

Adaptive Test-Time Compute Allocation for Block Diffusion Language Models in Complex Reasoning

Yi Lu^{1,3*}, Deyang Kong^{2,3*}, Jianing Wang³, Linsen Guo³, Xue Wang³, Qi Guo^{2,3},
Tao Gui^{1†}, Xuanjing Huang¹, Wei Ye^{2†}, Shikun Zhang², Wei Wang^{3†}

¹Fudan University ²Peking University ³Meituan LongCat Team
tgui@fudan.edu.cn, wye@pku.edu.cn, wangwei432@meituan.com

Abstract

Recent advances in block diffusion language models have demonstrated competitive performance and strong scalability on reasoning tasks. However, their test-time compute allocation remains largely unexplored, leaving a critical speed-effectiveness trade-off unresolved in long Chain-of-Thought reasoning. To address this, we propose a unified test-time compute allocation framework that introduces adaptivity in both step-wise decoding and blockwise generation. At the decoding level, we propose Bounded Adaptive Confidence Decoding (BACD), a difficulty-aware sampling strategy that dynamically adjusts denoising based on model confidence, accelerating inference while controlling error accumulation. Beyond step-wise adaptivity, we introduce the Think Coarse, Critic Fine (TCCF) paradigm that allocates large block sizes for exploratory thinking and smaller block sizes for precise refinement. To stabilize training under varying block configurations, we adopt Progressive Block Size Extension, which mitigates quality degradation when scaling up block sizes. Extensive evaluations on six benchmarks show that our TDAR-8B model with BACD and TCCF achieves a $2.38\times$ speedup and +3.4% average accuracy over the strong TraDo-8B baseline, unlocking the potential of block diffusion in complex reasoning. Our code and models are available at <https://github.com/LuLuLuyi/TDAR>.

1 Introduction

Recent advances in Diffusion Large Language Models (DLLMs), exemplified by LLaDA (Nie et al., 2025), Dream (Ye et al., 2025b), Gemini Diffusion (Gemini, 2025), have demonstrated the scalability of this paradigm (Nie et al., 2024; Gong et al., 2025a) and have achieved remarkable performance on mathematics (Ye et al., 2025b; Cheng

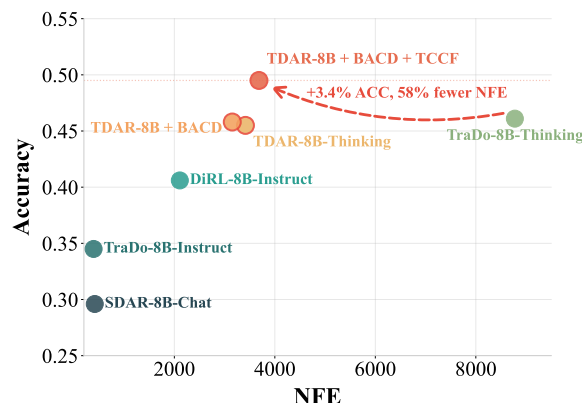


Figure 1: Performance and speed comparison of BDLMs. Our TDAR-8B-Thinking achieves $2.38\times$ speedup and +3.4% accuracy with BACD and TCCF compared to the best baselines.

et al., 2025), code generation (Gong et al., 2025b; Wang et al., 2025), and general reasoning tasks (Ye et al., 2025a; Shao et al., 2025). Building on the recent success of DLLMs, Block Diffusion Language Models (BDLMs) (Arriola et al., 2025) have emerged as a promising direction that integrates diffusion with traditional autoregressive (AR) decoding, enabling efficient KV caching and improving inference efficiency and generation flexibility.

However, existing BDLMs (Bie et al., 2025; Wang et al., 2025; Zhu et al., 2025c) primarily focus on simple reasoning tasks and offer limited exploration under the test-time scaling paradigm. While Wang et al. (2025) train a block diffusion model for complex reasoning, they do not systematically investigate how block diffusion should balance efficiency and effectiveness in such settings. Test-time scaling with long Chain-of-Thought (CoT) reasoning presents a double-edged scenario for block diffusion, i.e., although it naturally favors parallel decoding, the increased reasoning complexity substantially intensifies the trade-off between efficiency and effectiveness. In Figure 1, we observe that current BDLMs struggle

*Equal contribution.

†Corresponding authors.

to simultaneously achieve high performance and efficiency on complex reasoning tasks, where improvements in efficiency often lead to a substantial degradation in performance.

To address the efficiency–effectiveness trade-off of block diffusion under test-time compute allocation, we propose a unified framework that introduces adaptivity in both decoding and generation strategies. Our key insight is that long reasoning trajectories are inherently heterogeneous: different parts vary not only in local sampling difficulty, but also in their role within the overall reasoning process. Effectively exploiting this non-uniformity is crucial for achieving both efficiency and reliability in block diffusion. At the decoding stage, we introduce **Bounded Adaptive Confidence Decoding (BACD)**, a dynamic sampling strategy that adapts the denoising process to the varying difficulty of long reasoning trajectories. Using the average confidence of previously decoded tokens as a difficulty signal, our method enables aggressive acceleration when the model is highly confident via an upper-bound threshold, while enforcing a lower-bound threshold to prevent excessive error accumulation under uncertainty. Beyond step-wise decoding adaptivity, we propose **Think Coarse, Critic Fine (TCCF)**, a test-time scaling paradigm that exploits the heterogeneous roles within long reasoning trajectories. While exploratory reasoning can be generated efficiently with coarse-grained decoding, refinement and summarization demand finer granularity to ensure correctness. TCCF therefore allocates larger block sizes to exploratory segments and smaller block sizes to refinement-oriented segments, enabling an effective balance between efficiency and reasoning quality under test-time scaling. To enable efficient and effective decoding in a large block size, we adopt Progressive Block Size Extension, a multi-stage denoising fine-tuning strategy, which mitigates performance degradation when scaling block sizes. These designs enable BDLMs to effectively balance efficiency and reasoning quality, unlocking their potential for test-time scaling on complex reasoning tasks.

To evaluate the effectiveness and generalizability of our approach, we conduct extensive experiments on a diverse set of six reasoning benchmarks, including mathematics, code generation, and STEM. Results show that TDAR-8B-Thinking with BACD achieves state-of-the-art performance among 8B-scale block diffusion language models while reaching up to 4.14 Tokens Per Forward Pass.

In addition, the TCCF paradigm further improves reasoning performance and offers a better trade-off between speed and accuracy. In summary, our contributions are threefold:

- We propose Bounded Adaptive Confidence Decoding, a difficulty-aware decoding strategy that dynamically adapts denoising at test time, improving both performance and efficiency with strong robustness.
- We introduce Think Coarse, Critic Fine, a new test-time compute allocation paradigm that combines coarse-grained exploration with fine-grained refinement, significantly enhancing complex reasoning performance with minimal efficiency overhead.
- We identify block sizes as a key factor for BDLM acceleration and propose Progressive Block Size Extension, which mitigates performance degradation when scaling block sizes and unlocks BDLMs’ parallel potential.

2 Preliminary

2.1 Block Diffusion Language Models

Block Diffusion Language Models (Arriola et al., 2025; Wu et al., 2025a; Cheng et al., 2025) interpolate between autoregressive and diffusion paradigms. A token sequence $\mathbf{x}$ is partitioned into K non-overlapping blocks $\mathbf{x} = (\mathbf{x}^1, \dots, \mathbf{x}^K)$, where the block size of $\mathbf{x}^k$ represents as B . The model factorizes the likelihood autoregressively over blocks:

$$\log p_\theta(\mathbf{x}) = \sum_{k=1}^K \log p_\theta(\mathbf{x}^k | \mathbf{x}^{<k}), \quad (1)$$

where $\mathbf{x}^{<k}$ denotes the preceding clean blocks. Each conditional distribution $p_\theta(\mathbf{x}^k | \mathbf{x}^{<k})$ is modeled via a discrete masked diffusion process: a forward process and a reverse process. The forward process $q(\mathbf{x}_t^k | \mathbf{x}_0^k)$ gradually masks tokens within block $\mathbf{x}^k$. When adopt a linear noise schedule defined by $\alpha_t = 1 - t$ at time $t \in [0, 1]$, each token is independently replaced by a special [MASK] token $\mathbf{m}$ with probability $1 - \alpha_t$:

$$q(\mathbf{x}_t^k | \mathbf{x}_0^k) = \prod_{i=1}^B \text{Cat}((\mathbf{x}_t^k)_i; (1 - t)(\mathbf{x}_0^k)_i + t\mathbf{m}). \quad (2)$$

The reverse process reconstructs the original block $\mathbf{x}_0^k$ from the noisy state $\mathbf{x}_t^k$ given the history $\mathbf{x}^{<k}$.

