CROSS-DLLM: CROSS-BLOCK DIFFUSION DECODING WITH AUTOREGRESSIVE DRAFTING FOR COMPLEX REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models offer a promising alternative to autoregressive generation, but their iterative denoising process incurs high inference latency, limiting practical deployment for complex reasoning tasks. To address this issue, we propose **Cross-dLLM**, a cross-block diffusion decoding framework guided by lightweight autoregressive drafting. Specifically, an auxiliary autoregressive model first generates coarse-grained semantic reasoning drafts, which are subsequently used to guide diffusion decoding over multiple semantic blocks in parallel. By extending parallel decoding from token-level denoising to semantic-block-level generation, Cross-dLLM improves decoding efficiency and reasoning robustness by introducing a semantic skeleton as a structural prior, which guides the parallel diffusion process and mitigates the impact of imperfect planning drafts. We evaluate Cross-dLLM on diverse reasoning benchmarks, including GSM8K, MATH500, HellaSwag, ARC-C, and ARC-E, using LLaDA-8B as the base diffusion model. Experimental results show that Cross-dLLM consistently outperforms standard LLaDA decoding, achieving up to **27.1%** accuracy improvement while providing an approximately **2.29**$\times$ inference speedup. These results demonstrate that autoregressive semantic drafting provides an effective mechanism for improving both the efficiency and reasoning capability of diffusion language models. The code and implementation details are publicly available at `https://anonymous.4open.science/r/CBD-2278`.

1 INTRODUCTION

Large language models have achieved strong performance on complex reasoning tasks through autoregressive chain-of-thought generation Wei et al. (2022). However, token-by-token decoding causes inference time to increase with the length of the reasoning trajectory, particularly when complex problems require lengthy intermediate derivations. Diffusion language models (dLLMs) provide greater parallelism by recovering multiple masked tokens at each denoising step Sahoo et al. (2024). Nevertheless, aggressive parallel decoding may disrupt dependencies between intermediate reasoning steps, introducing a trade-off between generation efficiency and reasoning coherence Wu et al. (2025).

Recent work improves dLLM inference through confidence-based token selection, lightweight autoregressive guidance, and structured planning Wu et al. (2025); Israel et al. (2025); Li et al. (2026). Planned Diffusion Israel et al. (2025) first generates an autoregressive plan and then produces multiple plan-defined semantic segments through diffusion. Building on the broader planning-before-execution paradigm, Cross-dLLM considers a setting in which an external lightweight planner provides an explicit reasoning skeleton to a pretrained diffusion executor. Two challenges arise in this setting: reasoning regions must be generated concurrently despite their semantic dependencies, and the executor must remain effective when the planner produces incomplete or inaccurate skeletons. Addressing these challenges requires combining structural guidance with parallel decoding and robustness to imperfect planning signals.

We propose **Cross-dLLM**, as illustrated in Fig. 1, which connects autoregressive planning with diffusion-based reasoning generation. The planner produces a concise summary and ordered intermediate steps that serve as shared semantic anchors. Conditioned on these anchors, the diffusion executor

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

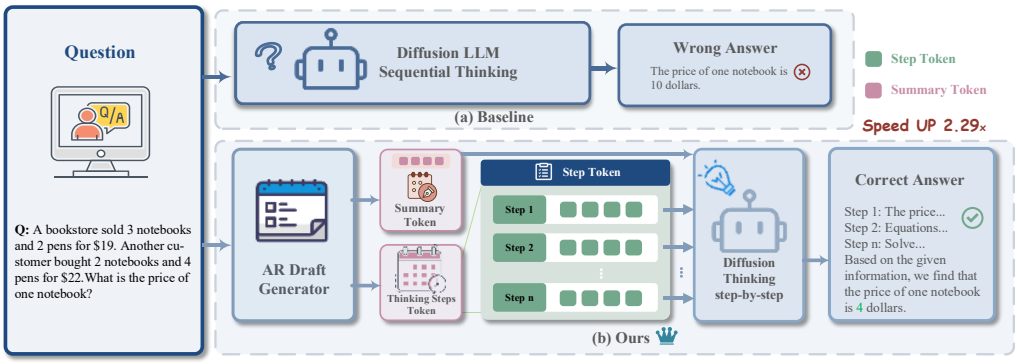


Figure 1: **Comparison of baseline diffusion reasoning and our draft-guided diffusion reasoning.** (a) In the illustrated baseline pipeline, reasoning proceeds through locally decoded regions without an explicit global reasoning skeleton, leading to an incorrect answer. (b) In our pipeline, an autoregressive draft generator first produces summary and step tokens, which guide the subsequent diffusion reasoning process and yield the correct answer with improved inference speed.

performs cross-block decoding, updating multiple skeleton-defined reasoning regions within the same denoising process rather than completing them sequentially. Each region receives a prescribed token-update budget, while token commitment is determined by confidence scores derived from shared global predictions. To mitigate training–inference mismatch, we further introduce a robust skeleton-conditioning strategy that combines reference, perturbed, planner-generated, and dropped skeletons. The dropout component, termed **Drop-AR**, reduces the executor's dependence on accurate planner outputs.

We evaluate Cross-dLLM on GSM8K Cobbe et al. (2021), MATH500 Hendrycks et al. (2021), HellaSwag Zellers et al. (2019), ARC-C, and ARC-E Clark et al. (2018). On a fixed 1,000-example GSM8K subset, Cross-dLLM improves strict-match accuracy from 67.6% to 74.8% and reduces end-to-end inference time from 293 to 128 minutes, corresponding to a 2.29× speedup. A matched-data LLaDA-SFT baseline trained on the same reasoning trajectories without skeleton conditioning reaches 68.9%, indicating that additional trajectory supervision alone does not account for the improvement. Controlled comparisons further distinguish the accuracy benefits of skeleton conditioning from the efficiency gains of cross-block decoding.

The main contributions are as follows.

- We propose **Cross-dLLM**, which connects a lightweight autoregressive planner and a diffusion executor through an explicit multi-step reasoning skeleton.
- We develop a **cross-block decoding strategy** that updates multiple skeleton-defined reasoning regions within the same denoising process, improving inference efficiency while retaining most of the accuracy of intra-block decoding.
- We introduce a robust skeleton-conditioning strategy, including **Drop-AR**, to reduce sensitivity to imperfect planner outputs. Controlled experiments distinguish the effects of trajectory supervision, skeleton conditioning, and cross-block decoding.

2 Related Work

Autoregressive Generation Paradigm. Autoregressive large language models (AR LLMs) generate text left-to-right, maintaining sequential dependencies that enable coherent multi-step reasoning. Chain-of-Thought prompting Wei et al. (2022) further improves reasoning performance on tasks such as mathematics and logic. Despite high-quality inference, sequential decoding inherently incurs latency that grows linearly with output length, limiting efficiency for long-text generation and high-concurrency scenarios.

Diffusion-based Generative Paradigm. The success of diffusion models in image generation has inspired discrete diffusion language models (dLLMs) as an alternative to sequential autoregressive

generation. Early works include Diffusion-LM Li et al. (2022), D3PM Austin et al. (2021), SEDD Lou et al. (2023), and DiffusionBERT He et al. (2022). More recently, masked diffusion models such as LLaDA Nie et al. (2025) have been extended to sparse MoE architectures Zhu et al. (2025), multimodal instruction tuning You et al. (2025), and long-context inference Liu et al. (2025).

