


# Improving the Throughput of Diffusion-based Large Language Models via a Training-Free Confidence-Aware Calibration

Jucheng Shen<sup>1,\*</sup> Gaurav Sarkar<sup>2,†</sup> Yeonju Ro<sup>3,\*</sup> Sharath Nittur Sridhar<sup>2</sup>  
Zhangyang Wang<sup>3</sup> Aditya Akella<sup>3</sup> Souvik Kundu<sup>2,\*</sup>

<sup>1</sup>Rice University <sup>2</sup>Intel <sup>3</sup>The University of Texas at Austin

jucheng.shen@rice.edu, yro@cs.utexas.edu, souvikk.kundu@intel.com

## Abstract

We present **CadLLM**, a *training-free* method to accelerate the inference throughput of diffusion-based LLMs (dLLMs). We first investigate on the dynamic nature of the token unmasking confidence across blocks and steps. Based on this observation, we then present a lightweight adaptive approach that can control the generation *block size*, *step size*, and *threshold* based on the average confidence score of the unmasked tokens. We further reduce the softmaxing overhead of token probability generation by dynamically leveraging a subset of *vocabulary size* to regulate sampling breadth. CadLLM is a plug-and-play model-agnostic with KV caching based dLLMs. Extensive experiments on four popular tasks demonstrate the efficacy of CadLLM to yield throughput improvements ranging from 1.1–2.28× over the state-of-the-art baseline with competitive accuracy.  [Code](#)

## 1 Introduction

Masked diffusion language models (dLLMs) have advanced rapidly, with open-sourced models such as LLaDA (Nie et al., 2025) and DREAM (Ye et al., 2025a) demonstrating strong generative capabilities (Austin et al., 2021; Campbell et al., 2022). However, unlike autoregressive language models that generate tokens sequentially in a single forward pass, diffusion-based models rely on a multi-step denoising Markov process that iteratively refines noisy latent states to generate clean text. This stochastic, multi-round refinement incurs substantial computational overhead during inference (Ye et al., 2025a; Nie et al., 2025), often causing significant slowdown in token generation.

To improve inference throughput, recently fast-dLLM (Wu et al., 2025) proposed parallel decoding that unmask all positions above a *static* global

confidence<sup>1</sup> threshold in parallel. While effective, this static thresholding has few key limitations. Firstly, fast-dLLM uses fixed block sizes with fixed step-size per block, essentially ignoring the change in confidence over the sequence. Secondly, it treats each block uniformly with a fixed sampling breadth (Fan et al., 2018; Holtzman et al., 2018), ignoring potential certainty differences. Thirdly, it employs a fixed token commit threshold, which also doesn't account for the variation in confidence across inference steps.

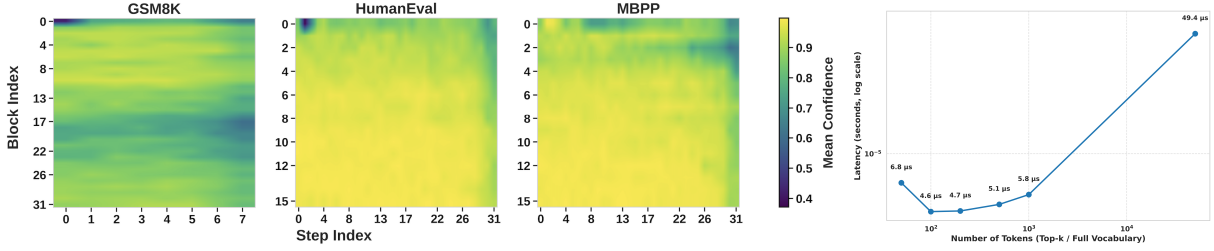
**Our Contributions.** To address the above limitations, in this paper we present **CadLLM**, a *training-free adaptive method* that uses lightweight confidence signals to allocate compute where uncertainty persists and save it where predictions stabilize. Specifically, we first analyze the difference in confidence across different blocks and denoising steps. Interestingly, we find that the confidence varies significantly across these dimensions (refer to Section §3). To more optimally balance the compute for token denoising in dLLM, we then present CadLLM. Based on the average confidence of the unmasked tokens in a block, CadLLM dynamically allocates the *block size*, *step size* and unmasking *threshold* during the inference steps. To further reduce the token prediction softmax operation cost we present an *adaptive vocabulary subset* selection based on the prediction confidence.

Extensive experiments across multiple datasets at different generation lengths ( $g$ ) demonstrate the efficacy of CadLLM with throughput gains ranging from **1.1–2.28×** over the state-of-the-art fast dLLM baseline (Wu et al., 2025) on LLaDA, and **1.1–1.4×** on DREAM, with competitive accuracy.

<sup>1</sup>In this work we define confidence score as the model's probability output score for its predicted tokens at each masked position.

\*Corresponding authors.

<sup>†</sup>Work done when the author was affiliated with Intel.



(a) Per-block, per-step confidence heatmaps. Lighter the color the higher the confidence. (b) Softmax latency.  
 Figure 1: (a) Confidence dynamics for three difference datasets. (b) Latency vs. vocabulary size.

## 2 Preliminaries

Generation in masked diffusion language model (MDM) is modeled by a network  $p_\theta$  that predicts clean tokens for masked positions conditioned on the partially noised context and (optionally) the time  $t$  (Austin et al., 2021). At each step, given the current evidence  $E = (x_t, t)$ , the model computes  $p_\theta(X^i = v | E)$  for each position  $i$  and selects tokens with the highest confidence  $c^i(E) = \max_{v \in \mathcal{V}} p_\theta(X^i = v | E)$  to unmask, iteratively updating the sequence until reaching  $t = 0$  or a stopping condition. Fast-dLLM (Wu et al., 2025) accelerates MDM generation by decoding masked tokens in parallel according to confidence signals. It organizes the output into  $K$  blocks of size  $B$  and performs at most  $S$  refinement steps per block via MDM mechanism.

**Threshold-Based Rule.** For a working sequence  $x$ , it compute confidences  $\{c^i(E)\}_{i \in \mathcal{M}(x)}$  and unmask all positions whose confidence exceeds a fixed global threshold  $\tau \in (0, 1)$  (Wu et al., 2025):

$$\mathcal{U}_\tau(E) = \{i \in \mathcal{M}(x) : c^i(E) \geq \tau\} \quad (1)$$

**Factor-Based Rule.** Factor-based rule controls the degree of parallelism via a *factor* parameter  $\phi > 0$ . For a sorted confidence order  $c^{(1)} \geq \dots \geq c^{(m)}$  for the  $m$  masked positions in the current block and choose the largest  $r$  satisfying:

$$(r + 1) \left(1 - c^{(r)}\right) < \phi,$$

It then unmask the top- $r$  positions. Intuitively, this decodes more tokens when individual confidences are uniformly high and fewer when they are borderline.

## 3 Motivational Case Study

**Confidence is Not Uniform.** Figure 1a shows per-block, per-step confidence for three datasets. Within each block, confidence rises quickly then plateaus; across blocks, difficulty varies—some stabilize immediately, others remain uncertain. This

relationship between confidence and decoding difficulty is independently corroborated by concurrent work (Lu et al., 2026), which shows that fixed block sizes force the model to commit low-confidence tokens prematurely, leading to higher decoding error. A fixed schedule thus over-refines easy blocks while under-serving harder ones. When confidence is high, large block size can exploit parallel decoding since more tokens exceed the threshold; when low, smaller blocks help stabilize outputs faster. Similarly, the step size should be large under low confidence and small once predictions are stable. Finally, the confidence threshold itself must adapt—should be high at situations to prevent premature commitments, and should be low at situations to accelerate decoding.

