CDLM: CONSISTENCY DIFFUSION LANGUAGE MODELS FOR FASTER SAMPLING

Minseo Kim [1] **Chenfeng Xu** [2 3] **Coleman Hooper** [2] **Harman Singh** [2] **Ben Athiwaratkun** [3] **Ce Zhang** [3]
Kurt Keutzer [2] **Amir Gholami** [2]

ABSTRACT

Diffusion Language Models (DLMs) offer a promising parallel generation paradigm but suffer from slow inference due to numerous refinement steps and the inability to use standard KV caching. We introduce CDLM (Consistency Diffusion Language Models), a training-based acceleration method that simultaneously tackles both bottlenecks. CDLM integrates consistency modeling to drastically reduce the number of required sampling steps by enabling multi-token finalization. Furthermore, we enforce a block-wise causal attention mask during fine-tuning, making the model fully compatible with KV caching. Experiments show CDLM achieves $3.6\times$–$14.5\times$ lower latency while maintaining competitive accuracy on math and coding tasks. The full training and evaluation code is available at https://github.com/SqueezeAILab/CDLM.

1 INTRODUCTION

Large Language Models (LLMs) have triggered a paradigm shift in AI, excelling at tasks such as instruction following, multi-turn dialogue, and code generation (Achiam et al., 2023). Most current LLMs are autoregressive, decoder-only Transformers trained to predict the next token under a causal attention mask (Grattafiori et al., 2024; Zhang et al., 2022). While training is parallelizable via teacher forcing, inference inherits a rigid sequential dependency, becoming a key efficiency bottleneck. Moreover, such models cannot exploit bidirectional context, which limits performance on tasks such as text infilling, editing, and global planning (Hu et al., 2024; Song et al., 2025).

As a promising alternative, diffusion language models (DLMs) have drawn attention for their potential in text generation. Inspired by the success of diffusion models in image, audio, and video synthesis (Ho et al., 2020), DLMs iteratively denoise random or masked token sequences into coherent text (Gong et al., 2023; Li et al., 2022). This process updates all token positions *in parallel* per step, removing the token-level sequential dependency inherent to autoregressive decoding. Closed-source models, primarily in code generation, report up to $10\times$ higher throughput than autoregressive models while preserving quality (Google DeepMind, 2025; Khanna et al., 2025; Song et al., 2025). However, open-source DLMs remain significantly slower (Nie et al., 2025b; Ye et al., 2025), largely because (i) bidirectional attention prohibits standard KV caching and (ii) they demand many refinement steps, often proportional to sequence length, to reach high fidelity (Kim et al., 2025).

Many recent works aim to accelerate DLM inference. One primary strategy leverages block-wise decoding with KV caching, either via approximate caching with periodic updates (Hu et al., 2025; Liu et al., 2025b; Ma et al., 2025a; Wu et al., 2025b) or by fine-tuning the model into a block-wise causal architecture so that caching becomes natural (Wang et al., 2025; Wu et al., 2025a). The other major line of work focuses on parallel sampling algorithms, which effectively decode multiple tokens per denoising step and reduce the total number of iterations (Ben-Hamu et al., 2025; Wu et al., 2025b).

In diffusion models for vision domains, many works have successfully reduced denoising to few-step or even one-step generation (Yin et al., 2024; 2025). A key technique behind this progress is *consistency modeling*, which trains models to produce consistent predictions across adjacent denoising states along the probability flow trajectory (Song et al., 2023; Song & Dhariwal, 2023). Although originally developed for continuous data, this paradigm has begun to influence discrete settings, including parallel decoding for autoregressive language models (Kou et al., 2024) and masked diffusion frameworks (Xu et al., 2025), highlighting its promise for language diffusion models.

In this work, we present **CDLM**, a training-based accelera-

[1]Seoul National University, Seoul, South Korea [2]University of California, Berkeley, CA, USA [3]Together AI, San Francisco, CA, USA. Correspondence to: Amir Gholami <amirgh@berkeley.edu>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

tion method that integrates **C**onsistency modeling (Song et al., 2023; Song & Dhariwal, 2023) into **D**iffusion **L**anguage **M**odels. CDLM trains a block-causal student via distillation from a bidirectional teacher to finalize multiple tokens per refinement step, and applies confidence-thresholded parallel decoding within each block at inference time. This design directly addresses two core bottlenecks of current DLMs: (i) excessive refinement steps (mitigated by distillation and consistency) and (ii) cache inefficiency (removed by block-wise causality). In particular, we make the following contributions:

- **Consistency-guided distillation.** We bring consistency modeling to DLMs by defining a token-level decoding trajectory and coupling it with distillation from a fully bidirectional teacher. This joint objective provides explicit supervision for multi-token finalization while aligning predictions between a less-informed state and a block-completion state within each block (Sections 3, 4.2).
- **Cache-friendly block causality.** By fine-tuning with a block-wise causal attention mask (Figure 2), we make bidirectional DLMs compatible with block-wise KV caching and early stopping at block boundaries (Sections 4.1, 4.3). We also provide a detailed system-level analysis of block-causal DLMs to characterize their efficiency and scalability (Section 5.4).
- **End-to-end speedups.** With 8 hours of training for Dream and 16 hours for LLaDA, CDLM reduces refinement steps by $3.4\times$–$7.9\times$ and latency by $3.6\times$–$14.5\times$ while maintaining accuracy across math and coding benchmarks. It also surpasses equal-size autoregressive LLMs in tokens-per-second (Section 5.2).

2 Background

2.1 Diffusion Language Models

Diffusion models (Ho et al., 2020) have been successfully applied across modalities such as text-conditioned image generation (Ramesh et al., 2022), audio synthesis (Chen et al., 2021b), and video generation (Esser et al., 2023). Early Diffusion Language Models (DLMs) operated in the continuous embedding space (Gao et al., 2024; Gong et al., 2023), but more recent models adopt masked diffusion models (MDMs) that work directly in the discrete token space (Gong et al., 2025a; Nie et al., 2025a). These models begin from a fully masked sequence and iteratively unmask tokens according to a scheduling policy. Architecturally, DLMs commonly employ Transformer backbones with *bidirectional* (non-causal) attention, permitting full-context interactions (Nie et al., 2025b). Despite their parallel update capability, DLMs are known to be slow for two main reasons. First, full bidirectional attention at every denoising step makes DLMs incompatible with standard KV caching. Second, high-quality generation demands a large number of denoising steps comparable to the sequence length, making inference inefficient (Kim et al., 2025). In this work, we propose a fine-tuning-based solution that addresses both limitations.

2.2 Efficient Inference in Diffusion Language Models

Inference in DLMs is an active area of research with significant room for improvement. Among training-free acceleration methods, one line of work exploits approximate caching via block-wise decoding, only recomputing the active block while reusing stale key–value states for inactive blocks (Ma et al., 2025a; Wu et al., 2025b). Others speed up parallel sampling by applying confidence thresholds (Wu et al., 2025b) or by leveraging lightweight autoregressive models as guidance (Hu et al., 2025), while recent analysis suggests that effective parallelism is difficult to adapt reliably at inference time (Kang et al., 2025). In contrast, training-based approaches focus on improving the intrinsic efficiency of DLMs through architectural or objective-level modifications. Some recent works fine-tune pretrained AR models or DLMs to introduce block-wise causality, enabling standard KV caching at the block level while retaining diffusion-style updates within each block (Wang et al., 2025; Wu et al., 2025a). Others design objectives that promote faster convergence by encouraging high-confidence token prediction (Chen et al., 2025). Our method is training-based: we jointly induce block-wise causality and reduce refinement steps through consistency-guided fine-tuning.

2.3 Knowledge Distillation in Language Models

Knowledge distillation (KD) (Hinton et al., 2015) transfers knowledge from a large, powerful teacher to a smaller, more efficient student. Early work on LLM distillation primarily relied on *black-box* KD, where students imitated textual outputs from closed-source APIs (Fu et al., 2023). With the rise of open-source LLMs, *white-box* distillation has become prevalent, enabling richer supervision from the teacher's internal representations, such as logits or embeddings (Agarwal et al., 2024; Gu et al., 2023). Beyond language, distillation has also been applied to diffusion-based image and video generation to reduce sampling steps while preserving quality (Yin et al., 2024; 2025). Our method can similarly be viewed as a form of *self-distillation*, where knowledge is transferred between models of identical size and type but with different architectural designs (a fully bidirectional teacher and a block-wise causal student). Like Deschenaux & Gulcehre (2025) and Ma et al. (2025b), we reduce decoding steps by distilling multi-step diffusion trajectories into *jumps* between non-consecutive denoising states. Our novelty lies in instantiating these jumps within a block-causal student and training it with *consistency*-oriented objectives.

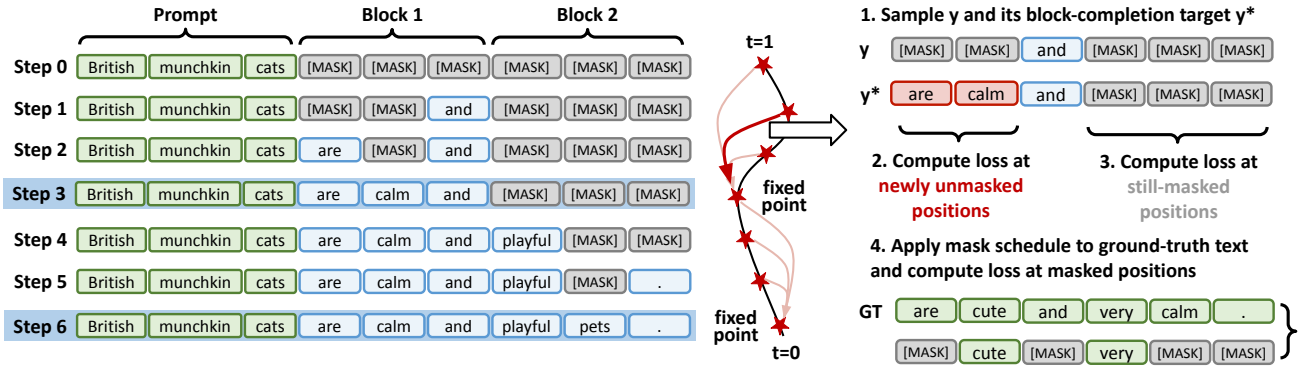

Figure 1. **Left:** Block-wise decoding trajectory of the teacher (steps $0 \to N$; diffusion time $t : 1 \to 0$). **Right:** The student's three-objective loss at an intermediate state y: (i) distillation from teacher logits on newly unmasked positions, (ii) consistency between y and its block-completion $y^\star$, and (iii) masked-denoising (DLM) loss on randomly masked ground-truth text.

2.4 Consistency Models and Language Models

Consistency models (Song et al., 2023) were introduced to overcome the slow, iterative sampling process of diffusion models. They learn a *consistency* property: any intermediate state along a probability flow ODE trajectory can be mapped directly back to its origin (the clean data) (Song & Dhariwal, 2023). This is achieved by minimizing discrepancies between predictions at adjacent denoising states. While originally developed for continuous domains such as vision, recent work has explored adapting this idea to discrete language modeling (Kou et al., 2024; Xu et al., 2025). Rather than explicitly modeling the ODE solver or its continuous trajectory, these methods reinterpret the notion of mapping intermediate states to final solutions within discrete token trajectories (Santilli et al., 2023). Our approach adopts this paradigm in DLMs, where trajectories arise naturally from iterative denoising. Conceptually, our objective can be viewed as an empirical generalization of the original consistency objective defined over continuous ODE trajectories, adapted to discrete, token-level diffusion processes.