Hybrid Generative Strategies. Recent studies combine autoregressive and diffusion models to exploit their complementary strengths. DEER Cheng et al. (2025) uses diffusion for draft generation and autoregressive verification, while Adaptive Parallel Decoding Israel et al. (2025) uses lightweight AR probabilities to guide parallel token commitment. Eso-LMs and LaDiR Sahoo et al. (2025); Kang et al. (2025) further explore autoregressive collaboration with masked or latent diffusion. Another line of work separates planning from generation. Plan-and-Solve Prompting, Skeleton-of-Thought, and Tree of Thoughts Wang et al. (2023); Ning & others (2023); Yao et al. (2023) introduce intermediate structures for reasoning, while Planner and Executor Berrayana et al. (2025) studies AR–diffusion collaboration across text and latent spaces. Planned Diffusion Israel et al. (2025) is most closely related to Cross-dLLM, using autoregressive plans to partition outputs into semantic spans for parallel diffusion generation. Cross-dLLM follows the same planning-before-execution paradigm but employs a lightweight planner to produce multi-step reasoning skeletons for a pretrained diffusion executor. Drop-AR further improves robustness to imperfect planning signals.

Efficient Inference for dLLMs. The iterative denoising process makes dLLM inference computationally expensive Sahoo et al. (2024). Fast-dLLM Wu et al. (2025) combines approximate blockwise KV-cache reuse with confidence-aware parallel decoding. Guided Diffusion Hu et al. (2025) uses a lightweight autoregressive model to guide token unmasking, while BlockSpec Pan et al. (2026) explores blockwise speculative decoding. Plan, Verify and Fill Li et al. (2026) is a training-free structured decoding method that constructs a hierarchical skeleton by selecting and verifying semantic anchors before filling the remaining content. In contrast, Cross-dLLM conditions a pretrained diffusion executor on an external multi-step reasoning skeleton and trains it to tolerate varying skeleton quality. During cross-block decoding, skeleton-defined regions are updated with separate token budgets within the same denoising process, enabling more reliable execution under imperfect reasoning plans.

3 METHODOLOGY

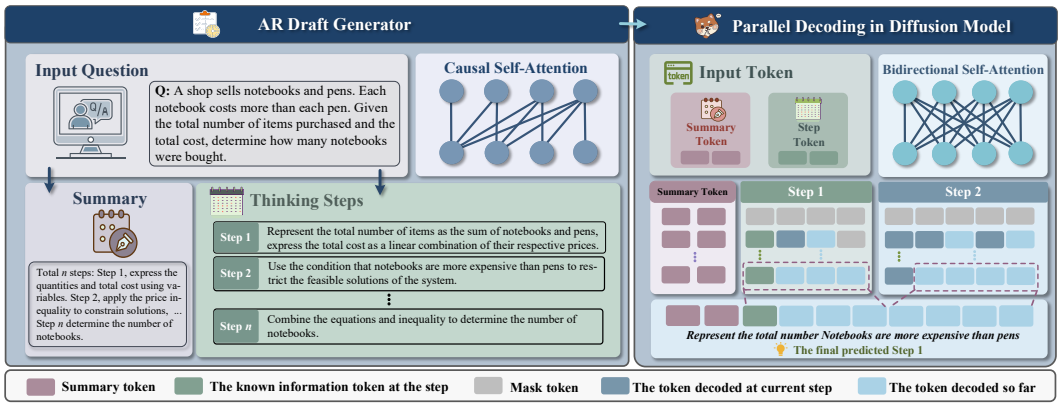


Figure 2: **Overview of Cross-dLLM. Left:** A lightweight autoregressive planner generates a compact reasoning skeleton with a global summary and ordered step-level anchors. **Right:** Conditioned on these anchors, the diffusion executor performs cross-block decoding, updating multiple reasoning regions within the same denoising process. Color legend: pink (summary), green (step anchors), gray (masked), dark blue (newly committed), and light blue (previously committed) tokens.

As illustrated in Fig. 2, Cross-dLLM first uses a lightweight autoregressive planner to generate a compact reasoning skeleton s. Conditioned on this skeleton, the diffusion executor reconstructs the detailed reasoning trajectory y by updating multiple skeleton-defined regions within the same denoising process.

3.1 Training Data Preparation

Our training data combines GSM8K, MATH, OBQA, and QASC to cover diverse reasoning patterns, including arithmetic, symbolic, and commonsense reasoning. Specifically, we use approximately 7.5k GSM8K samples, 300 high-difficulty MATH samples, and 1,000 OBQA/QASC examples. We include OBQA and QASC to enrich structured reasoning patterns and improve skeleton planning robustness. For each instance, we construct a structured reasoning skeleton s as a high-level planning signal. Following Figure 3, GPT-4o generates reasoning drafts containing concise summaries and ordered steps using a unified prompt. The generated drafts are filtered by automatic format and semantic consistency checks, producing the final skeleton s and reasoning trajectory y. For datasets without annotated reasoning, skeletons are synthesized using the same pipeline. The resulting training triplet (x, s, y) contains the input question x, skeleton s, and full reasoning trajectory y. The skeleton provides coarse-grained planning signals for the AR planner, while y provides dense supervision for the diffusion decoder to generate detailed reasoning conditioned on s.

3.2 Problem Formulation: The Hybrid Generative Paradigm

Autoregressive models capture sequential dependencies effectively, but token-by-token generation incurs linear latency, limiting scalability for complex reasoning. To address this, we model reasoning as a hierarchical generative process by introducing an explicit structural variable s to separate high-level planning from low-level execution. The planning skeleton s provides a high-level abstraction of the reasoning trajectory. It preserves core logical structure while omitting surface details.

Given an input context x and a target sequence y, the joint distribution is factorized as

$$p(y, s \mid x) = p_{\text{AR}}(s \mid x; \phi) \cdot p_{\text{Diff}}(y \mid s, x; \theta). \tag{1}$$

Here, $p_{\text{AR}}(s \mid x; \phi)$ denotes an autoregressive planner that generates the intermediate skeleton s sequentially, while $p_{\text{Diff}}(y \mid s, x; \theta)$ represents a diffusion-based decoder that refines the skeleton in parallel to produce the final output y. This factorization explicitly decouples high-level reasoning from fine-grained sequence generation, enabling serial planning together with parallelizable execution.

3.3 Autoregressive Skeleton Planning

In our framework, the reasoning skeleton s encodes the high-level logical structure of the reasoning process and serves as an intermediate planning representation for diffusion decoding. Each skeleton consists of a concise summary describing the overall reasoning objective and a sequence of coarse-grained reasoning steps capturing major intermediate transitions while omitting low-level lexical details and arithmetic derivations. This abstraction bridges serial planning and parallel execution across diverse reasoning tasks.

Skeletons are generated by a lightweight autoregressive planner $\mathcal{M}_{\text{AR}}$, parameterized as $p_\phi(s \mid x)$. Since skeletons are substantially shorter than full reasoning trajectories, the serial planning stage introduces only limited computational overhead while preserving global reasoning structure. Conditioned on the generated skeleton, the diffusion decoder reconstructs the complete reasoning trajectory in parallel. The AR planner is trained on (x, s) pairs using standard teacher-forcing autoregressive training.

