CAI-DLLM: Convergence Aware Inference for Diffusion Language Models

Farhana Amin
Virginia Tech
afarhana@vt.edu

Sabiha Afroz
Virginia Tech
sabihaafroz@vt.edu

Dimitrios S. Nikolopoulos
Virginia Tech
dsn@vt.edu

Abstract

Diffusion language models can generate many tokens in parallel, but they still require repeated denoising steps during inference. This makes generation costly, especially when the model continues to recompute tokens that are already stable. To address these limitations, we propose CAI-DLLM, a training-free inference method that uses first-step confidence to guide denoising and reduce inference time. Specifically, CAI-DLLM commits easy tokens earlier, allocates more denoising steps to harder tokens, and adjusts decoding schedules across output blocks. As it relies only on first-step confidence signals, it does not require retraining, extra predictors, or weight updates. We evaluate CAI-DLLM on LLaDA-8B-Instruct and Dream-7B-Instruct across math, code, reasoning, commonsense, and long-context (Dream-7B). CAI-DLLM achieves up to 18.2× wall clock inference speedup on LLaDA GSM8K while improving accuracy (77.41% vs. 76.27%), and up to 13.1× on Dream HumanEval at higher pass@1 than no-cache (48.17% vs. 46.95%). On harder reasoning tasks, speedups reach 44.8×, with a largest accuracy drop of 4.4 points, while energy consumption is reduced by up to 95.3%.

1 Introduction

Large language models are commonly built on autoregressive generation, producing text from left to right one token at a time (Vaswani et al., 2017; Brown et al., 2020; Grattafiori et al., 2024). This creates a sequential dependency during inference: each token depends on all previous tokens, so long outputs require many sequential model calls and generation is slow.

Masked diffusion language models offer a parallel alternative to autoregressive decoding by refining masked tokens over multiple denoising steps and updating many positions at once (Nie et al., 2025; Ye et al., 2025). This can improve throughput, but each denoising step still runs a full transformer forward pass over the sequence, including tokens that have already stabilized. Recent methods reduce the cost of each step through KV caching (Wu et al., 2025; Ma et al., 2025; Liu et al., 2025) or early layer token skipping (Zhu et al., 2026). However, they still use the same denoising budget for every token. Easy tokens continue to receive unnecessary steps, while hard tokens receive no extra refinement.

Our key observation is that the confidence from the first denoising step provides a free, reliable estimate of token difficulty. Tokens with high first-step confidence stabilize after only a few steps, while tokens with low confidence require more refinement. Since this signal is produced during the required first forward pass, it adds no extra cost.

Based on this observation, we propose CAI-DLLM, a novel training-free inference method that uses first-step confidence to assign each token its own step budget, set block level commit schedules, and detect and exit low yield late denoising. Easy tokens are committed early and hard tokens receive more steps, directly addressing the limitation that existing methods cannot vary per token computation. Our contributions are as follows.

- **First-step confidence as a token level control signal.** We show that confidence at the first denoising step predicts which tokens stabilize early and reuse this signal throughout decoding at no extra cost, without re-evaluating at every step as in Fast-dLLM (Wu et al., 2025).
- **Block level adaptive threshold scheduling.** We introduce a block end threshold $\theta_e^{(b)}$ that decreases across output blocks, reflecting that later blocks converge faster given more committed context.
- **Confidence gated token budgets and grind-**

ing phase detection. We assign each token a maximum denoising budget $B_i \in \{8, 32, 64\}$ based on first-step confidence and position, and introduce a low yield detector that stops denoising when newly committed tokens remain scarce for K consecutive steps.

2 Background and Motivation

Masked diffusion language models generate text through an iterative denoising process (Austin et al., 2021a; Lou et al., 2023; Sahoo et al., 2024). Given an input prompt, the model initializes a set of masked positions and progressively replaces them with predicted tokens. At each denoising step, the model predicts a vocabulary distribution for every masked position, and a decoding rule determines which tokens are committed. Unlike autoregressive models, which generate tokens sequentially, masked diffusion models can update multiple positions in a single forward pass. However, inference remains costly because generation still requires many denoising steps. As a result, the total cost depends on both the output length and the number of denoising iterations.

2.1 Denoising Computation is Uneven

A fixed decoding policy applies the same computation rule to all tokens and output blocks. This assumption is inefficient because denoising does not progress uniformly across the sequence. We analyze this behavior using LLaDA-8B-Instruct on GSM8K with 256 generated tokens divided into four 64-token blocks (Cobbe et al., 2021).

We measure the change in hidden states between adjacent denoising steps using relative L2 drift:

$$D^{(t)} = \frac{\|\mathbf{h}^{(t)} - \mathbf{h}^{(t-1)}\|_2}{\|\mathbf{h}^{(t-1)}\|_2}. \tag{1}$$

Here, $\mathbf{h}^{(t)} \in \mathbb{R}^{L \times d}$ denotes the hidden state at denoising step t. We average this value over token positions.

Our analysis shows that early layers change substantially less than deeper layers:

$$\begin{aligned} D_{\text{early}}(L0\text{–}L8) &\approx 2\text{–}3\%, \\ D_{\text{deep}}(L17\text{–}L31) &\approx 6\text{–}14\%. \end{aligned} \tag{2}$$

We also observe that first-step confidence separates tokens by convergence difficulty:

$$\begin{aligned} c_i^{(0)} > 0.50 &\Rightarrow t_i^* \approx 5, \\ c_i^{(0)} < 0.25 &\Rightarrow t_i^* \approx 30\text{–}50. \end{aligned} \tag{3}$$

Here, t_i^* denotes the first denoising step at which token i becomes stable. We define a token as stable when its top prediction remains unchanged for three consecutive steps. These trends are measured across 1,319 GSM8K test prompts. Figure 1 summarizes the same behavior across layers, output blocks, and tokens. Additional motivation analysis is provided in Appendix C.

2.2 First-Step Confidence as a Control Signal

The key question is whether token difficulty can be estimated early. We use the confidence from the first denoising step as this signal. For a masked position i, we define

$$c_i^{(0)} = \max_{v \in \mathcal{V}} p_\theta(x_i = v \mid x^{(0)}). \tag{4}$$

This value is already computed during standard decoding. It does not require another model, another forward pass, or retraining. As shown in Figure 1(c), tokens with $c_i^{(0)} > 0.50$ usually stabilize early, while tokens with $c_i^{(0)} < 0.25$ often need many more denoising steps. We therefore use $c_i^{(0)}$ to guide token commitment and step allocation.

3 Related Work

Autoregressive language models. Autoregressive models generate text one token at a time (Vaswani et al., 2017; Brown et al., 2020; Grattafiori et al., 2024). KV caching (Kwon et al., 2023) reduces repeated attention computation but does not remove the sequential bottleneck, motivating parallel diffusion decoding (Lee et al., 2018).

Diffusion language models. LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) show that masked diffusion models match autoregressive models on reasoning, math, and code, but every token receives the same fixed number of steps regardless of convergence speed. This uniform treatment is the core inefficiency CAI-DLLM addresses.

Caching for diffusion language models. Several methods reduce inference cost by reusing KV activations across denoising steps: dKV-Cache (Ma et al., 2025) delays reuse until tokens stabilize, dLLM-Cache (Liu et al., 2025) applies separate rules for prompt and response tokens, Fast-dLLM (Wu et al., 2025) commits tokens when confidence exceeds a fixed threshold and introduces DualCache, which we use as the Fast-dLLM representative baseline. FlashDLM (Hu et al., 2025)

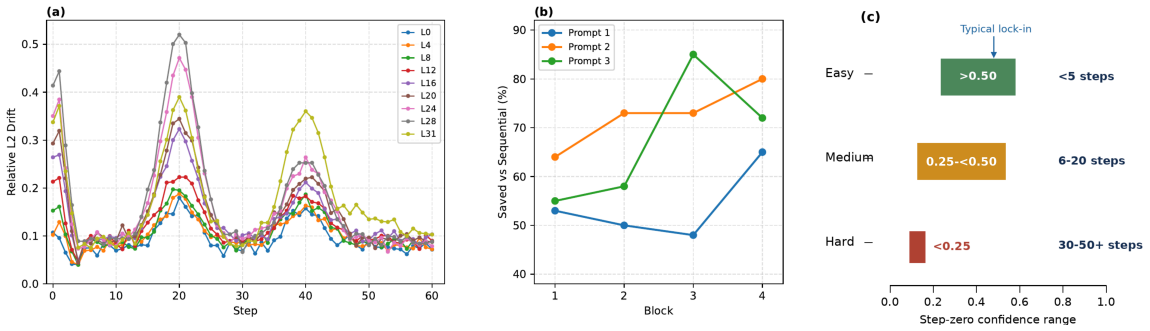

Figure 1: Denoising work is uneven across layers, blocks, and tokens. (a) Hidden state drift varies across transformer layers. (b) Saved steps increase in later blocks as earlier context becomes available. (c) First-step confidence separates easy tokens with $c_i^{(0)} > 0.50$, medium tokens with $0.25 \le c_i^{(0)} \le 0.50$, and hard tokens with $c_i^{(0)} < 0.25$.

combines caching with autoregressive guidance and Sparse dLLM (Song et al., 2025) prunes less useful cache entries. These methods reduce repeated computation through cache reuse or cache sparsity.