**Sampling Breadth has Latency Overhead.** Figure 1b evaluates the softmax latency for different vocabulary sizes. It shows that the latency grows sharply with vocabulary size: evaluating all  $\sim 50K$  tokens is nearly an order of magnitude slower than using a small subset. Since softmax is called at every step, this may become a major bottleneck. This necessitates the adaptive vocabulary size selection to perform the softmax computation on only a subset of total vocabulary.

## 4 CadLLM Methodology

As illustrated in Figure 2, we utilize diffusion LLM’s internal confidence for each token after each forward pass as the key signal to form a *feedback loop*. The confidence score serves as a single shared signal, and four policies form a closed-loop controller over four complementary resource dimensions: (1)  $B_t$ : how many tokens to decode in parallel (scope), (2)  $S_t$ : how many refinement steps to allocate (depth), (3)  $V_t$ : how broad the candidate set is (breadth), and (4)  $\tau_t$ : when to commit and move on (aggressiveness). Together, they jointly regulate the compute-per-token from a single signal, allocating resources where uncertainty persists and saving them where predictions have stabilized.

**Design Rationale.** All four policies use linear interpolation with clipping—a deliberate choice to isolate the benefit of *adaptivity itself* from any particular schedule form. Linear interpolation is the simplest monotonic mapping from a bounded signal (confidence  $\in [0, 1]$ ) to a bounded resource budget, introducing no additional inference-time computation beyond a single multiply-add per policy per block. Our key contribution is demonstrating that even this minimal adaptivity substantially outperforms static schedules, establishing a strong empirical lower bound for adaptive control in dLLMs.

#### 4.1 Adaptive block sizes ( $B_t$ )

Figure 1a shows that (i) confidence typically rises then plateaus within few refinements and (ii) unmasking difficulty varies across blocks. As demonstrated in the following Eq. 2, we set  $B_t$  proportional to the current average confidence  $\bar{c}$ : high-confidence blocks expand in size to amortize computing cost of forward passes, whereas low-confidence blocks shrink so the model can focus refinement where uncertainty concentrates.

$$B_t = \text{clip}\left(B_{\min} + (B_{\max} - B_{\min}) \cdot \bar{c}, B_{\min}, B_{\max}\right) \quad (2)$$

Here  $B_{\max}$  and  $B_{\min}$  represents the maximum and minimum allowable blocks sizes, respectively. The mean confidence  $\bar{c}$  is computed over a sliding window of the  $\Delta$  most recent per-step confidence values. We use a short window ( $\Delta=2$ ) for  $B_t$  and  $S_t$  so the controller reacts quickly to confidence shifts—critical because block and step decisions must adapt promptly to avoid premature commitment or wasted refinement. For  $V_t$ , a longer window ( $\Delta=5$ ) smooths transient fluctuations, since vocabulary breadth is less sensitive to instantaneous confidence changes.

#### 4.2 Adaptive steps ( $S_t$ )

We design the step size  $S_t$  to be complementarily related to  $\bar{c}$  and use it to pace the inner loop within the active block and also schedule the threshold  $\tau_t$ , as described in Eq. 3. This allows low-confidence mask locations to trigger more refinement steps, while high-confidence locations to have fewer steps.

$$S_t = \text{clip}\left(S_{\text{base}} + (S_{\text{max}} - S_{\text{base}})(1 - \bar{c}), S_{\text{base}}, S_{\text{max}}\right) \quad (3)$$

#### 4.3 Adaptive vocabulary size ( $V_t$ )

To amortize the cost of sampling from vocabulary, we adapt a sampling size  $V_t$  based on few key met-

ric: phase, confidence, and token repetition. In specific, we use a larger vocabulary subset at the early generation stage or during the phase of uncertainty in token generation. We also allow larger vocab size when repetitive tokens generate ( $r_t$ ). On the contrary we reduce its size for higher confidence situation to reduce compute cost. Eq. 4 defines the  $V_t$  allocation mechanism.

$$V_t = \text{clip}\left(V_{\text{phase}}(g_t) \cdot f_{\text{conf}}(\bar{c}) \cdot f_{\text{rep}}(r_t), V_{\min}, V_{\max}\right) \quad (4)$$

Here,  $V_{\text{phase}}$  widens early and narrows late based on the generation progress (% of tokens generated within a sequence)  $g_t$ ;  $f_{\text{conf}}$  expands under low confidence; and  $f_{\text{rep}}$  widens briefly under repetitions. In practice, the vocabulary subset is selected by taking the top- $V_t$  token indices ranked by raw logit magnitude (before softmax); the softmax is then computed only over these  $V_t$  logits, reducing cost from  $O(|\mathcal{V}|)$  to  $O(V_t)$  where  $V_t$  is typically 35–1,000.

**Same Token Repetition Detection.** A naive controller may over-narrow  $V_t$  once confidence is high, causing short loops (e.g., “the the the”). To avoid this, we introduce a lightweight *repetition detector* that scans recent outputs and returns  $r_t \in [0, 1]$ . When repetition spikes,  $f_{\text{rep}}(r_t) > 1$  briefly widens  $V_t$  to restore diversity, then reverts as  $r_t$  drops. This preserves fast decoding in stable regions while preventing local degeneracy.

#### 4.4 Adaptive threshold ( $\tau_t$ )

Adaptive confidence threshold *controls the token unmasking*. Early in a block or for low confidence, committing too many tokens in parallel by having a low threshold lead to poor quality output. Also, having a high threshold late in a block gates already stable tokens from being unmasked. As described in Eq. 5, we define a progress-aware threshold  $\tau_t$  that is initially strict (when confidence is low) and relaxes eventually (as confidence rises and remains high), as more and more tokens are unmasked and uses it to gate unmasking within the selected  $V_t$  candidates.

$$\tau_t = \tau_{\text{base}}(1 - g_t) + \tau_{\min}g_t \quad (5)$$

## 5 Experiments

### 5.1 Experimental Setup

All experiments are run on a single NVIDIA H100 GPU. We evaluate on GSM8K (5-shot), MATH (4-shot), MBPP (3-shot, pass@1), and HumanEval

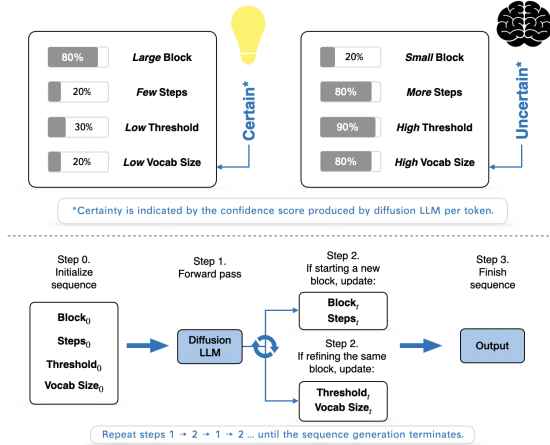


Figure 2: Overview of CadLLM’s adaptive controller. The controller dynamically updates various parameters based on a lightweight confidence and progress signals, replacing the static ones.