3.4 Skeleton-Conditioned Discrete Diffusion

Given an input x and skeleton s, the diffusion decoder reconstructs the full reasoning trajectory y, including intermediate steps and the final answer. The conditional distribution $p_{\text{Diff}}(y \mid s, x)$ is implemented as a discrete denoising diffusion model conditioned on the upstream AR-generated skeleton. We adopt an absorbing-state masked diffusion process where tokens are randomly replaced with $[MASK]$ according to schedule α_t. The reverse network $f_\theta(y_t, x, s)$ conditions on the partially masked sequence together with x and s to reconstruct globally consistent reasoning trajectories. The training objective minimizes cross-entropy over masked positions:

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{t, y_0} \left[\sum_{i \in \mathcal{M}_t} -\log p_\theta(y_0^i \mid y_t, x, s) \right], \tag{2}$$

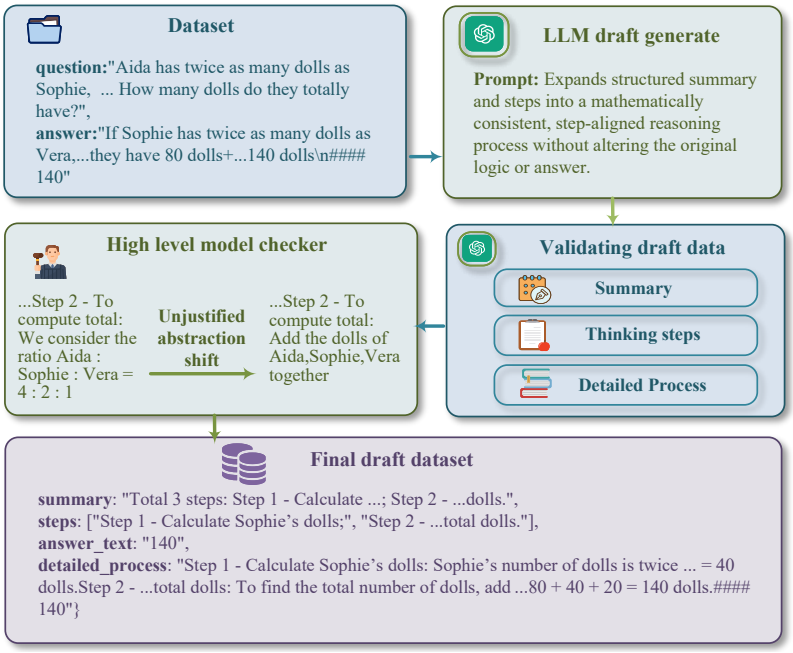


Figure 3: **Data construction pipeline of Cross-dLLM.** A GPT-powered generator expands QA pairs into reasoning drafts, followed by a lightweight validator for format consistency and step alignment and a high-level model checker for semantic errors. The cleaned drafts yield the skeleton s and full trajectory y, forming the training triplet (x, s, y).

where $\mathcal{M}_t$ denotes masked positions at step t. Conditioning on s constrains generation toward trajectories consistent with the high-level reasoning plan while preserving parallel decoding efficiency.

3.5 Robust Training Strategy

Hybrid AR-diffusion architectures face a training-inference mismatch: the diffusion decoder is conditioned on ground-truth skeletons s_{gt} during training, but relies on AR-generated skeletons $\hat{s}$ at inference, which may contain errors. This discrepancy can induce exposure bias, reducing generation quality. This design directly addresses the training-inference mismatch identified in our problem formulation, where the diffusion decoder is trained on perfect skeletons but must operate on potentially flawed AR-generated ones at test time.

To improve robustness, we construct the conditioning skeleton $\tilde{s}$ as a stochastic mixture:

$$\tilde{s} \sim \begin{cases} s_{gt} & 1-\gamma_1-\gamma_2-p_{\text{drop}}, \\ \text{Perturb}(s_{gt}, \epsilon) & \gamma_1, \\ \hat{s}_{AR}(x) & \gamma_2, \\ \emptyset & p_{\text{drop}}. \end{cases} \tag{3}$$

Here, $\text{Perturb}(s_{gt}, \epsilon)$ randomly modifies skeleton steps via deletion, swapping, or token masking, with intensity controlled by ϵ. We set $\gamma_1 = 0.4$, $\gamma_2 = 0.3$, $p_{\text{drop}} = 0.1$, and $\epsilon = 0.3$. This combination of stochastic perturbation and Drop-AR prevents overfitting to perfect skeletons, allowing the diffusion decoder to reconstruct coherent reasoning trajectories from noisy, partial, or missing skeletons, thereby improving inference stability and robustness.

3.6 Cross-Block Diffusion Decoding

Algorithm Overview. We integrate AR skeleton planning with block-wise confidence-based diffusion decoding. The generated sequence is partitioned into blockwise semantic regions. At each denoising

step, each block selects tokens based on confidence scores derived from shared global predictions. This eliminates cross-block sequential dependencies, enabling fully parallel generation.

Skeleton-Aware Initialization. Skeleton s is decomposed into K semantic anchors $[s^{(1)}, \ldots, s^{(K)}]$. The diffusion state y_T is initialized by interleaving anchors with masked spans:

$$y_T = \text{Concat}(x, s^{(1)}, \mathbf{m}, \ldots, s^{(K)}, \mathbf{m}),$$

where $\mathbf{m}$ denotes a fixed-length masked block. Each block is bounded by its surrounding anchors, reducing semantic drift while preserving global structure.

Per-Block Uniform Scheduling. Masked positions $\mathcal{M}$ are partitioned into K blocks of equal length L. For each block b, the number of tokens updated at diffusion step t is defined as

$$k_{t,b} = \left\lfloor \frac{L}{T} \right\rfloor + \mathbf{1}(t \leq L \bmod T). \tag{4}$$

The schedule is applied for $t = T, \ldots, 1$, ensuring balanced token allocation across diffusion steps while maintaining fully parallel block-wise execution and stable GPU utilization.

Blockwise Confidence-Based Unmasking. At each diffusion step, candidate tokens are sampled as

$$\hat{y}_i = \arg\max_{v \in \mathcal{V}} (\ell_{i,v} + g_{i,v}), \tag{5}$$

where $g_{i,v} \sim \text{Gumbel}(0, 1)$. Token-level confidence is then computed as $c_i = \max \text{softmax}(\ell)_{i,v}$.

Within each block, the top-$k_{t,b}$ most confident tokens are unmasked, while the remaining positions stay masked. Independent block execution enables parallel blockwise decoding, and the easy-first strategy stabilizes partial contexts, improving global consistency.

3.7 Theoretical Motivation

Parallel Decoding Approximation. Blockwise parallel decoding can be viewed as an approximation of the sequential joint distribution by reducing interactions among semantic blocks. Let $P = p(X_{B_i}, X_{B_j} \mid C_t)$ denote the true conditional joint distribution between two blocks and $Q = p(X_{B_i} \mid C_t)p(X_{B_j} \mid C_t)$ denote its factorized approximation. The discrepancy between P and Q can be characterized by their KL divergence:

$$D_{\text{KL}}(P \| Q) = I(X_{B_i}; X_{B_j} \mid C_t), \tag{6}$$

where $I(\cdot; \cdot | \cdot)$ measures the remaining dependency between blocks conditioned on the current context.

By Pinsker's inequality, the total variation distance between the joint distribution and its factorized approximation satisfies:

$$|P - Q|_{\text{TV}} \leq \sqrt{\frac{1}{2} I(X_{B_i}; X_{B_j} \mid C_t)}. \tag{7}$$

This provides an interpretation of blockwise parallel decoding: when semantic dependencies between blocks are weaker, the factorized approximation may become more accurate. However, this analysis characterizes distributional approximation error under an idealized setting and does not constitute a direct guarantee of generation accuracy.

Skeleton-Guided Dependency Reduction. The planning skeleton s serves as a shared semantic representation that provides high-level structural information across reasoning blocks. Conditioning on s introduces additional global context into blockwise generation, which can reduce uncertainty caused by missing cross-block information and facilitate more coherent parallel decoding. During training, the skeleton is treated as a conditioning signal with perturbation and dropout strategies to improve robustness against imperfect guidance. During inference, the AR-generated skeleton provides an approximate semantic plan that guides diffusion generation. Therefore, skeleton conditioning is expected to alleviate part of the dependency mismatch between independently generated blocks, although the actual effect depends on skeleton quality and decoding dynamics.

