AdaLook, an adaptive multi-step lookahead framework in this section.

4.1 General Framework

The general workflow of the method is shown in Figure 2. It consists of two key components: *Adaptive Rollout Continuation* and *Dynamic Branch Expansion*, which respectively address the **(L1)** and **(L2)** discussed in Section 1. For simplicity, the figure illustrates a special case with the maximum lookahead depth set to 2. The detailed algorithm is shown in Algorithm 1 in Appendix A.

4.2 Adaptive Rollout Continuation

To extend one-step lookahead to a multi-step setting, we maintain a set of k candidate decoding trajectories and evaluate each trajectory after every rollout step. For the j-th candidate trajectory, we compute a cumulative rollout score:

$$\tilde{s}^{(R)}(j) = \alpha \log c_t^j(\mathbf{x}_t) + \log \left(\sum_{r=1}^{R} \sum_{i \in \mathcal{T}_j^r} c_{t+r}^i(\mathbf{x}_{t+r;j}) \right) \tag{1}$$

where R denotes the current rollout depth, $\mathcal{T}_j^r$ is the set of positions that become high-confidence and are selected for commitment at the r-th rollout step under candidate j. The score is a direct extension of the original ETE score: when multi-step lookahead is used, we accumulate the downstream confidence gains across all rollout steps.

Before each additional rollout step, we compute the variance of the current candidate trajectory scores, denoted as $\mathrm{Var}(\{\tilde{s}^{(R)}(j)\}_{j=1}^k)$. If the variance falls below a threshold τ, indicating that the candidates remain not sufficiently discriminative, the decoder proceeds with another rollout step. Once the variance exceeds τ, or the rollout depth reaches the maximum $T_{\max}$, the decoder commits $j^* = \mathrm{argmax}_j(\tilde{s}^{(R)}(j))$ and returns the selected committed state to the main decoding loop for subsequent decoding.

4.3 Dynamic Branch Expansion

During multi-step rollout, individual branches may evolve into qualitatively different decoding states: some may reach stable high-confidence regions where continuing the current rollout is sufficient, while others may encounter new low-confidence regions that require re-triggering lookahead exploration. To handle this heterogeneity, after each extra forward step, every rollout branch independently determines whether further lookahead exploration should be triggered again according to the original ETE triggering condition. Based on the expansion status of all candidate branches, the decoder handles three cases:

Case 1: No branch requires further expansion. All hypotheses have reached stable decoding states. In this case, the decoder continues the adaptive rollout continuation procedure until the candidate trajectory scores become sufficiently distinguishable or the rollout depth reaches the maximum rollout step $T_{\max}$.

Case 2: All branches require further expansion. When all trajectories remain uncertain, the decoder first selects the highest-scoring candidate according to the updated rollout score:

$$j^* = \arg\max_j \left(\tilde{s}(j) + \log \bar{c}_{\mathcal{F}}^j \right),$$

where $\tilde{s}(j)$ accumulates the confidence of tokens already committed along hypothesis j before the latest rollout forward pass. $\bar{c}_{\mathcal{F}}^j$ denotes the average confidence over the decoding frontier after the latest rollout forward pass. Intuitively, the additional frontier-confidence term encourages selecting hypotheses whose latest rollout state is more promising for future decoding.

The current lookahead round then ends and the selected state $x_{t+1+r;j^*}$ is returned to the main decoding loop as the initialization for the next round of lookahead exploration. This allows the decoder to dynamically re-trigger lookahead from intermediate rollout states, enabling flexible branching rather than being restricted to a fixed linear rollout trajectory. Furthermore, since evaluating the trigger condition already requires a forward pass to obtain c_j for all hypotheses, the confidence predictions obtained for the selected state $\mathbf{x}_{t+1+r;j^*}$