The model predicts the distribution of original tokens in parallel:

$$p_\theta(\mathbf{x}_0^k | \mathbf{x}_t^k, \mathbf{x}^{<k}) = \prod_{i=1}^B p_\theta((\mathbf{x}_0^k)_i | \mathbf{x}_t^k, \mathbf{x}^{<k}). \quad (3)$$

Standard block discrete denoising diffusion models minimize the Negative Evidence Lower Bound (NELBO). Under the linear noise schedule $\alpha_t = 1 - t$, the objective is scaled by the inverse noise level $1/t$. The loss is computed exclusively on the set of masked positions $\mathcal{M}_t^k$ within each block:

$$\mathcal{L}(\theta) = \mathbb{E}_{t, \mathbf{x}} \left[-\frac{1}{t} \sum_{k=1}^K \sum_{i \in \mathcal{M}_t^k} \log p_\theta((\mathbf{x}_0^k)_i | \mathbf{x}_t^k, \mathbf{x}^{<k}) \right] \quad (4)$$

2.2 Sampling Algorithm for BDLMs

Generation in BDLMs proceeds block-by-block. For the k -th block $\mathbf{x}^k$, decoding starts from a fully masked state $\mathbf{x}_T^k$. At each denoising step t , the model predicts token probabilities for all masked positions $i \in \mathcal{M}_t^k$ conditioned on the current state $\mathbf{x}_t^k$, history $\mathbf{x}^{<k}$, and prompt $\mathbf{p}$. We define the *confidence* score for position i as:

$$c_i(\mathbf{x}_t^k) := \max_{v \in \mathcal{V}} p_\theta((\mathbf{x}_0^k)_i = v | \mathbf{x}_t^k, \mathbf{x}^{<k}, \mathbf{p}). \quad (5)$$

A subset of masked tokens $\mathcal{D}_t \subseteq \mathcal{M}_t^k$ is then selected for unmasking based on c_i . We introduce two widely used strategies for BDLMs:

Static Confidence Decoding. (Nie et al., 2025)

A fixed number of tokens, $n = \lceil B/N \rceil$, are decoded at each step, where N is the total denoising steps. Specifically, $\mathcal{D}_t$ consists of the n masked positions with the highest confidence scores.

Dynamic Confidence Decoding. (Wu et al., 2025b)

Tokens exceeding confidence threshold τ are decoded, with the selection set defined as $\mathcal{D}_t = \{i \in \mathcal{M}_t^k \mid c_i(\mathbf{x}_t^k) > \tau\}$. This accelerates inference by rapidly decoding easy tokens.

3 Method

We propose a unified framework for test-time scaling in Block Diffusion Language Models (BDLMs), which exploits the non-uniform nature of long reasoning trajectories through adaptivity at two levels. At the decoding level, we design a dynamic sampling strategy that adjusts denoising steps to local difficulty. At the reasoning-phase level, we introduce a new test-time scaling

paradigm that adapts block sizes to different stages of reasoning. An overview of our framework is shown in Figure 2.

3.1 Bounded Adaptive Confidence Decoding

Dynamic decoding for block diffusion models typically relies on a fixed confidence threshold, which becomes increasingly brittle as block size grows: high thresholds lead to inefficient decoding, while low thresholds cause error accumulation due to overly aggressive unmasking. A fundamental reason is that the confidence distribution within a block is non-stationary: it starts low when context is sparse and gradually rises as more tokens are unmasked.

To address this, we propose **Bounded Adaptive Confidence Decoding (BACD)**. Instead of using a fixed threshold, BACD dynamically adjusts the acceptance threshold based on the running mean of confidence scores from previously accepted tokens, allowing the criterion to naturally adapt as more tokens are revealed within a block. To prevent instability, we impose explicit upper and lower bounds on the threshold. The detailed decoding procedure is summarized in Algorithm 1.

BACD maintains a list of confidence scores from previously decoded tokens and uses their average to guide future decoding decisions. Let $\mathbf{c}_{t-1}$ denote the list of confidence scores collected before step t . In the initial state where $\mathbf{c}_{t-1}$ is empty, we set $\tau_t = \tau_h$. At each decoding step, we compute the average confidence $\bar{c}_{t-1} = \text{mean}(\mathbf{c}_{t-1})$. To stabilize decoding behavior, we introduce two thresholds: an upper bound τ_h and a lower bound τ_l . The effective decoding threshold τ_t is determined via a clipping operation:

$$\tau_t = \text{clip}(\bar{c}_{t-1}, \tau_l, \tau_h). \quad (6)$$

Using τ_t , we select the set of tokens to be unmasked at step t :

$$\mathcal{D}_t = \{i \in \mathcal{M}_t^k \mid c_i(\mathbf{x}_t^k) > \tau_t\}. \quad (7)$$

To guarantee convergence, if $\mathcal{D}_t = \emptyset$, we unmask the token with the highest confidence score. The confidence scores of newly decoded tokens are then appended to the list $\mathbf{c}$ for subsequent steps. By bounding the adaptive threshold, BACD prevents overly conservative or overly aggressive decoding, enabling a more stable trade-off between efficiency and generation quality.

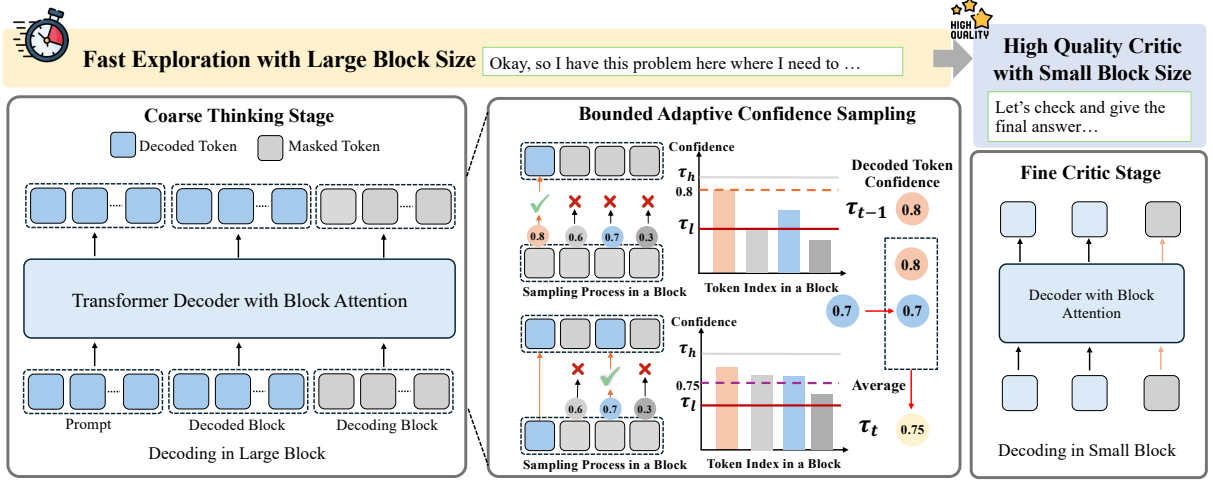


Figure 2: Overview of our reasoning process. We use Bounded Adaptive Confidence Decoding to enable fast exploration with large block sizes, and apply small block sizes for fine-grained refinement.