(0-shot, pass@1) with generation lengths  $g=256$  and  $g=512$ . We compare three decoding strategies: CadLLM (ours), and fast-dLLM with factor and threshold based approaches. Further hyperparameter details are provided in the Appendix.

## 5.2 Analysis of Main Results

Table 1 reports throughput (tokens/s), accuracy, and latency across benchmarks on LLaDA. CadLLM achieves 1.1–2.28× higher throughput than Fast-dLLM baselines while maintaining accuracy on par with the strongest models, including on MATH. We also evaluate on DREAM across all four benchmarks with consistent 1.1–1.4× gains (see Appendix Table 10). Adaptive control over blocks, steps, and vocabulary size reduces number of forward passes and compute cost without retraining. These gains persist at larger scales ( $g = 512$ ), demonstrating that CadLLM maintains both efficiency and accuracy across longer generations.

## 5.3 Ablations and Analysis

We conduct ablation studies to understand how each policy of CadLLM contribute to performance (Table 2). Additionally, we also investigate the effects of using different cache modes (Table 3), confidence calculation methods (Table 7), as well as enabling an optional early stopping via Prophet method that is proposed in recent literature (Table 8). We include the latter two in the Appendix. We also provide pairwise  $B_t \times S_t$  ablations confirming their complementary roles (Appendix Table 11) and a  $\pm 20\%$  hyperparameter sensitivity sweep demonstrating stable operating regions (Ap-

Benchmark	CadLLM (ours)	Fast-dLLM (factor)	Fast-dLLM (threshold)
GSM8K	78.01%	76.19%	79.00%
	120.07 (1.33×)	119.18 (1.32×)	90.40 (1.00×)
	75.44%	74.45%	75.28%
	107.79 (1.37×)	100.66 (1.27×)	78.77 (1.00×)
MATH	32.06%	32.22%	32.40%
	106.84 (1.34×)	109.97 (1.38×)	79.58 (1.00×)
	34.94%	35.40%	32.06%
	117.21 (1.14×)	111.86 (1.08×)	103.18 (1.00×)
MBPP	24.00%	21.20%	25.60%
	99.86 (1.37×)	96.01 (1.31×)	73.15 (1.00×)
	13.00%	13.20%	14.20%
	104.62 (1.35×)	100.73 (1.30×)	77.71 (1.00×)
HEval	35.97%	32.92%	37.19%
	220.81 (2.28×)	132.28 (1.37×)	96.84 (1.00×)
	43.29%	41.46%	45.12%
	163.72 (1.74×)	131.14 (1.38×)	94.41 (1.00×)

Table 1: Comprehensive results on LLaDA-Instruct (single NVIDIA H100). For each benchmark, the *upper row* is generation length 256 and the *lower row* is 512. Shot settings: GSM8K (5-shot), MATH (4-shot), MBPP (3-shot, pass@1), HumanEval (0-shot, pass@1). Each cell shows accuracy (top, in %) and throughput improvement (bottom: **tokens/s / improvement**). Improvement is relative to the Fast-dLLM threshold baseline at the same generation length.

pendix §7.4.7).

Mode	Token/s ↑	Time (s) ↓	Accuracy ↑	NFE ↓
ON	121.72	2501.10	78.01%	86,816
No $V_t$	119.67	2547.25	74.41%	86,666
No $S_t$	136.76	2235.68	76.12%	76,955
No $B_t$	111.19	2751.77	78.32%	92,344
No $\tau_t$	34.57	8841.89	78.17%	337,664
OFF	34.32	8908.60	78.01%	337,664

Table 2: GSM8K ablations at generation length 256. “ON” enables all four adaptive policies (adaptive block size, steps, vocabulary size, and threshold). Each subsequent row disables only the named policy. “OFF” disables all four adaptive policies.

**Adaptive Policy Ablations.** As shown in Table 2, disabling adaptive block size ( $B_t$ ) reduces throughput by 8.6%, while accuracy changes only slightly. This shows that block sizing mainly affects computation instead of accuracy. Larger blocks amortize computation cost when confidence is high, while smaller blocks localize refinement with some extra computation cost when confidence is low. Without adaptive steps ( $S_t$ ), throughput increases by 12.3% however, the accuracy drops by 1.9 percentage points. This indicates that steps alone balance inference speed and accuracy: fewer steps accelerate decoding with accuracy drop, and more steps preserve accuracy but incur higher computation cost.

Removing adaptive vocabulary size ( $V_t$ ) lowers throughput by 1.7% and sharply reduces accuracy by 4.6 percentage points, while NFE remains nearly unchanged. This confirms that adaptive vocabulary size acts as an exploration–exploitation lever: widening early or under uncertainty improves robustness, while narrowing later saves softmax cost without harming accuracy.

Eliminating the progress-aware threshold ( $\tau_t$ ) collapses efficiency. Throughput falls by 71.6%, latency increases by 254%, and NFE jumps by 289%, while accuracy remains nearly unchanged. This demonstrates that a dynamic commit gate is essential for practical runtime. The fully static **OFF** configuration shows nearly identical degradations, confirming that thresholding dominates the contribution to efficiency gains in the absence of other adaptive policies.

**Cache Modes (Prefix vs. Dual).** Additionally, as shown in Table 3, our method is *KV-cache agnostic*: it plugs into either prefix-only caching or dual (prefix+suffix) caching without any modification except for the few lines that replace the underlying cache methods.

Cache	Token/s $\uparrow$	Time (s) $\downarrow$	Accuracy $\uparrow$	NFE $\downarrow$
Prefix	145.20	2116.64	75.89%	82,677
	113.09	3155.55	76.72%	97,865
Dual	121.72	2501.10	78.01%	86,816
	107.79	3261.45	75.44%	104,215

Table 3: GSM8K (5-shot). CadLLM with prefix-only and dual KV caching. For each cache mode, the *upper row* is generation length 256 and the *lower row* is 512.

## 6 Conclusions

We present CadLLM, a *training-free* adaptive decoding method for masked dLLMs that replaces fixed schedules with four confidence-driven policies: adaptive block size, step size, vocabulary size (with repetition guard), and commitment threshold. These policies focus compute on uncertain regions and skip stable ones while remaining model-agnostic and KV-cache compatible. Evaluations on multiple datasets and two model architectures (LLaDA, DREAM) demonstrate that CadLLM retains accuracy similar to SoTA while improving throughput by 1.1–2.28 $\times$ .

## 7 Limitations

One limitation of CadLLM is that the accuracy and throughput improvements can potentially remain

sensitive to a few hyperparameters that may often require careful tuning. Automated search of the calibration hyperparameters is an interesting future research. Further evaluations on multimodal generative tasks with dLLMs remains another interesting exploration direction. Additionally, all four policies in CadLLM deliberately use linear interpolation with clipping as the schedule form, chosen to isolate the benefit of adaptivity itself (see Design Rationale in §4). While this minimal design already yields substantial gains, replacing it with learned policies (e.g., RL-based or Bayesian optimization over schedule forms) could further tighten the efficiency–quality trade-off.