3 Preliminary: Inference in Diffusion Language Models

We formalize DLM inference by defining the iterative refinement process, the sampling strategies, and the decoding trajectory that underpins our training objectives.

Iterative Refinement Process. DLM generation is an iterative refinement over N discrete sampling steps. It gradually transforms a fully masked sequence $\mathbf{x}_1$ at time $t = 1$ (step 0) into a clean sequence $\mathbf{x}_0$ at $t = 0$ (step N). Here, N serves as a hyperparameter controlling the trade-off between generation speed and sample quality. We denote the sequence at a discrete step k (where $k = 0, 1, \ldots, N$) as $\mathbf{x}_{t_k}$ with $t_k = 1 - \frac{k}{N}$ (so t_0=1, t_N=0). At each step, approximately L_g/N tokens are finalized on average, where L_g is the target generation length.

This refinement process is modeled probabilistically. Specifically, the transition from a state $\mathbf{x}_t$ to a subsequent state $\mathbf{x}_s$ $0 \le s < t \le 1$ is governed by the model p_θ's prediction of the clean sequence $\mathbf{x}_0$ given the current noisy sequence $\mathbf{x}_t$ and a conditioning prompt c, i.e.,

$$p_\theta(\mathbf{x}_0 \mid \mathbf{x}_t, c). \tag{1}$$

Following prior formulations (Austin et al., 2021a), the reverse-time transition $q_{s|t}(\mathbf{x}_s \mid \mathbf{x}_t)$ is defined for $0 \le s < t \le 1$ and factorizes across tokens. For each token i, the transition probability is defined as:

$$q_{s|t}(x_s^i \mid x_t^i) = \begin{cases} 1, & \text{if } x_t^i \neq \texttt{[MASK]},\ x_s^i = x_t^i, \\ \frac{s}{t}, & \text{if } x_t^i = \texttt{[MASK]},\ x_s^i = \texttt{[MASK]}, \\ \frac{t-s}{t}\, q_{0|t}(x_s^i \mid \mathbf{x}_t, c), & \text{if } x_t^i = \texttt{[MASK]},\ x_s^i \neq \texttt{[MASK]}. \end{cases} \tag{2}$$

where $q_{0|t}$ is the predictive distribution induced by $p_\theta(\mathbf{x}_0 \mid \mathbf{x}_t, c)$. This formulation reflects three token-level transitions: preserving an unmasked token, keeping a token masked, or unmasking it to a new token.

Sampling Strategies. To instantiate the probabilistic distribution $q_{s|t}$ into an actual generation procedure, practical deterministic strategies are employed. A common choice is *low-confidence remasking*, where instead of stochastic sampling, the model greedily selects the tokens to unmask (Nie et al., 2025b). Based on the confidence scores from $p_\theta(\mathbf{x}_0 \mid \mathbf{x}_t, c)$, the top-$m$ masked tokens with the highest probabilities are revealed, while all others remain

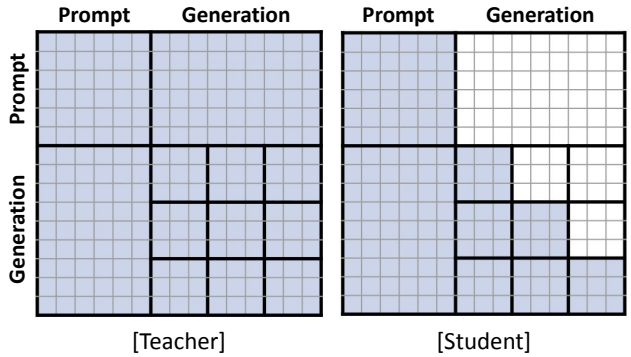

Figure 2. **Left:** Teacher DLM with full bidirectional attention, attending to the entire context. **Right:** Student DLM with a block-wise causal mask, attending to the prompt, previously completed blocks, and the current decoding block.

masked. In practice, this process is often constrained to operate within *blocks* of tokens, called *block-wise decoding*, limiting unmasking to localized regions of the sequence. This prevents unnatural early generation (Nie et al., 2025b) and, more importantly, enables KV caching for finalized blocks. Parallel denoising implicitly assumes *independence* among simultaneously finalized tokens, an idealized assumption that can degrade generation quality as more tokens are unmasked at once (Liu et al., 2025a).

Decoding Trajectory. Given this iterative and probabilistic refinement process, we define the *decoding trajectory* $\mathcal{T}$ as the sequence of states that the model traverses during inference:

$$\mathcal{T}_{\mathbf{x}} = \left(\mathbf{x}_{t_0}, \mathbf{x}_{t_1}, \ldots, \mathbf{x}_{t_N}\right), \quad t_k = 1 - \frac{k}{N}. \tag{3}$$

Each $\mathbf{x}_{t_k}$ is a partially refined sequence; the trajectory $\mathcal{T}_{\mathbf{x}}$ records its evolution across steps. Figure 1 (left) visualizes an example. This trajectory serves as the foundation for our consistency-based training below.

4 METHODOLOGY

We present **Consistency Diffusion Language Models (CDLM)**, a training scheme that accelerates DLM inference without sacrificing quality. We describe trajectory and hidden-state collection, our three-objective training, and the inference strategy.

4.1 Trajectory Collection for CDLM

Let p_θ denote the teacher DLM and $q_\phi(\cdot \mid x)$ the student DLM initialized from the teacher's weights. The teacher uses a *bidirectional* attention mask as in standard DLMs, whereas the student is trained with a *block-wise causal* mask, as illustrated in Figure 2. To enable consistency training,

Algorithm 1 Trajectory Collection for Training CDLM

Input: dataset $\mathcal{O}$ of pairs $(x, \hat{\mathbf{y}})$, generation length L_g, block size B, total steps $N{=}L_g$, teacher DLM p_θ, temperature set $\mathcal{S}_\tau$, target count n
Output: dataset $\mathcal{D}$ of quadruples $(x, \hat{\mathbf{y}}, \mathcal{T}_x, \mathbf{H}_x)$
Initialize $\mathcal{D} \leftarrow \emptyset$
repeat
  Sample $(x, \hat{\mathbf{y}}) \sim \mathcal{O}$; build input ids $\mathbf{x}$ with the answer span masked
  for each temperature $\tau \in \mathcal{S}_\tau$ **do**
    $\mathcal{T} \leftarrow [\,]$; $\mathbf{H} \leftarrow \mathbf{0} \in \mathbb{R}^{L_g \times d}$
    for blocks $b = 1, \ldots, \lceil L_g/B \rceil$ **do**
      for steps $s = 1, \ldots, B$ **do**
        Run $p_\theta(\mathbf{x})$; obtain logits ℓ and last hidden $\mathbf{h}$
        Sample $\mathbf{y}$ and compute confidence $\mathbf{c}$
        Finalize top-1 token y_i in the current block; write $\mathbf{h}_i \to \mathbf{H}$; append $\mathbf{x} \to \mathcal{T}$
      end for
    end for
    Append $(x, \hat{\mathbf{y}}, \mathcal{T}, \mathbf{H})$ to $\mathcal{D}$
  end for
until $|processed\_pairs| \geq n$

we collect trajectories offline by running the teacher on domain-specific prompts. For each prompt x, we denote the token-level trajectory as $\mathcal{T}_x$ and the corresponding hidden-state buffer by $\mathbf{H}_x \in \mathbb{R}^{L_g \times d}$; the resulting dataset is $\mathcal{D} = \{(x, \hat{\mathbf{y}}, \mathcal{T}_x, \mathbf{H}_x)\}$ with ground-truth text $\hat{\mathbf{y}}$ included. The overall procedure is summarized in Algorithm 1.

Prior work reports that block-wise decoding is effective for instruction-tuned models and attains high quality when the number of steps matches the generation length (Nie et al., 2025b). Accordingly, we adopt block-wise decoding with the number of sampling steps equal to the generation length ($N{=}L_g$) and finalize exactly one token per step within the current block (the highest-confidence token y_i). This configuration places the teacher at its most performant operating point and yields higher-quality trajectories. For white-box distillation, we store not only $\mathcal{T}_x$ but also the states required to reconstruct logits. Concretely, we record the last hidden states at the moments when tokens are finalized. We also augment each prompt with multiple trajectories generated at different temperatures. Additional details on dataset construction are provided in Appendix A.1.

4.2 CDLM Training

We jointly minimize three objectives to train CDLMs: (i) a **Distillation loss** that transfers the teacher's multi-token finalization signal to the block-wise causal student; (ii) a **Consistency loss** that enforces within-block temporal consistency between two states; and (iii) a **DLM loss** that pre-

Algorithm 2 Training Algorithm for CDLM

Input: dataset $\mathcal{D} = \{(x, \hat{\mathbf{y}}, \mathcal{T}_x, \mathbf{H}_x)\}$, block size B, generation length L_g, weights $w_{\text{cons}}, w_{\text{distill}}, w_{\text{dlm}}$, student DLM $q_\phi(\cdot \mid x)$ *with block-wise causal attention*, total steps $N{=}L_g$
Output: updated student parameters ϕ
repeat
  Sample $(x, \hat{\mathbf{y}}, \mathcal{T}_x, \mathbf{H}_x) \sim \mathcal{D}$
  Sample t_{start}; set $t_{\text{end}} \leftarrow \min\big(N, \lceil \frac{t_{\text{start}}}{B} \rceil B\big)$
  Retrieve states at t_{start} and t_{end} from $\mathcal{T}_x$
  Compute $\mathcal{L}_{\text{Distillation}}$ (Eq. (4)) using $\mathbf{H}_x$
  Compute $\mathcal{L}_{\text{Consistency}}$ (Eq. (5))
  Compute $\mathcal{L}_{\text{DLM}}$ (Eq. (6)) using ground-truth text $\hat{\mathbf{y}}$
  $\mathcal{L} \leftarrow w_{\text{distill}}\mathcal{L}_{\text{Distillation}} + w_{\text{cons}}\mathcal{L}_{\text{Consistency}} + w_{\text{dlm}}\mathcal{L}_{\text{DLM}}$
  Update ϕ with $\mathcal{L}$
until max epochs

serves the model's masked-token prediction capability. The detailed training procedure is depicted in Algorithm 2.

Notation Given the dataset $\mathcal{D} = \{(x, \hat{\mathbf{y}}, \mathcal{T}_x, \mathbf{H}_x)\}$, we first sample a prompt x and its trajectory $\mathcal{T}_x$, then sample a state $y \in \mathcal{T}_x$ and its block-completion state $y^\star$. Here, $y^\star$ denotes the state obtained by fully unmasking the active block of y (thus y and $y^\star$ are at most B steps apart). The right panel of Figure 1 visualizes this setup. We denote $\mathcal{U}_y = \{i \mid y_i = \texttt{[MASK]}, y^\star_i \neq \texttt{[MASK]}\}$ as the newly unmasked token indices between y and $y^\star$, and $\mathcal{S}_y = \{i \mid y_i = \texttt{[MASK]}, y^\star_i = \texttt{[MASK]}\}$ as the still-masked token indices at $y^\star$.