Parallel and hybrid decoding. D2F (Discrete Diffusion Forcing; Wang et al. 2025) mixes autoregressive and diffusion decoding to make KV reuse easier, but this reduces the parallelism that makes diffusion models fast in the first place. Spiffy (Agrawal et al., 2025) adapts speculative decoding (Leviathan et al., 2023) to diffusion models, but the draft-and-verify overhead adds complexity and extra computation at each step. Both methods require changes to the decoding structure itself. CAI-DLLM keeps the original model and decoding process unchanged, making it simpler to apply and compatible with existing caching methods.

Adaptive computation and token skipping. ES-dLLM skips low importance token computation in early layers (Zhu et al., 2026). This is effective, but it still runs the full denoising loop. ES-dLLM reduces computation inside each denoising step, while CAI-DLLM reduces the number of denoising steps assigned to each token.

4 Method

CAI-DLLM is a training-free inference method that uses first-step confidence to control masked diffusion decoding. Figure 2 summarizes the full pipeline: the model first runs the required step-zero (first step) forward pass to obtain token confidence scores, then a confidence aware controller uses these scores to guide adaptive parallel decoding, block level threshold schedules, token budgets, and low yield early exit. The method builds on confidence based parallel decoding (Wu et al., 2025), but replaces the fixed commit rule with a lightweight controller that commits easy tokens early, gives hard tokens more steps, and stops late denoising when few new tokens are being resolved.

4.1 Diffusion Decoding Setup

Let $\mathbf{x} = (x_1, \ldots, x_L)$ be a sequence of L tokens. At denoising step t, the model keeps a partially decoded sequence $\mathbf{x}^{(t)}$. Some positions contain generated tokens. Other positions still contain the masked tokens. Let $\mathcal{M}^{(t)}$ be the set of masked positions at step t. For each masked position, the model predicts a distribution over the vocabulary. Box 1 defines token confidence.

Box 1: Token Confidence

For each masked token $i \in \mathcal{M}^{(t)}$, the diffusion language model predicts

$$p_\theta(x_i \mid \mathbf{x}^{(t)}),$$

where $\mathcal{V}$ is the vocabulary. We define the confidence of token i at step t as

$$c_i^{(t)} = \max_{v \in \mathcal{V}} p_\theta(x_i = v \mid \mathbf{x}^{(t)}).$$

A high value of $c_i^{(t)}$ means that the model has a strong current prediction for token i. A low value means that the token is still uncertain.

4.2 First-Step Confidence

The first denoising step is required in every diffusion decoding process. We use this step to compute a difficulty signal for each token at no extra cost, since no additional forward pass is needed beyond what standard decoding already performs (Box 2).

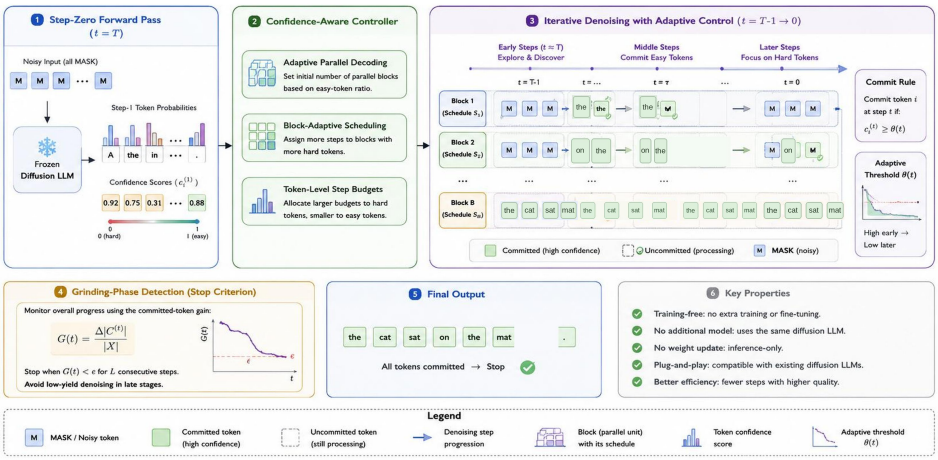

Figure 2: Overview of CAI-DLLM. ① The method runs a step-zero forward pass to obtain token confidence scores. ② These scores drive a confidence-aware controller that guides adaptive parallel decoding, block-adaptive scheduling, and token-level step budgets. ③ During iterative denoising, easy tokens are committed early while hard tokens receive more steps. ④ A grinding-phase detector stops low-yield late denoising. The method is training-free and does not change model weights.

Box 2: First-Step Confidence Signal

For each token i, we store the confidence from the first denoising step:

$$s_i = c_i^{(0)}.$$

The value s_i is used as an early estimate of token difficulty. Tokens with high s_i are treated as easier. Tokens with low s_i are treated as harder. This signal adds no extra model call because the first forward pass is already part of normal decoding.

As established in Section 2.2 and shown in Figure 1(c), our first observation is that tokens with high first-step confidence ($c_i^{(0)} > 0.50$) stabilize within 5 steps in nearly all cases, and tokens with low confidence ($c_i^{(0)} < 0.25$) continue changing for 30 steps or more. This pattern is consistent across prompts and tasks, which means $c_i^{(0)}$ is a reliable signal for controlling token commitment and step allocation.

Box 3: Adaptive Threshold Schedule

Let T be the maximum number of denoising steps for one block. Let θ_s and θ_e be the start and end thresholds. Let w be the warmup fraction.

$$u(t) = \frac{t - wT}{T - wT}$$

$$\alpha = \frac{\theta_s - \theta_e}{2}$$

$$\theta(t) = \begin{cases} \theta_s, & t < wT, \\ \theta_e + \alpha\big(1 + \cos(\pi u(t))\big), & t \geq wT. \end{cases}$$

A token is committed when

$$c_i^{(t)} \geq \theta(t).$$

We use $\theta_s = 0.90$, $\theta_e = 0.70$, and $w = 0.15$. The schedule stays strict early and relaxes later.

We use s_i as defined in Box 2 to assign each token a step budget, set commit thresholds, and schedule denoising across blocks. Storing s_i and the per-token budgets adds $\mathcal{O}(L)$ memory and $\mathcal{O}(L)$ work per step, negligible relative to the $\mathcal{O}(L^2 d)$ transformer forward pass. We do not measure this overhead in isolation; however, the end-to-end throughput results in Tables 1 and 2 confirm that it does not reduce the observed speedups.

4.3 Block Adaptive Scheduling

Long outputs are generated in blocks. Our second observation is that later blocks benefit from more committed context produced by earlier blocks, so they converge faster and can use a lower commit threshold without accuracy loss. Figure 1(b) shows that steps saved increase across the four output blocks. Savings are lowest in Block 1 and highest in the later blocks, indicating that later blocks need fewer denoising steps once earlier context has been committed. We use this observation by decreasing the end threshold across blocks.

Box 4: Block-Specific End Thresholds

Let $b \in \{1, 2, 3, 4\}$ be the block index, $\theta_s = 0.90$ the shared start threshold, and $w = 0.15$ the warmup fraction. The end threshold for block b is

$$\theta_e^{(b)} = 0.70 - 0.10\,(b - 1),$$

giving $\theta_e^{(b)} \in \{0.70, 0.60, 0.50, 0.40\}$ for blocks $b = 1, 2, 3, 4$ respectively. The commit threshold at step t is

$$\theta^{(b)}(t) = \begin{cases} \theta_s & t < wT, \\ \theta_e^{(b)} + \Delta^{(b)}(t) & t \geq wT, \end{cases}$$

where

$$\Delta^{(b)}(t) = \frac{\theta_s - \theta_e^{(b)}}{2} \left(1 + \cos \frac{\pi(t - wT)}{T - wT}\right)$$

and T is the total steps per block. Block 1 keeps the highest end threshold as it has the least decoded context. Each later block lowers it because later blocks converge more easily (Figure 1(b)).

4.4 Token Budgets and Position Aware Thresholds

Adaptive thresholds reduce many unnecessary steps, but late denoising can still become inefficient. In this phase, the model may run several forward passes while committing very few new tokens. We call this the grinding phase. To reduce this cost, CAI-DLLM assigns each token a maximum denoising budget. The budget depends on the first-step confidence and the token position inside the block. Per token compute budgeting draws on the adaptive computation literature (Graves, 2016; Schuster et al., 2022); our key distinction is that CAI-DLLM uses the first-step confidence $c_i^{(0)}$ as a *free* control signal, no additional forward passes are needed beyond the one required by standard decoding.

Box 5: Confidence-Gated Token Budget

Let $s_i = c_i^{(0)}$ be the first-step confidence of token i. Let $p = \text{pos}(i)$ be the position of token i inside its block. The token budget B_i is the maximum number of denoising steps allowed for token i. Cases are evaluated from top to bottom.

$$B_i = \begin{cases} 8, & s_i > 0.50 \text{ and } p < 32, \\ 64, & s_i < 0.25 \text{ or } p \geq 48, \\ 32, & \text{otherwise.} \end{cases} \quad (5)$$

If token i is still masked after B_i steps, it is committed using its current best prediction:

$$x_i \leftarrow \arg\max_{v \in \mathcal{V}} p_\theta(x_i = v \mid x^{(t)}).$$

The budget levels follow the stability tiers in Figure 1(c). Our third observation is that token position also matters : tokens at $p < 32$ commit 37% faster on average than those at $p \geq 48$ on GSM8K with LLaDA-8B, motivating the position-aware threshold in Box 6.