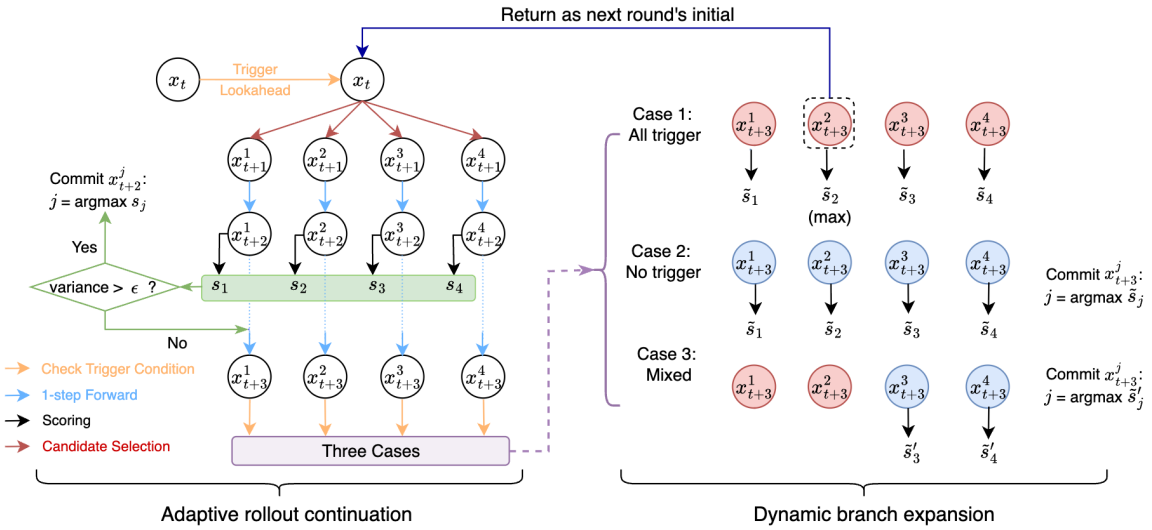

Figure 2: Overview of the adaptive multi-step lookahead mechanism (when the maximum lookahead depth is 2).

are cached and reused as the initial forward pass of the next lookahead round, avoiding one redundant model forward.

Case 3: Mixed expansion status. When only part of the candidate trajectories require further expansion, the decoder prunes branches that still trigger lookahead and retains only stable branches for subsequent rollout iterations. The adaptive rollout continuation loop resumes over the remaining stable candidates. This process continues until a single branch remains, the score variance exceeds τ, or the rollout budget T_{max} is exhausted, at which point the best remaining candidate is committed. This design prioritizes trajectories that have already transitioned into stable decoding regimes, while avoiding excessive exploration on repeatedly uncertain branches that may lead to diminishing returns in decoding efficiency.

Hyperparameter Selection Our algorithm introduces several hyperparameters that control candidate selection, rollout triggering, and decoding progression. Since exhaustively tuning all hyperparameters on each dataset would be computationally expensive, we discuss our hyperparameter selection strategy in this subsection.

Specifically, we group the hyperparameters into three main categories:

1. Fixed hyperparameters. We fix $k = 4$, $\alpha = 0.1$, $\beta = 0.01$, , $T_{\text{max}} = 2$ and $c_{\text{info}} = 0.2$ across all experiments. These values are either adopted from prior work (Fu et al., 2025) or chosen based on calibration results. We observe no significant dataset-specific variation for these hyperparameters.

2. Calibration-guided hyperparameters. For hyperparameters that are not fixed, we use a small calibration set to evaluate different combinations and identify general selection rules. Since parameters such as C and N induce different accuracy-efficiency trade-offs, we analyze the configurations on the empirical Pareto frontier to identify the best parameter patterns.

In particular, we observe a clear interaction between the confidence threshold C and the per-block unlocking budget N. Both parameters affect the decoding pace: smaller N unlocks subsequent blocks earlier, while smaller C commits more tokens per step. Both choices can reduce the number of decoding steps, but may also make the decoding process more aggressive. Among Pareto-frontier configurations, the preferred threshold tends to increase slowly with the decoding budget. This empirical trend can be roughly summarized by the following range-valued guideline:

$$C^*(N) \in [0.5 + 0.05 \log_2 N,\ 0.5 + 0.10 \log_2 N],$$

where the selected value is clipped to the calibration range $[0.5, 0.9]$.