## References

- Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. [Structured denoising diffusion models in discrete state-spaces](#). *Preprint*, arXiv:2107.03006.
- Andrew Campbell, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. 2022. [A continuous time framework for discrete denoising models](#). *Preprint*, arXiv:2205.14987.
- Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.
- Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. *arXiv preprint arXiv:1805.06087*.
- Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. 2025. [Diffusion language models know the answer before decoding](#). *Preprint*, arXiv:2508.19982.
- Guanxi Lu, Hao Mark Chen, Yuto Karashima, Zhican Wang, Daichi Fujiki, and Hongxiang Fan. 2026. [Adablock-dllm: Semantic-aware diffusion llm inference via adaptive block size](#). *Preprint*, arXiv:2509.26432.
- 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](#). *Preprint*, arXiv:2502.09992.
- 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](#). *Preprint*, arXiv:2505.22618.
- Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong.

2025a. *Dream 7b: Diffusion large language models. Preprint*, arXiv:2508.15487.

Jiacheng Ye, Zihui 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*.

## Appendix

### 7.1 Algorithm

Algorithm 1 describes the detailed working principle of adaptive thresholding based token generation for CadLLM.

---

#### Algorithm 1 CadLLM.<sup>2</sup>

---

**Require:** model  $p_\theta$ , prompt  $x_{\text{prompt}}$ , length  $g$   
**Require:**  $(B_{\min}, B_{\max}), (S_{\min}, S_{\max})$   
**Require:**  $(V_{\min}, V_{\max}), (\tau_0, \tau_{\min})$   
**Require:** window  $\Delta$

- 1:  $x \leftarrow [x_{\text{prompt}}; [\text{MASK}]^g]$
- 2:  $pos \leftarrow |x_{\text{prompt}}|$
- 3:  $remain \leftarrow g$
- 4:  $C \leftarrow []$
- 5: **while**  $remain > 0$  **do**
- 6:    $g_t \leftarrow (g - remain)/g$
- 7:    $\bar{c} \leftarrow \text{MEANCONF}(C, \Delta)$
- 8:    $f_{\text{conf}}(\bar{c}) \in \{1.5, 1.2, 0.8, 1.0\}$  by  $\bar{c}$
- 9:    $r_t \leftarrow \text{repeat-score}$
- 10:    $f_{\text{rep}}(r_t) \leftarrow 1 + 2r_t$  if  $r_t > 0.5$ , else 1
- 11:    $B_t \leftarrow \text{clip}(B_{\min} + (B_{\max} - B_{\min})\bar{c})$
- 12:    $S_t \leftarrow \text{clip}(S_{\min} + (S_{\max} - S_{\min})(1 - \bar{c}))$
- 13:    $V_t \leftarrow \text{clip}(V_{\text{phase}}(g_t) f_{\text{conf}}(\bar{c}) f_{\text{rep}}(r_t))$
- 14:    $\tau_t \leftarrow \tau_0(1 - g_t) + \tau_{\min}g_t$
- 15:    $M \leftarrow \text{mask in } [pos : pos + B_t]$
- 16:   **for**  $t' = 1$  **to**  $S_t$  **do**
- 17:      $\ell \leftarrow \text{LOGITS}(x)$
- 18:      $x_{\text{pred}} \leftarrow \arg \max(\ell)$
- 19:      $p[i] \leftarrow \text{conf. of } x_{\text{pred}}[i] \text{ in } V_t, i \in M$
- 20:      $\text{commit} \leftarrow \{i \in M : p[i] \geq \tau_t\}$
- 21:      $x[\text{commit}] \leftarrow x_{\text{pred}}[\text{commit}]$
- 22:     append  $\text{MEAN}(p[i] | i \in M)$  to  $C$
- 23:     **if** no  $[\text{MASK}]$  in  $M$  **then break**
- 24:     **end if**
- 25:   **end for**
- 26:    $pos \leftarrow pos + B_t$
- 27:    $remain \leftarrow remain - B_t$
- 28: **end while**
- 29: **return**  $x$

---

<sup>2</sup>For all initial variables and hyperparameters not specified here, see Table 4.

### 7.2 Related Works

**Fast-dLLM.** Fast-dLLM accelerates masked diffusion language models via confidence-aware parallel decoding with (i) a fixed-threshold rule that unmask all positions with  $c^i \geq \tau$  and (ii) a factor-based rule selecting the largest  $r$  with  $(r+1)(1 - c^{(r)}) < \phi$ , executed in fixed-size blocks with a preset number of refinement steps (Wu et al., 2025). Our **CadLLM** departs from this static design by turning these knobs into policies: *adaptive block size* (compute allocation to easy vs. hard blocks), *adaptive steps* (precision–speed dial), *adaptive vocabulary size* (confidence-aware search breadth), and *adaptive threshold* (phase-aware commit threshold). This training-free controller closes the loop between signals and compute, improving throughput while preserving accuracy.

**Early exiting in diffusion LMs.** Concurrent work observes early answer convergence to propose *Prophet*, that triggers an “all-in” decode using the top-2 confidence gap as a stop criterion (Li et al., 2025). Our focus is orthogonal: a training-free compute allocation (adaptive blocks, steps, vocabulary size, and threshold). To further demonstrate the adaptability of CadLLM we integrate the early-exit strategy of prophet as an option for early block-level generation. We leave Prophet disabled in main results because it overlaps with adaptive steps and reduces accuracy (see Appendix Sec. 7.4 for a detailed comparison).

### 7.3 Hyperparameter Settings

Parameter	Value
Generation length $g$	(256, 512)
Temperature $T$	0
Remasking	low confidence
Confidence method	softmax
Confidence window $(B_t, S_t)$	$\Delta_{BS} = 2$
Confidence window $(V_t)$	$\Delta_V = 5$
Initial block size $B_0$	24
Block size range $(B_{\min}, B_{\max})$	(4, 64)
Initial step $S_0$	24
Step range $(S_{\min}, S_{\max})$	(24, 90)
Initial vocabulary size $V_0$	100
Vocabulary size range $(V_{\min}, V_{\max})$	(35, 1000)
Initial threshold $\tau_0$	0.85
Threshold minimum $\tau_{\min}$	0.4
Early stopping via Prophet	False
Repetition detector	window $w = 8$
Repetition detector	min length = 2
Caching	Dual Cache

Table 4: Hyperparameters for CadLLM used in Table 1.

Parameter	Value
Generation length $g$	(256, 512)
Temperature $T$	0
Caching	DualCache
Selection rule	factor (f); threshold (t)
Factor / Threshold	$\phi = 1.0$ (f); $\tau = 0.9$ (t)
Confidence method	softmax

Table 5: Hyperparameters for Fast-dLLM baselines used in Table 1.

Parameter	Value
Generation length $g$	(256, 512)
Temperature $T$	0
Remasking	low confidence
Confidence method	softmax
Confidence window ( $B_t, S_t$ )	$\Delta_{BS} = 2$
Confidence window ( $V_t$ )	$\Delta_V = 5$
Initial block size $B_0$	24
Block size range ( $B_{\min}, B_{\max}$ )	(8, 48)
Initial step $S_0$	24
Step range ( $S_{\min}, S_{\max}$ )	(24, 70)
Initial vocabulary size $V_0$	100
Vocabulary size range ( $V_{\min}, V_{\max}$ )	(35, 1000)
Initial threshold $\tau_0$	0.85
Threshold minimum $\tau_{\min}$	0.4
Early stopping via Prophet	False
Repetition detector	window $w = 8$
Repetition detector	min length = 2
Caching	Dual Cache

Table 6: Hyperparameter choices for CadLLM on DREAM.