Box 6: Position-Aware Commit Rule

Let $p = \text{pos}(i)$ be the position of token i inside its block. Let $\theta_b(t)$ be the block-level threshold at denoising step t. The token-specific threshold is

$$\theta_i(t) = \text{clip}\left(\theta_b(t) \cdot r_{\text{pos}(i)}, 0.20, 0.98\right). \quad (6)$$

Here, $r_{\text{pos}(i)}$ is a position factor:

$$r_{\text{pos}(i)} = \begin{cases} 0.80, & p < 16, \\ 0.90, & 16 \leq p < 32, \\ 1.05, & 32 \leq p < 48, \\ 1.10, & p \geq 48. \end{cases}$$

The final token commit rule is

$$c_i^{(t)} \geq \theta_i(t) \quad \text{or} \quad t \geq B_i, \quad (7)$$

where B_i is the token budget from Box 5.

The position factors are selected from a 200 sample held out subset of LLaDA GSM8K to improve speed while keeping accuracy degradation below one point. We use the same factors for Dream-7B without further tuning. More details are provided in Appendix A.2.

4.5 Grinding Phase Detection

Token budgets act at the token level. We also use a block level detector to stop low yield denoising. At each step, we count how many tokens were committed. If this count remains low for several consecutive steps, the block is treated as being in the grinding phase.

Box 7: Grinding Phase Detector

Let $\eta^{(t)}$ be the number of tokens committed at step t. A block enters the grinding phase when

$$\eta^{(t)} < \bar{\eta} \quad \text{for } K \text{ consecutive steps} \quad (8)$$

where $\eta^{(t)} = |\{i : x_i^{(t)} \neq \texttt{[MASK]}, x_i^{(t-1)} = \texttt{[MASK]}\}| \,/\, L$ is the fraction of newly committed tokens at step t, $\bar{\eta}$ is the low yield threshold (default $\bar{\eta} = 1.5\%$), $K = 4$ is the patience window, and L is the sequence length.
We use

$$\bar{\eta} = 1.5, \qquad K = 4.$$

When the detector triggers, remaining masked tokens are committed only if

$$c_i^{(t)} > \theta_{\text{fb}}, \qquad \theta_{\text{fb}} = 0.40.$$

The threshold $\bar{\eta} = 1.5$ separates normal progress from low yield denoising. The window $K = 4$ avoids reacting to one noisy step. The fallback threshold keeps very uncertain tokens from being forced.

4.6 Model Adaptive Confidence Gating

Different models respond differently to confidence gating. When gating is ON, the token budgets B_i from Box 5 and the force commit threshold θ_{fb} from Box 7 are active. When gating is OFF, only the adaptive threshold and block schedules are used. Confidence gating is set via GSM8K ablation (Appendix B.1): ON for LLaDA, OFF for Dream, consistent with the HumanEval ablation in Section 6.2.

4.7 Algorithm

Algorithm 1 summarizes CAI-DLLM for one block. Line 1 initializes the low efficiency counter ℓ. Lines 3–8 run the step-zero forward pass to store confidence scores s_i, assign budgets B_i, and set the block threshold $\theta_e^{(b)}$. Lines 10–18 commit each masked token when its confidence meets the position aware threshold or its budget is exhausted. Lines 19–23 increment or reset ℓ based on the newly committed count $\eta^{(t)}$. When $\ell \geq K$ (lines 24–29), remaining high confidence tokens are force committed and the block exits early. The final stopping condition exits if all tokens are committed before the budget runs out.

Algorithm 1: CAI-DLLM inference for one block

Input : $\mathbf{x}^{(0)}$: masked sequence; b: block index; T: max steps per block; $\theta_s, \theta_e^{(b)}$: start/end thresholds; θ_{fb}: force commit threshold; w: warmup fraction; K: patience window; $\bar{\eta}$: yield threshold

Output : Decoded block

// $t \in \{0, \ldots, T-1\}$ is local to this block; B_i in local steps

1 $\ell \leftarrow 0$ // ℓ: low-efficiency step counter
2 **for** $t = 0$ **to** $T - 1$ **do**
3 Run forward pass on $\mathbf{x}^{(t)}$
4 Compute $c_i^{(t)}$ for all masked tokens i
5 **if** $t = 0$ **then** // step-zero init
6 $s_i \leftarrow c_i^{(0)}$ // store first-step confidence
7 Assign budget B_i per token (Box 5)
8 Set threshold $\theta_e^{(b)}$ (Box 4)
9 **end**
10 Compute $\theta^{(b)}(t)$ (Box 3); $\eta^{(t)} \leftarrow 0$ // newly committed count
11 **foreach** *masked token* i **do**
12 Compute $\theta_i(t)$ (Box 6)
13 **if** $c_i^{(t)} \geq \theta_i(t)$ ***or*** $t \geq B_i$ **then**
14 $x_i \leftarrow \arg\max_{v \in \mathcal{V}} p_\theta(x_i{=}v \mid \mathbf{x}^{(t)})$
15 $\eta^{(t)}$ += 1
16 **end**
17 **end**
18 **if** $\eta^{(t)} < \bar{\eta}$ **then** // track low-yield steps
19 ℓ += 1
20 **else**
21 $\ell \leftarrow 0$
22 **end**
23 **if** $\ell \geq K$ **then** // grinding phase detected
24 **foreach** *masked* i *with* $c_i^{(t)} > \theta_{fb}$ **do**
25 $x_i \leftarrow \arg\max_{v \in \mathcal{V}} p_\theta(x_i{=}v \mid \mathbf{x}^{(t)})$
26 **end**
27 **break**
28 **end**
29 **if** *no masked tokens remain* **then break**
30 **end**
31 **return** decoded block

5 Implementation Details

Setup. CAI-DLLM is developed on top of the open-source PyTorch based ES-dLLM codebase, reusing its KV cache backend and integrating our confidence aware controller. We evaluate LLaDA-8B-Instruct and Dream-7B-Instruct on a single NVIDIA H200 SXM5 GPU with 141 GB HBM3e (NVIDIA Corporation, 2024). We generate 256 tokens for reasoning and commonsense tasks, and 512 tokens for code and long-context tasks. The block length is 64 for all tasks except LongBench, where we use 32. All runs use batch size 8 and temperature 0.

Decoding settings. No-cache decoding uses 64 denoising steps per block without KV caching or parallel decoding. All methods use the same generation settings for fair comparison. For CAI-DLLM, we set the warmup fraction to $w = 0.15$, the start threshold to $\theta_s = 0.90$, and the base end threshold to $\theta_e = 0.70$. Confidence gating is enabled for LLaDA and disabled for Dream based on GSM8K ablations. Following ES-dLLM, the KV cache update frequency is 16 for LLaDA and 8 for Dream (Zhu et al., 2026).

Baselines and metrics. We compare with no-cache decoding, DualCache from Fast-dLLM (Wu et al., 2025), and ES-dLLM (Zhu et al., 2026) baselines under the same task settings. For evaluation, we use exact match for GSM8K, MathQA, and BBH, pass@1 for HumanEval and MBPP, and F1/ROUGE-L for LongBench. Throughput is output tokens per second after one warm-up pass, and

speedup is wall-clock time relative to no-cache decoding. Energy is computed as $E = \bar{P} \times T$, where $\bar{P}$ is average GPU power from `nvidia-smi` and T is wall-clock generation time. Full settings, sample sizes, baselines, and software versions are in Appendix A.

6 Results

6.1 Main Results

We evaluate on seven benchmark datasets covering math, code, reasoning, commonsense, and long-context generation (Chen et al., 2021; Suzgun et al., 2023; Austin et al., 2021b; Amini et al., 2019; Bisk et al., 2020; Sakaguchi et al., 2020; Bai et al., 2024). Tables 1 and 2 report the main results.

Bench.	Method	TPS	Spd.	Score	Eng.
GSM8K	No Cache	19.8	1.0×	79.68	3052
	DualCache	103.2	9.3×	78.47	291
	ES-dLLM	97.9	9.0×	77.48	219
	CAI-DLLM	**369.5**	**18.7×**	77.26±1.18	**146**
Human Eval	No Cache	44.3	1.0×	46.95	374
	DualCache	258.1	5.8×	45.12	60
	ES-dLLM	308.5	7.0×	41.46	41
	CAI-DLLM	**581.5**	**13.1×**	48.17±3.76	**24**
BBH	No Cache	24.8	1.0×	61.48	12655
	DualCache	226.9	9.1×	58.27	1318
	ES-dLLM	292.2	11.8×	57.84	927
	CAI-DLLM	**571.4**	**23.0×**	56.95±0.53	**500**
MBPP	No Cache	29.1	1.0×	57.40	868
	DualCache	170.4	5.9×	55.40	104
	ES-dLLM	141.0	4.9×	56.40	78
	CAI-DLLM	**581.7**	**20.0×**	56.60±2.16	**34**
MathQA	No Cache	58.8	1.0×	35.40	54
	DualCache	59.1	1.0×	34.40	43
	ES-dLLM	45.2	0.8×	32.60	47
	CAI-DLLM	**530.8**	**9.0×**	33.80±2.13	**9**
PIQA	No Cache	62.5	1.0×	49.62	158
	DualCache	62.7	1.0×	49.56	140
	ES-dLLM	57.1	0.9×	51.47	135
	CAI-DLLM	**130.9**	**2.1×**	49.08±1.17	**61**
Wino-grande	No Cache	64.6	1.0×	46.80	42
	DualCache	64.7	1.0×	48.40	39
	ES-dLLM	59.8	0.9×	47.60	38
	CAI-DLLM	**107.6**	**1.7×**	44.60±2.22	**24**

Table 1: Results on Dream-7B-Instruct. TPS = tokens/sec; Eng. = energy in Wh. **Bold** = best. DualCache from Wu et al. (2025). CAI-DLLM scores show std. err. as ±.