To explain this relationship, when N is small, the decoding frontier advances more quickly because subsequent blocks are unlocked after fewer decoding rounds. In this case, a lower confidence threshold is often preferred so that the decoder can commit enough reliable tokens before the frontier moves forward, providing useful context for later blocks. In contrast, when N is large, the decoder spends more refinement rounds on the current block before unlocking later blocks. We provide justification about the log-scale relationship in Appendix B.

We also find that N_e, the minimum number of

remaining frontier tokens required to trigger lookahead, has little impact on performance within a reasonable range. To reduce the number of tunable hyperparameters, we set $N_e = N$ by default.

Notably, these trends are generally consistent across different datasets, suggesting that the calibration rules are not strongly dataset-specific.

3. Empirical hyperparameters without stable trends. The main hyperparameter that does not show a stable pattern on the calibration set is γ, which controls when lookahead is triggered. We find that γ generally performs reasonably within the range $[0.1, 0.4]$, but no single value consistently dominates across datasets or decoding budgets. We therefore consider both a universal setting and an optimized setting in our experiments; the details of these settings are discussed in Section 5.

5 Empirical Evaluation

5.1 Experimental Setups

Models, Benchmarks and Baselines We conduct experiments mainly using **LLaDA-8B-Instruct** (Nie et al., 2026). We evaluate AdaLook on four widely used benchmarks, including **MMLU** (Hendrycks et al., 2020), **GSM8K** (Cobbe et al., 2021), **MATH500** (Hendrycks et al., 2021), and **BBH** (Suzgun et al., 2023), covering both general knowledge and mathematical reasoning tasks. For GSM8K, we use the full test set with 1,319 examples, and for MATH500, we use all 500 examples. For BBH and MMLU, we randomly sample 500 examples from the full test set for evaluation. For all benchmarks, we use a generation length of 512 and a block size of 64. All experiments are performed using greedy decoding and run on NVIDIA H200 GPUs. We compare AdaLook with ETE (Fu et al., 2025) and the confidence-aware parallel decoding strategy from Fast-dLLM (Wu et al., 2025), which adaptively adjusts the decoding progress according to both the number of decoded tokens and their confidence scores.

Hyperparameter Configurations. Both ETE and AdaLook involve several hyperparameters. As discussed in Section 4.3, the hyperparameters that need to be varied across configurations for both methods are the per-block unlocking budget N, the confidence threshold C, and the lookahead triggering threshold γ. Importantly, AdaLook does not introduce additional tunable hyperparameters beyond those required by ETE; the extra hyperparameters specific to our adaptive multi-step mechanism are determined once on a small calibration set and fixed across all experiments.

For N and C, we follow the calibration guideline in Section 4.3. Specifically, we choose N from powers of two in the range $[2, 64)$, and pair each N with confidence thresholds C suggested by the corresponding range-valued rule, forming a set of accuracy–efficiency trade-off configurations. For γ, since no stable calibration pattern is observed, we follow the discussion in Section 4.3 and consider values in $[0.1, 0.4]$. For both ETE and AdaLook, we report two settings. The *optimized* setting scans all combinations of $\gamma \in [0.1, 0.4]$ and the calibrated N/C pairs, and reports the best Pareto-frontier results. The *standard* setting fixes $\gamma = 0.2$, a default value that appears frequently among optimized configurations, and uses the same N/C calibration rule.

5.2 Main Results

We report the accuracy–efficiency trade-off obtained under different N/C configurations in Figure 3. Each point corresponds to one decoding configuration, and the curves compare how accuracy changes as the decoding budget varies across different datasets. Here, decoding steps refer to the average number of model forward passes used during generation across evaluation examples. Based on Figure 3, we have the following observations.