Distillation Loss We use the student's predictions at state y together with the teacher's hidden-state buffer $\mathbf{H}_x$. For each $i \in \mathcal{U}_y$, we reconstruct the teacher's distribution by applying $\ell_i^{(T)} = \texttt{lm\_head}(\mathbf{h}_{x,i})$ and $p_i^{(T)} = \text{softmax}(\ell_i^{(T)})$ to the stored hidden state $\mathbf{h}_{x,i}$. With these teacher distributions, we define the distillation objective as

$$\mathcal{L}_{\text{Distillation}} = \mathbb{E}_{(x,\mathcal{T}_x,\mathbf{H}_x)\sim\mathcal{D}}\, \mathbb{E}_{y\sim\mathcal{T}_x}\Bigg[\frac{1}{|\mathcal{U}_y|} \sum_{i\in\mathcal{U}_y} D_{\text{KL}}\Big(p_i^{(T)} \,\Big\|\, q_\phi(\cdot \mid y, x)_i\Big)\Bigg], \tag{4}$$

which applies forward KL divergence on the positions newly unmasked between y and $y^\star$. This serves as the main anchor that transfers the bidirectional teacher's guidance to the block-wise student, effectively providing *supervision for multi-token finalization* within each block.

Consistency Loss We compare the student's predictive distributions at state y and block-completion state $y^\star$, enforcing agreement on still-masked positions. Concretely, we minimize the following objective:

$$\mathcal{L}_{\text{Consistency}} = \mathbb{E}_{(x,\mathcal{T}_x)\sim\mathcal{D}}\, \mathbb{E}_{y\sim\mathcal{T}_x}\Bigg[\frac{1}{|\mathcal{S}_y|} \sum_{i\in\mathcal{S}_y} D_{\text{KL}}\Big(q_{\phi^-}(\cdot \mid y^\star, x)_i \,\Big\|\, q_\phi(\cdot \mid y, x)_i\Big)\Bigg]. \tag{5}$$

Here, q_{ϕ^-} denotes a stop-gradient target (detached from backpropagation) for stable training (Song et al., 2023), and D_{KL} is the forward KL divergence between two distributions. Intuitively, Eq. (5) aligns the student at a *less-informed* state with itself at a *more-informed* state, encouraging stable multi-step jumps along the decoding trajectory. Since DLMs are trained to predict only `[MASK]` tokens, restricting the loss to still-masked positions ($\mathcal{S}_y$) avoids ill-defined supervision.

DLM loss We include an auxiliary loss, identical to the masked denoising objective used in standard DLM pre-training (Nie et al., 2025b; Ye et al., 2025), to preserve the model's mask prediction capability. For a prompt x and its ground-truth response $\hat{\mathbf{y}}$, we randomly sample a masking ratio $t \sim \mathcal{U}[0, 1]$ and independently mask each token with a probability t to obtain the masked sequence $\hat{\mathbf{y}}_t$. The resulting objective is

$$\mathcal{L}_{\text{DLM}} = -\mathbb{E}_{(x,\hat{\mathbf{y}})\sim\mathcal{D}}\, \mathbb{E}_t\Bigg[\frac{1}{t} \sum_{i=1}^{L_g} \mathbf{1}\big[\hat{y}_{t,i} = \texttt{[MASK]}\big] \log q_\phi\big(\hat{y}_i \mid \hat{\mathbf{y}}_t, x\big)\Bigg]. \tag{6}$$

Here, $\mathbf{1}[\cdot]$ denotes the indicator function.

Consequently, the total training objective is

$$\mathcal{L}(\phi) = w_{\text{distill}}\, \mathcal{L}_{\text{Distillation}} + w_{\text{cons}}\, \mathcal{L}_{\text{Consistency}} + w_{\text{dlm}}\, \mathcal{L}_{\text{DLM}}. \tag{7}$$

4.3 Inference

At inference time, the student decodes block-wise under a block-causal mask (Figure 2), reusing the KV cache for the prompt and all previously finalized blocks. Within each block, we apply *confidence-thresholded* parallel finalization: at each step, among the masked positions of the current block, we reveal every token whose confidence exceeds a threshold τ_{conf}, following Fast-dLLM (Wu et al., 2025b). We also adopt early stopping: decoding terminates once an `<endoftext>` token is produced within the current block, analogous to AR decoding. We intentionally avoid additional heuristics such as inter-block parallelism (Wang et al., 2025), which introduce extra hyperparameters whose optimal values are task- and domain-dependent.

Table 1. Evaluation results for **Dream-7B-Instruct**. Arrows in headers indicate whether higher (↑) or lower (↓) is better. *Notes:* Par. = parallel decoding; D.C. = dual-cache KV.

Benchmark	Method	TPS ↑	Latency (s) ↓	Total Steps ↓	Gen. Length	Score ↑
GSM8K -CoT (8-shot)	Dream-7B-Instruct	4.1 (×1.0)	23.5 (×1.0)	256.0 (×1.0)	95.1	79.1
	dLLM-Cache	6.9 (×1.7)	12.6 (×1.9)	256.0 (×1.0)	87.5	75.2
	Fast-dLLM (Par.)	19.2 (×4.7)	5.0 (×4.7)	53.7 (×4.8)	96.0	**79.9**
	Fast-dLLM (Par.+D.C.)	36.6 (×8.9)	2.5 (×9.4)	60.8 (×4.2)	90.3	77.3
	CDLM–Dream (ours)	**51.7** (×12.6)	**2.1** (×11.2)	**44.1** (×5.8)	107.4	78.8
HumanEval -Instruct (0-shot)	Dream-7B-Instruct	15.4 (×1.0)	13.4 (×1.0)	256.0 (×1.0)	206.7	48.2
	dLLM-Cache	9.4 (×0.6)	12.0 (×1.1)	256.0 (×1.0)	113.6	43.9
	Fast-dLLM (Par.)	66.4 (×4.3)	3.2 (×4.2)	61.6 (×4.2)	210.2	45.7
	Fast-dLLM (Par.+D.C.)	**79.9** (×5.2)	2.5 (×5.4)	71.6 (×3.6)	200.9	46.3
	CDLM–Dream (ours)	43.3 (×2.8)	**2.2** (×6.1)	**49.6** (×5.2)	96.9	**50.0**
MATH (4-shot)	Dream-7B-Instruct	8.3 (×1.0)	21.9 (×1.0)	256.0 (×1.0)	182.2	**38.0**
	dLLM-Cache	11.3 (×1.4)	13.0 (×1.7)	256.0 (×1.0)	146.9	35.8
	Fast-dLLM (Par.)	23.9 (×2.9)	7.6 (×2.9)	87.1 (×2.9)	181.0	37.8
	Fast-dLLM (Par.+D.C.)	48.8 (×5.9)	3.7 (×5.9)	98.2 (×2.6)	179.8	37.4
	CDLM–Dream (ours)	**53.8** (×6.5)	**2.9** (×7.6)	**63.2** (×4.1)	158.4	32.4
MBPP -Instruct (0-shot)	Dream-7B-Instruct	2.3 (×1.0)	21.7 (×1.0)	256.0 (×1.0)	49.5	51.8
	dLLM-Cache	5.1 (×2.2)	11.3 (×1.9)	256.0 (×1.0)	57.3	**56.0**
	Fast-dLLM (Par.)	15.5 (×6.7)	3.1 (×7.0)	35.3 (×7.3)	47.3	55.6
	Fast-dLLM (Par.+D.C.)	25.4 (×11.0)	1.9 (×11.4)	43.6 (×5.9)	47.1	53.4
	CDLM–Dream (ours)	**48.1** (×20.9)	**1.5** (×14.5)	**33.2** (×7.7)	74.3	53.0

5 EXPERIMENTS

5.1 Experimental Setup

This subsection summarizes datasets, CDLM training, evaluation protocol, baselines, hardware, and metrics.

Target Models and Training Datasets We consider two representative open-source DLMs: Dream-7B-Instruct (Ye et al., 2025) and LLaDA-8B-Instruct (Nie et al., 2025b). We derive 7.5k prompts from Bespoke-Stratos-17k (Labs, 2025), filtering to those with prompt length ≤ 512 tokens. For LLaDA, we further augment the corpus with 7.5k math-style prompts from DParallel (Chen et al., 2025), using the same length budget. When generating teacher trajectories, we fix the generation length to L_g=256 and the block size to B=32. Additional details are provided in Appendix A.1.

Training Details We fine-tune all models with LoRA (Hu et al., 2022) applied to both attention and MLP modules, using a block-wise causal mask (Figure 2) with B=32 and L_g=256. End-to-end training takes approximately **8 hours** for CDLM–Dream and **16 hours** for CDLM–LLaDA on 4× NVIDIA A100 (80GB) GPUs. Additional details are in Appendix A.2.

Benchmarks. Following established conventions (Wang et al., 2025; Wu et al., 2025b), we evaluate mathematical reasoning and code generation on GSM8K, GSM8K-CoT (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), HumanEval (Chen et al., 2021a), and MBPP (Austin et al., 2021b). Coding tasks are evaluated zero-shot, and few-shot settings for math follow the original Dream and LLaDA papers. All efficiency measurements are taken on 4× NVIDIA A100 (80 GB) GPUs with batch size 1 under data parallelism. Latency, total steps, and generation length are reported as per-sample averages over the evaluation set. Additional evaluation details are provided in Appendix A.3.

Baselines. As references, we evaluate the original DLMs using block-wise decoding under their official inference settings. Because our work targets two orthogonal axes of inference acceleration, (1) *KV caching* and (2) *step reduction*, we include three inference-time baselines: (i) dLLM-Cache (Liu et al., 2025b), which focuses on (1) via adaptive feature caching; (ii) Fast-dLLM (Parallel) (Wu et al., 2025b), which targets (2) via confidence thresholding; and (iii) Fast-dLLM (Parallel + Dual Cache) (Wu et al., 2025b), which combines (1) + (2) via approximate dual-cache KV caching. We omit D2F (Wang et al., 2025) here because it is trained for L_g=512 and reports on Dream-7B-Base rather than Dream-7B-Instruct. For autoregressive comparisons, we evaluate Qwen2.5-7B-Instruct (Qwen et al., 2024) alongside Dream and Llama-3.1-8B-Instruct (Grattafiori et al., 2024) alongside LLaDA. For fair comparison across methods, we fix L_g=256, use greedy decoding (temperature 0.0), apply the same B=32, and set the token-confidence threshold to τ_{conf}=0.9. Further details are in Appendix A.3.

5.2 Main Results

We compare CDLM against vanilla DLMs, accelerated DLM variants, and autoregressive baselines.

Table 2. Evaluation results for **LLaDA-8B-Instruct**. Arrows in headers indicate whether higher (↑) or lower (↓) is better. We report HumanEval (not HumanEval-Instruct), as the latter lowers overall baseline scores. *Notes:* Par. = parallel decoding; D.C. = dual-cache KV.