CAI-DLLM achieves the highest throughput on every evaluated dataset (Figure 3). Gains scale with generation length: on longer tasks, speedup reaches 44.8× on LLaDA BBH, 28.5× on LLaDA MBPP, 23.0× on Dream BBH, 20.0× on Dream MBPP, and 19.0× on Dream LongBench. This length-scaling trend holds across generation lengths,

Bench.	Method	TPS	Spd.	Score	Eng.
GSM8K	No Cache	17.1	1.0×	76.27	3559
	DualCache	196.4	11.7×	**77.18**	264
	ES-dLLM	175.3	10.6×	77.10	223
	CAI-DLLM	**335.1**	**18.2×**	77.41±1.18	**169**
Human Eval	No Cache	39.0	1.0×	37.20	425
	DualCache	173.0	4.4×	35.98	55
	ES-dLLM	139.8	3.6×	35.98	42
	CAI-DLLM	**435.3**	**11.2×**	35.98±3.76	**33**
BBH	No Cache	11.1	1.0×	56.75	28425
	DualCache	130.1	11.8×	53.29	1193
	ES-dLLM	168.8	14.5×	54.51	915
	CAI-DLLM	**495.1**	**44.8×**	52.33±0.53	**579**
MBPP	No Cache	15.3	1.0×	40.00	1029
	DualCache	93.7	6.1×	38.20	97
	ES-dLLM	75.6	4.9×	38.00	81
	CAI-DLLM	**435.6**	**28.5×**	37.00±2.16	**51**
MathQA	No Cache	48.5	1.0×	35.20	68
	DualCache	42.5	0.9×	33.40	49
	ES-dLLM	34.1	0.7×	32.20	56
	CAI-DLLM	**411.4**	**8.5×**	33.60±2.13	**10**
PIQA	No Cache	51.0	1.0×	50.05	178
	DualCache	46.6	0.9×	46.79	163
	ES-dLLM	44.5	0.9×	50.54	162
	CAI-DLLM	**105.3**	**2.1×**	47.55±1.17	**82**
Wino-grande	No Cache	52.4	1.0×	49.20	48
	DualCache	48.5	0.9×	46.80	45
	ES-dLLM	46.2	0.9×	50.00	44
	CAI-DLLM	**74.3**	**1.4×**	48.20±2.23	**31**

Table 2: Results on LLaDA-8B-Instruct. TPS = tokens/sec; Eng. = energy in Wh. **Bold** = best. Sample sizes in Appendix A. CAI-DLLM scores show std. err. as ±.

with 4.59× on LLaDA-Instruct and 15.29× on Dream-Instruct at length 512 (Appendix E). On shorter commonsense tasks (PIQA, Winogrande), CAI-DLLM remains the only method with clear speedup, though at a modest accuracy trade-off versus ES-dLLM (Appendix D.3). The controller overhead is below 2%, while early commitment and grinding-phase exit reduce per-iteration time by 13.6% on LLaDA and 22.5% on Dream.

Accuracy shows a clear efficiency-quality trade-off. On Dream HumanEval, CAI-DLLM improves pass@1 from 46.95 to 48.17 while giving a 13.1× speedup; with a standard error of ±3.76, we treat this as accuracy parity. On harder reasoning tasks, the trade-off is larger. For example, LLaDA BBH drops from 56.75 to 52.33 while reaching 44.8× speedup. Dream GSM8K also drops from 79.68 to 77.26, but gives an 18.7× speedup.

Long-context results. On LongBench, evaluated with Dream-7B, CAI-DLLM reaches 19.04× speedup with an average score of 23.11, compared with 24.63 for no-cache decoding. All cached meth-

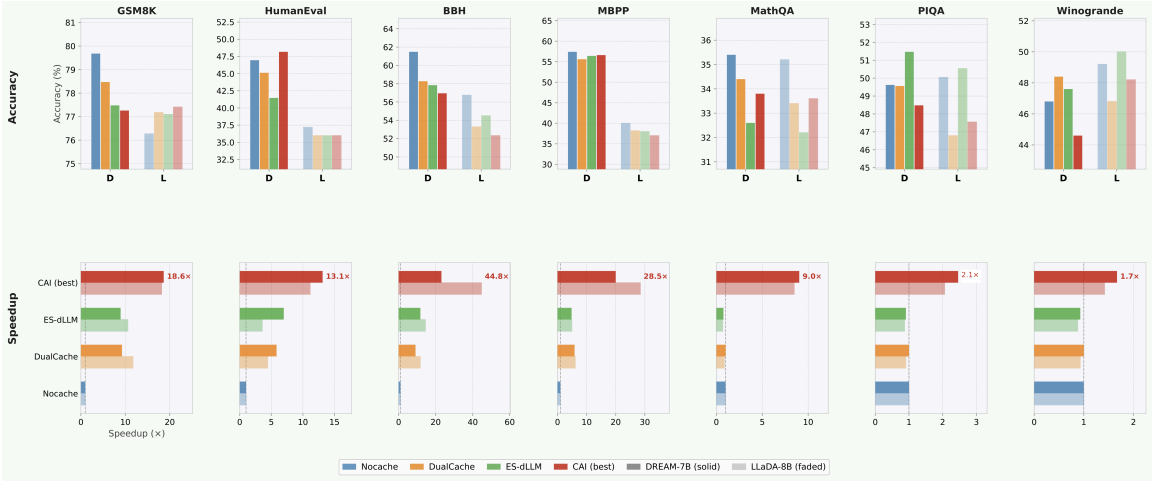

Figure 3: Accuracy and speedup comparison across the seven main benchmarks for Dream-7B and LLaDA-8B. LongBench is reported in detail in Appendix D.2.

ods slightly reduce the average score in this setting, but CAI-DLLM provides more than twice their throughput. Energy follows the same trend as runtime. For example, on LLaDA GSM8K, CAI-DLLM reduces energy from 3559 Wh to 169 Wh, a 95.3% reduction.

6.2 Ablation Study

We ablate the main components of CAI-DLLM on HumanEval. Table 3 reports results for both models. GSM8K model adaptation results are reported in Appendix B.1, Table 11.

LLaDA-8B-Instruct. APD gives the largest throughput gain, increasing throughput from 139.8 TPS with ES-dLLM to 424.3 TPS. This comes with a small pass@1 drop from 35.98% to 35.37%. Adding the per-block schedule gives a small further speedup, reaching 434.2 TPS with the same pass@1. Adding token budgets gives the highest throughput for LLaDA, reaching 435.3 TPS, while keeping pass@1 at 35.37%. This shows that most of the gain comes from APD, while block scheduling and budgets add smaller improvements.

Dream-7B-Instruct. For Dream, the gated budget configuration reaches very high throughput, but it lowers pass@1 to 40.34%. Disabling confidence gating recovers pass@1 to 46.34%, which is higher than DualCache and close to the no-cache baseline, while keeping 614.6 TPS. This supports our model-adaptive policy: LLaDA can use confidence-gated budgets, while Dream performs better when gating is disabled.

Denoising-step reduction. Table 4 confirms that CAI-DLLM reduces denoising iterations directly, not only per-step cost, with 39–56% step reduction on GSM8K and 38–47% on HumanEval.

Model	Config.	Pass@1	TPS
LLaDA	ES-dLLM	35.98	139.8
	+APD	35.37	424.3
	+APD+Block	35.37	434.2
	+APD+Block+Budget ⋆	35.37	435.3
Dream	DualCache	45.12	72.5
	ES-dLLM	41.46	56.7
	+APD+Block+Budget	40.34	614.7
	Full CAI, gate off ⋆	46.34	614.6

Table 3: HumanEval ablation of CAI-DLLM components. ⋆ marks the recommended configuration for each model.

Task	Model	Fixed	CAI	Reduction
GSM8K	LLaDA	256	114	55.5%
GSM8K	Dream	256	141	44.9%
HumanEval	LLaDA	512	315	38.5%
HumanEval	Dream	512	271	47.1%

Table 4: Average denoising steps per request and step reduction relative to fixed budget.

7 Conclusion

We present CAI-DLLM, a training-free inference method for diffusion language models that uses first-step confidence to guide token commitment, block schedules, and token budgets without extra models, retraining, or weight updates. Across seven benchmarks on LLaDA-8B-Instruct and Dream-7B-Instruct, CAI-DLLM achieves consistent speedups, up to 44.8×, with the largest gains on longer generation tasks and up to 95.3% energy reduction. These results show that first-step confidence, available at no extra cost, is sufficient to substantially reduce redundant denoising computation while keeping the original model unchanged.