First, AdaLook consistently achieves a better accuracy–efficiency trade-off than the baselines across datasets. When comparing (*AdaLook (Standard)*, *ETE (Standard)*, Fast-dLLM), and (*AdaLook (Optimized)*, *ETE (Optimized)*, and Fast-dLLM), we generally observe the ordering *AdaLook* $>$ *ETE* $>$ Fast-dLLM in terms of the trade-off between accuracy and decoding steps. For example, at around 45 decoding steps, *AdaLook (Optimized)* achieves approximately 4.5% higher accuracy than *ETE (Optimized)*. This suggests that adaptive multi-step lookahead improves upon one-step lookahead by exploring more informative future decoding trajectories, while still avoiding unnecessary rollout computation. In contrast, Fast-dLLM only achieves a comparable trade-off on the relatively easier GSM8K benchmark when sufficient decoding steps are allowed. This suggests that confidence-aware decoding without more strategic exploration mechanisms is insufficient for harder reasoning tasks where low-confidence regions are more frequently encountered.

Second, the improvement varies across datasets.

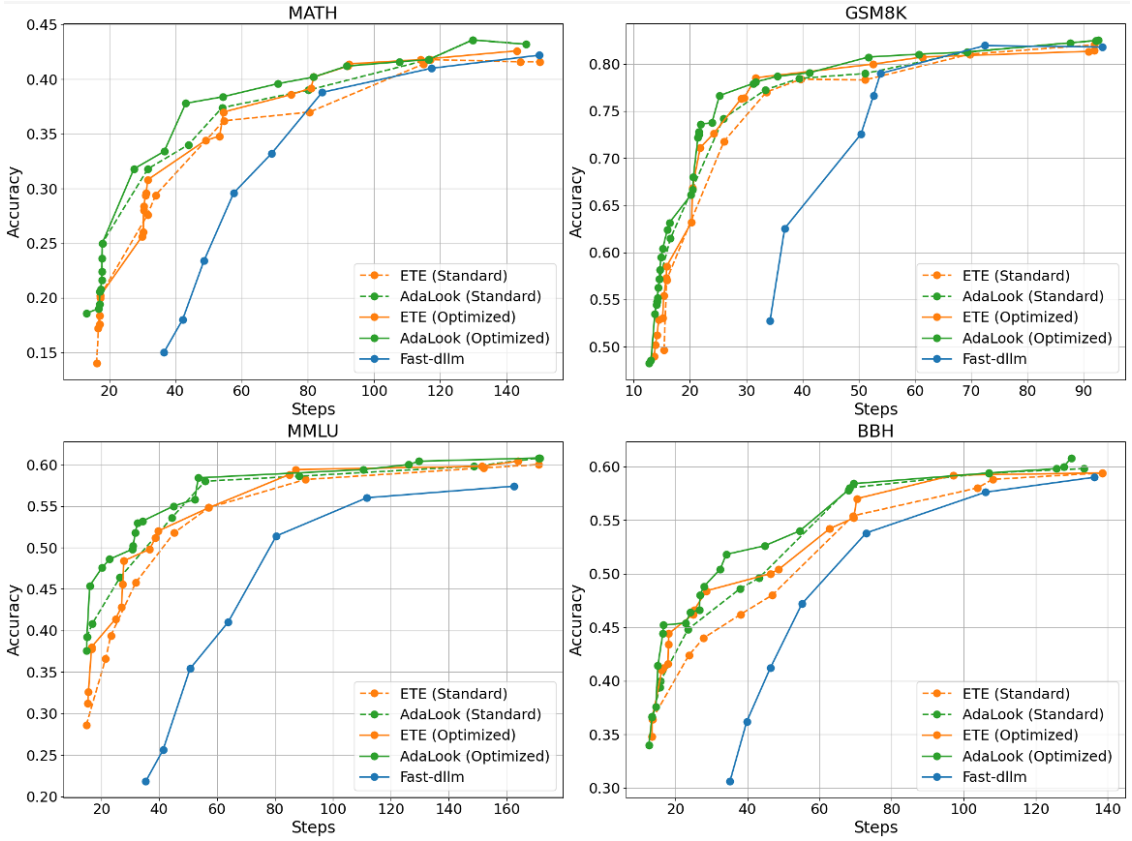

Figure 3: Accuracy vs decoding steps across different benchmarks

The gain is relatively modest on GSM8K, but becomes more pronounced on more challenging benchmarks such as MATH and BBH. This indicates that harder examples may benefit more from additional lookahead steps, as they often require longer-range exploration to identify reliable token commitments. To explain this, more difficult tasks may involve higher uncertainty in decoding, where locally confident token decisions may still lead to suboptimal reasoning paths, making shallow exploration insufficient. In contrast, simpler examples often exhibit more stable and less ambiguous decoding trajectories, where local confidence is already a strong indicator of correct token commitments, leaving less room for deeper lookahead to further improve the trade-off.