Tables 4 and 5 detail the hyperparameter settings for ours and the baseline Fast-dLLM, respectively. For both cases, we used the LLaDA-8B-Instruct model. Table 6 report the hyperparameter settings of our method used on DREAM-7B-Base model.

## 7.4 Additional Experimental Analysis

### 7.4.1 Confidence Method Choice

We compare two confidence computations for our controller: maximum softmax probability (“Softmax”) and entropy-based confidence (“Entropy”). Results on GSM8K at  $g=256$  (single H100) show that softmax is both faster and slightly more accurate (Table 7), with substantially fewer function evaluations.

Mode	Token/s $\uparrow$	Time(s) $\downarrow$	Accuracy $\uparrow$	NFE $\downarrow$
Softmax	121.72	2501.10	78.01%	86,816
Entropy	76.69	3981.49	77.03%	135,963

Table 7: Comparison of two confidence calculation modes (softmax vs. entropy) on GSM8K (5-shot), for generation length  $g=256$ .

### 7.4.2 With Early Commit

We evaluate integrating *Prophet* early-commit (Li et al., 2025) into **CadLLM** (Table 8). On GSM8K at  $g=256$  (single H100), enabling Prophet increases throughput and reduces function evaluations, but it also lowers accuracy, overlapping with the role of our adaptive-steps policy. As a result, we keep Prophet *disabled by default*; it remains an optional speed knob for deployments that can trade a few accuracy points for lower latency. It is also noteworthy that the original idea of Prophet was explored on top of LLaDA. On the other hand, we are the first to exhaustively explore the validity of this idea in the context of fast-dLLM and added their orthogonal support to ours. In our evaluation, fast-dLLM remains the main baseline, as it is the current training-free SoTA baseline with static decoding framework.

Mode	Token/s $\uparrow$	Time(s) $\downarrow$	Accuracy $\uparrow$	NFE $\downarrow$
Prophet	144.50	2113.72	75.28%	70,335
Default	121.72	2501.10	78.01%	86,816

Table 8: Comparison of enabling early stopping via Prophet vs. default mode on GSM8K (5-shot),  $g=256$ .

### 7.4.3 Ablations with $B_t, S_t, V_t$

Table 9 demonstrates the contributions of each of the three components, namely,  $B_t, S_t$ , and  $V_t$ .

Dataset	$B_t$	$S_t$	$V_t$	Token/s $\uparrow$	Accuracy $\uparrow$
GSM8K	×	×	×	34.3	78.01
	✓	×	×	62.5	74.75
	×	✓	×	34.8	78.01
	×	×	✓	33.7	<b>78.17</b>
	✓	✓	✓	<b>122.9</b>	78.01

Table 9: Ablation results on LLaDA with  $g = 256$ .

### 7.4.4 On Hyperparameter Sensitivity

Diffusion based decoding output often depends on hyperparameter settings. However, their hyperparameter sensitivity is not unique to our method. For example, Fast-dLLM (Wu et al., 2025) requires tuning to find the factor value, refresh interval, block length, and threshold settings. In practice, we found that the hyperparameters primarily depend on the model architecture rather than downstream task. Notably, we used one fixed hyperparameter set for all the task evaluations with LLaDA. Additionally, we now demonstrate the efficacy of a single fixed hyperparameter setting on a separate model, namely DREAM (Ye et al., 2025b), across

all four benchmarks at both generation lengths in Table 10. CadLLM consistently delivers throughput gains of 1.1–1.4 $\times$  on DREAM, confirming that it can be used as an out-of-the-box plug-and-play solution without requiring per-task tuning.

Benchmark	CadLLM (ours)	Fast-dLLM (threshold)
GSM8K	72.40%	73.54%
	<b>86.14 (1.10<math>\times</math>)</b>	<b>78.24 (1.00<math>\times</math>)</b>
	73.31%	73.24%
	<b>103.86 (1.21<math>\times</math>)</b>	<b>85.77 (1.00<math>\times</math>)</b>
MATH	35.98%	37.36%
	<b>128.80 (1.28<math>\times</math>)</b>	<b>101.01 (1.00<math>\times</math>)</b>
	37.20%	38.38%
	<b>161.10 (1.20<math>\times</math>)</b>	<b>134.00 (1.00<math>\times</math>)</b>
MBPP	52.20%	53.40%
	<b>137.83 (1.40<math>\times</math>)</b>	<b>98.22 (1.00<math>\times</math>)</b>
	53.00%	54.80%
	<b>185.22 (1.23<math>\times</math>)</b>	<b>150.23 (1.00<math>\times</math>)</b>
HEval	37.19%	51.00%
	<b>104.31 (1.16<math>\times</math>)</b>	<b>89.91 (1.00<math>\times</math>)</b>
	39.02%	50.60%
	<b>102.66 (1.20<math>\times</math>)</b>	<b>85.62 (1.00<math>\times</math>)</b>

Table 10: Full results on DREAM-7B-Base (single NVIDIA H100) with a single fixed hyperparameter setting. For each benchmark, the *upper row* is generation length 256 and the *lower row* is 512. Each cell shows accuracy (top, in %) and throughput improvement (bottom: **tokens/s / improvement**). Improvement is relative to the Fast-dLLM threshold baseline.

#### 7.4.5 Scaling Behavior of Adaptive Vocab Size

We examine how the adaptive vocabulary size  $V_t$  behaves under substantially different tokenizer scales. Across both architectures evaluated, we find that the effective operating range of  $V_t$  remains stable even when the total vocabulary size increases by more than 3 $\times$ .

LLaDA uses a vocabulary of 50,257 tokens and achieves strong throughput–accuracy trade-offs with  $V_t=1000$  (about 2% of the vocabulary). DREAM, despite using a much larger vocabulary of 151,936 tokens, exhibits similarly stable behavior under the same  $V_t=1000$  setting (about 0.65%). This indicates that  $V_t$  does not need to scale proportionally with vocabulary size.

#### 7.4.6 Pairwise $B_t \times S_t$ Ablation

We conduct the full pairwise  $B_t \times S_t$  ablation (with  $\tau_t$  and  $V_t$  held ON throughout) on both GSM8K and HumanEval for LLaDA and DREAM. Recall that  $B_t$  governs refinement *scope* (how many tokens are revised per block), while  $S_t$  governs refinement *depth* (how many denoising steps are applied); to-

gether they represent orthogonal dimensions of the adaptive refinement mechanism.