Alg. 1: Bounded Adaptive Confidence Decoding

Input: Masked block $\mathbf{x}_T^k$, bounds τ_h, τ_l
Output: Clean block $\mathbf{x}_0^k$
Initialize confidence list $\mathbf{c} \leftarrow []$;
while masked tokens exist in $\mathbf{x}^k$ **do**
 Let $\mathcal{M} \leftarrow \{i \mid (\mathbf{x}^k)_i = [\text{MASK}]\}$;
 Compute confidence scores c_i for all $i \in \mathcal{M}$;
 Calculate mean $\bar{c} \leftarrow \text{mean}(\mathbf{c})$ if $\mathbf{c} \neq []$ else τ_h ;
 Set threshold $\tau \leftarrow \text{clip}(\bar{c}, \tau_l, \tau_h)$;
 Select candidates $\mathcal{D} \leftarrow \{i \in \mathcal{M} \mid c_i > \tau\}$;
 if $\mathcal{D} = \emptyset$ **then** $\mathcal{D} \leftarrow \{\arg \max_{i \in \mathcal{M}} c_i\}$;
 Unmask tokens at indices $\mathcal{D}$, append scores to $\mathbf{c}$;
end while

Coarse Thinking. The model first generates an exploratory reasoning trajectory $\mathbf{r}$ using a large block size B_{think} to maximize efficiency:

$$\mathbf{r} \sim p_{\theta}(\cdot \mid \mathbf{p}; B_{\text{think}}). \quad (8)$$

Fine Critic. Conditioned on $\mathbf{p}$ and the exploratory trajectory $\mathbf{r}$, the model performs refinement and consolidation using a smaller block size $B_{\text{critic}} < B_{\text{think}}$ to improve reasoning reliability:

$$\mathbf{y} \sim p_{\theta}(\cdot \mid \mathbf{p}, \mathbf{r}; B_{\text{critic}}). \quad (9)$$

3.2 Think Coarse, Critic Fine

Long CoT reasoning is not uniform across its trajectory. Different segments of a reasoning process serve distinct functional roles. In particular, early segments often focus on exploratory reasoning, while later segments emphasize refinement, verification, and summarization. However, block diffusion models exhibit a clear trade-off between block size and generation quality: larger block sizes enable faster decoding but may degrade accuracy, whereas smaller block sizes improve precision at the cost of efficiency. As a result, a single block size is insufficient to balance efficiency and reasoning quality across all stages.

Motivated by this observation, we propose Think Coarse, Critic Fine (TCCF), a stage-wise test-time compute allocation paradigm that assigns different block sizes to different functional stages of reasoning. TCCF explicitly exploits the block-size–quality trade-off in BDLMs. Given an input prompt $\mathbf{p}$, TCCF decomposes inference into two stages:

Training with Large Block Sizes. TCCF relies on large block sizes during coarse thinking, which can lead to performance degradation if the model is trained with a fixed small block size. To enable stable training under large block sizes, we adopt **Progressive Block Size Extension**, a multi-stage supervised fine-tuning strategy that gradually increases the block size. Given a training instance $(\mathbf{p}, \mathbf{x})$ consisting of a prompt $\mathbf{p}$ and a target response $\mathbf{x}$, the supervised fine-tuning objective is:

$$\mathcal{L}_{\text{SFT}}(\theta) = \mathbb{E}_{t, (\mathbf{p}, \mathbf{x})} \left[-\frac{1}{t} \sum_{k=1}^K \sum_{i \in \mathcal{M}_t^k} \log p_{\theta}((\mathbf{x}_0^k)_i \mid \mathbf{x}_t^k, \mathbf{x}^{<k}, \mathbf{p}) \right]. \quad (10)$$

This progressive scaling mitigates performance degradation during expansion, which is crucial for enabling TCCF’s coarse thinking stage.

Method	Math500		AIME24		AIME25		AMC23		LCB		GPQA		AVG		
	TPF	ACC	TPF	AVG@32	TPF	AVG@32	TPF	AVG@32	TPF	AVG@8	TPF	AVG@8	TPF	NFE	ACC
<i>Masked Diffusion Language Model</i>															
LLaDA	3.92	41.2	3.44	6.7	3.66	0.0	4.07	12.5	2.83	4.7	2.91	17.2	3.47	233	13.7
LLaDA-1.5	3.97	42.2	3.35	0.0	3.69	0.0	4.01	10.0	2.86	4.3	3.01	24.2	3.48	236	13.5
LLaDA-MoE	2.71	56.6	2.89	3.3	2.71	0.0	3.17	32.5	2.06	12.9	2.18	27.8	2.62	259	22.2
<i>Block Diffusion Language Model</i>															
Fast-dLLM-v2	2.81	59.4	2.59	0.0	2.58	0.0	2.78	25.0	1.73	6.8	2.09	28.3	2.43	259	19.9
SDAR-8B-Chat	2.59	71.2	3.01	7.0	2.33	9.5	2.63	45.3	2.32	13.7	1.98	30.9	2.48	416	29.6
DiRL-8B-Instruct	2.09	80.0	1.88	21.0	1.82	19.9	1.99	62.2	2.00	15.9	1.76	44.4	1.92	2109	40.6
TraDo-8B-Instruct	2.39	74.8	2.16	10.7	2.01	13.0	2.22	55.8	1.73	17.2	1.44	35.5	1.99	393	34.5
TraDo-8B-Thinking	1.42	85.8	1.32	30.6	1.32	24.5	1.36	69.7	1.22	22.2	1.30	44.0	1.32	8777	46.1
SDAR-30B-Thinking	1.66	91.6	1.63	61.6	1.60	53.6	1.62	91.8	1.31	38.2	1.37	64.9	1.53	7449	67.0
TDAR-8B-Thinking (ours)	3.98	78.0	3.76	35.7	3.50	29.8	4.00	68.8	2.67	30.6	4.64	30.2	3.76	3415	45.5
+ BACD	4.61	78.8	4.05	36.1	3.79	30.5	4.43	69.9	2.96	29.4	4.98	30.2	4.14	3153	45.8
+ BACD +TCCF	3.67	82.8	3.84	39.8	3.56	32.7	3.86	73.9	2.75	33.1	4.27	34.4	3.66	3684	49.5
TDAR-30B-Thinking (ours)	1.55	90.6	1.50	66.8	1.54	56.5	1.56	93.4	1.33	50.7	1.46	52.8	1.49	6402	68.5
+ BACD	1.63	90.8	1.56	69.1	1.60	55.7	1.60	93.3	1.38	51.5	1.51	52.6	1.55	6128	68.8
+ BACD +TCCF	1.62	91.0	1.57	69.1	1.60	57.4	1.62	94.7	1.39	51.4	1.52	56.8	1.55	6580	70.1

Table 1: Performance comparison of various models on six reasoning benchmarks.

4 Experiment

4.1 Experiment Setup

Adaptation to Long CoT Reasoning BDLMs. We adapt Qwen3-8B-base into Block DLMs following Cheng et al. (2025). We first train with a blockwise diffusion objective ($B = 4$) on a 50B-token annealing corpus, then perform SFT on long CoT data with progressive block-size expansion from $B = 4$ to $B = 64$. Based on the performance-efficiency trade-off analysis in Section 5.2, we select the model trained with a block size of $B = 16$ for 8B models and $B = 4$ for 30B models as our reasoning model denoted as **TDAR-8B-thinking** and **TDAR-30B-thinking**. Implementation details are provided in Appendix B.1.

Datasets and Baselines. To comprehensively evaluate BDLMs, we select representative benchmarks covering mathematical reasoning (Math500 (Hendrycks et al., 2021), AIME2024 (MAA, 2024), AIME2025 (MAA, 2025), AMC2023 (AMC, 2023)), code generation (LiveCodeBench v5¹ (Jain et al., 2024)), and STEM reasoning (GPQA-diamond (Rein et al., 2023)). Specifically, we configure 32 runs for AIME2024, AIME2025, and AMC2023; 8 runs for LiveCodeBench and GPQA-diamond.

We compare our method against state-of-the-art open-source BDLMs, including Fast-dllm-v2 (Wu et al., 2025a), SDAR-8B-Chat (Cheng et al., 2025), SDAR-30B-Thinking (Cheng et al., 2025), DiRL-8B-Instruct (Zhu et al., 2025c), TraDo-

8B-Instruct (Wang et al., 2025), and TraDo-8B-Thinking (Wang et al., 2025). To compare with masked diffusion language models, we involve LLaDA (Nie et al., 2025), LLaDA-1.5 (Zhu et al., 2025a), LLaDA-MoE (Zhu et al., 2025b).

Evaluation Setup We adopt the block size B recommended in their official repositories: $B = 4$ for SDAR-8B-Chat, DiRL-8B-Instruct, and TraDo-8B; $B = 32$ for Fast-dllm-v2. For decoding algorithms, we adapt widely used algorithms from DLLMs: Dynamic Confidence-Aware Decoding (Wu et al., 2025b). We employ Dynamic Confidence Decoding with a threshold $\tau = 0.9$ as the default algorithm for the main results. For BACD, we set the upper bound τ_h to 0.9 and the lower bound τ_l to 0.7. Other details are shown in Appendix B.2. For the TCCF paradigm, we configure TDAR-8B-thinking with $B_{\text{think}} = 16$ and $B_{\text{critic}} = 4$ and TDAR-30B-thinking with $B_{\text{think}} = B_{\text{critic}} = 4$. TCCF implementation details are provided in Appendix C.