Third, AdaLook also tends to achieve a higher maximum reachable accuracy, particularly on more challenging benchmarks such as MATH, MMLU, and BBH. For example, on MATH, the best performance achieved by AdaLook is 43.6%, compared to 42.6% for ETE and 42.2% for Fast-dLLM. This suggests that adaptive multi-step lookahead not only improves decoding efficiency but also helps the decoder discover higher-quality reasoning trajectories that may be difficult to reach with shallow exploration alone.

To further evaluate the generality of AdaLook across different model backbones, we also conduct experiments on Dream-v0-Instruct-7B (Ye et al., 2025); the corresponding results are provided in Appendix C.

5.3 Analysis on Latency

In Section 5.2, we use the average number of forward steps per example to evaluate decoding efficiency. This metric provides a hardware-independent estimate of inference cost. However, our adaptive multi-step lookahead may require more batched forward passes during rollout, since multiple candidate trajectories are evaluated in parallel within a single lookahead step. As a result, this additional batched computation may increase the overall generation latency, even under a similar number of effective decoding steps.

To better understand this effect, we further analyze the generation latency under different beam sizes k, which corresponds to the batch size used for parallel candidate evaluation. Table 1 reports the average time per decoding step on H200 and B200 GPUs, measured in milliseconds. Notably, this value is not computed by timing an isolated model forward pass. Instead, we measure the

Table 1: Average latency on MATH with different GPUs

Beam Size	H200		B200	
	ETE	AdaLook	ETE	AdaLook
2	0.070	0.074	0.052	0.0541
4	0.086	0.102	0.056	0.0606
6	0.104	0.124	0.068	0.0736

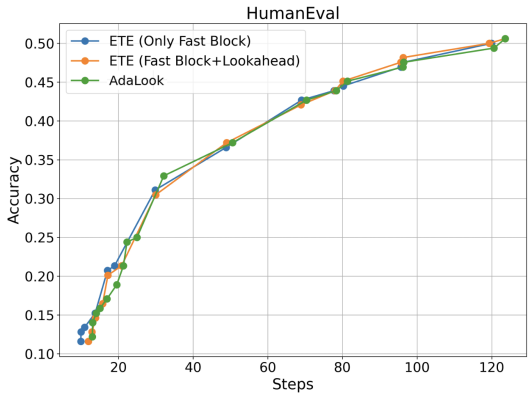

Figure 4: Performance of different methods in HumanEval

end-to-end generation time, including model forward passes, score computation, candidate selection, branch evaluation, and other decoding overhead, and divide it by the average number of forward steps. Therefore, the reported value should be interpreted as the average time corresponding to one decoding step reported in Figure 3 for different decoding methods.

According to the results in Table 1, the latency per decoding step increases for both ETE and AdaLook as the beam size grows, reflecting the additional cost of evaluating more candidate trajectories in parallel during lookahead decoding. Due to the additional rollout and branch expansion operations, AdaLook introduces slightly higher latency than ETE. However, this gap becomes much smaller on stronger GPUs such as the B200. Under the beam size $k = 4$ used in our main experiments, AdaLook increases the average per-step latency by only around 7% compared with ETE on B200 GPUs. This suggests that, with a reasonable beam size and modern GPU hardware, the additional latency introduced by our adaptive multi-step lookahead is nearly negligible in practice.

5.4 Discussion on Code Generation Tasks

In this section, we analyze the behavior of lookahead-based decoding on code generation tasks, which exhibit a qualitatively different response to lookahead exploration compared to mathematical reasoning and general knowledge benchmarks. Specifically, we find that confidence-based lookahead signals are less effective on coding benchmarks such as HumanEval (Chen et al., 2021): even the one-step lookahead strategy in ETE provides only marginal gains over Fast Block Diffusion Sampling. As shown in Figure 4, the trade-off curves of Fast Block Diffusion Sampling and Fast Block Diffusion Sampling + Lookahead remain highly similar, with only minor differences across decoding budgets. Moreover, our adaptive multi-step lookahead method also lies roughly on the same frontier, suggesting that both one-step and multi-step lookahead provide limited benefits for code generation tasks.