$B_t$	$S_t$	Token/s $\uparrow$	Time(s) $\downarrow$	Accuracy $\uparrow$	NFE $\downarrow$
<i>GSM8K (g=256, LLaDA)</i>					
$\checkmark$	$\checkmark$	110.21	2777.19	76.95	86,987
$\times$	$\checkmark$	98.52	3106.57	77.79	92,365
$\checkmark$	$\times$	127.33	2406.37	76.57	76,996
$\times$	$\times$	100.85	3035.45	77.03	89,542
<i>HumanEval (g=256, LLaDA)</i>					
$\checkmark$	$\checkmark$	206.07	311.32	32.93	9,228
$\times$	$\checkmark$	192.78	329.71	31.71	9,546
$\checkmark$	$\times$	213.19	300.06	31.10	8,373
$\times$	$\times$	186.98	338.72	29.88	9,203
<i>GSM8K (g=256, DREAM)</i>					
$\checkmark$	$\checkmark$	85.47	3934.71	71.65	183,695
$\times$	$\checkmark$	82.48	4077.22	72.55	187,157
$\checkmark$	$\times$	91.91	3659.20	70.05	167,716
$\times$	$\times$	30.89	10888.96	72.02	179,033
<i>HumanEval (g=256, DREAM)</i>					
$\checkmark$	$\checkmark$	104.88	399.13	35.37	20,298
$\times$	$\checkmark$	107.26	390.25	31.71	20,615
$\checkmark$	$\times$	113.89	367.53	36.59	18,949
$\times$	$\times$	33.48	1250.25	31.71	19,820

Table 11: Pairwise ablation between  $B_t$  and  $S_t$  (with  $\tau_t$  and  $V_t$  ON). The OFF/OFF configuration yields the lowest accuracy on HumanEval for both LLaDA (29.88%) and DREAM (31.71%), confirming  $B_t$  and  $S_t$  play complementary roles.

#### 7.4.7 Hyperparameter Sensitivity Analysis ( $\pm 20\%$ )

We present a systematic  $\pm 20\%$  sensitivity sweep, varying one hyperparameter at a time while holding all others at their default values across both LLaDA and DREAM on GSM8K and HumanEval. Bold rows indicate the default setting. Experiments are run on 100 samples per task.

	LLaDA (GSM8K)	LLaDA (HEval)	DREAM (GSM8K)	DREAM (HEval)
$B_{\text{init}}$	24	48	16	16
$S_{\text{init}}$	24	16	24	24
$B_{\text{max}}$	64	96	48	48
$S_{\text{max}}$	90	32	70	70
$B_{\text{min}}$	4	12	8	8
$\tau$	0.85	0.85	0.85	0.85

Table 12: Default hyperparameter values used in the sensitivity sweep.

	Variant	Token/s $\uparrow$	Time(s) $\downarrow$	Accuracy $\uparrow$	NFE $\downarrow$
<i>Sensitivity to <math>B_{init}</math></i>					
19	-20%	103.06	224.5	78.0	6,809
<b>24</b>	<b>default</b>	<b>106.84</b>	<b>216.4</b>	<b>78.0</b>	<b>6,589</b>
29	+20%	106.64	214.2	76.0	6,552
<i>Sensitivity to <math>S_{init}</math></i>					
19	-20%	109.07	211.4	80.0	6,568
<b>24</b>	<b>default</b>	<b>107.53</b>	<b>215.0</b>	<b>78.0</b>	<b>6,589</b>
29	+20%	108.42	213.6	76.0	6,602
<i>Sensitivity to <math>B_{max}</math></i>					
51	-20%	104.22	221.6	80.0	6,671
<b>64</b>	<b>default</b>	<b>108.09</b>	<b>213.9</b>	<b>78.0</b>	<b>6,589</b>
77	+20%	107.48	216.3	75.0	6,681
<i>Sensitivity to <math>S_{max}</math></i>					
72	-20%	106.11	217.2	80.0	6,583
<b>90</b>	<b>default</b>	<b>108.82</b>	<b>212.5</b>	<b>78.0</b>	<b>6,589</b>
108	+20%	105.29	219.7	75.0	6,629
<i>Sensitivity to <math>B_{min}</math></i>					
3	-20%	106.67	216.8	79.0	6,592
<b>4</b>	<b>default</b>	<b>106.29</b>	<b>217.5</b>	<b>78.0</b>	<b>6,589</b>
5	+20%	108.79	212.5	77.0	6,539
<i>Sensitivity to <math>\tau</math></i>					
0.68	-20%	131.62	178.3	77.0	5,156
<b>0.85</b>	<b>default</b>	<b>108.39</b>	<b>213.3</b>	<b>78.0</b>	<b>6,589</b>
1	+20%	29.91	779.8	80.0	25,600

Table 13: Sensitivity analysis on LLaDA — GSM8K ( $g=256$ , 100 samples).

	Variant	Token/s $\uparrow$	Time(s) $\downarrow$	Accuracy $\uparrow$	NFE $\downarrow$
<i>Sensitivity to <math>B_{init}</math></i>					
38	-20%	223.44	164.2	45.0	5,484
<b>48</b>	<b>default</b>	<b>236.14</b>	<b>157.5</b>	<b>45.0</b>	<b>5,264</b>
58	+20%	245.23	150.7	42.0	5,126
<i>Sensitivity to <math>S_{init}</math></i>					
13	-20%	242.98	150.7	43.0	5,059
<b>16</b>	<b>default</b>	<b>242.52</b>	<b>153.3</b>	<b>45.0</b>	<b>5,264</b>
19	+20%	239.08	153.5	43.0	5,178
<i>Sensitivity to <math>B_{max}</math></i>					
77	-20%	232.81	159.3	41.0	5,405
<b>96</b>	<b>default</b>	<b>243.36</b>	<b>152.8</b>	<b>45.0</b>	<b>5,264</b>
115	+20%	233.40	158.3	40.0	5,359
<i>Sensitivity to <math>S_{max}</math></i>					
26	-20%	245.54	152.1	45.0	5,243
<b>32</b>	<b>default</b>	<b>238.82</b>	<b>155.7</b>	<b>45.0</b>	<b>5,264</b>
38	+20%	241.26	153.7	45.0	5,282
<i>Sensitivity to <math>B_{min}</math></i>					
10	-20%	245.42	152.2	46.0	5,278
<b>12</b>	<b>default</b>	<b>240.96</b>	<b>154.3</b>	<b>45.0</b>	<b>5,264</b>
14	+20%	240.08	155.4	45.0	5,289
<i>Sensitivity to <math>\tau</math></i>					
0.68	-20%	256.61	143.7	42.0	5,061
<b>0.85</b>	<b>default</b>	<b>241.72</b>	<b>153.8</b>	<b>45.0</b>	<b>5,264</b>
1	+20%	50.26	730.2	47.0	25,600

Table 14: Sensitivity analysis on LLaDA — HumanEval ( $g=256$ , 100 samples).