Training–Inference Distribution Shift. Training typically conditions on ground-truth skeletons $p_{\text{train}}(s) = \delta_{s_{\text{gt}}}$, whereas inference relies on AR-generated skeletons $p_{\text{inf}}(s)$. This discrepancy introduces a distribution shift between training and deployment. To improve robustness, our Drop-AR

Table 1: **Zero-shot comparison with autoregressive, diffusion, and hybrid baselines.** Results are reported on GSM8K, MATH500, HellaSwag, ARC-C, and ARC-E.

Category	Method	Backbone (Param)	GSM8K ↑	MATH500 ↑	HellaSwag	ARC-C	ARC-E
Autoregressive Baselines							
AR	Qwen-2.5-7B Qwen et al. (2024)	7B	80.1	45.1	55.8	54.7	77.4
AR	Qwen-2.5-3B Qwen et al. (2024) (Planner)	3B	66.2	36.2	50.1	52.1	70.0
Diffusion & Hybrid Baselines							
Diff	LLaDA-8B Nie et al. (2025) (Baseline)	8B	67.6	26.1	50.6	32.1	70.5
Diff	Dream Ye et al. (2025)	7B	70.3	31.4	73.3	56.3	82.3
Hybrid	LaDiR Kang et al. (2025)	8B	53.9	25.4	51.7	49.8	73.5
Ours	Draft-Conditioned Parallel Diffusion	**3B + 8B**	**74.8**	**36.8**	**68.3**	**59.2**	**75.3**

strategy constructs a broader training distribution p_{mix} by combining perturbed, generated, and partially removed skeleton conditions. This encourages the diffusion executor to remain effective under imperfect skeleton guidance and reduces sensitivity to the difference between training-time and inference-time skeletons.

4 Experiments

All experiments are conducted on NVIDIA H100 GPUs with 80GB memory. We evaluate **Cross-dLLM** on reasoning accuracy and inference efficiency. Five-seed stability and statistical uncertainty analyses on GSM8K are provided in Appendix B.7. We evaluate Cross-dLLM on mathematical reasoning and commonsense/science reasoning benchmarks, including GSM8K Cobbe et al. (2021), MATH500 Hendrycks et al. (2021), HellaSwag Zellers et al. (2019), ARC-Easy, and ARC-Challenge Clark et al. (2018). We compare against representative autoregressive, diffusion, and hybrid generation baselines under the same evaluation protocol and hardware settings. Unless otherwise specified, no in-context examples are provided during inference. Detailed benchmark descriptions, baseline configurations, and implementation details are provided in Appendix B.

4.1 Main Results

Comparison with Baselines. Table 1 compares Cross-dLLM with representative autoregressive, diffusion, and hybrid baselines under zero-shot evaluation. Relative to the LLaDA-8B diffusion baseline Nie et al. (2025), Cross-dLLM achieves higher accuracy across all five benchmarks. Specifically, accuracy increases from 67.6% to 74.8% on GSM8K, from 26.1% to 36.8% on MATH500, and from 32.1% to 59.2% on ARC-C. These improvements demonstrate the effectiveness of combining explicit reasoning skeletons with skeleton-conditioned diffusion execution. The gains on HellaSwag and ARC-E further indicate that the framework's benefits extend beyond mathematical reasoning.

Compared with the lightweight Qwen-2.5-3B planner Qwen et al. (2024), Cross-dLLM achieves higher accuracy on all five tasks. This suggests that the final system performance is not simply inherited from the planner itself, but benefits from the subsequent skeleton-conditioned diffusion execution. The larger Qwen-2.5-7B model remains stronger on GSM8K and MATH500, while Cross-dLLM achieves higher accuracy on ARC-C, illustrating that the relative advantage depends on the task and model scale.

Among diffusion and hybrid baselines, Cross-dLLM outperforms LaDiR Kang et al. (2025) across all five evaluated tasks. Compared with Dream Ye et al. (2025), Cross-dLLM performs better on GSM8K, MATH500, and ARC-C, whereas Dream remains stronger on HellaSwag and ARC-E. These results indicate that the relative benefit of skeleton-guided diffusion varies across task types. Planned Diffusion Israel et al. (2025) is the closest conceptual prior, but its reported evaluation follows a different instruction-following and latency-oriented protocol rather than the reasoning benchmarks in Table 1. We therefore discuss it separately in Related Work instead of mixing non-comparable cross-paper results into the main table.

Generalization Analysis. We further evaluate Cross-dLLM's generalization beyond mathematical reasoning. On HellaSwag, ARC-C, and ARC-E, Cross-dLLM consistently improves over the LLaDA baseline, with the largest gain observed on ARC-C, where accuracy increases by 27.1 percentage points, from 32.1% to 59.2%. These results suggest that skeleton-conditioned diffusion can benefit

commonsense and science reasoning in addition to arithmetic reasoning. However, Cross-dLLM does not uniformly outperform other diffusion models: Dream achieves higher accuracy on HellaSwag and ARC-E. This variation highlights the task-dependent nature of the observed performance gains.

4.2 Ablation Studies

4.2.1 Training Strategies

Table 2 shows the impact of different training strategies. The baseline LLaDA-8B Nie et al. (2025) achieves 67.6% accuracy. Drop-AR (randomly dropping AR drafts during training) improves accuracy to 71.8%, suggesting that this strategy helps the diffusion model maintain reasoning performance even with partial guidance. Ground-truth training (conditioning on accurate skeletons) achieves 70.1%, showing limited gains from perfect skeletons alone. The hybrid strategy combining Drop-AR and ground-truth guidance reaches 74.8% accuracy, higher than either alone, reflecting the complementary effect: robustness to imperfect drafts while retaining the ability to learn from accurate skeletons.

Table 2: **Ablation Study on Training Strategies.** We evaluate the impact of different training strategies on GSM8K Cobbe et al. (2021) using exact-match (EM) accuracy. *Baseline* refers to the original LLaDA-8B model. Δ denotes improvement over the baseline.

Training Strategy	Acc (EM)	Δ
Baseline	67.6	–
Drop-AR	71.8	+4.2
Ground-truth	70.1	+2.5
Hybrid	**74.8**	**+7.2**

Decoding Strategies. Table 3 examines the effects of guidance input, executor training, and decoding parallelism on GSM8K. A pre-generated reasoning draft refers to an offline-prepared intermediate reasoning chain with the final answer removed. Direct preserves the original draft layout, whereas Step-aligned aligns reasoning steps with their corresponding masked generation regions. Naive multi-block increases decoding parallelism without step alignment. Providing a pre-generated draft improves the accuracy of raw LLaDA from 67.6% to 77.1% under intra-block decoding. However, naive multi-block decoding reduces accuracy to 66.4%, illustrating the difficulty of increasing parallelism without explicit structural alignment. Step-aligned cross-block decoding recovers accuracy to 69.3%, which further increases to 73.8% with executor fine-tuning. The pre-generated draft configurations serve as diagnostic settings and are not directly comparable to the online Cross-dLLM pipeline. Under the same AR-generated skeleton and fine-tuned executor, cross-block decoding reduces inference time from 273 to 128 minutes, while accuracy decreases from 76.6% to 74.8%. This controlled comparison demonstrates that cross-block decoding substantially improves efficiency while retaining most of the accuracy achieved by intra-block decoding.