We hypothesize that this limitation stems from the structural nature of code generation. Unlike mathematical reasoning tasks, where informative token commitments can effectively resolve local ambiguity and lead to cascades of high-confidence future predictions, code generation often involves long-range structural and semantic dependencies that span much of the generated sequence. As a result, the quality of a token update is difficult to reliably assess within a limited rollout horizon, and the confidence gain observed during short lookahead trajectories may provide a weak signal for identifying truly beneficial token commitments. Consequently, although our method extends one-step lookahead to adaptive multi-step exploration, the rollout horizon may still remain insufficient to capture the long-range dependencies required for code generation, limiting the benefit of strategic lookahead. We leave the development of lookahead mechanisms better suited for code generation to future work.

6 Conclusion

In this work, we present AdaLook, an adaptive multi-step lookahead framework for efficient masked diffusion language model decoding. Unlike prior one-step lookahead methods that focus on immediate information gain, AdaLook explores longer decoding trajectories while dynamically controlling rollout depth and branch expansion. By using candidate-score variance to decide whether to continue rollout and re-evaluating branches at intermediate states, the proposed method avoids unnecessary computation while enabling flexible exploration. Experiments on multiple benchmarks and model backbones show that AdaLook improves the accuracy–decoding steps trade-off over existing decoding baselines.

Limitations

While our method consistently improves the accuracy–efficiency trade-off in terms of the number of decoding steps, the adaptive multi-step lookahead introduces slightly higher computational overhead per decoding step due to the batched forward passes required for hypothesis evaluation, even on high-end GPUs. We believe that this overhead will become increasingly negligible as hardware capabilities continue to advance and batched inference becomes more efficient.

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 S Nolte, and Brian Karrer. 2026. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *Advances in Neural Information Processing Systems*, 38:55981–56007.

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*.

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. 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*.

Hengyu Fu, Baihe Huang, Virginia Adams, Charles Wang, Venkat Srinivasan, and Jiantao Jiao. 2025. From bits to rounds: Parallel decoding with exploration for diffusion language models. *arXiv preprint arXiv:2511.21103*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

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*.

Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. 2020. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761*.

Sanghyun Lee, Seungryong Kim, Jongho Park, and Dongmin Park. 2025. Lookahead unmasking elicits accurate decoding in diffusion language models. *arXiv preprint arXiv:2511.05563*.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. 2025. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2026. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646.

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 *International Conference on Learning Representations*, volume 2025, pages 64972–65009.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. 2024. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2020. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and 1 others. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051.

Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Dongrui Liu, and Linfeng Zhang. 2025. Accelerating diffusion large language models with slowfast: The three golden principles. *arXiv e-prints*, pages arXiv–2506.

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 preprint arXiv:2505.22618*.

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

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

A Algorithm

We present the detailed algorithm of AdaLook in Algorithm 1.

B Justification for the log-scale calibration rule

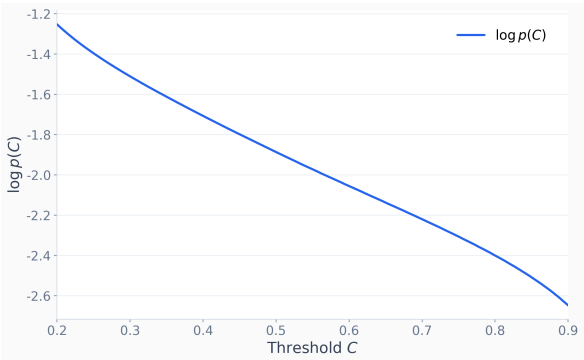

Figure 5: Relationship between $\log(p(c))$ and C (evaluated with MATH500).

We provide an additional justification for why the preferred confidence threshold C increases approximately logarithmically with the per-block unlocking budget N.