	Variant	Token/s $\uparrow$	Time(s) $\downarrow$	Accuracy $\uparrow$	NFE $\downarrow$
<i>Sensitivity to <math>B_{init}</math></i>					
13	-20%	84.04	303.3	69.0	14,315
<b>16</b>	<b>default</b>	<b>82.64</b>	<b>308.5</b>	<b>69.0</b>	<b>14,315</b>
19	+20%	84.81	300.6	69.0	14,315
<i>Sensitivity to <math>S_{init}</math></i>					
19	-20%	84.38	302.2	70.0	14,018
<b>24</b>	<b>default</b>	<b>84.52</b>	<b>301.6</b>	<b>69.0</b>	<b>14,315</b>
29	+20%	83.47	305.4	70.0	14,263
<i>Sensitivity to <math>B_{max}</math></i>					
38	-20%	84.41	302.0	71.0	13,867
<b>48</b>	<b>default</b>	<b>82.90</b>	<b>307.5</b>	<b>69.0</b>	<b>14,315</b>
58	+20%	86.43	295.0	71.0	14,033
<i>Sensitivity to <math>S_{max}</math></i>					
56	-20%	84.26	302.6	69.0	14,160
<b>70</b>	<b>default</b>	<b>82.51</b>	<b>309.0</b>	<b>69.0</b>	<b>14,315</b>
84	+20%	82.79	307.9	69.0	14,356
<i>Sensitivity to <math>B_{min}</math></i>					
6	-20%	85.07	299.7	70.0	14,252
<b>8</b>	<b>default</b>	<b>84.08</b>	<b>303.2</b>	<b>69.0</b>	<b>14,315</b>
10	+20%	85.83	297.1	71.0	13,865
<i>Sensitivity to <math>\tau</math></i>					
0.68	-20%	93.01	274.1	71.0	12,811
<b>0.85</b>	<b>default</b>	<b>83.87</b>	<b>303.9</b>	<b>69.0</b>	<b>14,315</b>
1	+20%	48.32	527.6	76.0	25,600

Table 15: Sensitivity analysis on DREAM — GSM8K ( $g=256$ , 100 samples).

	Variant	Token/s $\uparrow$	Time(s) $\downarrow$	Accuracy $\uparrow$	NFE $\downarrow$
<i>Sensitivity to <math>B_{init}</math></i>					
13	-20%	112.38	227.2	41.0	12,032
<b>16</b>	<b>default</b>	<b>110.20</b>	<b>231.7</b>	<b>41.0</b>	<b>12,032</b>
19	+20%	110.86	230.3	41.0	12,032
<i>Sensitivity to <math>S_{init}</math></i>					
19	-20%	108.56	235.2	39.0	11,963
<b>24</b>	<b>default</b>	<b>109.80</b>	<b>232.5</b>	<b>41.0</b>	<b>12,032</b>
29	+20%	109.53	233.0	42.0	12,204
<i>Sensitivity to <math>B_{max}</math></i>					
38	-20%	108.84	234.5	44.0	12,472
<b>48</b>	<b>default</b>	<b>112.48</b>	<b>227.0</b>	<b>41.0</b>	<b>12,032</b>
58	+20%	108.80	234.6	45.0	12,188
<i>Sensitivity to <math>S_{max}</math></i>					
56	-20%	110.06	232.0	41.0	11,812
<b>70</b>	<b>default</b>	<b>110.86</b>	<b>230.3</b>	<b>41.0</b>	<b>12,032</b>
84	+20%	106.68	239.3	43.0	12,293
<i>Sensitivity to <math>B_{min}</math></i>					
6	-20%	111.71	228.5	43.0	12,043
<b>8</b>	<b>default</b>	<b>110.25</b>	<b>231.5</b>	<b>41.0</b>	<b>12,032</b>
10	+20%	110.18	231.7	41.0	12,062
<i>Sensitivity to <math>\tau</math></i>					
0.68	-20%	122.46	208.4	43.0	10,873
<b>0.85</b>	<b>default</b>	<b>112.23</b>	<b>227.5</b>	<b>41.0</b>	<b>12,032</b>
1	+20%	51.03	500.3	43.0	25,600

Table 16: Sensitivity analysis on DREAM — HumanEval ( $g=256$ , 100 samples).

Dataset	$g$	Token/s	NFE	Accuracy
GSM8K	256	120.07 ( $\pm 0.51$ )	86816	77.79 ( $\pm 1.14$ )
	512	107.79 ( $\pm 0.53$ )	103792	76.42 ( $\pm 1.17$ )
MATH	256	102.04 ( $\pm 0.25$ )	423857	31.96 ( $\pm 0.62$ )
	512	117.21 ( $\pm 0.34$ )	643229	34.42 ( $\pm 0.63$ )
MBPP	256	99.86 ( $\pm 0.61$ )	27979	26.00 ( $\pm 1.96$ )
	512	104.62 ( $\pm 0.43$ )	50776	12.8 ( $\pm 1.49$ )
HEval	256	220.81 ( $\pm 1.62$ )	9066	31.09 (NA)
	512	163.72 ( $\pm 1.64$ )	17527	37.20 (NA)

Table 17: CadLLM: Throughput and accuracy statistical deviation analysis with multiple runs.

**Default hyperparameter values.** Across the five block and step hyperparameters ( $B_{\text{init}}$ ,  $S_{\text{init}}$ ,  $B_{\text{max}}$ ,  $S_{\text{max}}$ ,  $B_{\text{min}}$ ), accuracy varies by at most  $\pm 3$  pp on GSM8K and  $\pm 5$  pp on HumanEval within the  $\pm 20\%$  range, and NFE shifts by less than 5%, confirming that the default configuration lies in a stable operating region. The confidence threshold  $\tau$  exhibits the expected monotonic trade-off: lowering it reduces NFE and speeds up inference, while raising it toward 1.0 forces near-exhaustive computation; the practical range  $\tau \in [0.68, 0.90]$  remains well-behaved on both tasks. Crucially, **one fixed set of hyperparameters is used across all three of the four tasks for LLaDA (except HumanEval), and all four tasks for DREAM**, with no or very little per-task tuning.

#### 7.4.8 Statistical Uncertainty Analysis

We have run all experiments on LLaDA with a total of *three* different seeds to validate the stability of our results. We now report mean  $\pm$  standard deviation for tokens/sec and accuracy in Table 17. Note that (1) as NFE are the same across all runs for each task, we report the single NFE result without standard deviation; (2) since we derive accuracy for HumanEval (HEval) from the post-processing script, we do not obtain the standard deviation of accuracy and thus only report the single accuracy.

#### 7.4.9 Failure Modes and Practical Guardrails

Like other diffusion-based decoding methods, CadLLM exhibits failure modes under overly aggressive hyperparameter configurations. To characterize these limits, we perform 100-sample diagnostic runs on GSM8K ( $g=256$ ), perturbing one hyperparameter at a time. Under stressed settings, accuracy drops sharply on both models, as summarized in Table 18.

Setting	LLaDA (%)	DREAM (%)
$V_t=1$	8	1
$\tau_0=0.3$ , $\tau_{\text{min}}=0.3$	35	28
$S_0=6$ , $S_{\text{max}}=12$	29	20
$B_0/B_{\text{max}}/B_{\text{min}}$	25	24

Table 18: Accuracy drops under hyperparameter stress on GSM8K ( $g=256$ ). Each row perturbs one hyperparameter. For blocks,  $B_0/B_{\text{max}}/B_{\text{min}} = (\text{LLaDA: } 112/128/96; \text{ DREAM: } 80/96/64)$ .

These results show that extremely small vocabulary sizes, overly low thresholds, insufficient refinement steps, or excessively large blocks can all lead to severe accuracy collapse, consistent with premature commitment and under-refinement in dLLM.

To illustrate the qualitative failure pattern, below we include one representative output produced by LLaDA under an unstable configuration ( $\tau_0=0.3$ ,  $\tau_{\text{min}}=0.3$ ):

##### Question:

Kylar went to the store to buy glasses for his new apartment. One glass costs \$5, but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How much does he need to pay for them?