4.3 System Efficiency

Our system consists of an AR planner (Qwen-2.5-3B Qwen et al. (2024)) and a diffusion generator (LLaDA-8B Nie et al. (2025)). On 4 H100 GPUs, the planner achieves 157.79 tokens/s with 0.61s latency per sample. The diffusion generator dominates single-sample inference time, while the optimized design reduces latency by approximately 74% relative to the original LLaDA-8B, improving suitability for offline reasoning. Batching improves diffusion throughput despite the cost of iterative denoising. As shown in Table 6 in Appendix B.1.2, increasing the batch size from 1 to 32 raises throughput from 14.6 to 21.5 tokens/s; the previously reported 3.3$\times$ gain is relative to non-batched execution at 6.5 tokens/s. The maximum batch size is 32 due to GPU memory constraints (60.6 GB), while the AR planner adds limited overhead at larger batches.

Overall, the hybrid design improves reasoning accuracy at the cost of additional computation. Batching and cross-block decoding reduce diffusion overhead, making the system suitable for offline reasoning where accuracy is prioritized over strict latency constraints.

Table 3: **Comparison of guidance, executor training, and decoding configurations.** We report strict-match accuracy and inference latency on a fixed 1,000-example subset of GSM8K using 4 H100 GPUs. Relative latency is normalized to standard LLaDA decoding. "4 / 3.34" denotes four concurrently decoded reasoning blocks with an average of 3.34 committed tokens per denoising step.

Guidance Input	Layout	Executor	Decoding	Parallelism	Strict Match	Latency	Time
None	N/A	Raw LLaDA	Intra-block	1	67.6%	1.00×	293s
Pre-generated reasoning draft	Direct	Raw LLaDA	Intra-block	1	77.1%	0.95×	279s
Pre-generated reasoning draft	Direct	Raw LLaDA	Naive multi-block	4	66.4%	0.47×	137s
Pre-generated reasoning draft	Step-aligned	Raw LLaDA	Cross-block	4 / 3.34	69.3%	0.36×	105s
Pre-generated reasoning draft	Step-aligned	Fine-tuned LLaDA	Cross-block	4 / 3.34	73.8%	0.43×	125s
AR-generated reasoning skeleton	Step-aligned	Fine-tuned LLaDA	Intra-block	1	76.6%	0.93×	273s
AR-generated reasoning skeleton	Step-aligned	Fine-tuned LLaDA	Cross-block	4 / 3.34	74.8%	0.44×	128s

4.4 Generalization with Lightweight AR Planners

To evaluate generalization across different autoregressive planners, we fix the diffusion decoder (LLaDA-8B Nie et al. (2025)) and replace the AR planner with lightweight open-source models. Experiments are conducted under zero-shot inference on GSM8K Cobbe et al. (2021), MATH500 Hendrycks et al. (2021), HellaSwag Zellers et al. (2019), ARC-C, and ARC-E Clark et al. (2018). We evaluate three planners with different capacities: Qwen-2.5-0.5B Qwen et al. (2024), DeepSeek-Distill-Qwen-1.5B Guo et al. (2025), and Gemma-2B Gemma Team & Google DeepMind (2024). All settings share the same training and inference pipeline, including Drop-AR, hybrid supervision, and cross-block decoding, with only the AR planner varied.

As shown in Table 7 in Appendix B.2, Cross-dLLM consistently improves over LLaDA-8B Nie et al. (2025) with planners ranging from 0.5B to 2B parameters, demonstrating robustness to lightweight AR models. Larger planners generally provide better skeletons and yield higher performance, while even the smallest planner achieves clear gains, validating the effectiveness of Drop-AR under imperfect drafts. All configurations maintain similar efficiency improvements, achieving $2.4\times$–$2.8\times$ speedup, showing that cross-block decoding benefits are largely independent of planner scale.

4.5 Impact of Skeleton Quality

Table 8 in Appendix B.4 compares different skeleton extraction strategies and their impact on reasoning performance. The Refined method first generates a logical skeleton using an LLM, then extracts reasoning-relevant tokens, numerals, operators, and step markers through regular expressions, followed by rule-based consistency verification and keyword augmentation. In contrast, Coarse Extraction relies only on predefined keywords without semantic modeling, while Heuristic Chain segments text by sentences and treats each sentence as an independent reasoning unit, often producing fragmented structures. On GSM8K Cobbe et al. (2021), the Refined method achieves 74.8% accuracy, outperforming Coarse (57.3%) and Heuristic (48.2%). This demonstrates that preserving logical structure, multi-step dependencies, and semantic coherence in skeletons is important for improving downstream reasoning performance. Due to space constraints, we provide a few-shot evaluation in Appendix B.3, where Cross-dLLM exhibits more stable token-level predictions and higher log-likelihood than the baseline when trained with only 400 examples.

5 Conclusion

We introduced Cross-dLLM, a hybrid framework that combines autoregressive drafting with semantic-aware cross-block diffusion decoding. Across diverse reasoning benchmarks, Cross-dLLM consistently improves over standard diffusion baselines, achieving accuracy gains of up to **27.1%** and up to a **2.29× speedup** in inference. By using structured drafts to guide parallel generation across semantic regions, Cross-dLLM further outperforms conventional intra-block parallelism under comparable settings. These results demonstrate that lightweight autoregressive planning can provide effective structural guidance for improving the accuracy–efficiency trade-off of diffusion language model reasoning.

AI USE STATEMENT

Generative AI tools were used in several stages of this work. For research ideation and execution, they were used to assist with brainstorming and refining methodological variants, implementation and debugging, and the design of experimental checks. Generative AI was also used during data construction to generate structured reasoning drafts and auxiliary reasoning annotations used in our training pipeline; these generated outputs were subsequently subjected to filtering and quality-control procedures before use. In addition, generative AI tools were used to draft portions of the manuscript and to improve the clarity, grammar, and readability of author-written text.

All AI-assisted research outputs, code, experimental results, and manuscript text were reviewed and verified by the authors. The authors made the final decisions regarding the methodology, experimental design, interpretation of results, and scientific claims, and take full responsibility for the content of this work.

REFERENCES

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured Denoising Diffusion Models in Discrete State-Spaces. *arXiv e-prints*, art. arXiv:2107.03006, July 2021. doi: 10.48550/arXiv.2107.03006.

Lina Berrayana, Ahmed Heakl, Muhammad Abdullah Sohail, Thomas Hofmann, Salman Khan, and Wei Chen. Planner and executor: Collaboration between discrete diffusion and autoregressive models in reasoning. *arXiv preprint arXiv:2510.15244*, 2025.

Zicong Cheng, Guo-Wei Yang, Jia Li, Zhijie Deng, Meng-Hao Guo, and Shi-Min Hu. DEER: Draft with Diffusion, Verify with Autoregressive Models. *arXiv e-prints*, art. arXiv:2512.15176, December 2025. doi: 10.48550/arXiv.2512.15176.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv e-prints*, art. arXiv:1803.05457, March 2018. doi: 10.48550/arXiv.1803.05457.

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. Training Verifiers to Solve Math Word Problems. *arXiv e-prints*, art. arXiv:2110.14168, October 2021. doi: 10.48550/arXiv.2110.14168.

Gemma Team and Google DeepMind. Gemma: Open Models Based on Gemini Research and Technology. *arXiv e-prints*, art. arXiv:2403.08295, March 2024. doi: 10.48550/arXiv.2403.08295. URL https://arxiv.org/abs/2403.08295.

Daya Guo, Dejian Yang, Haowei Zhang, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Zhengfu He, Tianxiang Sun, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. DiffusionBERT: Improving Generative Masked Language Models with Diffusion Models. *arXiv e-prints*, art. arXiv:2211.15029, November 2022. doi: 10.48550/arXiv.2211.15029. URL https://arxiv.org/abs/2211.15029.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. *arXiv e-prints*, art. arXiv:2103.03874, March 2021. doi: 10.48550/arXiv.2103.03874.