Benchmark	Method	TPS ↑	Latency (s) ↓	Total Steps ↓	Gen. Length	Score ↑
GSM8K (4-shot)	LLaDA-8B-Instruct	8.2 (×1.0)	28.3 (×1.0)	256.0 (×1.0)	232.1	77.1
	dLLM-Cache	18.7 (×2.3)	12.3 (×2.3)	256.0 (×1.0)	229.6	78.5
	Fast-dLLM (Par.)	26.7 (×3.3)	8.7 (×3.3)	77.5 (×3.3)	231.3	**78.6**
	Fast-dLLM (Par.+D.C.)	**54.6** (×6.7)	4.2 (×6.7)	85.0 (×3.0)	230.4	76.5
	CDLM–LLaDA (ours)	54.3 (×6.6)	**3.3** (×8.6)	**57.7** (×4.4)	177.3	73.9
HumanEval (0-shot)	LLaDA-8B-Instruct	7.4 (×1.0)	11.3 (×1.0)	256.0 (×1.0)	83.9	37.8
	dLLM-Cache	12.6 (×1.7)	10.8 (×1.0)	256.0 (×1.0)	135.9	39.0
	Fast-dLLM (Par.)	17.9 (×2.4)	4.6 (×2.5)	100.3 (×2.6)	83.1	36.6
	Fast-dLLM (Par.+D.C.)	18.9 (×2.6)	4.2 (×2.7)	97.7 (×2.6)	80.2	36.0
	CDLM–LLaDA (ours)	**50.9** (×6.9)	**1.9** (×5.9)	**32.3** (×7.9)	95.1	**40.2**
MATH (4-shot)	LLaDA-8B-Instruct	8.8 (×1.0)	25.7 (×1.0)	256.0 (×1.0)	226.4	24.1
	dLLM-Cache	22.7 (×2.6)	10.9 (×2.4)	256.0 (×1.0)	248.5	33.3
	Fast-dLLM (Par.)	25.1 (×2.9)	9.9 (×2.6)	96.9 (×2.6)	248.6	**33.4**
	Fast-dLLM (Par.+D.C.)	49.7 (×5.7)	5.0 (×5.1)	107.0 (×2.4)	248.2	32.5
	CDLM–LLaDA (ours)	**50.2** (×5.7)	**4.2** (×6.1)	**75.3** (×3.4)	210.3	28.3
MBPP -Instruct (0-shot)	LLaDA-8B-Instruct	17.7 (×1.0)	11.4 (×1.0)	256.0 (×1.0)	201.0	40.8
	dLLM-Cache	18.5 (×1.0)	10.8 (×1.1)	256.0 (×1.0)	199.4	40.8
	Fast-dLLM (Par.)	21.2 (×1.2)	6.2 (×1.8)	59.1 (×4.3)	131.8	**42.0**
	Fast-dLLM (Par.+D.C.)	40.1 (×2.3)	3.4 (×3.4)	66.9 (×3.8)	134.7	35.0
	CDLM–LLaDA (ours)	**60.6** (×3.4)	**3.2** (×3.6)	**58.0** (×4.4)	195.3	38.4

5.2.1 *Results on CDLM–Dream*

Table 1 summarizes throughput, latency, step counts, generation length, and accuracy for Dream-7B-Instruct and its acceleration baselines.

Steps and scores. Our objective is to reduce the number of sampling steps so that the model reaches a coherent answer more quickly. As shown in Table 1, CDLM–Dream achieves the largest step reductions across benchmarks, cutting refinement steps by roughly $4.1\times$–$7.7\times$ with minor accuracy changes on most benchmarks. Naive Dream is run with 256 refinement steps at its most performant setting; simply reducing the step count markedly degrades generation quality (see Section 5.3.2). Despite aggressive step reduction, CDLM–Dream matches or improves accuracy: it rises from 51.8→53.0 on MBPP-Instruct and from 48.2→50.0 on HumanEval-Instruct, with only a negligible drop on GSM8K-CoT (79.1→78.8). In contrast, dLLM-Cache keeps the step budget fixed at 256 and accelerates via KV caching rather than step reduction.

We attribute the accuracy drop on MATH to two factors. First, the small training mixture (∼7.5k prompts) provides limited exposure to the advanced reasoning patterns required by MATH. The data are drawn from filtered subsets that largely contain problems already solvable by Qwen2.5, resulting in a narrower effective difficulty range. Second, teacher trajectories are capped at 256 generation tokens during training. This shorter reasoning budget may be inadequate for complex multi-step problems. A natural next step is to train with a longer budget (e.g., 512 tokens), as in D2F (Wang et al., 2025).

Latency and Throughput (TPS). CDLM–Dream shows the largest latency reductions across all benchmarks, with speedups of up to $11.2\times$ on GSM8K-CoT and $14.5\times$ on MBPP-Instruct. These gains stem from combining *step reduction* with *block-wise KV caching*. The impact of caching is evident when comparing methods with similar refinement steps. For example, on GSM8K-CoT, Fast-dLLM (Par.+D.C.) halves latency (5.0 s→2.5 s) relative to Fast-dLLM (Par.) despite using more refinement steps (53.7→60.8) by avoiding redundant computation. CDLM–Dream further benefits from block-wise causal training (Figure 2), which enables exact KV caching and early termination at block boundaries, reducing unnecessary decoding.

CDLM–Dream attains the highest TPS on three out of four benchmarks. The sole exception is HumanEval-Instruct, where TPS is lower than Fast-dLLM baselines due to a substantially shorter effective generation length (97 vs. ∼200). Differences in decoding dynamics can arise because CDLM is strictly block-causal and cannot attend to future blocks, lacking explicit budget awareness, whereas other baselines retain full bidirectional attention that preserves a form of global budget awareness. Despite shorter generations, CDLM–Dream achieves higher pass@1, indicating that it can emit fewer tokens while preserving solution quality.

5.2.2 *Results on CDLM–LLaDA*

Table 2 summarizes throughput, latency, step counts, generation length, and accuracy for LLaDA-8B-Instruct and its

acceleration baselines.

Steps and scores. As shown in Table 2, CDLM–LLaDA delivers the largest step reductions across all four benchmarks, cutting steps by roughly $3.4\times$–$7.9\times$. Despite this aggressive reduction, CDLM–LLaDA improves HumanEval from 37.8→40.2 and MATH from 24.1→28.3. MBPP-Instruct shows a slight drop (40.8→38.4), yet still outperforms Fast-dLLM (Par.+D.C.) by 3.4 points at comparable throughput. GSM8K is the only benchmark with a clear degradation (77.1→73.9), which we analyze below.

We attribute the GSM8K degradation primarily to limited data and LLaDA's sensitivity to fine-tuning. In initial experiments, training CDLM–LLaDA on bespoke ∼7.5k prompts already reduced GSM8K accuracy (77.1→72.1), consistent with prior observations that even simple SFT can harm math performance unless data are carefully curated and sufficiently large. Augmenting the data with an additional 7.5k math-focused prompts partially recovers performance (GSM8K 72.1→73.9, MATH 25.4→28.3) while preserving coding scores. In addition, the fixed generation budget (L_g=256) may further constrain reasoning depth, as LLaDA tends to produce longer outputs when allowed larger budgets. We expect that jointly scaling the dataset and increasing the generation budget would close the remaining GSM8K gap.

Latency and Throughput (TPS). CDLM–LLaDA attains the strongest latency reductions across all tasks; for example, on GSM8K latency falls from 28.3 s→3.3 s. Throughput gains are likewise substantial, ranging from $3.4\times$ to $6.9\times$ relative to the naive LLaDA baseline. The role of KV caching is evident when step counts are similar: LLaDA vs. dLLM-Cache keeps 256 steps yet nearly halves latency on GSM8K (28.3 s→12.3 s), and Fast-dLLM (Par.) vs. (Par.+D.C.) reduces GSM8K latency further (8.7 s→4.2 s) even as steps increase (77.5→85.0). CDLM–LLaDA goes beyond caching alone by also reducing refinement steps via *consistency* training while retaining block-wise causality for exact KV caching, yielding the best overall efficiency in both latency and TPS.

5.2.3 *Comparison with Autoregressive Models*

Throughput. CDLM improves naive DLMs in throughput by $3\times$–$21\times$ and also surpasses equal-size autoregressive (AR) baselines (Figure 3). CDLM–Dream achieves $1.2\times$ and $1.1\times$ higher throughput than its AR baseline on GSM8K-CoT (8-shot) and MBPP-Instruct (0-shot), respectively, while CDLM–LLaDA attains $1.3\times$ and $4.2\times$ higher throughput on GSM8K (4-shot) and HumanEval (0-shot). Throughput here reflects both per-step cost and the number of refinement steps. A CDLM refinement step is more expensive than an AR step due to matrix-matrix multiplications for decoding. However, CDLM can finalize multiple tokens per iteration, reducing total steps and increasing effective throughput. For example, from Table 1, CDLM–Dream produces on average $107.4/44.1 \approx 2.4$ tokens per step on GSM8K-CoT and $74.3/33.2 \approx 2.2$ tokens per step on MBPP-Instruct.

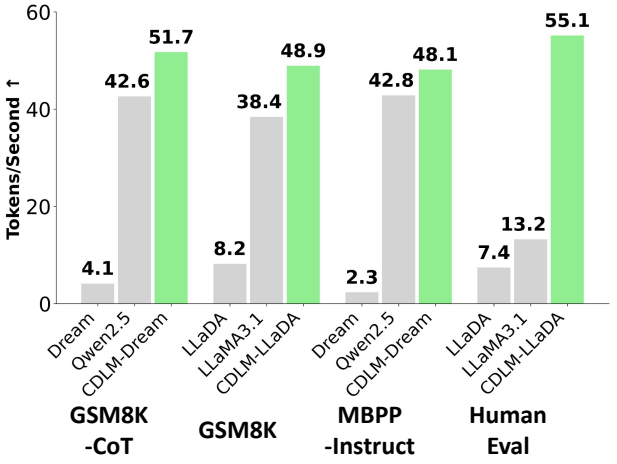

Figure 3. **Throughput comparison across benchmarks.** Tokens per second on GSM8K, MBPP, and HumanEval for Dream-7B-Instruct and LLaDA-8B-Instruct under naive diffusion decoding, autoregressive (AR) baselines, and CDLM.

Accuracy. Although throughput is our focus, accuracy may be lower than that of AR models because CDLMs remain bounded by the strength of their DLM backbones. For CDLM–Dream versus its AR baseline, the accuracies are 78.8 vs. 73.8 on GSM8K-CoT and 53.0 vs. 81.7 on MBPP-Instruct (CDLM vs. AR). For CDLM–LLaDA versus its AR baseline, the respective accuracies are 73.9 vs. 80.3 on GSM8K and 40.2 vs. 60.4 on HumanEval (CDLM vs. AR). As stronger DLM backbones emerge, applying our consistency-based fine-tuning should yield further throughput gains with improved quality.

5.3 Ablation Studies

We analyze how training and inference choices affect efficiency and accuracy.

5.3.1 *Loss-Weight Composition*

Table 3 varies the loss weights $(w_{\text{distill}}, w_{\text{cons}}, w_{\text{dlm}})$ to study the trade-off between convergence speed and final accuracy. We train the Dream model for four epochs with a constant learning rate `2e-5`, then evaluate on GSM8K (4-shot) and HumanEval-Instruct (0-shot).