Limitations

CAI-DLLM has several limitations. First, our model-specific confidence gating choice is selected using GSM8K ablations on the full test set: gating is enabled for LLaDA-8B and disabled for Dream-7B. This is a limitation of the current study because the gating decision is not selected on a held out validation set. For deployment on a new model, the gating choice should be selected on a small held out validation subset of a representative task, such as 200–500 samples. Second, the speedups can come with accuracy trade-offs. For example, accuracy drops from 56.75 to 52.33 on BBH with LLaDA-8B, and from 40.00 to 37.00 on MBPP with LLaDA-8B. This makes the method more suitable for settings where lower latency or energy use is important and moderate quality loss is acceptable. Third, all experiments are run on a single NVIDIA H200 GPU, so results may differ on other hardware. Finally, some evaluations use reduced sample sizes to control computation cost, such as 500 samples for MathQA and reduced task subsets for LongBench, which may increase variance.

Ethical Considerations

This work presents a training-free inference acceleration method for masked diffusion language models. It does not add new model capabilities, change model weights, or expand the types of content the models can generate. The main goal is to reduce inference cost and energy use for existing diffusion language model deployments. The ethical risks of this work are therefore mainly the risks of the underlying models, such as LLaDA-8B and Dream-7B. Standard responsible-use practices for large language models still apply. Generative AI tools were used only for language polishing and editing of author written text. They were not used to generate experimental results, technical claims, citations, or research conclusions. All authors reviewed and approved the final manuscript and are responsible for its content. All technical claims, experiments, citations, and final text were checked and approved by the authors.

Code Release. We will release the complete CAI-DLLM implementation, including inference scripts, configuration files, and evaluation code, upon acceptance of this paper.

References

Sudhanshu Agrawal, Risheek Garrepalli, Raghavv Goel, Mingu Lee, Christopher Lott, and Fatih Porikli. 2025. Spiffy: Multiplying diffusion llm acceleration via lossless speculative decoding. *arXiv preprint arXiv:2509.18085*.

Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

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

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021b. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiayi Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, and Yejin Choi. 2020. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few shot learners. In *Advances in Neural Information Processing Systems*.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. In *arXiv preprint arXiv:2110.14168*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h,

Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2023. Language model evaluation harness.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S. Abdelfattah, Jae Sun Seo, Zhiru Zhang, and Udit Gupta. 2025. Flashdlm: Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv preprint arXiv:2505.21467*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*.

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

Aaron Lou, Chenlin Meng, and Stefano Ermon. 2023. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning*.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025. dkv cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji Rong Wen, and Chongxuan Li. 2025. Large language diffusion models. *arXiv preprint arXiv:2502.09992*.

NVIDIA Corporation. 2024. NVIDIA H200 Tensor Core GPU.

Subham Sekhar Sahoo, Mariano Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, and Volodymyr Kuleshov. 2024. Simple and effective masked diffusion language models. *arXiv preprint arXiv:2406.07524*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9).

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, volume 35.

Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2025. Sparse dllm: Accelerating diffusion llms with dynamic cache eviction. *arXiv preprint arXiv:2508.02558*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. 2025. Diffusion llms can do faster than ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*.

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

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

Zijian Zhu, Fei Ren, Zhanhong Tan, and Kaisheng Ma. 2026. Es-dllm: Efficient inference for diffusion large language models by early-skipping. In *International Conference on Learning Representations*.

A Evaluation Setup and Reproducibility Details

This appendix gives the detailed evaluation settings used in Section 5. All compared methods use the same generation length, block length, batch size, temperature, and evaluation split for each benchmark.

Setting	Reasoning	Code	Commonsense	LongBench
Gen. tokens	256	512	256	512
Block length	64	64	64	32
Batch size	8	8	8	8
Temperature	0	0	0	0

Table 5: Generation settings, identical across all compared methods.

Benchmark	Split	Used	Full	Notes
GSM8K	test	1,319	1,319	Full test set
BBH	test	6,511	6,511	Full; 23 subtasks
HumanEval	test	164	164	Full test set
MBPP	test	257	257	Full sanitized set
WinoGrande	validation	500	1,267	Standard eval split
PIQA	validation	1,838	1,838	Full validation set
MathQA	validation	500	4,475	Subset; eval cost
LongBench	test	100	200	Per task; 4 tasks

Table 6: Sample sizes per benchmark. Subsets are used only to control evaluation cost, not due to any limitation of CAI-DLLM.

A.1 Generation Settings

Table 5 summarizes the generation settings used across benchmark groups. For reasoning and commonsense benchmarks (GSM8K, BBH, MathQA, PIQA, WinoGrande), we generate 256 tokens. For code benchmarks (HumanEval, MBPP) and LongBench, we generate 512 tokens. Block length 64 follows the standard block wise decoding setup used in recent diffusion language model inference work (Wu et al., 2025; Zhu et al., 2026). For LongBench, we use block length 32 for finer commit control over longer generations. All experiments use batch size 8 and temperature 0.

For CAI-DLLM, the adaptive threshold schedule uses $w = 0.15$, $\theta_s = 0.90$, and $\theta_e = 0.70$. Confidence gating refers to whether token budgets B_i and force commitment at θ_{fb} are active (ON) or disabled (OFF), while keeping the adaptive threshold and block schedules in both cases. We enable gating for LLaDA and disable it for Dream because Dream's confidence scores are more uniform across token positions, making forced token commitment counterproductive; disabling it gives a better accuracy-speed trade-off on Dream as shown in Table 11 (Appendix B). . The KV cache update frequency follows the ES-dLLM evaluation scripts (Zhu et al., 2026): 16 for LLaDA and 8 for Dream, meaning the cache is refreshed every f denoising steps.

A.2 Hyperparameter Justification

Position aware commit rule. The position aware commit rule uses four factors, $r_{\mathrm{pos}(i)} \in \{0.80, 0.90, 1.05, 1.10\}$. These factors reflect the within block stability pattern observed in our motivation analysis. In LLaDA GSM8K, tokens in the first quarter of a block commit faster than tokens in the last quarter. We therefore use smaller factors for early positions and larger factors for late positions.

The factors are selected on a 200-sample held out subset of LLaDA GSM8K. We choose the setting that improves speed while keeping accuracy degradation below one point. The same factors are then used for Dream-7B without additional tuning.

The clipping range $(0.20, 0.98)$ acts as a safety bound. The upper bound prevents very high thresholds from blocking commitment. The lower bound prevents overly aggressive commitment when thresholds are changed outside the main setting.

A.3 Benchmark Sample Sizes

Table 6 reports the exact number of samples used for each benchmark. Full test or validation sets are used wherever feasible. The two exceptions are MathQA and LongBench, where we use subsets to keep total GPU time within a practical budget: running all four methods across both models on the full MathQA validation set (4,475 samples) or the full LongBench set would require several additional days of H200 compute. The subset sizes (500 for MathQA, 100 per task for LongBench) are standard in prior work on diffusion language model inference (Zhu et al., 2026) and are large enough to produce stable accuracy estimates, as confirmed by the standard errors reported in Table 9.

The subset choice reflects evaluation cost only and does not indicate any limitation of CAI-DLLM on longer or harder inputs. In practice, CAI-DLLM is designed precisely for varied request types: it assigns easy tokens fewer steps and hard tokens more, so it naturally adapts to short and long outputs within the same batch. The LongBench results (Appendix D.2) confirm that CAI-DLLM transfers to long-context inference with a 19.04× speedup and minimal quality loss.

A.4 LongBench Task Selection

LongBench (Bai et al., 2024) is evaluated on Dream-7B only. LLaDA-8B is excluded because its maximum context length is insufficient for most LongBench tasks. We evaluate four tasks: NarrativeQA, MultifieldQA-en, GovReport, and QM-

Task	Metric	Samples	Max input
NarrativeQA	F1	100	4,000
MultifieldQA-en	F1	100	4,000
GovReport	ROUGE-L	100	4,000
QMSum	ROUGE-L	100	4,000
Total	–	400	–

Table 7: LongBench tasks used in evaluation.

Benchmark	Metric	Tool
GSM8K	Exact match	`lm-eval`
BBH	Exact match	`lm-eval`
HumanEval	pass@1	`lm-eval`
MBPP	pass@1	`lm-eval`
MathQA	Exact match	`lm-eval`
WinoGrande	Exact match	`lm-eval`
PIQA	Exact match	`lm-eval`
LongBench	F1 / ROUGE-L	LongBench scripts

Table 8: Evaluation metric and tool per benchmark.

Sum. We remove HotpotQA and Qasper because they showed high variance under the 100-sample setting. Thus, our LongBench result should be interpreted as a selected-task LongBench evaluation rather than the full LongBench suite. All inputs are truncated to 4,000 tokens using `-max_input_len 4000`. Table 7 lists the selected tasks, metrics, sample counts, and input length limit.

A.5 Evaluation Metrics

Table 8 lists the evaluation metric and tool for each benchmark. GSM8K, BBH, MathQA, WinoGrande, and PIQA use exact match. HumanEval and MBPP use pass@1. LongBench uses F1 and ROUGE-L.