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

Daniel Israel, Tian Jin, Ellie Cheng, Guy Van den Broeck, Aditya Grover, Suvinay Subramanian, and Michael Carbin. Planned diffusion. *arXiv preprint arXiv:2510.18087*, 2025.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating Diffusion LLMs via Adaptive Parallel Decoding. *arXiv e-prints*, art. arXiv:2506.00413, June 2025. doi: 10.48550/arXiv.2506.00413. URL https://arxiv.org/abs/2506.00413.

Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Nicklas Majamaki, Navdeep Jaitly, Yi-An Ma, and Lianhui Qin. LaDiR: Latent Diffusion Enhances LLMs for Text Reasoning. *arXiv e-prints*, art. arXiv:2510.04573, October 2025. doi: 10.48550/arXiv.2510.04573.

Miao Li, Hanyang Jiang, Sikai Cheng, Hengyu Fu, Yuhang Cai, Baihe Huang, Tinghan Ye, Xuanzhou Chen, and Pascal Van Hentenryck. Plan, Verify and Fill: A Structured Parallel Decoding Approach for Diffusion Language Models. *arXiv e-prints*, art. arXiv:2601.12247, January 2026. doi: 10.48550/arXiv.2601.12247. URL https://arxiv.org/abs/2601.12247.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-lm improves controllable text generation, 2022. URL https://arxiv.org/abs/2205.14217.

Xiaoran Liu, Yuerong Song, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. Longllada: Unlocking long context capabilities in diffusion llms, 2025. URL https://arxiv.org/abs/2506.14429.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution. *arXiv e-prints*, art. arXiv:2310.16834, October 2023. doi: 10.48550/arXiv.2310.16834. URL https://arxiv.org/abs/2310.16834.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large Language Diffusion Models. *arXiv e-prints*, art. arXiv:2502.09992, February 2025. doi: 10.48550/arXiv.2502.09992.

Xuefei Ning and others. Skeleton-of-Thought: Prompting LLMs for Efficient Parallel Generation. *arXiv e-prints*, art. arXiv:2307.15337, July 2023. doi: 10.48550/arXiv.2307.15337. URL https://arxiv.org/abs/2307.15337.

Tianxiang Pan, Baitao Gong, Mo Guang, Hongwei Yong, Tianpeng Jiang, Yaqian Li, Zheng Cao, and Kaiwen Long. Blockspec: Blockwise speculative decoding for diffusion LLMs, 2026. URL https://openreview.net/forum?id=hmAviop5rm.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Tingyu Zheng, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report. *arXiv e-prints*, art. arXiv:2412.15115, December 2024. doi: 10.48550/arXiv.2412.15115.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models, 2024. URL https://arxiv.org/abs/2406.07524.

Subham Sekhar Sahoo, Zhihan Yang, Yash Akhauri, Johnna Liu, Deepansha Singh, Zhoujun Cheng, Zhengzhong Liu, Eric Xing, John Thickstun, and Arash Vahdat. Esoteric Language Models: Bridging Autoregressive and Masked Diffusion LLMs. *arXiv e-prints*, art. arXiv:2506.01928, June 2025. doi: 10.48550/arXiv.2506.01928.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. *arXiv e-prints*, art. arXiv:2305.04091, May 2023. doi: 10.48550/arXiv.2305.04091. URL https://arxiv.org/abs/2305.04091.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv e-prints*, art. arXiv:2201.11903, January 2022. doi: 10.48550/arXiv.2201.11903.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free Acceleration of Diffusion LLM by Enabling KV Cache and Parallel Decoding. *arXiv e-prints*, art. arXiv:2505.22618, May 2025. doi: 10.48550/arXiv.2505.22618.

Algorithm 1 Cross-dLLM Inference

Require: Prompt x, AR planner M_{AR}, diffusion model M_{Diff}, denoising steps T
Ensure: Generated sequence y

Phase I: Skeleton Planning

$s \leftarrow M_{\mathrm{AR}}(x)$
$\{s^{(1)}, \ldots, s^{(K)}\} \leftarrow \mathrm{Parse}(s)$ ▷ Extract semantic anchors
$y_T \leftarrow \mathrm{Init}(x, s^{(1)}, \ldots, s^{(K)})$
$\mathcal{M} \leftarrow \{i \mid y_T[i] = \texttt{[MASK]}\}$
Partition $\mathcal{M}$ into $\{\mathcal{M}_b\}_{b=1}^{K}$
Precompute $\{k_{t,b}\}$ using $\mathrm{GetNumTransfer}(\mathcal{M}_b, T)$

Phase II: Cross-Block Diffusion Decoding

for $t = T, T-1, \ldots, 1$ **do**
  $\ell \leftarrow M_{\mathrm{Diff}}(y_t, x, s)$ ▷ Predict logits
  $\hat{y} \leftarrow \arg\max(\ell + g)$, where $g \sim \mathrm{Gumbel}(0, 1)$
  $c_i \leftarrow \max_{v \in \mathcal{V}} \mathrm{softmax}(\ell_i)_v$ for all $i \in \mathcal{M}$
  $\mathcal{C} \leftarrow \emptyset$
  for $b = 1, \ldots, K$ **in parallel do**
    $\mathcal{C}_b \leftarrow \mathrm{TopK}(\mathcal{M} \cap \mathcal{M}_b, k_{t,b}, c)$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_b$
  end for
  $y_{t-1} \leftarrow y_t$
  for all $i \in \mathcal{C}$ **do**
    $y_{t-1}[i] \leftarrow \hat{y}[i]$
  end for
  $\mathcal{M} \leftarrow \mathcal{M} \setminus \mathcal{C}$
end for
return y_0

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *arXiv e-prints*, art. arXiv:2305.10601, May 2023. doi: 10.48550/arXiv.2305.10601.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion Large Language Models. *arXiv e-prints*, art. arXiv:2508.15487, August 2025. doi: 10.48550/arXiv.2508.15487.

Zebin You, Shen Nie, Xiaolu Zhang, Jun Hu, Jun Zhou, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. Llada-v: Large language diffusion models with visual instruction tuning, 2025. URL https://arxiv.org/abs/2505.16933.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? *arXiv e-prints*, art. arXiv:1905.07830, May 2019. doi: 10.48550/arXiv.1905.07830.

Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, Hongrui Guo, Jiaqi Hu, Wentao Ye, Tieyuan Chen, Chenchen Li, Chengfu Tang, Haibo Feng, Jun Hu, Jun Zhou, Xiaolu Zhang, Zhenzhong Lan, Junbo Zhao, Da Zheng, Chongxuan Li, Jianguo Li, and Ji-Rong Wen. Llada-moe: A sparse moe diffusion language model, 2025. URL https://arxiv.org/abs/2509.24389.

A Appendix

A.1 Algorithm

For completeness, Algorithm 1 summarizes the inference procedure of Cross-dLLM, including autoregressive skeleton planning, skeleton-aware initialization, and cross-block diffusion decoding. The algorithm complements the methodology presented in Section 3.6.

A.2 Experimental Setup

Table 4 summarizes the hyperparameters used across all models. The full configuration details are provided in subsequent tables in this appendix.

B Experimental Details

Evaluation Benchmarks. We test on GSM8K Cobbe et al. (2021) and MATH500 Hendrycks et al. (2021) for mathematical reasoning, and HellaSwag Zellers et al. (2019), ARC-Easy, and ARC-Challenge Clark et al. (2018) for commonsense and science reasoning.