Evaluation Metrics of Efficiency. We evaluate computational efficiency using two key metrics: Tokens Per Forward Pass (TPF) and Number of Function Evaluations (NFE). While generating a complete reasoning sequence, NFE denotes the actual number of forward passes and TPF measures the average number of tokens generated per forward model call, defined as:

$$\text{TPF} = \frac{\text{Total Generated Tokens}}{\text{NFE}}$$

Higher TPF and lower NFE values indicate greater decoding parallelism and lower inference overhead. In our main evaluation, we report benchmark-

¹The evaluation time range spans from August 2024 to May 2025.

specific TPF alongside the overall average TPF and NFE, with more exhaustive efficiency and throughput analyses provided in Appendix A.

4.2 Main Results

Performance and Speed As shown in Table 1, TDAR-8B-thinking achieves competitive reasoning performance alongside outstanding decoding efficiency across a wide range of benchmarks. Specifically, our base model, TDAR-8B-thinking, achieves an average decoding speed of 3.76 TPF, which represents a $2.57\times$ speedup in NFE compared to the strongest BDLM baseline, TraDo-8B-Thinking, while maintaining comparable accuracy. When fully equipped with our proposed BACD decoding algorithm and the TCCF paradigm, TDAR-8B-thinking achieves a final average accuracy of **49.5%**. This configuration outperforms TraDo-8B-Thinking by exactly 3.4 percentage points while maintaining a highly efficient decoding speed of 3.66 TPF, delivering a $2.38\times$ speedup in NFE. Practical GPU-time measurements and a controlled comparison with a matched autoregressive baseline are provided in Appendix A.2.

Effectiveness of BACD and TCCF Integrating the BACD decoding algorithm and the TCCF paradigm further optimizes the speed-accuracy trade-off of TDAR-8B-thinking. Specifically, BACD dynamically accelerates the decoding process by reducing average NFE while maintaining or slightly improving overall generation accuracy. Meanwhile, the TCCF paradigm significantly enhances reasoning performance on complex tasks by allocating computing budgets adaptively. For instance, on the highly challenging AIME24 benchmark, TCCF improves the reasoning accuracy of TDAR-8B-thinking to **39.8%** (a 3.7% absolute gain over the BACD variant) while still preserving a high decoding speed of 3.84 TPF. Across all evaluations, the synergy of BACD and TCCF demonstrates that adaptive block sizing and confidence bounding can robustly balance inference cost and deep reasoning quality.

Generalization Across Model Scales We further evaluate the scalability and generalization of our approach by scaling up to a larger model size. As shown in Table 1, TDAR-30B-thinking achieves a base average accuracy of **68.5%** with **1.49 TPF**, already outperforming the strong SDAR-30B-Thinking baseline in both accuracy and efficiency. Applying BACD to the 30B model increases the

average speed to **1.55 TPF** while slightly boosting accuracy to **68.8%**. Finally, equipped with TCCF, TDAR-30B-thinking reaches a strong average accuracy of **70.1%**. This demonstrates that our training and decoding paradigms generalize robustly to larger scales.

5 Analysis

5.1 Analysis on Sampling Algorithm

Comparison with Existing Decoding Algorithms.

We compare BACD ($\tau_h=0.9, \tau_l=0.7$) against mainstream decoding algorithms on TDAR-8B-Thinking. The baselines include: (1) **Static methods** (LowConfidenceStatic, EntropyStatic) with a fixed step budget of $N=B/4=4$; (2) **Dynamic Confidence** (Wu et al., 2025b) (our default setting, with $\tau=0.9$); (3) **EB-Sampler** (Ben-Hamu et al., 2025) using confidence, entropy, and margin proxies ($\gamma=0.5$); and (4) **Saber** (Dong et al., 2025) ($\mu=2$). Detailed configurations for each baseline are provided in Appendix D.

Results. As shown in Table 2, static methods suffer from catastrophic accuracy collapse, confirming that fixed decoding schedules are unsuitable for BDLMs. Among dynamic methods, EB-Sampler (entropy) achieves the highest accuracy (48.1%) but incurs severe computational overhead (7,358 NFE). Saber yields suboptimal accuracy despite high NFE. In contrast, BACD achieves the best overall trade-off, outperforming the default Dynamic Confidence baseline in both accuracy (+0.3%) and efficiency (reducing NFE to 3,153, a 7.7% reduction), while delivering the highest throughput (4.14 TPF) among all baselines.

Method	Acc	NFE	TPF
StaticConfidence	17.8	3707	4.00
StaticEntropy	3.5	3535	4.00
DynamicConfidence	45.5	3415	3.76
EB-Sampler (confidence)	44.9	3353	3.66
EB-Sampler (entropy)	48.1	7358	1.55
EB-Sampler (margin)	43.3	7677	1.56
Saber	34.9	3977	2.95
BACD	45.8	3153	4.14

Table 2: Comparison of BACD with existing decoding algorithms on TDAR-8B-Thinking, averaged over six benchmarks.

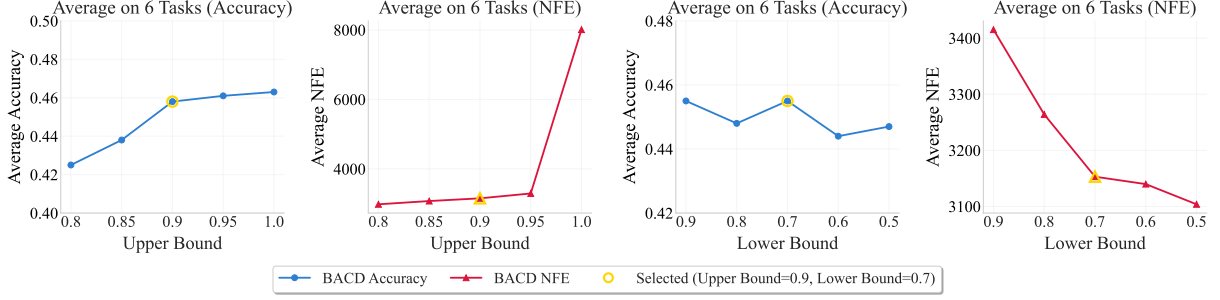


Figure 3: Accuracy and Speed under different thresholds across 6 tasks.

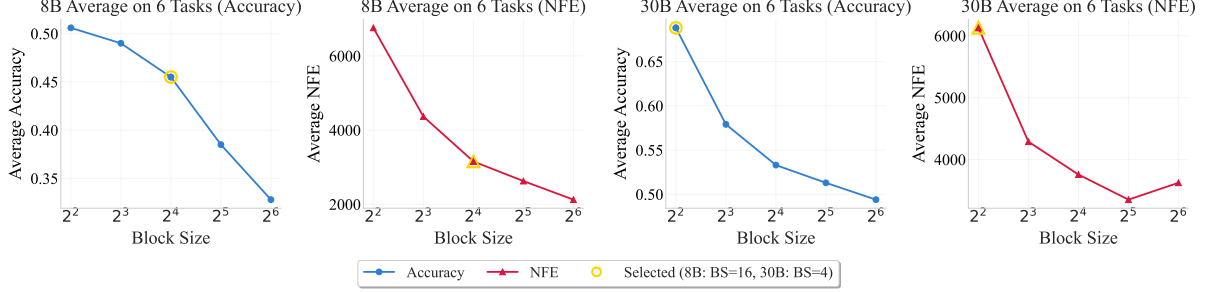


Figure 4: Impact of block size on 8B and 30B models' performance and efficiency. Gold marker indicates our selected checkpoint.

Performance and Speed under Different Thresholds. We ablate the impact of varying upper bound τ_h and lower bound τ_l in BACD across all 6 tasks in Figure 3. As τ_h increases from 0.8 to 0.95, accuracy steadily improves with only a modest NFE increase, since a higher upper bound makes the clipping operation more conservative, allowing fewer tokens to pass the threshold per step and thus improving generation quality. However, setting $\tau_h = 1.0$ causes the NFE to surge dramatically, because it reaches the theoretical ceiling of confidence scores, forcing the algorithm to repeatedly fall back to the single-token fallback rule ($\mathcal{D} \leftarrow \{\arg \max_i c_i\}$) and degenerating toward token-by-token decoding. For τ_l , varying it from 0.9 down to 0.5 yields a slight NFE reduction while accuracy degrades gradually, as a lower bound permits more aggressive early unmasking when the adaptive mean $\bar{c}$ drops, accepting lower-confidence tokens that may introduce errors. We select $\tau_h = 0.9$ and $\tau_l = 0.7$ as the final configuration, which achieves competitive accuracy (45.8%) at a moderate NFE (3153) while avoiding the efficiency collapse at extreme upper bounds.