A.6 Baselines

We compare against No-cache, DualCache (Wu et al., 2025), and ES-dLLM (Zhu et al., 2026). No-cache is vanilla diffusion decoding without KV caching or parallel decoding. DualCache caches KV activations for both prompt and response tokens. ES-dLLM is run using its official implementation and published settings. We use importance score $\alpha = 0.5$ and proportion steps $[(1.0, 0.0), (0.5, 0.125), (0.25, 0.25)]$. No per-benchmark tuning is performed for any baseline.

FlashDLM (Hu et al., 2025), Sparse dLLM (Song et al., 2025), and D2F (Wang et al., 2025) are not included as direct baselines because compatible public implementations for both LLaDA-8B and Dream-7B were not available at the time of evaluation. Spiffy (Agrawal et al., 2025) is orthogonal to our method because it uses speculative decoding, and a combined comparison is left for future work.

Bench.	Dream acc.	LLaDA acc.	Std. err.
GSM8K	77.26%	77.41%	±1.18 pp
BBH	56.95%	52.33%	±0.53 pp
MBPP	56.60%	37.00%	±2.16 pp
HumanEval	48.17%	35.98%	±3.76 pp

Table 9: CAI-DLLM accuracy and standard errors.

A.7 Statistical Errors and Determinism

Standard errors are computed by `lm-eval` (Gao et al., 2023) as the standard error of the mean across evaluation samples. Table 9 reports the standard errors for CAI-DLLM. Errors for other methods are of similar magnitude.

The largest accuracy difference between CAI-DLLM and No-cache is 4.4 points on LLaDA BBH. The standard error for this result is ±0.53, so the gap is about eight times larger than the estimated uncertainty. This suggests that the BBH difference is unlikely to be caused only by sampling noise as BBH contains multi-step reasoning tasks, where early token commitment can sometimes preserve a wrong intermediate choice and reduce the chance of later correction. HumanEval has higher uncertainty because it has only 164 problems, so its differences should be interpreted with more care.

Accuracy evaluation uses `lm-eval` with fixed seeds: random seed 0, NumPy seed 1234, and torch manual seed 1234. Decoding is deterministic at temperature 0. Timing measurements are taken after one warm-up pass.

A.8 Software Versions

All experiments use PyTorch 2.3.1, CUDA 12.4, Transformers 4.44.0, and Python 3.11. The LLaDA-8B-Instruct and Dream-7B-Instruct weights are loaded from HuggingFace Hub using official checkpoints. Flash Attention 2 is enabled for all methods. CUDA deterministic mode is enabled for all runs. Results may differ on earlier PyTorch versions due to different CUDA kernel availability on H200.

A.9 Compute Budget

All experiments use a single NVIDIA H200 GPU. As shown in Table 10, total inference compute across all benchmarks, models, and methods is ap-

Method	Hours	% Total
No cache	77.38	77.9%
ES-dLLM	8.19	8.2%
DualCache	8.07	8.1%
CAI-DLLM	3.89	3.9%
CAI-DLLM no CG	1.85	1.9%
Total	**99.38**	100%

Table 10: Inference GPU-hours by method. CG denotes confidence gating.

Model	Config.	Acc.	TPS
LLaDA	ES-dLLM	77.10	175.3
	+APD	76.80	490.2
	+APD+Block	76.95	**496.0**
	+APD+Block+Budget ⋆	**77.41**	335.1
	Full CAI, gate off	76.57	320.0
Dream	ES-dLLM	**77.48**	97.9
	+APD	75.36	372.9
	+APD+Block	75.51	**383.0**
	+APD+Block+Budget	75.51	**383.0**
	Full CAI, gate off ⋆	77.26	369.5

Table 11: GSM8K ablation of CAI-DLLM components. ⋆ marks the recommended configuration for each model.

proximately 99.4 GPU-hours. No-cache inference alone requires 77.4 hours, or 77.9% of the total. In comparison, CAI-DLLM completes the same evaluation in 3.9 hours, reducing inference compute by 95.0% and saving 73.5 GPU-hours relative to no-cache decoding.

B Additional Ablation Results

B.1 GSM8K Ablation

Table 11 reports the GSM8K component ablation. The results follow the same pattern as the HumanEval ablation in Section 6.2. APD gives the main throughput gain for both models. The per-block schedule gives a small additional gain. For LLaDA, adding token budgets with confidence gating improves accuracy to 77.41%, which is higher than ES-dLLM. For Dream, confidence gating does not improve accuracy or speed, so the best configuration keeps gating off.

C Additional Background and Motivation Analysis

This appendix provides additional evidence for the observations in Section 2. The main paper focuses on the most direct motivation: denoising behavior is uneven across layers, blocks, and tokens. Here, we include extra analysis on attention sparsity, APD schedule sensitivity, and hardware behavior.

Attention sparsity across layers. LLaDA-8B has 32 transformer layers (L0–L31). We sample six layers at roughly equal intervals (L0, L4, L8, L16, L24, L31) to represent early, middle, and late processing stages without profiling all 32 layers at full cost.

Figure 4 shows three complementary signals across these layers. First, attention entropy decreases in deep layers (L24, L31) and increases in shallow layers (L0, L4) as denoising progresses, showing that deeper layers become progressively more selective over time. Second, the number of effective tokens attended drops sharply in deep layers, confirming that later layers concentrate on fewer positions while early layers attend broadly. Third, the top-10 attention mass is highest in deep layers, indicating that a small set of tokens captures most of the attention weight in those layers.

Together, these observations show that compute requirements differ substantially across layers: early layers process all tokens broadly while deep layers focus selectively. A single uniform compute policy across all layers is therefore wasteful, which motivates layer-aware or token-aware inference strategies such as CAI-DLLM.

APD schedule sensitivity. Figure 5 shows that decoding accuracy depends on the confidence schedule. Fixed thresholds and overly aggressive schedules can reduce accuracy. The selected cosine schedule gives the best accuracy-speed trade-off among the tested settings. This supports using an adaptive threshold schedule rather than a single fixed threshold.

Hardware behavior. Figure 6 compares throughput, latency, bandwidth use, and compute use across batch sizes on the H200 GPU. The four curves correspond to different decoding configurations: `ES-dLLM (no par.)` is ES-dLLM without parallel decoding; `Parallel th=0.9` and `Parallel th=0.7` are confidence based parallel decoding with fixed commit thresholds of 0.9 and 0.7 respectively, used here as ablation points to understand the effect of threshold choice on hardware utilization; and `APD C 0.9→0.5` is our adaptive cosine schedule, which starts at 0.9 and relaxes to 0.5. These are not separate systems but configurations used to profile hardware behavior under different decoding policies.

As batch size increases, compute utilization (bottom-right panel) grows much more steeply than bandwidth utilization (bottom-left panel) across all configurations. At batch size 8, compute utilization exceeds 150–350% of baseline while bandwidth

utilization stays below 16%. This gap confirms that the bottleneck is compute, not memory bandwidth. On a compute bound workload, reducing the number of denoising steps directly reduces wall-clock time and energy, which is the primary optimization axis of CAI-DLLM.

D Additional Results and Analysis

This appendix reports memory usage, task-level LongBench results, and detailed energy measurements. These results support the main paper and show that CAI-DLLM improves efficiency without large memory overhead.

D.1 Memory Usage

Table 12 reports peak GPU memory on HumanEval. HumanEval is used for profiling because its 164 problems make repeated profiling practical. Memory use is mainly determined by model weights and KV cache size, so the results are representative across tasks.

For LLaDA, CAI-DLLM uses 32,787 MB, only 480 MB more than DualCache (1.5% overhead). This extra memory comes from three small components: per-token first-step confidence scores s_i ($\mathcal{O}(L)$ floats), token step budgets B_i ($\mathcal{O}(L)$ integers), and the position aware threshold lookup table ($\mathcal{O}(L)$ floats). Together these are negligible relative to the model weights and KV cache.

For Dream, CAI-DLLM uses 36,533 MB, which is 5,822 MB more than DualCache but still 7,500 MB below no-cache decoding. The larger increase is not caused by the confidence controller itself. Dream uses more aggressive parallel decoding, which fills a larger output token buffer simultaneously. Specifically, the adaptive threshold schedule commits more tokens in parallel per step on Dream than on LLaDA, increasing the size of the active output buffer. The CAI-DLLM bookkeeping overhead is the same $\mathcal{O}(L)$ as for LLaDA; the difference is entirely due to Dream's parallel decoding behavior.

D.2 Detailed LongBench Results

Table 13 reports task-level LongBench results on Dream-7B. Each task uses its own metric: F1 for NarrativeQA and MultifieldQA-en, and ROUGE-L for GovReport and QMSum. The average score is the unweighted mean of these four task scores, where each score is on a 0–100 scale.

CAI-DLLM increases average throughput from 2.7 to 51.4 tokens per second, giving a 19.04× speedup over no-cache decoding. The average score decreases from 24.63 to 23.11, a drop of 1.52 points, which is similar to the drops seen for DualCache (1.10 points) and ES-dLLM (0.94 points). This shows that all caching methods reduce long-context quality by a similar margin, while CAI-DLLM provides more than double their throughput. On QMSum, CAI-DLLM slightly improves over both cached baselines (14.66 vs. 14.59 and 14.38), suggesting that for summarization tasks the confidence aware commit schedule does not hurt quality. The larger drop on MultifieldQA-en (43.72 vs. 47.95 for ES-dLLM) indicates that multi-field retrieval tasks are more sensitive to early token commitment. Overall, these results confirm that confidence aware decoding transfers to long-context inference with competitive quality and substantially higher throughput.