Baselines. Cross-dLLM is compared with representative autoregressive baselines (Qwen-3B/7B Qwen et al. (2024)), diffusion baselines (LLaDA-8B Nie et al. (2025), Dream7B Ye et al. (2025)), and hybrid baselines (LaDiR Kang et al. (2025)). All baselines are evaluated under the same protocol and hardware.

Decoding and Training. Inference includes intra-block and cross-block decoding configurations for parallel generation. Models are trained with AdamW, and the diffusion model uses a batch size of 4. Full hyperparameters and additional implementation details are provided in Appendix B.

B.1 System Efficiency

B.1.1 End-to-End Latency and Accuracy–Latency Trade-off

We report end-to-end latency on the same fixed 1,000-example GSM8K subset using the same evaluation script and hardware. For Cross-dLLM variants, the total time includes both autoregressive skeleton generation and diffusion decoding. Fast-dLLM is evaluated using its protocol-matched implementation on the same LLaDA backbone and evaluation subset.

Table 5 shows that Fast-dLLM provides substantially higher acceleration, reaching 68.3% accuracy in 21 minutes. In contrast, Cross-dLLM reaches 74.8% accuracy in 128 minutes, improving accuracy by 6.5 points over Fast-dLLM at higher latency. We therefore position Cross-dLLM as targeting a different accuracy–latency operating point, rather than as the fastest diffusion decoding method.

The comparison between the skeleton-conditioned executor and full Cross-dLLM isolates the decoding strategy under the same AR planner, diffusion executor, and predicted skeletons. Cross-block decoding reduces total inference time from 273 to 128 minutes, corresponding to a 53.1% latency reduction, while accuracy changes from 76.6% to 74.8%.

B.1.2 Batch Inference Scaling

Table 6 reports the throughput and memory consumption of the diffusion executor under different batch sizes. Increasing the batch size from 1 to 32 improves throughput from 14.6 to 21.5 tokens/s, corresponding to a 1.47$\times$ batching gain. The 3.31$\times$ ratio instead compares the batch-size-32 implementation against a separate non-batched implementation operating at 6.5 tokens/s. We report these two comparisons separately to avoid conflating batch scaling with implementation-level acceleration.

B.2 Generalization with Lightweight AR Planners

We report generalization results of different lightweight autoregressive (AR) planners in Table 7.

B.3 Few-shot Evaluation

To examine the effectiveness of CBD under limited data regimes, we conduct a few-shot evaluation on GSM8K Cobbe et al. (2021), where the model is trained with only 400 training examples. In addition to final answer accuracy, we analyze the token-level generation behavior by comparing log-likelihood distributions across different decoding strategies. Figure 4 shows both the total log-likelihood and the average log-probability per token under the few-shot setting. Compared to the standard diffusion decoding baseline, CBD exhibits a consistent rightward shift in both distributions, indicating a higher

Table 4: Hyperparameter overview for baseline models and Cross-dLLM (optimal settings).

Category	Hyperparameter	Value / Notes
Qwen2.5-3B / 7B (Autoregressive Baselines & Planner)		
Architecture	Attention	Standard Transformer causal self-attention, SwiGLU-like FFN
	Position Encoding	Causal (left-to-right)
Decoding	Temperature	0.7
	Top-p	0.8
	Repetition penalty	1.05
	Max generation tokens	512
Fine-tuning	Optimizer	AdamW ($\beta_1 = 0.9, \beta_2 = 0.98$), $\epsilon = 10^{-6}$
	Learning rate	2×10^{-5}
	Batch size	4
LLaDA-8B (Masked Diffusion Baseline)		
Training	Epochs	$\sim$3 on SFT data
	LR schedule	Linear warmup
	Batch size	4
	Block length	1024
Inference	Generation length	1024
	Remasking strategy	Low-confidence remasking
Dream-7B (Diffusion Baseline)		
Inference	Diffusion steps	1024 (default)
	Schedule	Cosine
	Guidance	Classifier-free (optional)
LaDiR (Hybrid Baseline)		
Inference	Planner	AR transformer (Qwen-like config)
	Denoiser	Diffusion-based, settings from original paper
Cross-dLLM (Ours)		
Architecture	Planner	Qwen2.5-3B (frozen or fine-tuned)
	Diffusion decoder	Masked diffusion, block-wise parallel generation
Training (CBD decoder)	Optimizer	AdamW ($\beta_1 = 0.95, \beta_2 = 0.98$)
	Learning rate	2×10^{-5} (tuned on validation)
	Batch size	4
	Block length	1024
	Diffusion steps	1024
	Noise schedule	Cosine
Inference (CBD decoding)	Intra-block decoding	Token-level parallel within each semantic block
	Cross-block decoding	Parallel decoding across semantic blocks
	Remasking	Low-confidence remasking
	Max generation length	1024
Shared / General		
Optimizer	AdamW	$\beta_1 = 0.95, \beta_2 = 0.98$, weight decay 0.1
LR schedule	Warmup	1000–2000 steps or 0.1–0.4 warmup ratio
Activation	FFN	SwiGLU or GELU variants

likelihood assigned to reference outputs at the token level. This trend suggests that CBD produces more stable and confident predictions even when trained and evaluated under limited data regimes.

B.4 Impact of Skeleton Quality

To further analyze the role of intermediate skeleton quality, we compare several alternative skeleton construction strategies with different levels of structural granularity and semantic coherence. The results in Table 8 show that naive heuristic decomposition substantially degrades performance, while our logic-aware semantic-level decomposition produces significantly better downstream reasoning accuracy.

Table 5: End-to-end accuracy and latency comparison on the fixed 1,000-example GSM8K subset. Speedup is computed relative to standard LLaDA decoding. The skeleton-conditioned executor and full Cross-dLLM use the same AR planner and diffusion executor, isolating the effect of cross-block decoding.

Method	**Acc. (%)**	**Planner Time (min)**	**Diffusion Time (min)**	**Total Time (min)**	**Speedup**
Standard LLaDA	67.6	–	293	293	1.00×
Fast-dLLM	68.3	–	21	21	13.95×
Skeleton-conditioned executor	**76.6**	8	265	273	1.07×
Full Cross-dLLM	74.8	8	120	128	2.29×

Table 6: Batch inference performance of the diffusion model (LLaDA-8B Nie et al. (2025)). Throughput and peak GPU memory measured on 4×H100 GPUs with 128 denoising steps. The maximum feasible batch size is 32 due to OOM beyond 60.6 GB. All latency measurements use identical maximum decoding steps and stopping criteria across methods.

Batch Size	Generated Tokens	Throughput (tokens/s)	Peak Memory (GB)
Non-batched	583	6.5	30.2
1	583	14.6	30.8
2	1166	18.7	31.8
4	2332	20.3	33.7
8	4664	20.5	37.6
16	9328	21.3	45.3
24	13992	21.5	52.9
32	18656	21.5	60.6

B.5 Quantitative Examples

We systematically analyze model behavior across multiple samples. Full case examples are provided in the supplementary material; here we summarize the key observations.

Baseline failures. The baseline frequently suffers from arithmetic errors and improper handling of intermediate states during multi-step reasoning. Our method, by strictly following the planner's stepwise structural constraints, effectively mitigates these failures.

Correct but different. Even when both methods produce correct answers, their reasoning styles differ substantially. The baseline tends to compress reasoning into short implicit operations, whereas our method unfolds the trajectory step by step according to the skeleton, yielding more structured and interpretable outputs.
Overall, introducing explicit stepwise constraints significantly improves reasoning correctness and reduces errors linked to intermediate state management. Remaining errors stem primarily from numerical inaccuracies during execution. Detailed examples and error-type statistics are available in the supplementary material.