5.2 Analysis on Block Size Extension

Larger block sizes accelerate inference by reducing NFE but degrade generation quality due to the challenges of parallel prediction. As the block size B

grows, approximating $p(x_{t+1}, \dots, x_{t+B} \mid x_{\leq t})$ in a single forward pass diverges further from the true autoregressive factorization, compounding prediction errors, as Figure 4 shown.

To balance this trade-off, we select the block size at the knee point of the Accuracy–NFE curve (details in Appendix E), where the marginal accuracy loss per unit NFE reduction sharply increases. For the 8B model, the marginal cost (accuracy loss per 1,000 NFE saved) rises from 0.029 ($B = 8 \rightarrow 16$) to 0.133 ($B = 16 \rightarrow 32$), a $4.6\times$ increase that identifies $B = 16$ as the optimal knee point. A similar analysis identifies $B = 4$ for the 30B model, leading us to adopt $B = 16$ for the 8B model and $B = 4$ for the 30B model.

6 Ablation Study

6.1 Ablation on Progressive Block Size Extension

We conduct an ablation study to evaluate the effectiveness of our progressive block-size expansion strategy compared to directly training with a large fixed block size. We train two variants of the TDAR-8B-Base model with a target block size of $B = 16$ for an equivalent number of tokens.

As shown in Table 3, the progressive strategy yields superior results across all benchmarks, suggesting that it effectively mitigates the optimization difficulties associated with large block sizes. On

average across six tasks, the progressive approach outperforms the direct baseline by 6.7 points in accuracy, while also reducing NFE by 213.9, indicating that progressive training leads to both better performance and more efficient inference.

Strategy	Acc	NFE	TPF
Direct Expansion	39.1	3366.9	4.40
Progressive Expansion	45.8	3153.0	4.14

Table 3: Ablation on block size expansion strategies.

These results clarify the roles of training-time block-size adaptation and test-time compute allocation in our framework. PBSE improves the model’s ability to reason with larger block sizes, while BACD and TCCF further optimize the inference-time accuracy–efficiency trade-off. These components are therefore complementary, with PBSE facilitating large-block decoding and BACD and TCCF controlling its use during inference.

6.2 Ablation on Think Coarse Critic Fine Paradigm

We conduct an ablation study on the block sizes used in the thinking stage (B_{think}) and the critic stage (B_{critic}) of the TCCF paradigm, with results shown in Table 4. Reducing B_{critic} consistently enhances reasoning accuracy, supporting our hypothesis: while the exploratory phase benefits from larger blocks to quickly navigate the search space, the final answer-formulation phase requires fine-grained prediction to prevent cascading errors.

However, accuracy is not always monotonic. For TDAR-8B (trained progressively up to $B = 16$), performance peaks at $B_{\text{critic}} = 4$ and declines at $B_{\text{critic}} = 1$. This suggests that abrupt single-token decoding creates a distribution mismatch with the blockwise training objective; thus, $B_{\text{critic}} = 4$ acts as a “sweet spot” balancing block-level coherence with precision. Conversely, TDAR-30B shows monotonic gains down to $B_{\text{critic}} = 1$ due to its smaller base size ($B_{\text{think}} = 4$). Crucially, this transition incurs minimal efficiency overhead: reducing TDAR-8B’s B_{critic} from 16 to 4 yields a substantial 1.5-point accuracy gain with under 7% NFE overhead, confirming TCCF’s Pareto-optimal efficiency in concentrating compute on the critique phase.

Model	B_{think}	B_{critic}	Acc	NFE	TPF
TDAR-8B	16	1	48.9	4251.2	3.15
TDAR-8B	16	2	49.5	3930.9	3.42
TDAR-8B	16	4	49.5	3684.4	3.66
TDAR-8B	16	8	48.3	3531.6	3.83
TDAR-8B	16	16	48.0	3453.9	3.94
TDAR-30B	4	4	70.1	6579.7	1.55
TDAR-30B	4	2	70.1	6730.3	1.51
TDAR-30B	4	1	70.3	6982.4	1.45

Table 4: Ablation on TCCF paradigm with different block sizes, averaged over six benchmarks.

7 Related Work

7.1 Test Time Scaling and Efficient Reasoning

The success of OpenAI’s o1 popularized test-time compute scaling, which improves reasoning performance by scaling up inference-time computation (OpenAI et al., 2024). However, this paradigm often leads to increasingly lengthy reasoning trajectories and “overthinking” behaviors (Chen et al., 2025). Existing solutions primarily focus on controlling generation length via length-controlled policy optimization (Aggarwal and Welleck, 2025) or adaptive fine-tuning based on task complexity (Hao et al., 2024; Liu et al., 2025; Fang et al., 2025; Arora and Zanette, 2025; Zhang et al., 2025). Recent work has further revealed heterogeneous reasoning dynamics from a model-internal perspective (Tan et al., 2025). In contrast, we explore leveraging Block Diffusion Language Models (BDLMs) to achieve parallel decoding and flexible sequence manipulation, offering a hardware-friendly pathway to accelerate test-time scaling.

7.2 Diffusion Language Models

Diffusion Language Models (DLMs) (Nie et al., 2025; Ye et al., 2025b; Austin et al., 2023; Sahoo et al., 2024) present a promising alternative to sequential generation by enabling parallel multi-token predictions. To support efficient Key-Value (KV) caching in bidirectional setups, BDLMs (Arriola et al., 2025) employ a blockwise generation structure. Recent works adapt pre-trained language models to diffusion objectives (Gong et al., 2025a; Ye et al., 2025b; Wu et al., 2025a; Cheng et al., 2025) and incorporate reinforcement learning to enhance reasoning capabilities (Wang et al., 2025; Zhu et al., 2025c). However, BDLMs suffer from an inherent trade-off between parallelizability and

generation quality. While several advanced dynamic samplers have been developed for masked diffusion models (Ben-Hamu et al., 2025; Hong et al., 2025; Dong et al., 2025), existing block diffusion baselines typically rely on static step budgets or constant confidence thresholds (Cheng et al., 2025; Bie et al., 2025).

8 Conclusion

In this paper, we propose a unified test-time compute allocation framework for BDLMs that adaptively balances reasoning quality and decoding efficiency. By introducing Bounded Adaptive Confidence Decoding (BACD) and the Think Coarse, Critic Fine (TCCF) paradigm, our approach dynamically adjusts step-level confidence bounds and block-level reasoning stages. Extensive evaluations across six benchmarks demonstrate that our framework significantly accelerates inference while improving reasoning accuracy. Overall, this work advances the efficiency-effectiveness Pareto frontier for block diffusion language models and lays a solid foundation for future research.

Limitations

We observe different large-block adaptation behaviors across model scales: the 8B model preserves reasoning quality up to $B = 16$, whereas the 30B model reaches its Accuracy–NFE knee at $B = 4$. One likely contributor is our fixed adaptation budget: due to training-resource constraints, the 8B and 30B variants use the same 50B-token CPT corpus and 3B-token long-CoT SFT budget, resulting in substantially fewer adaptation tokens per parameter for the larger model. Consequently, our current experiments cannot fully disentangle intrinsic scale sensitivity from insufficient large-block adaptation. Data-scaled adaptation for larger BDLMs is therefore an important direction for future work.