##### Answer:

To determine the total cost for 16 glasses, we need to consider the cost of the first glass and the subsequent discounted.

```
- First glass:  $5.00
- Second glass: 0.60 * $5.00 = $3.00
- Total::  $ $ .000 + $ $ .00 = $
$8.00
- Total::  $ $8.00 + $ $3.00 = $
$11.00
- Total:  $ $ $ $ $ .00 $ $ $ $ $ $ $
$ $ $ $ $
```

These observations motivate simple practical guardrails: avoid overly restrictive  $V_t$ , maintain moderate initial thresholds, and allow sufficient refinement steps early in decoding. Within the empirically validated ranges reported in this appendix, CadLLM remains stable across architectures while consistently improving throughput.

#### 7.5 Qualitative Case Studies

We compare answers produced by our controller in three settings: *ON* (all four adaptive policies enabled), an *ablated* variant with exactly one policy disabled, and *OFF* (all four adaptive policies disabled). The answers are shown in Table 19.

**Question:**

Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for \$2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?

---

<b>ON</b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Calculate the total number of eggs laid per day. 2) Subtract eggs eaten and baked. 3) Compute revenue. Total eggs = 16. Breakfast = 3; baking = 4. Eggs left to sell = $16 - 3 - 4 = 9$ . She sells each egg for \$2, so revenue = $9 \times 2 = 18$ . Therefore, Janet makes \$18 per day.
<b>No <math>V_t</math></b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Calculate the total number of eggs laid per day. 2) Subtract eggs eaten and baked. 3) Compute revenue. Total eggs = 16. Breakfast = 3; baking = 4. Eggs left to sell = $16 - 3 - 4 = 9$ . Price = \$2/egg. Revenue = $9 \times 2 = 18$ . Therefore, Janet makes \$18 per day.
<b>OFF</b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Total eggs laid per day: 16. 2) Eggs eaten for breakfast and baked for friends: $3 + 4 = 7$ . 3) Eggs left to sell: $16 - 7 = 9$ . 4) Revenue: $9 \times 2 = 18$ . Therefore, Janet makes \$18 every day.

---

Table 19: Case study (adaptive vocabulary size ablation). Each variant is a single, full-width row.

**Adaptive vocabulary size (case study).** In Table 19, all three variants (ON, No adaptive vocabulary size, OFF) compute the correct answer (18). The qualitative difference is that *ON* presents a concise, single-pass derivation with no redundant enumeration, whereas *No adaptive vocabulary size* and *OFF* tend to include extra scaffolding. This reflects the role of vocabulary size: keep breadth wide only when uncertainty is high, then narrow once scores concentrate. On harder prompts, this setting reduces token/phrase repetition by briefly widening the candidate set when the model hesitates, and then tightening it as confidence stabilizes.

**Adaptive steps (case study).** In Table 20, all rows again reach 18, but *ON* keeps the reasoning succinct while *No-adaptive steps* uses comparable logic with slightly more intermediate narration.

---

<b>ON</b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Calculate the total number of eggs laid per day. 2) Subtract eggs eaten and baked. 3) Compute revenue. Total eggs = 16. Breakfast = 3; baking = 4. Eggs left to sell = $16 - 3 - 4 = 9$ . Price = \$2. Revenue = $9 \times 2 = 18$ . Therefore, \$18 per day.
<b>No <math>S_t</math></b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Calculate the total number of eggs laid per day. 2) Subtract eggs eaten and baked. 3) Compute revenue. Total eggs = 16. Breakfast = 3; baking = 4. Eggs left for sale = $16 - 3 - 4 = 9$ . Price = \$2/egg. Revenue = $9 \times 2 = 18$ . Therefore, \$18 per day.
<b>OFF</b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Total eggs laid per day: 16. 2) Eggs eaten for breakfast and baked for friends: $3 + 4 = 7$ . 3) Eggs left to sell: $16 - 7 = 9$ . 4) Revenue: $9 \times \$2 = \$18$ . Therefore, Janet makes \$18 every day.

---

Table 20: Case study (adaptive steps ablation).

Adaptive steps mainly protect quality on trickier questions: they allocate just enough refinement to ambiguous spans before committing, which avoids under-refinement errors (e.g., off-by-one arithmetic or prematurely truncated derivations) without forcing needless extra passes when the solution is already settled.

**Adaptive block size (case study).** Table 21 shows that all settings deliver the same correct outcome, indicating the policy does not harm content fidelity on short arithmetic. Its qualitative value appears as sequences grow: *ON* focuses work on the uncertain span only, preventing spillover from already-stable text and preserving step-by-step clarity. The ablated and all-off rows remain correct here, but on longer chains the adaptive span reduces cross-span interference and keeps explanations tighter.

**Adaptive threshold (case study).** In Table 22, each variant still returns 18 with coherent steps. The adaptive threshold's contribution is pacing: it raises the bar early (to avoid premature commitment when context is thin) and relaxes it late (to avoid redundant re-checking once the answer is evident). The *No-adaptive threshold* row looks similar on this easy prompt, but on more ambiguous ones the adaptive gate prevents vacillation and unnecessary edits, yielding consistent derivations while preserving efficiency.

---

<b>ON</b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Calculate the total number of eggs laid per day. 2) Subtract eggs eaten and baked. 3) Compute revenue. Total eggs = 16. Breakfast = 3; baking = 4. Eggs left to sell = $16 - 3 - 4 = 9$ . Price = \$2. Revenue = $9 \times 2 = 18$ . Therefore, \$18 per day.
<b>No <math>B_t</math></b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Total eggs laid per day: 16. 2) Eggs eaten for breakfast and baked for friends: $3 + 4 = 7$ . 3) Eggs left to sell: $16 - 7 = 9$ . 4) Revenue: $9 \times \$2 = \$18$ . Therefore, Janet makes \$18 every day.
<b>OFF</b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Total eggs laid per day: 16. 2) Eggs eaten for breakfast and baked for friends: $3 + 4 = 7$ . 3) Eggs left to sell: $16 - 7 = 9$ . 4) Revenue: $9 \times \$2 = \$18$ . Therefore, Janet makes \$18 every day.

---

Table 21: Case study (adaptive block size ablation).

---

<b>ON</b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Calculate the total number of eggs laid per day. 2) Subtract eggs eaten and baked. 3) Compute revenue. Total eggs = 16. Breakfast = 3; baking = 4. Eggs left to sell = $16 - 3 - 4 = 9$ . Price = \$2. Revenue = $9 \times 2 = 18$ . Therefore, \$18 per day.
<b>No <math>\tau_t</math></b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Total eggs laid per day: 16. 2) Eggs eaten for breakfast: 3. 3) Eggs baked for muffins: 4. 4) Eggs left to sell: $16 - 3 - 4 = 9$ . 5) Revenue: $9 \times \$2 = \$18$ . Therefore, Janet makes \$18 at the farmers' market.
<b>OFF</b>	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1) Total eggs laid per day: 16. 2) Eggs eaten for breakfast and baked for friends: $3 + 4 = 7$ . 3) Eggs left to sell: $16 - 7 = 9$ . 4) Revenue: $9 \times \$2 = \$18$ . Therefore, Janet makes \$18 every day.

---

Table 22: Case study (adaptive threshold ablation).