Model	Method	Peak Mem.	vs DualCache
LLaDA	No cache	41,567 MB	+9,260 MB
LLaDA	DualCache	32,307 MB	baseline
LLaDA	ES-dLLM	32,503 MB	+196 MB
LLaDA	CAI-DLLM	32,787 MB	+480 MB
Dream	No cache	44,033 MB	+13,322 MB
Dream	DualCache	30,711 MB	baseline
Dream	ES-dLLM	30,807 MB	+96 MB
Dream	CAI-DLLM	36,533 MB	+5,822 MB

Table 12: Peak GPU memory on HumanEval. The extra memory for CAI-DLLM over DualCache comes from per-token confidence scores, budgets, and threshold tables. Dream's larger overhead is due to fuller parallel output generation, not CAI-DLLM bookkeeping.

Task	Metric	No cache	DualCache	ES-dLLM	CAI-DLLM
NarrativeQA	F1	18.18	16.96	17.05	16.96
MultifieldQA-en	F1	47.24	45.35	47.95	43.72
GovReport	ROUGE-L	19.99	17.23	16.26	17.09
QMSum	ROUGE-L	13.69	14.59	14.38	14.66
Average score		24.63	23.53	23.69	23.11
Average TPS		2.7	23.4	24.2	51.4
Speedup		1.00×	8.7×	9.0×	19.04×

Table 13: Detailed LongBench results on Dream-7B. Average score is the unweighted mean of per-task scores.

D.3 Short Commonsense Tasks

On PIQA and WinoGrande, CAI-DLLM improves throughput but loses accuracy compared with ES-dLLM. These tasks have short outputs, so there are fewer redundant denoising steps to remove. In this setting, early commitment can force incorrect tokens before enough context is available. Thus, CAI-DLLM is less suitable for short, quality-critical tasks where the small speedup does not justify the accuracy cost.

D.4 Energy Measurements

Energy is computed as $E = \bar{P} \times T$, where $\bar{P}$ is the average GPU power and T is wall-clock generation time. We report energy in both Joules and Watt-hours, with $\mathrm{Wh} = E/3600$. GPU power is sampled using `nvidia-smi`. These measurements include GPU power only and do not include CPU, DRAM, or system level power.

Table 14 shows that CAI-DLLM reduces total energy in all measured cases. The main reason is shorter generation time. For example, LLaDA GSM8K drops from 3559.1 Wh to 168.7 Wh, a 95.3% reduction.

E Throughput Benchmark Results

We report wall-clock throughput in tokens per second (TPS) and speedup over the `nocache` baseline. We evaluate four generation lengths, $L \in \{64, 128, 256, 512\}$, on two instruction tuned diffusion LLMs: LLaDA-Instruct (Nie et al., 2025) and Dream-Instruct (Ye et al., 2025). All experiments use a single NVIDIA H200 GPU under the same hardware and software setup. For each generation length, speedup is computed as the ratio between a method's TPS and the `nocache` TPS.

As shown in Table 15, CAI-DLLM achieves the highest throughput in every setting across both models. On LLaDA-Instruct, speedup ranges from 2.07× at $L = 64$ to 4.59× at $L = 512$. This shows that longer generation lengths give more room for adaptive step reduction. On Dream-Instruct, the gain is larger and reaches 15.29× over `nocache` at $L = 512$ (692.6 vs. 45.3 TPS). In contrast, `dualcache` and `esdllm` do not improve throughput at short generation lengths and sometimes slightly reduce it. This happens because their fixed caching overheads are not fully amortized when $L \in \{64, 128\}$. Overall, these results show that CAI-DLLM is most effective at longer sequence lengths, where token difficulty varies more and early commitment saves more computation.

Task	Model	Method	Avg Power	Energy (J)	Energy (Wh)	Saved
GSM8K	LLaDA	No cache	688.0 W	12,812,627	3559.1	–
GSM8K	LLaDA	DualCache	587.2 W	950,104	263.9	92.6%
GSM8K	LLaDA	ES-dLLM	441.7 W	801,455	222.6	93.7%
GSM8K	LLaDA	CAI-DLLM	558.3 W	607,297	168.7	95.3%
HumanEval	Dream	No cache	673.4 W	1,347,290	374.25	–
HumanEval	Dream	DualCache	550.5 W	215,714	59.92	84.0%
HumanEval	Dream	ES-dLLM	358.9 W	149,309	41.47	88.9%
HumanEval	Dream	CAI-DLLM	464.1 W	85,134	23.65	93.7%
HumanEval	LLaDA	No cache	672.2 W	1,530,033	425.01	–
HumanEval	LLaDA	DualCache	529.9 W	197,734	54.93	87.1%
HumanEval	LLaDA	ES-dLLM	348.6 W	152,845	42.46	90.0%
HumanEval	LLaDA	CAI-DLLM	474.0 W	118,921	33.03	92.2%
MBPP	Dream	No cache	678.3 W	3,123,006	867.5	–
MBPP	Dream	DualCache	380.0 W	375,572	104.3	88.0%
MBPP	Dream	ES-dLLM	378.1 W	281,825	78.3	91.0%
MBPP	Dream	CAI-DLLM	412.6 W	122,798	34.1	96.1%
MBPP	LLaDA	No cache	676.3 W	3,704,466	1029.0	–
MBPP	LLaDA	DualCache	413.4 W	349,628	97.1	90.6%
MBPP	LLaDA	ES-dLLM	371.7 W	291,715	81.0	92.1%
MBPP	LLaDA	CAI-DLLM	434.4 W	180,389	50.1	95.1%

Table 14: Detailed energy measurements. The Saved column reports reduction relative to the no-cache row for the same task and model.

Model	Method	$L = 64$		$L = 128$		$L = 256$		$L = 512$	
		TPS	Speedup	TPS	Speedup	TPS	Speedup	TPS	Speedup
LLaDA	No cache	52.3	1.00×	53.8	1.00×	54.0	1.00×	40.6	1.00×
	DualCache	51.3	0.98×	50.2	0.93×	50.8	0.94×	51.4	1.27×
	ES-dLLM	45.4	0.87×	46.9	0.87×	46.8	0.87×	46.9	1.16×
	CAI-DLLM	**108.0**	**2.07**×	**202.1**	**3.76**×	**123.6**	**2.29**×	**186.3**	**4.59**×
Dream	No cache	61.5	1.00×	63.1	1.00×	61.4	1.00×	45.3	1.00×
	DualCache	62.4	1.01×	63.5	1.01×	64.3	1.05×	64.0	1.41×
	ES-dLLM	57.5	0.93×	58.7	0.93×	59.7	0.97×	59.6	1.32×
	CAI-DLLM	**98.6**	**1.60**×	**199.2**	**3.16**×	**242.9**	**3.96**×	**692.6**	**15.29**×

Table 15: Throughput and speedup over No cache across generation lengths on LLaDA-Instruct and Dream-Instruct. Best TPS for each generation length is shown in bold.