References

- Pranjal Aggarwal and Sean Welleck. 2025. [L1: Controlling how long a reasoning model thinks with reinforcement learning](#). *Preprint*, arXiv:2503.04697.
- AMC. 2023. [American mathematics competition - amc](#). In *American Mathematics Competition - AMC*.
- Daman Arora and Andrea Zanette. 2025. [Training language models to reason efficiently](#). *Preprint*, arXiv:2502.04463.
- Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. [Block diffusion: Interpolating between autoregressive and diffusion language models](#). *arXiv preprint arXiv:2503.09573*.
- Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2023. [Structured denoising diffusion models in discrete state-spaces](#). *Preprint*, arXiv:2107.03006.
- Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. 2025. [Accelerated sampling from masked diffusion models via entropy bounded unmasking](#). *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*.
- Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuqi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. [Do not think that much for 2+3=? on the overthinking of o1-like llms](#). *Preprint*, arXiv:2412.21187.
- Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and 1 others. 2025. [Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation](#). *arXiv preprint arXiv:2510.06303*.
- Yihong Dong, Zhaoyu Ma, Xue Jiang, Zhiyuan Fan, Jiaru Qian, Yongmin Li, Jianha Xiao, Zhi Jin, Rongyu Cao, Binhua Li, and 1 others. 2025. [Saber: An efficient sampling with adaptive acceleration and backtracking enhanced remasking for diffusion language model](#). *arXiv preprint arXiv:2510.18165*.
- Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. [Thinkless: Llm learns when to think](#). *Preprint*, arXiv:2505.13379.
- Gemini. 2025. [Gemini diffusion, our state-of-the-art, experimental text diffusion model](#).
- 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](#). *Preprint*, arXiv:2410.17891.
- Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatiao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. 2025b. [Diffucoder: Understanding and improving masked diffusion models for code generation](#). *Preprint*, arXiv:2506.20639.

- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. [Training large language models to reason in a continuous latent space](#). *Preprint*, arXiv:2412.06769.
- 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*.
- 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](#). *Preprint*, arXiv:2507.18578.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. [Live-codebench: Holistic and contamination free evaluation of large language models for code](#). *Preprint*, arXiv:2403.07974.
- Wei Liu, Ruochen Zhou, Yiyun Deng, Yuzhen Huang, Junteng Liu, Yuntian Deng, Yizhe Zhang, and Junxian He. 2025. [Learn to reason efficiently with adaptive length-based reward shaping](#). *Preprint*, arXiv:2505.15612.
- MAA. 2024. [American invitational mathematics examination-aim 2024](#).
- MAA. 2025. [American invitational mathematics examination-aim 2025](#).
- Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. 2024. Scaling up masked diffusion models on text. *arXiv preprint arXiv:2410.18514*.
- 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. In *ICLR 2025 Workshop on Deep Generative Model in Machine Learning: Theory, Principle and Efficacy*.
- OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. [Openai o1 system card](#). *Preprint*, arXiv:2412.16720.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dierani, Julian Michael, and Samuel R. Bowman. 2023. [Gpqa: A graduate-level google-proof qa benchmark](#). *Preprint*, arXiv:2311.12022.
- Subham Sekhar 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](#). *Preprint*, arXiv:2406.07524.
- Chenyang Shao, Sijian Ren, Fengli Xu, and Yong Li. 2025. Diffuse thinking: Exploring diffusion language models as efficient thought proposers for reasoning. *arXiv preprint arXiv:2510.27469*.
- Yuqiao Tan, Minzheng Wang, Shizhu He, Huanxuan Liao, Chengfeng Zhao, Qiunan Lu, Tian Liang, Jun Zhao, and Kang Liu. 2025. Bottom-up policy optimization: Your language model policy secretly contains internal policies. *arXiv preprint arXiv:2512.19673*.
- Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. 2025. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv:2509.06949*.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. 2025a. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328*.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. 2025b. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*.
- 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](#). *Preprint*, arXiv:2410.14157.
- 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 preprint arXiv:2508.15487*.
- Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025. [Adaptthink: Reasoning models can learn when to think](#). *Preprint*, arXiv:2505.13417.
- Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025a. [Llada 1.5: Variance-reduced preference optimization for large language diffusion models](#). *Preprint*, arXiv:2505.19223.
- Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, Hongrui Guo, Jiaqi Hu, Wentao Ye, Tiejuan Chen, Chenchen Li, Chengfu Tang, Haibo Feng, Jun Hu, Jun Zhou, and 7 others. 2025b. [Llada-moe: A sparse moe diffusion language model](#). *Preprint*, arXiv:2509.24389.
- Ying Zhu, Jiaxin Wan, Xiaoran Liu, Siyanag He, Qiqi Wang, Xu Guo, Tianyi Liang, Zengfeng Huang, Ziwei He, and Xipeng Qiu. 2025c. Dirl: An efficient post-training framework for diffusion language models. *arXiv preprint arXiv:2512.22234*.

A Detailed Efficiency and Runtime Analysis

A.1 Token Length and Forward-Evaluation Statistics

Table 5 reports the average number of function evaluations (NFE) and response token length (Tok) across six reasoning benchmarks. NFE characterizes the algorithmic cost of generation, while Tok reflects the length of model outputs. Compared with BACD alone, TCCF moderately increases NFE (e.g., from 3,153 to 3,684 for TDAR-8B) by allocating additional computation to the fine-grained critic stage. This additional computation improves the average accuracy from 45.8% to 49.5%, illustrating the accuracy–efficiency trade-off introduced by stage-wise test-time compute allocation.

A.2 Practical Runtime and Matched Autoregressive Comparison

In addition to the algorithmic efficiency metrics above, we measure the aggregate GPU time required to complete the full six-task evaluation. We also construct a matched autoregressive baseline initialized from the same Qwen3-8B base model and trained with the same long-CoT data, optimization steps, and evaluation protocol as TDAR. This controlled setup isolates the effect of the model architecture from differences in training data or optimization budget.

Table 6 shows that the algorithmic efficiency gains generally translate to measured GPU time. On TDAR-8B, BACD reduces aggregate GPU time from 66.82 to 63.02 hours (5.7%) while slightly improving average accuracy; on TDAR-30B, it reduces GPU time from 97.36 to 92.27 hours (5.2%). TCCF instead allocates additional computation to fine-grained verification: relative to the default decoding setting, it improves average accuracy from 45.5% to 49.5% on TDAR-8B with 5.8% additional GPU time, and from 68.5% to 70.1% on TDAR-30B with 2.3% additional GPU time. The matched AR model achieves 57.9% average accuracy, 8.4 percentage points higher than TDAR-8B + BACD + TCCF, but requires 10,434 forward evaluations and 137.87 GPU hours. This corresponds to $2.83\times$ as many forward evaluations and $1.95\times$ as much aggregate GPU time; equivalently, TDAR-8B + BACD + TCCF uses 48.7% less GPU time.

B Implementation Details

B.1 Model Training Configuration

We transform Qwen3-8B-base into Block DLMs following the methodology of Cheng et al. (2025). Initially, we modify the generation mechanism through blockwise diffusion objective training ($B = 4$) on a 50B-token annealing corpus subset, yielding TDAR-8B-Base ($B = 4$).

Next, we conduct Supervised Fine-Tuning (SFT) on 3B long Chain-of-Thought (CoT) datasets employing our progressive block-size expansion approach. Starting from block size 4, we gradually increase the training block size to 64 while maintaining consistent training configurations. Each stage undergoes 3 training epochs. Detailed training specifications are presented in Table 7.

B.2 Inference Configuration

We utilize SGLang² as our inference engine, which provides efficient support for both autoregressive and block diffusion decoding. All experiments are conducted on NVIDIA H200 GPUs.

We configure the maximum generation length to 30,000 tokens for SDAR-8B-Chat, DiRL-8B-Instruct, all TraDo-8B variants and our TDAR-8B-thinking model, which is sufficient for long Chain-of-Thought reasoning trajectories. We use a temperature of 1.0 to encourage diverse responses, while top-p and top-k sampling are not employed in our main experiments.

C Detailed Usage of TCCF Paradigm

The Think Coarse, Critic Fine (TCCF) paradigm implements a two-stage reasoning process that adaptively adjusts block sizes according to the functional roles of different reasoning segments. The paradigm exploits the observation that early-stage exploratory reasoning can tolerate coarse-grained generation, while later-stage refinement and verification require fine-grained precision. Algorithm 2 presents the complete TCCF inference procedure.