For a fixed confidence threshold C, the expected fraction of tokens that can be committed in one refinement step is

$$p(C) = \Pr(c \geq C).$$

After N rounds the expected fraction of tokens committed within the block can be approximated as $1 - (1 - p(C))^N$. If we want to maintain a comparable block progression rate with different values of N, this probability should remain roughly at a constant level, then we have

$$(1 - p(C))^N \approx \text{const}.$$

Taking log for both sides:

$$N \log(1 - p(C)) = -\text{const}.$$

When p is small, we have $\log(1 - p) \approx -p$, then we have

$$p(C) \approx \frac{\text{const}}{N} \tag{2}$$

Thus, as N increases, the required per-step commitment probability $p(C)$ decreases, meaning that a larger confidence threshold C can be used.

To connect this relationship to the logarithmic schedule, we empirically examine the tail probability of masked-token confidence. As shown in Figure 5, over the practical calibration range, the log-tail probability $\log \Pr(c \geq C)$ decreases approximately linearly with C when $C \in [0.2, 0.9]$. This suggests that the high-confidence tail can be approximated by an exponential form:

$$p(C) \approx \exp(-\lambda C),$$

for some $\lambda > 0$. Combining this empirical approximation with Eq.(2) yield:

$$\exp(-\lambda C) \approx \frac{\text{const}}{N}.$$

Therefore,

$$C \approx \frac{1}{\lambda} \log N + \text{const}.$$

This provides a theoretical justification for using a log-scale guideline to select C as the unlocking budget N varies, while the exact threshold is chosen empirically on the calibration set.

C Additional Empirical Results for DREAM-v0-Instruct-7B

In this subsection, we provide additional evaluation of AdaLook using **Dream-v0-Instruct-7B** (Ye et al., 2025). The results on GSM8K and MMLU are presented in Figure 6. As shown in the figure, AdaLook consistently achieves a better accuracy–efficiency trade-off than the two baselines, demonstrating that the benefits of adaptive multi-step lookahead generalize across different DLM backbones.

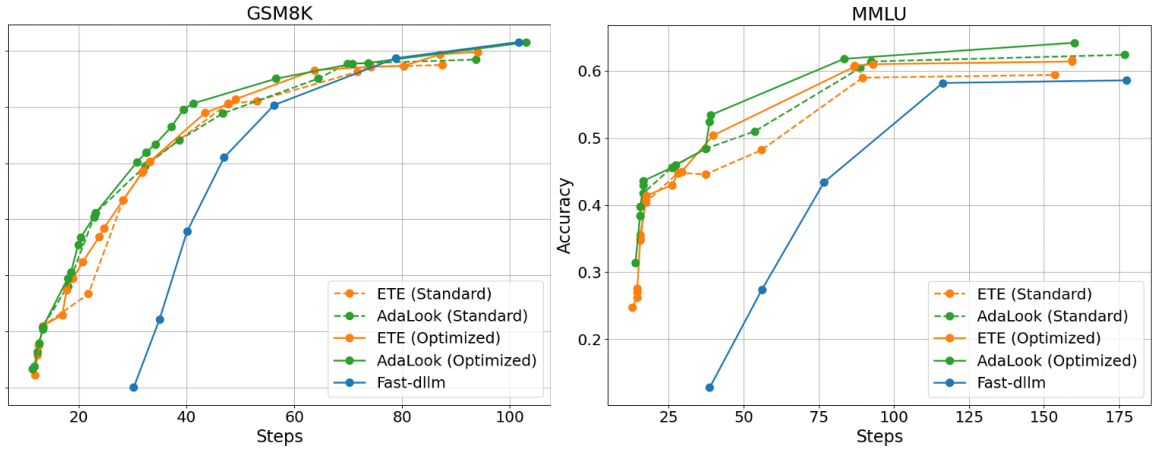

Figure 6: Accuracy vs decoding steps across different benchmarks with DREAM-v0-Instruct-7B

D Disclosure of AI Assistant Usage

AI assistants were used solely for language polishing and proofreading during manuscript preparation. Specifically, they were used to improve grammar, spelling, clarity, and overall readability of the text. All technical content, experiments, analyses, and conclusions were developed and verified by the authors.