Distillation vs. consistency in isolation. With a small auxiliary DLM loss (w_{dlm}=0.01), the *distillation-only* setting (row 1) acts as a strong anchor: it converges quickly (about

Table 3. **Ablation of loss weights.** Effects of varying $(w_{\text{distill}}, w_{\text{cons}}, w_{\text{dlm}})$ on *score* (black) and *steps to convergence* (blue) on GSM8K and HumanEval-Instruct for CDLM–Dream.

w_{distill}	w_{cons}	w_{dlm}	**GSM8K** (4-shot)	**HumanEval-Inst.** (0-shot)
1.0	✗	0.01	73.2 (46.7)	42.7 (61.0)
✗	1.0	0.01	6.9 (100.6)	0.0 (124.3)
1.0	1.0	0.01	74.1 (49.4)	42.7 (49.4)
1.0	1.0	✗	73.3 (46.2)	48.2 (51.5)
1.0	0.1	0.01	75.1 (48.0)	45.7 (59.9)
1.0	0.1	✗	73.7 (48.1)	50.6 (69.0)

46.7 and 61.0 steps for a generation length of 256), with a slight degradation in score. In contrast, *consistency-only* (row 2) collapses: optimizing for self-agreement without teacher supervision causes failure.

Coupling yields synergy. When we couple *consistency* with *distillation* (rows 3 and 5), convergence becomes both faster and more stable. GSM8K improves from 73.2 to 74.1–75.1 with similar or fewer steps, and HumanEval is comparable to or higher (42.7→45.7) while requiring fewer iterations. We fix w_{distill}=1.0 and set w_{cons}=0.5 as a good balance between speed and quality.

Role of the auxiliary DLM loss. Removing the ground-truth masked-denoising term (rows 4 and 6) raises HumanEval but lowers GSM8K, indicating a loss of math reasoning ability. A small auxiliary weight helps preserve this ability, and we find that general coding performance can be recovered with longer training. Accordingly, we use w_{dlm}=0.01 for Dream and w_{dlm}=0.1 for LLaDA, as the DLM loss on LLaDA has a smaller absolute scale.

5.3.2 *Effective Step Reduction*

We evaluate a naive step-truncation baseline by forcing the teacher DLMs to use a similar number of refinement steps as our CDLMs (Table 4). Because decoding is block-wise, the step count must be a multiple of the block factor (L_g/B=8). For Dream, we set the step budget to 48 and measure GSM8K-CoT (8-shot) as in Table 1; for LLaDA, we set it to 56 and measure GSM8K (4-shot) as in Table 2. Naively truncating the step budget, in other words, forcing a non-retrained DLM to finalize multiple tokens per iteration, leads to marked accuracy degradation. By contrast, CDLM maintains quality at comparable step counts, underscoring that consistency training is necessary for stable multi-token refinement. Furthermore, with KV caching, our approach reduces latency by roughly half relative to the naive DLM at comparable iteration counts.

Table 4. **Ablation of refinement steps.** GSM8K results obtained by naively truncating the baseline DLM step counts to match the step budgets used by CDLM–Dream and CDLM–LLaDA.

Method	**Latency (s)** ↓	**Steps** ↓	**Score** ↑
Dream-7B-Instruct	4.4	48	41.8
CDLM–Dream (ours)	2.1	44.1	78.8
LLaDA-8B-Instruct	6.0	56	60.3
CDLM–LLaDA (ours)	3.3	57.7	73.9

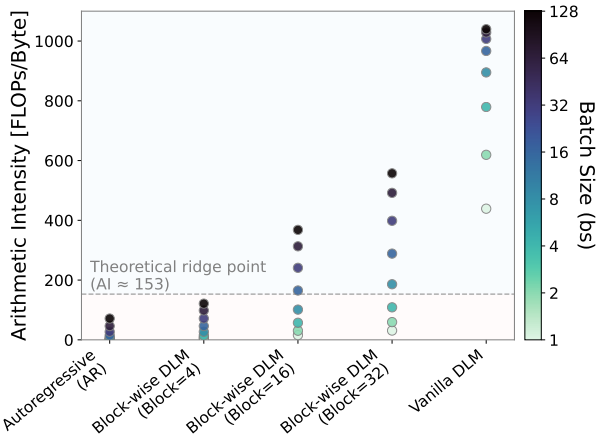

Figure 4. **Arithmetic intensity across batch sizes.** Arithmetic Intensity (AI) during decoding as a function of batch size (bs $\in \{1, 2, 4, 8, 16, 32, 128\}$) for autoregressive (AR) models, vanilla DLMs, and block-wise DLMs (CDLM). The dashed line indicates the theoretical ridge point, separating memory-bound (red) and compute-bound (blue) regimes.

5.4 System-Level Scalability via Arithmetic Intensity

To understand CDLM's hardware-utilization characteristics, we analytically model the inference for autoregressive (AR) models, vanilla DLMs, and block-wise DLMs with block sizes $B \in \{4, 16, 32\}$. Building on prior modeling frameworks (Tiwari et al., 2025; Kim et al., 2025), we parameterize the AR baseline following the LLaMA-3.1 (Grattafiori et al., 2024) configuration, while vanilla and block-wise DLMs follow the LLaDA (Nie et al., 2025b) configuration. We examine *arithmetic intensity* (AI), the ratio of computation to memory traffic (FLOPs per byte moved), to diagnose whether inference is memory- or compute-bound and to characterize batch-level scalability. Concretely, we consider a prompt length L_p=512 and generation length L_g=256 to match Section 5.2, and track AI as the batch size (bs) increases from 1 to 128 (Figure 4). To focus on decoding behavior, we exclude one-time prefill costs (KV-cache initialization) and ignore memory-capacity constraints (e.g., out-of-memory).

Autoregressive (AR) Models. AR decoding operates in a strongly memory-bound regime, with AI close to 1 at $\mathrm{bs} = 1$. This is because each decoding step processes a single token with substantial weight/KV-cache traffic. As the batch size increases, weight traffic is amortized across sequences, resulting in approximately linear AI scaling at small batch sizes (1.0→2.0→4.0→7.8 for $\mathrm{bs} \in \{1, 2, 4, 8\}$). However, even at $\mathrm{bs} = 128$ (AI 71.3), AR decoding remains below the compute-bound regime, indicating that performance is still primarily constrained by memory traffic.

Vanilla DLMs. In contrast, vanilla DLMs are compute-bound even at $\mathrm{bs} = 1$, with AI (438.9) already exceeding the ridge point. Each denoising step recomputes the entire sequence with full bidirectional attention, rather than reusing a KV cache across steps, leading to a highly compute-intensive workload. As a result, increasing bs yields limited additional AI scaling (438.9→619.2→779.3 for $\mathrm{bs} \in \{1, 2, 4\}$), and AI nearly saturates beyond $\mathrm{bs} = 64$ (1028.6→1039.7 from 64 to 128).

Block-wise DLMs (CDLM). Block-wise DLMs operate in an intermediate regime. At $\mathrm{bs} = 1$, their AI (4.0, 15.8, 31.1 for $B \in \{4, 16, 32\}$) exceeds AR decoding but remains below vanilla DLMs. This gain stems from intra-block parallelism: each step processes B tokens under comparable memory traffic as a single-token AR step, effectively amortizing weight loads across the block and scaling AI by nearly B. The AI crosses the ridge point at relatively small batch sizes (e.g., B=32 at $\mathrm{bs} \approx 8$ and B=16 at $\mathrm{bs} \approx 16$), allowing block-wise DLMs to better utilize available compute in small-batch settings. At higher batch sizes, however, the marginal AI gain diminishes as bs increases, since block-wise DLMs already exploit both batch- and block-level parallelism.

Overall, this analysis shows that CDLM-like block-wise DLMs sit between the memory-bound regime of AR models and the compute-saturated regime of full-attention DLMs. This balanced operating point explains CDLM's superior throughput and scalability in small-batch inference. A detailed roofline analysis with AI and corresponding simulated performance is provided in Appendix B.4.

6 Discussion

Here, we discuss key trade-offs, potential extensions, and limitations of our approach.

Expressiveness vs. efficiency trade-off Full bidirectional attention in DLMs requires recomputing $\mathcal{O}(L^2)$ attention at every denoising step, making inference highly compute-intensive and impractical at scale. CDLM enables exact KV caching to mitigate this cost while preserving bidirectional context within each block, retaining local refinement capabilities such as infilling inside the current block. We further believe that the mild left-to-right inductive bias introduced by block-wise decoding aligns well with reasoning-style generation.

Inference-only acceleration techniques Our method is orthogonal to and can be combined with various inference-only (training-free) techniques. At inference time, we only use block-wise KV caching and confidence-based multi-token finalization, but more sophisticated mechanisms can be layered on top. For example, incorporating D2F (Wang et al., 2025)'s inter-block parallelism could further reduce wall-clock latency.

Scaling with stronger DLM backbones CDLM is a post-training recipe that can be applied to any block-diffusion models to accelerate convergence, and its effectiveness is expected to grow as stronger DLMs become available. Recent models such as SDAR (Cheng et al., 2025) are reported to outperform Dream and LLaDA and have been released at multiple scales. A promising direction is to generate trajectories from large-scale DLM teachers (e.g., 30B) and train mid-scale students (e.g., 8B) using the CDLM post-training recipe.

Limitations. Training currently relies on offline, static trajectories that are predominantly math-focused. Despite careful dataset selection, the student may overfit to the teacher's prior and domain, potentially limiting generalization beyond the training distribution. For LLaDA, we observe that augmenting the training set with additional math-style prompts improves math performance by 2–3 percentage points while preserving coding ability, suggesting that scaling and diversifying the corpus are straightforward directions for future work. Furthermore, CDLM's performance is ultimately bounded by the teacher: a bidirectional DLM distilled into a block-causal student cannot exceed the teacher's knowledge. Distilling from stronger DLMs or autoregressive teachers is a natural way to lift this ceiling. Additional discussion appears in Appendix C.

7 Conclusion

We presented **CDLM**, a training-based acceleration scheme that brings *consistency modeling* to DLMs. By enforcing within-block temporal consistency and fine-tuning a block-wise causal student, CDLM reduces refinement steps and enables exact KV caching. Across math and coding tasks, it yields faster inference, fewer steps, lower latency, and higher throughput while maintaining competitive accuracy. Promising directions include scaling the distillation corpus, broadening domain coverage, and distilling from stronger teachers.

ACKNOWLEDGEMENTS

We acknowledge gracious support from the FuriosaAI, Intel, Apple, NVIDIA, Macronix, and Mozilla team. Furthermore, we appreciate support from Google Cloud, the Google TRC team and Prof. David Patterson. Prof. Keutzer's lab is sponsored by the Intel corporation, UC Berkeley oneAPI Center of Excellence, Intel VLAB team, as well as funding through BDD and BAIR. We also acknowledge support by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. DOE SciGPT grant. Minseo also gratefully acknowledges the SNUCSE GPU Service 3.0 program and Bacchus for providing A100 GPU resources that supported this research. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

REFERENCES

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Agarwal, R., Vieillard, N., Zhou, Y., Stanczyk, P., Garea, S. R., Geist, M., and Bachem, O. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and van den Berg, R. Structured denoising diffusion models in discrete state-spaces. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021a. URL https://openreview.net/forum?id=h7-XixPCAL.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b.

Ben-Hamu, H., Gat, I., Severo, D., Nolte, N., and Karrer, B. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *arXiv preprint arXiv:2505.24857*, 2025.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021a. URL https://arxiv.org/abs/2107.03374.

Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., and Chan, W. Wavegrad: Estimating gradients for waveform generation. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=NsMLjcFaO8O.

Chen, Z., Fang, G., Ma, X., Yu, R., and Wang, X. dparallel: Learnable parallel decoding for dllms. *arXiv preprint arXiv:2509.26488*, 2025.

Cheng, S., Bian, Y., Liu, D., Zhang, L., Yao, Q., Tian, Z., Wang, W., Guo, Q., Chen, K., Qi, B., et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*, 2025.

Christopher, J. K., Bartoldson, B. R., Ben-Nun, T., Cardei, M., Kailkhura, B., and Fioretto, F. Speculative diffusion decoding: Accelerating language generation through diffusion. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 12042–12059, 2025.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Deschenaux, J. and Gulcehre, C. Beyond autoregression: Fast LLMs via self-distillation through time. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=uZ5K4HeNwd.

Esser, P., Chiu, J., Atighehchian, P., Granskog, J., and Germanidis, A. Structure and content-guided video synthesis with diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 7346–7356, 2023.

Fu, Y., Peng, H., Ou, L., Sabharwal, A., and Khot, T. Specializing smaller language models towards multi-step reasoning. In *International Conference on Machine Learning*, pp. 10421–10430. PMLR, 2023.

Gao, Z., Guo, J., Tan, X., Zhu, Y., Zhang, F., Bian, J., and Xu, L. Empowering diffusion models on the embedding space for text generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4664–4683, 2024.

Gong, S., Li, M., Feng, J., Wu, Z., and Kong, L. Diffuseq: Sequence to sequence text generation with diffusion models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=jQj-_rLVXsj.

Gong, S., Agarwal, S., Zhang, Y., Ye, J., Zheng, L., Li, M., An, C., Zhao, P., Bi, W., Han, J., Peng, H., and Kong, L. Scaling diffusion language models via adaptation from autoregressive models. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=j1tSLYKwg8.

Gong, S., Zhang, R., Zheng, H., Gu, J., Jaitly, N., Kong, L., and Zhang, Y. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025b.

Google DeepMind. Gemini Diffusion. https://deepmind.google/models/gemini-diffusion/, 2025.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Gu, Y., Dong, L., Wei, F., and Huang, M. Minillm: Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Hu, J. C., Cavicchioli, R., and Capotondi, A. Bidirectional awareness induction in autoregressive seq2seq models. *arXiv preprint arXiv:2408.13959*, 2024.

Hu, Z., Meng, J., Akhauri, Y., Abdelfattah, M. S., Seo, J.-s., Zhang, Z., and Gupta, U. Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv preprint arXiv:2505.21467*, 2025.

Kang, W., Galim, K., Oh, S., Lee, M., Zeng, Y., Zhang, S., Hooper, C., Hu, Y., Koo, H. I., Cho, N. I., et al. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*, 2025.

Khanna, S., Kharbanda, S., Li, S., Varma, H., Wang, E., Birnbaum, S., Luo, Z., Miraoui, Y., Palrecha, A., Ermon, S., et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 1, 2025.

Kim, M., Hooper, C., Tomar, A., Xu, C., Farajtabar, M., Mahoney, M. W., Keutzer, K., and Gholami, A. Beyond next-token prediction: A performance characterization of diffusion versus autoregressive language models. *arXiv preprint arXiv:2510.04146*, 2025.

Kou, S., Hu, L., He, Z., Deng, Z., and Zhang, H. CLLMs: Consistency large language models. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=8uzBOVmh8H.

Labs, B. Bespoke-stratos: The unreasonable effectiveness of reasoning distillation. https://www.bespokelabs.ai/blog/bespoke-stratos-the-unreasonable-effectiveness-of-reasoning-distillation, 2025. Accessed: 2025-10-23.

Lansechen. Lansechen/bs17k_collection_filtered_easy_maxlength600. https://huggingface.co/datasets/Lansechen/bs17k_collection_filtered_easy_maxlength600, 2025a. Accessed: 2025-10-23.

Lansechen. Lansechen/bs17k_collection_filtered_hard_maxlength600. https://huggingface.co/datasets/Lansechen/bs17k_collection_filtered_hard_maxlength600, 2025b. Accessed: 2025-10-23.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Li, X. L., Thickstun, J., Gulrajani, I., Liang, P., and Hashimoto, T. Diffusion-LM improves controllable text generation. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=3s9IrEsjLyk.

Liu, J., Dong, X., Ye, Z., Mehta, R., Fu, Y., Singh, V., Kautz, J., Zhang, C., and Molchanov, P. Tidar: Think in diffusion, talk in autoregression. *arXiv preprint arXiv:2511.08923*, 2025a.

Liu, Z., Yang, Y., Zhang, Y., Chen, J., Zou, C., Wei, Q., Wang, S., and Zhang, L. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025b.

Ma, X., Yu, R., Fang, G., and Wang, X. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025a.

Ma, Y., Du, L., Wei, L., Chen, K., Xu, Q., Wang, K., Feng, G., Lu, G., Liu, L., Qi, X., et al. dinfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv:2510.08666*, 2025b.

Nie, S., Zhu, F., Du, C., Pang, T., Liu, Q., Zeng, G., Lin, M., and Li, C. Scaling up masked diffusion models on text. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=WNvvwK0tut.

Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., ZHOU, J., Lin, Y., Wen, J.-R., and Li, C. Large language diffusion models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=KnqiC0znVF.

NVIDIA. Nvidia ampere architecture whitepaper. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf, 2020. Accessed: 2026-02-01.

NVIDIA. Nvidia a100 tensor core gpu architecture: Unprecedented acceleration at every scale. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf, May 2022. Accessed: 2026-02-01.

Qwen, :, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Tang, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

Santilli, A., Severino, S., Postolache, E., Maiorca, V., Mancusi, M., Marin, R., and Rodola, E. Accelerating transformer inference for translation via parallel decoding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12336–12355. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.acl-long.689/.

Song, Y. and Dhariwal, P. Improved techniques for training consistency models. *arXiv preprint arXiv:2310.14189*, 2023.

Song, Y., Dhariwal, P., Chen, M., and Sutskever, I. Consistency models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pp. 32211–32252, 2023. URL https://openreview.net/forum?id=FmqFfMTNnv.

Song, Y., Zhang, Z., Luo, C., Gao, P., Xia, F., Luo, H., Li, Z., Yang, Y., Yu, H., Qu, X., et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Tiwari, R., Xi, H., Tomar, A., Hooper, C., Kim, S., Horton, M., Najibi, M., Mahoney, M. W., Keutzer, K., and Gholami, A. Quantspec: Self-speculative decoding with hierarchical quantized kv cache. *arXiv preprint arXiv:2502.10424*, 2025.

Wang, X., Xu, C., Jin, Y., Jin, J., Zhang, H., and Deng, Z. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*, 2025.

Wu, C., Zhang, H., Xue, S., Diao, S., Fu, Y., Liu, Z., Molchanov, P., Luo, P., Han, S., and Xie, E. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328*, 2025a.

Wu, C., Zhang, H., Xue, S., Liu, Z., Diao, S., Zhu, L., Luo, P., Han, S., and Xie, E. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025b.

Xu, C., Wang, X., Liao, Z., Li, Y., Hou, T., and Deng, Z. Unicms: A unified consistency model for efficient multimodal generation and understanding. *arXiv preprint arXiv:2502.05415*, 2025.

Ye, J., Xie, Z., Zheng, L., Gao, J., Wu, Z., Jiang, X., Li, Z., and Kong, L. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Yin, T., Gharbi, M., Zhang, R., Shechtman, E., Durand, F., Freeman, W. T., and Park, T. One-step diffusion with distribution matching distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6613–6623, 2024.

Yin, T., Zhang, Q., Zhang, R., Freeman, W. T., Durand, F., Shechtman, E., and Huang, X. From slow bidirectional to fast autoregressive video diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 22963–22974, 2025.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

A Additional Experimental Details

A.1 Dataset and Preprocessing

Sources. We use two publicly available datasets from the Hugging Face Hub (Lansechen, 2025a;b), which are post-processed versions of Bespoke-Stratos 17k (Labs, 2025). We choose them because (i) D2F (Wang et al., 2025), a representative fine-tuning-based DLM acceleration method, uses the same source, and (ii) they are filtered to a maximum length of 600 tokens and predominantly contain math word-style reasoning problems. For both Dream and LLaDA, we treat Qwen2.5-7B's responses as ground-truth answers.

Dream is trained only on this Bespoke-derived subset. To strengthen math performance for LLaDA, we further augment its corpus with an additional 7.5k math-style prompts sampled from the DParallel dataset (Chen et al., 2025). The released DParallel corpus does not include Qwen2.5-7B outputs, so we generate reference answers with Qwen2.5-7B for these prompts, yielding an LLaDA training set whose size is approximately $2\times$ that of Dream. After trying multiple datasets, we empirically observe that dataset selection is crucial; overly rigid formats (e.g., multiple-choice math) tend to hurt the model's ability to generalize.

Preprocessing and Generation Configuration. We retain prompts whose input length is at most 512 tokens. For each prompt, we generate teacher trajectories using the bidirectional, block-wise DLM, with generation length L_g=256, block size B=32, total diffusion steps N=256, and prompts left-padded to a length of 512 tokens. Ground-truth responses are truncated or padded to a length of 256 tokens.

Table 5. Training configuration for **CDLM–Dream-7B-Instruct**.

Learning rate	2×10^{-5}
Scheduler (warmup)	constant (5%)
Epochs (best)	16 (best: 12)
Effective batch size	64
Optimizer	AdamW
LoRA rank / α	32 / 32
LoRA targets	`q_proj, k_proj, v_proj, o_proj, up_proj, down_proj, gate_proj`
Prompt / generation length	512 / 256
Loss weights $(w_{\text{distill}}, w_{\text{cons}}, w_{\text{dlm}})$	(1.0, 0.5, 0.01)

Table 6. Training configuration for **CDLM–LLaDA-8B-Instruct**.

Learning rate	1×10^{-5}
Scheduler (warmup)	constant (5%)
Epochs (best)	16 (best: 12)
Effective batch size	64
Optimizer	AdamW
LoRA rank / α	64 / 64
LoRA targets	`q_proj, k_proj, v_proj, attn_out, up_proj, ff_proj, ff_out`
Prompt / generation length	512 / 256
Loss weights $(w_{\text{distill}}, w_{\text{cons}}, w_{\text{dlm}})$	(1.0, 0.5, 0.1)

Temperature augmentation. Effective training benefits from trajectories that are both diverse and reliable. In DLMs, temperature influences not only token choices but also the *order* in which tokens are revealed (Gong et al., 2025b), enabling additional diversity in the generated trajectories. We therefore generate multiple trajectories per prompt at different temperatures. As shown in Figure 5, using temperature τ=1.0 can destabilize the reasoning chain and lead to incorrect conclusions. Consequently, we restrict trajectory collection to $\tau \in \{0.0, 0.5\}$.

Hidden-state buffer. Beyond token trajectories, we store the teacher's decisions in a compact hidden-state buffer. Whenever a token is finalized, we record the teacher's last-layer hidden state at that position; over a generation of length L_g, this yields a buffer of shape $d \times L_g$ per example (see Figure 6). Directly saving logits is impractical because their dimensionality is $|V|$ (vocabulary size); using hidden states of dimension d instead gives roughly a $30\times$ reduction in storage. The trajectory datasets fit on a single server: each shard is 25-30 GiB for 15k samples.