Stage 1: Coarse Thinking ($B_{think} = 16$): In the first stage, the model performs exploratory reasoning with a large block size to efficiently generate the initial reasoning trajectory. The model processes the input prompt p and generates reasoning content until it reaches the designated thinking boundary marker `</think>`. During this stage,

²<https://github.com/sgl-project/sglang>

Method	Math500		AIME24		AIME25		AMC23		LCB		GPQA		AVG	
	NFE	Tok	NFE	Tok	NFE	Tok	NFE	Tok	NFE	Tok	NFE	Tok	NFE	Tok
<i>Autoregressive Language Model</i>														
Qwen3-8B-Thinking	-	3395	-	13950	-	14223	-	6448	-	13056	-	11533	-	10434
<i>Block Diffusion Language Model</i>														
SDAR-8B-Chat	252	652	766	2303	454	1058	414	1088	142	328	467	924	416	1059
DiRL-8B-Instruct	1110	2322	3650	6867	3397	6187	2167	4313	382	764	1949	3427	2109	3980
TraDo-8B-Instruct	261	625	551	1190	513	1033	408	907	122	211	504	724	393	782
TraDo-8B-Thinking	3379	4787	11546	15216	11318	14951	6749	9191	10863	13244	8804	11429	8777	11470
SDAR-30B-Thinking	3557	5910	8646	14069	9440	15117	5324	8641	10100	13217	7624	10424	7449	11230
TDAR-8B-Thinking (ours)	1403	5590	4142	15589	4469	15648	2355	9424	5212	13937	2910	13510	3415	12283
+ BACD	1317	6065	3842	15578	4123	15649	2175	9632	4758	14093	2701	13457	3153	12412
+ BACD + TCCF	1490	5466	4458	17108	4768	16965	2483	9582	5490	15113	3419	14585	3684	13136
TDAR-30B-Thinking (ours)	1894	2933	7972	11983	8505	13086	3370	5268	9810	13091	6858	10035	6402	9399
+ BACD	1805	2947	7469	11638	8326	13364	3201	5134	9371	12958	6597	9983	6128	9337
+ BACD + TCCF	2002	3254	8193	12840	8810	14116	3357	5437	10174	14171	6943	10535	6580	10059

Table 5: Average number of function evaluations (NFE) and response token length (Tok) across six reasoning benchmarks.

Model	Setting	Acc. (%)	NFE	GPU Hours
TDAR-8B	Default Dynamic Confidence	45.5	3,415	66.82
TDAR-8B	+ BACD	45.8	3,153	63.02
TDAR-8B	+ BACD + TCCF	49.5	3,684	70.71
TDAR-30B	Default Dynamic Confidence	68.5	6,402	97.36
TDAR-30B	+ BACD	68.8	6,128	92.27
TDAR-30B	+ BACD + TCCF	70.1	6,580	99.58
Matched Qwen3-8B AR	Autoregressive	57.9	10,434	137.87

Table 6: Aggregate GPU time under the same evaluation workload, together with a matched autoregressive baseline.

Config	CPT	SFT
Tokens	50B	3B
Epochs	1	3
Learning Rate	1e-4	3e-5
Batch Size	1024	128

Table 7: Training configuration details for each stage.

the model rapidly explores the solution space, proposes potential approaches, and conducts preliminary analysis. The large block size ($B_{think} = 16$) enables efficient parallel decoding, significantly accelerating the exploration process.

Transition Mechanism: Once the model completes the thinking stage, the system automatically replaces this marker with a transition prompt: “Let’s check if there are any mistakes and give the final answer.” This prompt serves as an explicit instruction to shift the model’s focus from exploration to verification and refinement.

Stage 2: Fine Critic ($B_{critic} = 4$): In the second stage, the model switches to a smaller block size ($B_{critic} = 4$) to perform careful verification

and refinement. The model reviews the reasoning generated in Stage 1, identifies potential errors, corrects mistakes, and produces the final answer with high reliability. The fine-grained decoding ensures precision in the critical refinement phase while maintaining computational efficiency overall, since Stage 2 typically generates significantly fewer tokens than Stage 1.

Prompt 1: Transition prompt used in TCCF

Let’s check if there are any mistakes and give the final answer.

Figure 5: Transition prompt used in TCCF paradigm to shift from coarse thinking ($B_{think} = 16$) to fine critic stage ($B_{critic} = 4$). This prompt is automatically inserted at the position where $\langle /think \rangle$ marker appears in the Stage 1 output.

Discussion on marker for boundary detection: The TCCF paradigm is highly flexible and generalizable. Depending on the task requirements or model characteristics, the transition mechanism can be adapted using alternative prompt formulations

Alg. 2: Think Coarse, Critic Fine (TCCF) Inference

Require: Input prompt p , model θ , thinking block size B_{think} , critic block size B_{critic} , transition prompt p_{trans}
Ensure: Final response y

- 1: *// Stage 1: Coarse Thinking*
- 2: Initialize reasoning trajectory $r \leftarrow$ empty string
- 3: Set current block size $B \leftarrow B_{think}$
- 4: **while** r does not contain `</think>` **do**
- 5: Generate next block: $r_{new} \sim p_{\theta}(\cdot | p, r; B)$
- 6: Append to trajectory: $r \leftarrow r + r_{new}$
- 7: **end while**
- 8: *// Transition: Replace marker with refinement prompt*
- 9: $r \leftarrow \text{replace}(\text{</think> in } r \text{ with } p_{trans})$
- 10: *// Stage 2: Fine Critic*
- 11: Initialize final response $y \leftarrow r$
- 12: Set current block size $B \leftarrow B_{critic}$
- 13: **while** not end-of-sequence **do**
- 14: Generate next block: $y_{new} \sim p_{\theta}(\cdot | p, y; B)$
- 15: Append to response: $y \leftarrow y + y_{new}$
- 16: **if** y contains end-of-sequence token **then**
- 17: **break**
- 18: **end if**
- 19: **end while**
- 20: **return** y

(e.g., “Review the steps above” or “Summarize the solution”), or by employing different structural markers. Our implementation provides a foundational framework for test-time scaling in Block DLMs, offering a broad design space for future exploration in efficient and reliable long-context reasoning.

D Detailed Decoding Configurations and Algorithms

In this section, we provide formal formulations and hyperparameter configurations for the mainstream DLM decoding algorithms evaluated in Section 5.1 and Table 2.

D.1 Static Decoding

Static decoding methods utilize a fixed step-budget schedule that is independent of token prediction confidence.

- **StaticConfidence:** In each denoising step within a block of size B , a fixed number of tokens $n = \lceil B/N \rceil$ are unmasked, where N is the total pre-allocated denoising steps. The tokens with the highest confidence (Equation 5) are selected. Under our evaluation setup, we set $N = 4$ for a block size $B = 16$.
- **StaticEntropy:** Similar to StaticConfidence, this method unmaskes a fixed n tokens per step but uses the negative entropy of the pre-

dicted token distribution as the selection metric. The entropy at position i is defined as $H_i = -\sum_{v \in \mathcal{V}} p_i(v) \log p_i(v)$, and the n positions with the lowest entropy are unmasked at each step.

D.2 Dynamic Confidence Decoding

Dynamic Confidence Decoding (Wu et al., 2025b) serves as our default baseline and adapts the number of generated tokens dynamically. At each step t , the candidate set of tokens to be unmasked is defined as:

$$\mathcal{D}_t = \{i \in \mathcal{M}_t^k \mid c_i(\mathbf{x}_t^k) > \tau\} \quad (11)$$

where τ is a constant threshold. If $\mathcal{D}_t = \emptyset$, the algorithm falls back to unmasking the single token with the highest confidence score to guarantee convergence. In our experiment, we set the default threshold $\tau = 0.9$ following official implementations.

D.3 EB-Sampler

Introduced by Ben-Hamu et al. (2025), the **EB-Sampler** is an adaptive multi-token unmasking strategy designed for masked diffusion models. Its central insight is that a partially unmasked sequence often deterministically determines multiple masked tokens, allowing aggressive parallel unmasking without sacrificing generation quality.

At each step, EB-Sampler dynamically estimates the cumulative error of predicting multiple tokens and unmaskes them simultaneously, provided that the total error proxy remains below a predefined budget γ . We evaluate three distinct error proxy variants of EB-Sampler:

- **Confidence Proxy:** Measures error using the predicted probability of the argmax token, ensuring that the joint confidence approximation remains bounded.
- **Entropy Proxy:** Uses the predicted entropy at each position to gauge the information gain and uncertainty.
- **Margin Proxy:** Gauges prediction certainty by calculating the difference between the top-1 and top-2 predicted token probabilities: $\text{Margin}_i = p_i(v^{(1)}) - p_i(v^{(2)})$.

We configure the error budget to $\gamma = 0.5$ for all three variants in our benchmarks.