E Artifacts, Licensing, and Intended Use

All datasets and pretrained models used in this work are publicly available and are utilized in accordance with their respective licenses. Our use is strictly limited to research and evaluation purposes, consistent with the intended scope of each artifact. No third-party artifacts are redistributed or modified in terms of their access conditions.

Algorithm 1 Adaptive Multi-Step Lookahead Algorithm

Require: Model p_θ, current state $\mathbf{x}_t$, masked token set $\mathcal{M}$, beam width k, medium-confidence target c^{info}, position bias β, downstream threshold C, sample-quality weight α, variance threshold τ, lookahead budget M

// Step 1: Select k exploration candidates
Score each masked position $i \in \mathcal{M}$: $\quad \phi_i = -|c_i - c^{\text{info}}| + \beta\,(i - (b-1)\,n_b)$
$\mathcal{H} \leftarrow \text{Top-}k\big(\{\phi_i\}_{i\in\mathcal{F}}\big)$
// Step 2: Build k hypotheses and run one-step lookahead
for each candidate $j \in \mathcal{H}$ **do**
  $\mathbf{x}^j \leftarrow \mathbf{x}_t$ with position j committed
end for
$\{(\hat{x}^j, c^j)\}_{j\in\mathcal{H}} \leftarrow \textsc{BatchedForward}(p_\theta,\ \{\mathbf{x}^j\}_{j\in\mathcal{H}})$
for each $j \in \mathcal{H}$ **do**
  $s_j \leftarrow$ Eq.(1) ▷ Score j by unlocked confidence mass
end for
// Step 3: Variance gate — decide whether to run extra forward chain
if $\mathrm{Var}(\{s_j\}_{j\in\mathcal{H}}) < \tau$ **then**
  for $r = 1, \ldots, M-1$ **do**
    $\{x^j\} \leftarrow \textsc{Commit}(\{(\hat{x}^j, c^j)\})$ for all $j \in \mathcal{H}$
    $\{(\hat{x}^j, c^j)\} \leftarrow \textsc{BatchedForward}(p_\theta,\ \{\mathbf{x}^j\}_{j\in\mathcal{H}})$
    Check retrigger: $\mathrm{r}_j \leftarrow \textsc{CheckTrigger}(\{(\hat{x}^j, c^j)\})$ for all j
    if all $\mathrm{r}_j =$ True **then** ▷ Case 2: all trigger
      $j^* \leftarrow \arg\max_j\,(s_j + \log \bar{c}^j_{\mathcal{F}})$
      return $(\hat{x}^{j^*}, c^{j^*})$ as initial state for next decoding round
    else if no $\mathrm{r}_j =$ True **then** ▷ Case 1: no trigger
      $s_j \leftarrow$ Eq.(1) for all $j \in \mathcal{H}$
      if $\mathrm{Var}(s_j) \geq \tau$ **then**
        $j^* \leftarrow \arg\max_j\ s_j$; **go to** Commit
      end if
    else ▷ *Case 3: Mixed — shrink to stable hypotheses*
      $\mathcal{H} \leftarrow \{j \in \mathcal{H} : \mathrm{r}_j = \texttt{False}\}$
      $s_j \leftarrow$ Eq.(1) for all $j \in \mathcal{H}$
      if $|\mathcal{H}| = 1$ **then**
        $j^* \leftarrow$ the unique element in $\mathcal{H}$; **go to** Commit
      end if
      if $\mathrm{Var}(s_j) \geq \tau$ **then**
        $j^* \leftarrow \arg\max_j\ s_j$; **go to** Commit
      end if
    end if
    $j^* \leftarrow \arg\max_j\ s_j$; **go to** Commit
  end for
else $j^* \leftarrow \arg\max_j\ s_j$; **go to** Commit
end if
// Final Step: Commit
$\mathbf{x}^{j^*} \leftarrow \textsc{Commit}(\hat{x}^{j^*}, c^j)$
return $\mathbf{x}^{j^*}$