A.2 Training

Training Configurations. Models are optimized with AdamW using a constant learning-rate schedule with warmup. Prompts are left-padded to 512 tokens and decoded with generation length 256, resulting in a fixed se-

Prompt: Return your final response within \boxed{}. Evaluate the product (n-1) · n · (n+1) · (n+2) · (n+3), when n=2.

temp=0.0

First, substitute \(n = 2 \) into the expression: \[(n-1) \cdot n \cdot (n+1) \cdot (n+2) \cdot (n+3) \] \[(2-1) \cdot 2 \cdot (2+1) \cdot (2+2) \cdot (2+3) \] Now, simplify each term: \[(2-1) = 1 \] \[n = 2 \] \[(2+1) = 3 \] \[(2+2) = 4 \] \[(2+3) = 5 \] Next, multiply these values together: \[1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \] \[1 \cdot 2 = 2 \] \[2 \cdot 3 = 6 \] \[6 \cdot 4 = 24 \] \[24 \cdot 5 = 120 \] Thus, the product is: \[**\boxed{120}**\]

temp=0.5

To evaluate the product \((n-1) \cdot n \cdot (n+1) \cdot (n+2) \cdot (n+3)\) where \(n = 2\), we will follow these steps: 1. Substitute \(n = 2\) into the expression. 2. Simplify each term in the product. 3. Multiply the simplified terms together. Let's start by substituting \(n = 2\) into the expression: \[(2-1) \cdot 2 \cdot (2+1) \cdot (2+2) \cdot (2+3) \] Simplify each term: \[1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \] Now, multiply these terms together: \[1 \cdot 2 = 2 \] \[2 \cdot 3 = 6 \] \[6 \cdot 4 = 24 \] \[24 \cdot 5 = 120 \] Thus, the product is: \[**\boxed{120}** \]

temp=1.0

First, substitute \(n = 2 \) into the expression \((n-1) \cdot n \cdot (n+1) \cdot (n+2) \cdot (n+3)\): \[(n-1) \cdot n \cdot (n+1) \cdot (n+2) \cdot (n+3) = (2-1) \cdot 2 \cdot (2+1) \cdot (2+2) \cdot (2+3) \] Simplify each term: \[(2-1) = 1 \] \[2 = 2 \] \[(2+1) = 3 \] \[(2+2) = 4 \] \[(2+3) = 5 \] Now, multiply these values together: \[1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \] Calculate the product: \[1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120 \] \[**\boxed{24}** \]

Figure 5. **Teacher outputs vs. temperature.** Final outputs from LLaDA-8B-Instruct at sampling temperatures $\tau \in \{0.0, 0.5, 1.0\}$. Answers are marked with boxed$\{\cdot\}$ (blue = correct; red = incorrect).

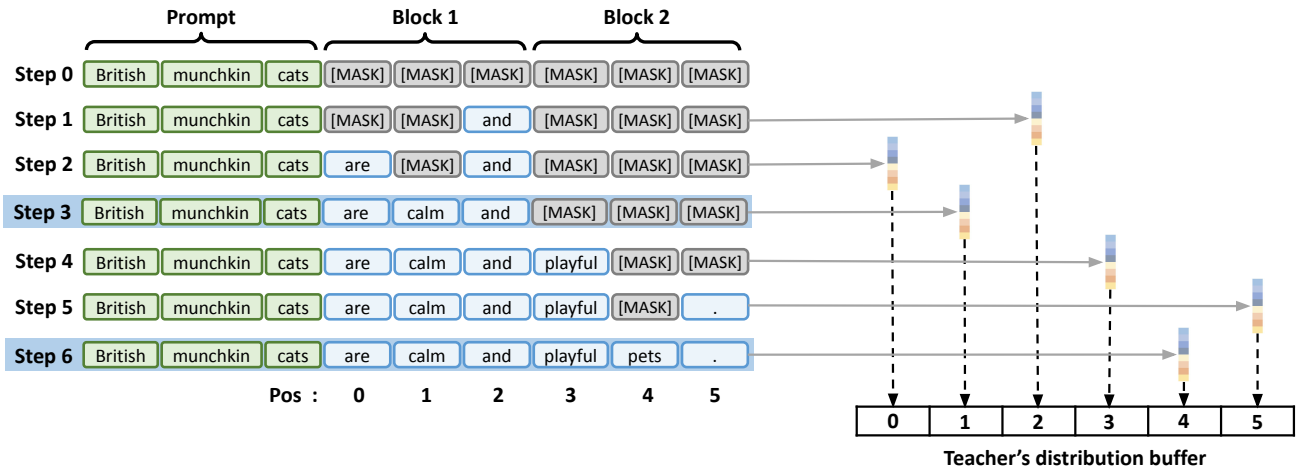

Figure 6. **Teacher trajectory and hidden-state buffer. Left:** Block-wise decoding trajectory. **Right:** As each token is finalized, the teacher's last hidden state at that position is written to a fixed-length buffer of size L_g; arrows indicate write indices across steps (toy example with B=3, L_g=6).

quence length of 768. Based on the ablation study in Section 5.3.1, we set the loss weights $(w_{\text{distill}}, w_{\text{cons}}, w_{\text{dlm}})$ to $(1.0, 0.5, 0.01)$ for Dream and $(1.0, 0.5, 0.1)$ for LLaDA. All models are trained for 16 epochs with an effective batch size of 64, and we select the best checkpoint based on validation performance. Training ran on 4× NVIDIA A100 (80 GB) or 8× NVIDIA RTX A6000 (48 GB) GPUs; each epoch took roughly 30 minutes for 15k trajectories. We used per-device batch sizes of 1–2 with gradient accumulation to keep the effective batch size fixed at 64. Full configurations are listed in Tables 5 and 6.

Loss formulations. Empirically, the *forward* KL divergence yielded more stable, better-calibrated training dynamics and more monotonic convergence than the *reverse* KL divergence for distribution matching. We also found that distillation in *logit space* outperformed embedding-space distillation using mean squared error (MSE).

A.3 Evaluation

Hardware and parallelism. All measurements in Section 5 except Table 3 were taken on 4× NVIDIA A100-80 GiB GPUs using data parallelism with batch size 1 per GPU.

Tooling and post-processing. We use the `lm_eval` framework (EleutherAI's `lm-eval-harness`) for dataset loading, prompting, and evaluation. We apply the harness's default post-processing rules, truncating decoded outputs at any task-specific `stop_sequences` prior to scoring. Evaluation metrics include exact match scores on GSM8K and math-verify scores on MATH. For code-generation benchmarks (HumanEval and MBPP), we additionally employ Python post-processing scripts adapted from the DParallel repository (Chen et al., 2025) to execute generated code and compute pass@1.

Timing and metrics. We report *per-sample averages*. End-to-end latency is computed by summing the wall-clock

time of the generation routine across all ranks and dividing by the total number of evaluated samples. *Total steps* denotes the average number of refinement steps actually executed per sample (summed across ranks, then divided by the sample count). *Generation length* is the average number of valid tokens produced per sample, excluding `<endoftext>` and any content following a task-specific `stop_sequence`.

Prompt formatting. We follow each benchmark's standard few-shot protocol and, by default, apply the model's chat template. For HumanEval with LLaDA-8B-Instruct, however, enabling the chat template caused a sharp pass@1 decline (naive LLaDA: 37.8→7.9), so we evaluate HumanEval without the chat template. For math benchmarks, we do not use `--fewshot_as_multiturn`. Instead, all few-shot examples are merged into a single user prompt.

Inference settings. For LLaDA, we enable block-wise decoding with block size B=32. Dream does not natively support block-wise decoding, so we extend the baseline implementation to support block-wise decoding under the same configuration.

B Additional Results

B.1 Validation Trends During Training

We track validation performance and average refinement iterations across training epochs for CDLM–Dream, following the training setup in Table 5. Figure 7 shows the trends on GSM8K-CoT (left) and MBPP-Instruct (right). On GSM8K-CoT, validation accuracy steadily improves in early epochs and then saturates, while the average number of refinement iterations consistently decreases. This indicates that training encourages the model to reach correct solutions in fewer steps, aligning with our goal of accelerating inference. On MBPP-Instruct, validation accuracy exhibits moderate fluctuations and a downward trend in later epochs, whereas the average number of refinement iterations continues to decrease. This suggests that continued training further reduces the number of refinement iterations, but may also bias the model toward math-style reasoning due to the composition of the training data. We therefore select checkpoints before overfitting becomes pronounced.

B.2 Token-Level Confidence Threshold

Table 7 sweeps the token-level confidence threshold $\tau_{\text{conf}} \in \{0.85, 0.90, 0.95\}$ to characterize conservative-aggressive decoding behavior. Increasing τ_{conf} makes finalization more conservative, tokens are only accepted when the model is highly confident, so throughput (TPS) decreases and per-sample latency increases. Decreasing τ_{conf} makes decoding more aggressive and has the opposite effect. This monotonic speed trend holds on both GSM8K-CoT and HumanEval-Instruct. Concretely, on GSM8K-CoT, TPS/latency moves from $42.7/2.5\,\text{s}$ at τ=0.95 to $57.7/1.9\,\text{s}$ at τ=0.85; on HumanEval-Instruct, from $34.4/2.8\,\text{s}$ to $47.3/2.0\,\text{s}$.

Table 7. **Ablation of the confidence threshold.** Performance under different token-level confidence thresholds τ_{conf} for CDLM–Dream on GSM8K-CoT (8-shot) and HumanEval-Instruct (0-shot).

Benchmark	τ_{conf}	TPS ↑	Latency (s) ↓	Score ↑
GSM8K -CoT (8-shot)	0.95	42.7	2.5	**78.8**
	0.90	51.7	2.1	**78.8**
	0.85	**57.7**	**1.9**	78.4
HumanEval -Instruct (0-shot)	0.95	34.4	2.8	**51.2**
	0.90	43.3	2.2	50.0
	0.85	**47.3**	**2.0**	48.2

Accuracy shows a similar monotonic trend in τ_{conf}. On GSM8K-CoT, scores are essentially flat at 78.8 for τ=0.95 and 0.90, with a slight drop at τ=0.85 (78.4). Similarly, on HumanEval-Instruct, the best score appears at the most conservative setting (51.2 at τ=0.95), 3 points higher than the most aggressive setting (τ=0.85). These results suggest that raising τ_{conf} trades speed for quality, but the gain is not linear. In practice, we find τ_{conf}=0.90 to be a robust default that balances speed and quality across tasks.

B.3 Sensitivity Analysis on Inference Block Size

We study the effect of varying the inference-time block size by evaluating CDLM models trained with B=32 under block sizes $B \in \{4, 8, 16, 32, 64\}$. Figure 8 reports accuracy and throughput (TPS) on representative benchmarks.

Across both CDLM–Dream and CDLM–LLaDA, throughput increases monotonically as the block size grows from B=4 to B=32, reflecting more aggressive exploitation of parallelism and a gradual move away from AR-style decoding. However, increasing the block size beyond the training configuration, from B=32 to B=64, does not consistently yield further speedups. Instead, it often results in saturated or even reduced TPS, indicating a train–inference discrepancy because the models are not trained to predict and complete such large blocks.