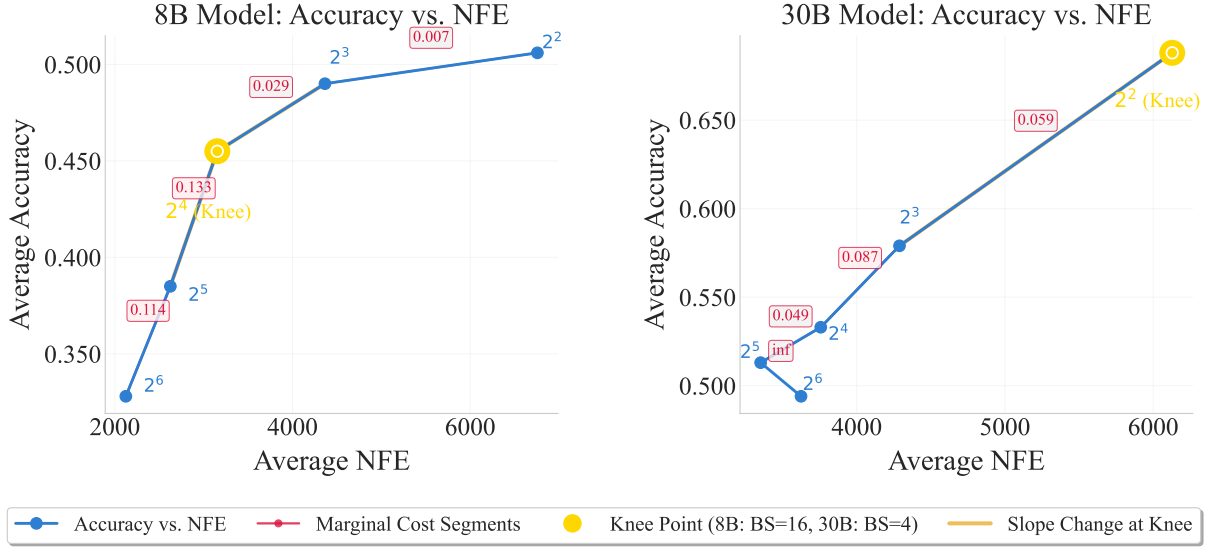


Figure 6: Accuracy vs. NFE trade-off for different block sizes. Numbers on each segment denote the marginal cost (accuracy gain per 1,000 additional NFE). The golden circle marks the knee point where the marginal return drops sharply.

D.4 Saber

The Saber algorithm, proposed by Dong et al. (2025), addresses the brittleness of DLMs in structured generation tasks (e.g., code reasoning) by introducing a training-free backtracking-enhanced remasking solver. Saber implements two core mechanisms:

1. **Adaptive Acceleration:** The solver dynamically adjusts the number of unmasked tokens at each step according to the model’s evolving confidence distribution.
2. **Backtracking Enhanced Remasking:** As generation proceeds and new context is established, Saber re-evaluates the confidence of previously unmasked tokens. If a token’s confidence drops below a dynamic backtracking threshold, Saber applies a re-masking operation (reverting the token back to [MASK]), thereby pruning compounding errors in early generation steps.

For our evaluation on BDLMs, we configure the momentum-based early termination coefficient $\mu = 2$.

E Block Size Selection via Knee Point Analysis

We determine the optimal block size by identifying the *knee point* of the Accuracy–NFE curve, i.e., the point beyond which further NFE investment yields diminishing accuracy gains.

8B Model. As shown in Figure 6 (left), the marginal cost drops by 4.6× from the segment $B=32 \rightarrow 16$ (0.133) to $B=16 \rightarrow 8$ (0.029), indicating that $B=16$ is the knee point where additional NFE provides substantially less accuracy improvement.

30B Model. For the 30B model (Figure 6, right), the marginal cost peaks at $B=16 \rightarrow 8$ (0.087) and declines to 0.059 at $B=8 \rightarrow 4$, identifying $B=4$ as the knee point. We therefore adopt $B=16$ for the 8B model and $B=4$ for the 30B model throughout our experiments.

F Case Study

Figure 7 presents a detailed case study demonstrating why the TCCF paradigm is crucial for complex reasoning tasks. The problem asks for the least possible sum of distinct positive integers with product 84, which requires searching through multiple factor combinations.

Stage 1: Efficient Exploration via Coarse Thinking ($B = 16$). In this phase, the model acts as a rapid explorer. By using a large block size, it can quickly traverse the reasoning path and identify high-probability solutions. As shown in the case, the model correctly identifies the factorization of 84 and proposes several valid sets, arriving at a suboptimal answer of 15. Although the coarse granularity ($B = 16$) accelerates the generation of the reasoning chain, it inherently carries a trade-off in precision, leading the model to overlook the opti-

mal combination $\{3, 4, 7\}$ in the vast search space. This stage provides the necessary *breadth* of reasoning without consuming excessive computational time.

Stage 2: Precision Correction via Fine Critic ($B = 4$). The transition prompt (“Let’s check...”) triggers a mode switch, acting as a verifier. By switching to the finest granularity ($B = 4$), the model allocates its full computational capacity to scrutinize the previous conclusion. In this high-precision mode, the model successfully detects the oversight from the coarse stage and discovers the subtle combination $\{3, 4, 7\}$ (sum=14).

Effectiveness Analysis: This case illustrates the core advantage of TCCF. It decouples reasoning generation from answer verification. Pure coarse decoding ($B = 16$) is fast but prone to detail errors (finding 15 instead of 14), while pure fine decoding ($B = 4$) is accurate but computationally expensive. TCCF using the first stage to efficiently narrow down the solution space and the second stage to ensure the final answer is precise. This coarse-to-fine mechanism allows the model to correct its own hallucinations or oversights, directly contributing to the accuracy improvements observed in our benchmarks.

Prompt 2: Model Output

Problem: The product of a set of distinct positive integers is 84. What is the least possible sum of these integers?

Stage1 Output:

<think>

We need to find a set of distinct positive integers (i.e., the integers are all different) whose product is 84, and we want to minimize the sum of the integers.

First, note that factorization of $84 = 2^2 * 3 * 7$...

Potential sets:

- Single integer: 84 sum = 84.
- Two numbers: distinct positive integers a, b , $a*b=84$... minimal sum among two numbers without 1 is 19 ($7+12$).
- Three numbers: $a*b*c=84$, distinct >1 ... We found such: $2,3,14$ sum=19; $2,4,21$ sum=27; $2,6,14$ sum=22; $2,7,12$ sum=21; $3,4,14$ sum=21. So minimal sum among three numbers is 19.
- Four numbers: ... we could have $4,3,7,1$ product = $4*3*7*1 = 84$... Sum = $4+3+7+1 = 15$.

Thus the minimal sum among sets of size 4 is 15.

Now we need to produce final answer.

Let's check if there are any mistakes and give the final answer.

Stage2 Output:

Let me pause and think about this more carefully... But we need to also consider if there is any other set with sum 14? Let's see if there's any set of size >1 with sum 14? The set $1,3,4,7$ sum = 15. No, sum 14 would require $1,3,4,6$? Product = 72 not 84...

But I should also consider whether the problem intends the set to contain at least two integers... However, the set with sum 14 includes 3 numbers, all distinct >0 , product 84. So if we allow any cardinality ≥ 2 , the minimum sum is 14. But we need to verify if there is a set of size 2 with sum less than 14. The minimum sum among pairs not including 1 is 19. So no. So if cardinality ≥ 2 , the answer is 14. But we have a set of size 3 and sum 14. So answer would be 14, not 15.

But we need to see what the problem expects... Actually $14 < 15$. But we found a set of three numbers with sum 14 ($3+4+7$). So sum 14 is less. So the minimal sum must be ≤ 14 . But we found a set $3,4,7$ sum = 14, product = 84. So sum 14 is possible. Wait we need to double-check: Are there any sets of size ≥ 4 with sum <14 ? Possibly $1,3,4,7$ sum = 15 which is higher than 14. So minimal sum is 14...

But earlier we thought of including 1 to reduce sum further. The minimal sum is 14 from $3,4,7$. So answer is 14.

Thus final answer: 14. Let's produce final answer accordingly.

</think>

****Final Answer:****

The prime factorisation of $84 = 2^2 * 3 * 7$. Considering all possible sets of distinct positive integers with product 84:

- One integer: $84 \rightarrow$ sum = 84
- Two integers: minimal sum = $7+12 = 19$
- Three integers: minimal sum = $3+4+7 = 14$ (set $3,4,7$)
- Four integers: must include 1, minimal sum = $1+3+4+7 = 15$
- Five or more integers: impossible

The smallest achievable sum is 14.

Figure 7: Case study of a representative problem from the Math500 benchmark.