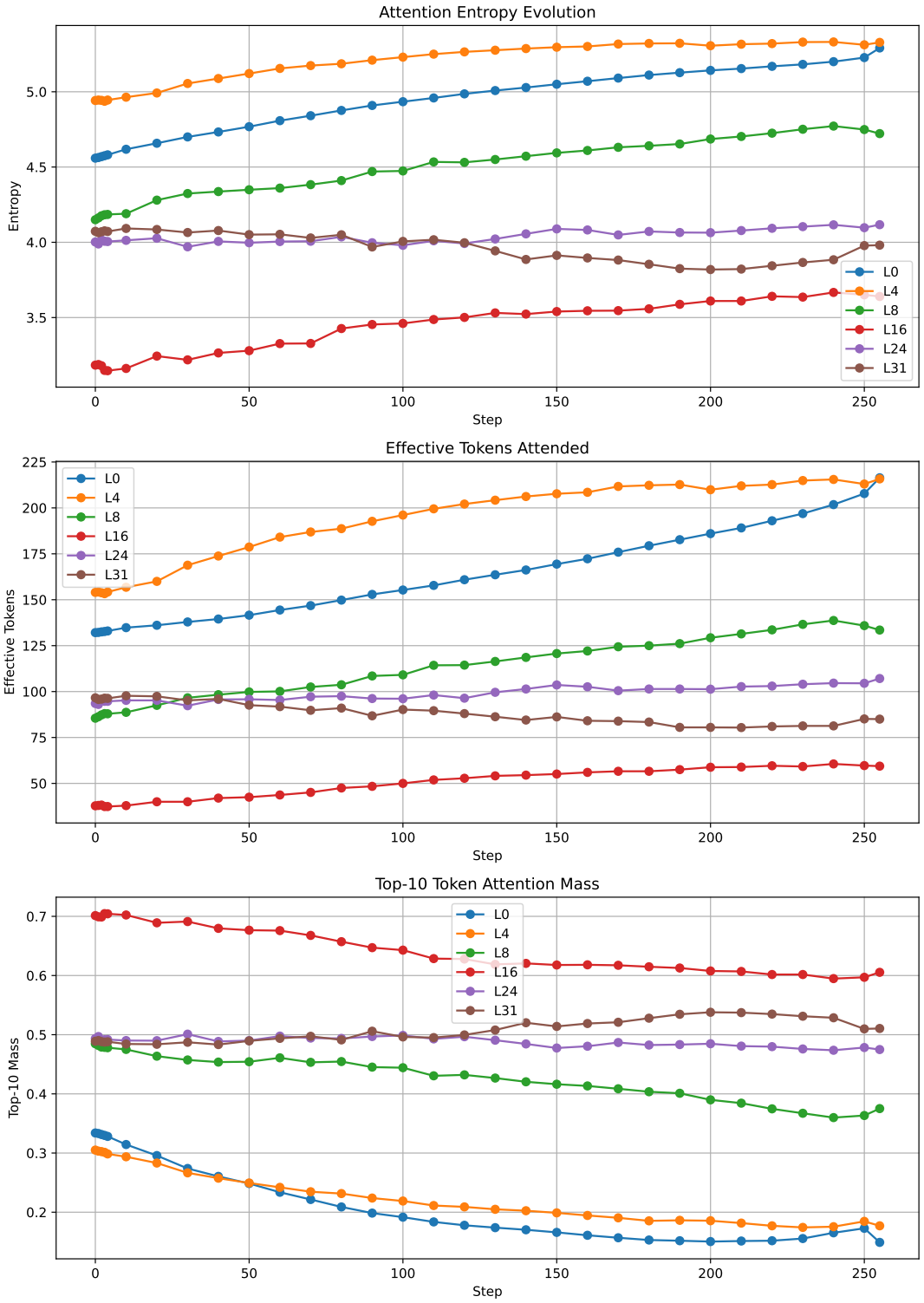

Figure 4: Attention sparsity across denoising steps for sampled layers of LLaDA-8B on GSM8K. We sample early, middle, and late transformer layers. The plots show attention entropy, effective tokens attended, and top-10 attention mass. The strong layer-to-layer variation supports layer-aware analysis of diffusion language model inference.

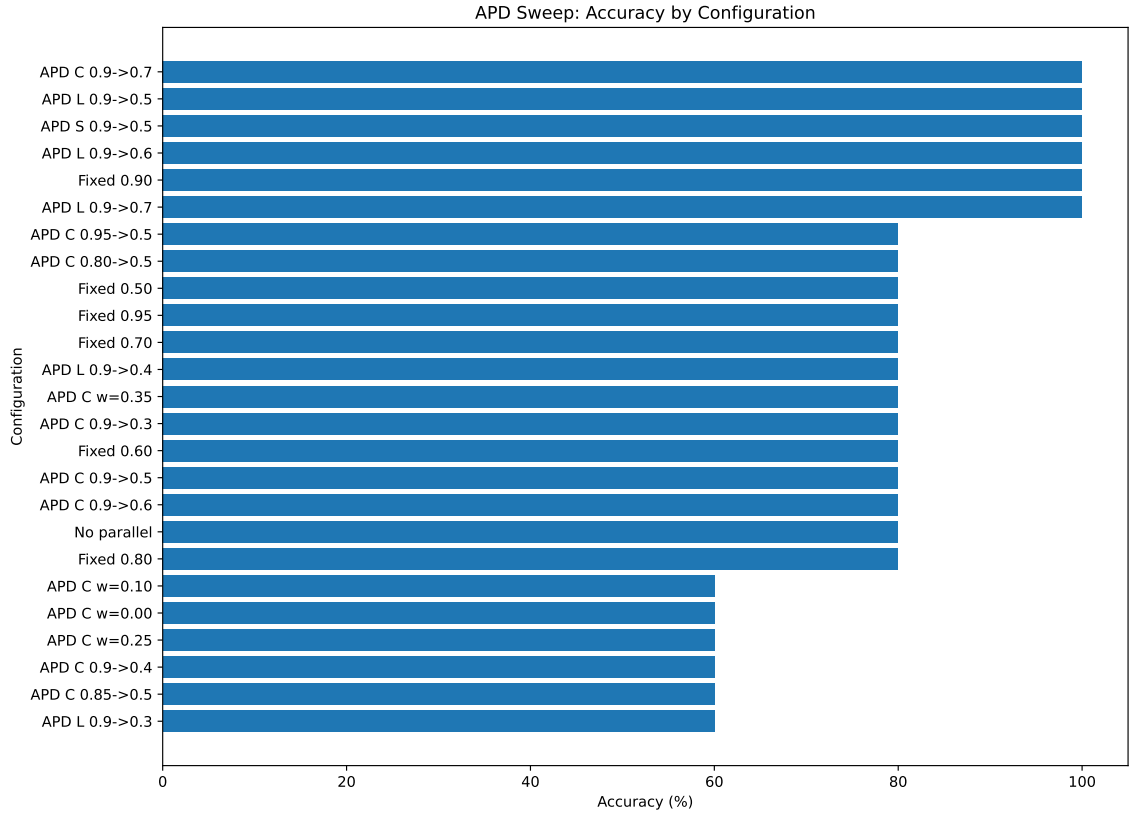

Figure 5: Accuracy across APD schedule configurations on LLaDA-8B GSM8K. The selected cosine schedule with $\theta_s = 0.90$, $\theta_e = 0.70$, and $w = 0.15$ gives the best accuracy-speed trade-off.

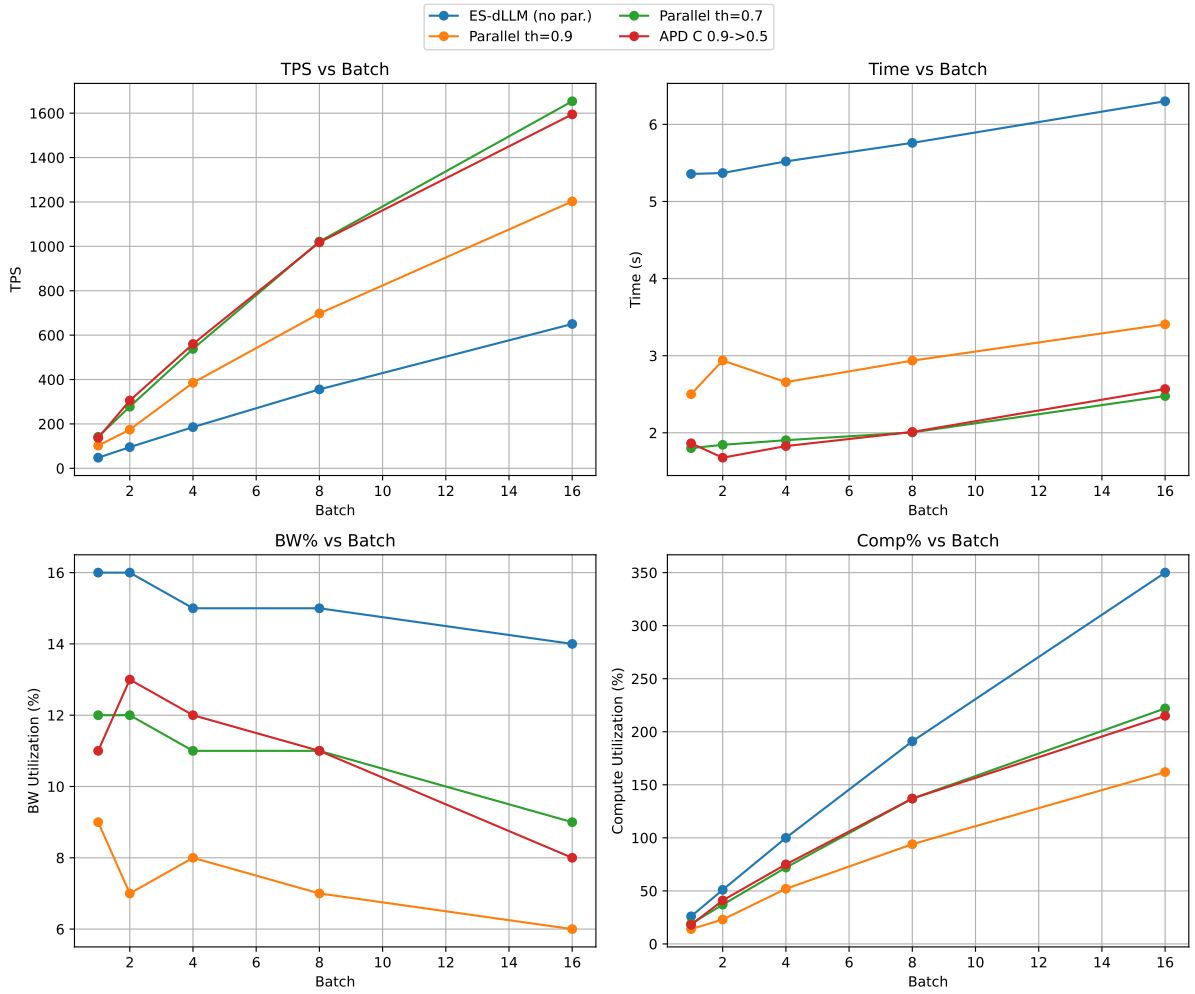

Figure 6: Hardware behavior across batch sizes. Fixed-threshold parallel decoding and APD are compared with ES-dLLM. The H200 runs the workload in a compute-bound regime, making denoising-step reduction an effective optimization target.