For most tasks, the best accuracy is achieved when the inference-time block size matches the training block size (B=32), suggesting that aligning training and inference configurations helps avoid performance degradation. To support a wider range of block sizes at inference time, future work could explore curriculum-based training schemes that begin with smaller block sizes (e.g., B=8 or 16) and gradually increase the difficulty of completing multiple blocks.

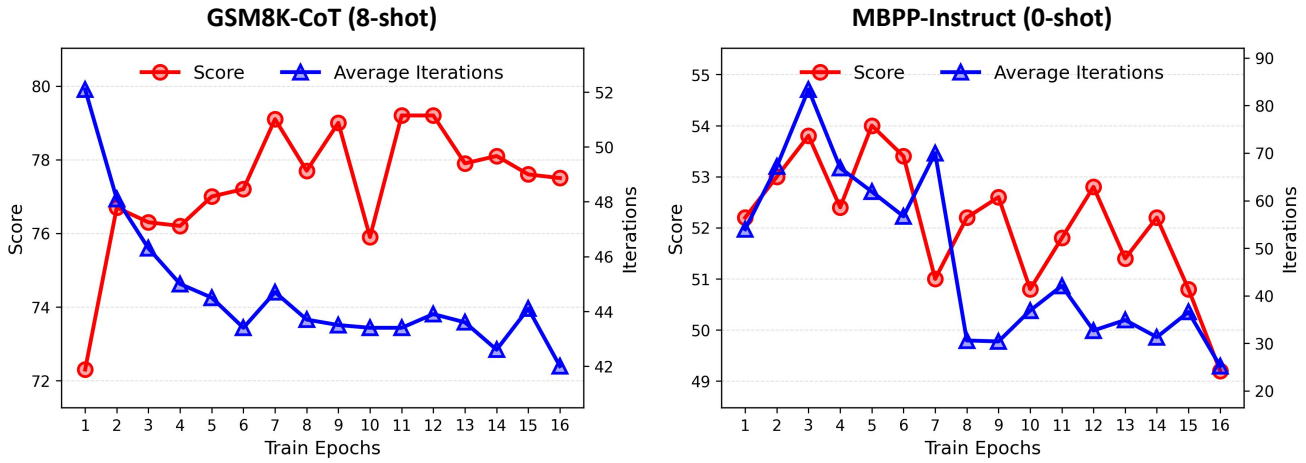

Figure 7. **Validation trends during CDLM–Dream training.** Validation score (red) and average refinement iterations (blue) across training epochs on GSM8K-CoT (8-shot, left) and MBPP-Instruct (0-shot, right).

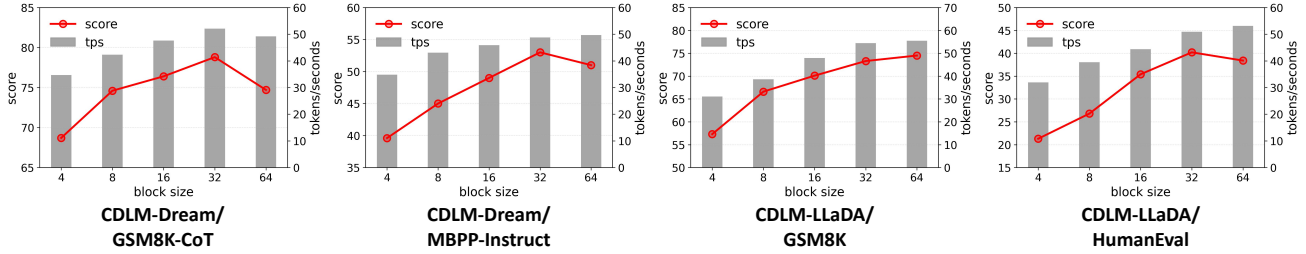

Figure 8. **Impact of inference-time block size.** Accuracy and throughput (TPS) across different block sizes $B \in \{4, 8, 16, 32, 64\}$ for CDLM–Dream and CDLM–LLaDA. All models are trained with B=32 and evaluated by varying the inference-time block size.

B.4 Roofline Analysis of AR Models, Vanilla DLMs, and Block-wise DLMs

Following Section 5.4, we also perform a roofline analysis to characterize the hardware efficiency of AR decoding, full-attention vanilla DLMs, and block-wise DLMs (i.e., CDLM). We derive roofline parameters for an NVIDIA A100-SXM4-80GB (GA100) GPU (NVIDIA, 2020; 2022), assuming dense FP16 Tensor Core computation (without 2:4 structured sparsity) at the boost clock frequency. The theoretical peak FP16 performance is

$$\begin{aligned}\text{Peak FP16 (dense)} &= 108\ \text{SM} \times 4\ \text{Tensor Core/SM} \\ &\quad \times 256\ \text{FMA/(cycle} \cdot \text{Tensor Core)} \\ &\quad \times 1.41\ \text{GHz} \times 2\ \text{FLOP/FMA} \\ &\approx 311.9\ \text{TFLOP/s}.\end{aligned}$$

With memory bandwidth of 2039.0 GB/s, the theoretical ridge point is

$$\text{AI}_{\text{ridge}} = \frac{311.9\ \text{TFLOP/s}}{2039.0\ \text{GB/s}} \approx 153.0\ \text{FLOP/Byte}.$$

Note that the AR and DLM backbones differ (e.g., GQA vs. MHA), so absolute AI values are not directly comparable across model families.

AR Models and Vanilla DLMs. In Figure 9 (left), AR models start in a strongly memory-bound regime, with AI near 1 at $\text{bs} = 1$. Even at $\text{bs} = 128$, AI remains well below the ridge point, and performance scales noticeably with batch size. In contrast, vanilla DLMs are compute-bound even at $\text{bs} = 1$, with AI (≈ 438.9) far exceeding the ridge point. Accordingly, performance quickly saturates and shows little improvement with increasing batch size. The observed plateau lies slightly below the theoretical peak, as non-linear operations such as layer normalization and softmax are executed on vector units with lower peak throughput than Tensor Cores. This heterogeneity in execution units results in a lower effective performance ceiling during inference.

Block-wise DLMs. Figure 9 (right) shows the block-wise DLMs with block sizes $B = 4, 16, 32$. At $\text{bs} = 1$, their AI values (4.0, 15.8, and 31.1, respectively) are higher than AR decoding but lower than vanilla DLMs. This reflects increased intra-block data reuse: processing multiple tokens within a shared KV-cache context and reusing the same model weights across tokens in the block amortizes memory access and raises AI. For small batch sizes, block-wise DLMs remain memory-bound and exhibit strong scaling as batch size increases. As B grows, performance tends to

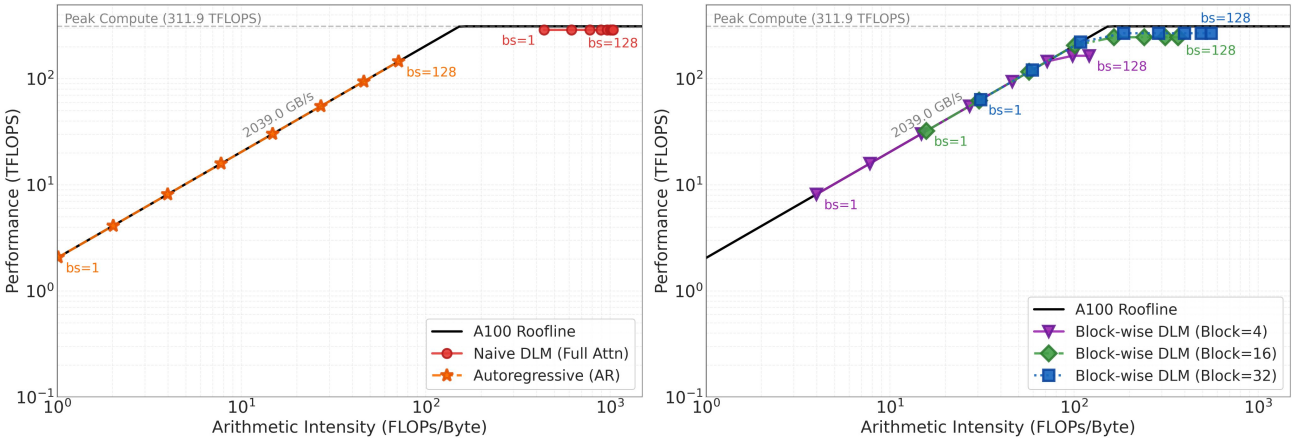

Figure 9. **Roofline analysis of batched inference across language model variants.** Roofline placement of decoding in autoregressive (AR) models, vanilla DLMs, and block-wise DLMs with block sizes $B \in \{4, 16, 32\}$ under batch sizes $\mathrm{bs} \in \{1, 2, 4, 8, 16, 32, 64, 128\}$. All results are obtained via roofline simulation assuming an NVIDIA A100-SXM4-80GB GPU with peak FP16 performance of 311.9 TFLOP/s and memory bandwidth of 2039.0 GB/s.

saturate around $\mathrm{bs} = 64$ for $B = 4$, $\mathrm{bs} = 16$ for $B = 16$, and $\mathrm{bs} = 8$ for $B = 32$.

Overall, block-wise DLMs occupy a middle ground between AR models, which are memory-bound, and vanilla DLMs, which are compute-bound. By exploiting parallelism within blocks while retaining moderate AI, block-wise decoding improves compute utilization in low-batch regimes without immediately saturating available compute.

C Limitations and Future Directions

Our current training relies on offline, static trajectories. Although we carefully select datasets to limit domain bias, the student can still overfit to the teacher's priors because supervision does not adapt during training. A straightforward direction is to broaden the data mixture and scale beyond our current $\sim$15k samples to cover more diverse domains.

A complementary path is to move from offline supervision to on-the-fly generation. If the student generates trajectories during training and the teacher verifies or refines them online, on-policy learning could reduce the training–inference discrepancy. In practice, this is currently constrained by DLM sampling speed: slow generation limits the feasibility of closed-loop training without prior trajectory materialization. Improving DLM inference throughput is therefore a prerequisite for large-scale online training.

Another promising direction is to use CDLM as a draft model within speculative decoding frameworks (Leviathan et al., 2023), where diffusion models can serve as fast proposers for autoregressive verifiers. A naive DLM drafter yields limited benefit because standard DLMs require many refinement steps to produce coherent text. In contrast, a *consistency*-trained DLM (CDLM) can generate reasonable drafts in far fewer steps. Pairing a CDLM drafter with an autoregressive (AR) verifier (cf. (Christopher et al., 2025)) is promising: the CDLM proposes multi-token candidates quickly, and the AR model supplies strong final validation.

CDLM's performance is ultimately bounded by the teacher: a bidirectional DLM distilled into a block-causal student cannot exceed the teacher's knowledge. To lift this ceiling, distillation from stronger autoregressive (AR) teachers is a natural next step. In this work we intentionally used high-quality, third-party-curated Qwen2.5-7B generations (Lansechen, 2025a;b) as ground-truth text, hoping to inject AR knowledge into the student. A more direct, and likely more effective, approach is to distill from an AR teacher: AR models are already scaled up, typically stronger, and much faster at generation than naive DLMs, making large-scale dataset construction easier. Identifying effective ways to extract suitable training trajectories from AR models is therefore a key prerequisite.