B.6 Controlled Attribution of Performance Gains

A potential confounding factor in the main comparison is that Cross-dLLM is trained using GPT-4o-generated reasoning trajectories, whereas the original LLaDA model does not receive the same supervision. We therefore conduct a controlled attribution analysis to separate the effects of additional trajectory supervision, skeleton-conditioned generation, and cross-block decoding.

All configurations are evaluated on the same fixed 1,000-example subset of GSM8K using identical hardware, maximum generation length, stopping criteria, answer extraction, and evaluation scripts. The matched-data LLaDA-SFT baseline is fine-tuned using exactly the same GPT-4o-generated full reasoning trajectories as Cross-dLLM, but all skeleton and draft fields are removed during both training and inference.

As shown in Table 9, matched-data supervised fine-tuning improves raw LLaDA from 67.6% to 68.9%, corresponding to a gain of only 1.3 points. Therefore, the additional GPT-4o-generated reasoning trajectories alone do not explain the overall improvement of Cross-dLLM.

Table 7: Generalization performance on GSM8K Cobbe et al. (2021), MATH500 Hendrycks et al. (2021), HellaSwag Zellers et al. (2019), ARC-C Clark et al. (2018), and ARC-E Clark et al. (2018) benchmarks using different lightweight autoregressive (AR) planners:Qwen-2.5-0.5B Qwen et al. (2024), DeepSeek-Distill-Qwen-1.5B Guo et al. (2025), and Gemma-2B Gemma Team & Google DeepMind (2024) (all paired with LLaDA-8B Nie et al. (2025) as executor).

AR Planner	**Params**	**GSM8K (EM %)**	**MATH500 (%)**	**HellaSwag (%)**	**ARC-C (%)**	**ARC-E (%)**	**Speedup**
LLaDA-8B (Baseline)	8B	67.6	26.1	50.6	32.1	70.5	1.0×
Qwen-0.5B + LLaDA-8B	0.5B+8B	69.1	29.3	51.7	32.3	71.7	≈2.4×
DeepSeek-1.5B + LLaDA-8B	1.5B+8B	70.5	31.7	53.4	33.4	72.3	≈2.6×
Gemma-2B + LLaDA-8B	2B+8B	71.8	34.2	53.8	42.5	73.9	≈2.8×

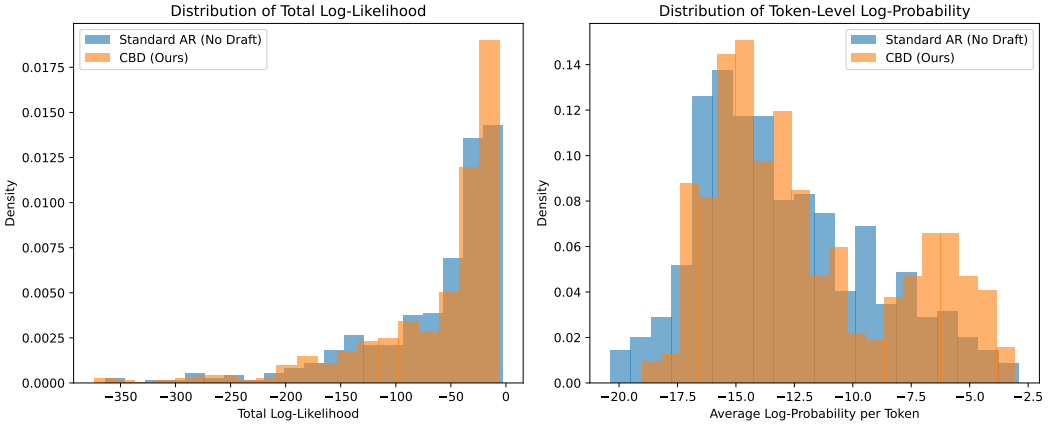


Figure 4: Distribution of log-likelihood metrics under few-shot evaluation. We compare **LLaDA (No Draft)** and **CBD (Ours)** using approximately 400 training examples. Left: distribution of total log-likelihood over evaluation samples. Right: distribution of average token-level log-probability. Higher values (less negative) indicate better model fit. All results are obtained under the same evaluation protocol.

The improvement from 68.9% to 76.6% jointly reflects skeleton-conditioned training, Drop-AR, and inference-time skeleton guidance. We do not attribute this difference solely to inserting a skeleton at inference time.

Finally, under the same AR planner and diffusion executor, cross-block decoding changes accuracy from 76.6% to 74.8% while reducing end-to-end inference time from 273 to 128 minutes. This corresponds to a 53.1% latency reduction and characterizes cross-block decoding as an accuracy–latency trade-off rather than an unconditional accuracy improvement.

B.7 Statistical Significance and Reproducibility

Evaluation uncertainty. All GSM8K comparisons in the main controlled experiments are conducted on the same fixed 1,000-example subset using the same answer-extraction and evaluation

Table 8: Impact of skeleton quality on downstream performance. Coarse denotes rule-based keyphrase extraction; Refined corresponds to our logic-aware composite instruction tuning pipeline.

Data Construction	Skeleton Definition	Acc.
Raw Text	None (Standard LLaDA)	67.6
Coarse Extraction	Rule-based / Keyphrase	57.3
Heuristic Chain	Sentence-level Split	48.2
Ours	**Semantic-level Split**	**74.8**

Table 9: Controlled attribution analysis on the fixed 1,000-example GSM8K subset. "Same-trajectory SFT" indicates whether the model is fine-tuned using the same GPT-4o-generated full reasoning trajectories as Cross-dLLM. All latency measurements include the complete inference pipeline where applicable.

Setting	Same-trajectory SFT	Skeleton	Decoding	Acc. (%)	Time (min)
Raw LLaDA	No	None	Standard	67.6	293
Matched-data LLaDA-SFT	Yes	None	No-draft parallel	68.9	127
Skeleton-conditioned executor	Yes	AR planner	Intra-block	76.6	273
Full Cross-dLLM	Yes	Same AR planner	Cross-block	74.8	128

Table 10: Strict-match accuracy across five random seeds on the fixed 1,000-example GSM8K subset.

Method	1234	42	78	1145	11	Mean $\pm$ Std.
LLaDA	66.9	67.2	68.3	69.2	67.8	67.88 ± 0.91
Cross-dLLM	74.9	73.7	74.2	73.8	75.1	74.34 ± 0.63

procedure. Standard LLaDA achieves 67.6% strict-match accuracy, while Cross-dLLM achieves 74.8%, corresponding to an improvement of 7.2 percentage points. The corresponding Wilson 95% confidence intervals are [64.6%, 70.4%] for LLaDA and [72.0%, 77.4%] for Cross-dLLM.

Multi-seed stability. The main tables report the default evaluation run, while this analysis reports additional runs under multiple random seeds to assess stochastic robustness. We further evaluate both Cross-dLLM and the LLaDA baseline under five random seeds: 1234, 42, 78, 1145, and 11. Cross-dLLM obtains strict-match accuracies of 74.9%, 73.7%, 74.2%, 73.8%, and 75.1%, respectively, yielding an average of 74.34% with a standard deviation of 0.63 percentage points. Under the same seeds, LLaDA obtains 66.9%, 67.2%, 68.3%, 69.2%, and 67.8%, respectively, with an average of 67.88% and a standard deviation of 0.91 percentage points. Cross-dLLM therefore outperforms the baseline under all five evaluated seeds, with an average improvement of 6.46 percentage points. The relatively small variation across seeds suggests that the observed gain is stable with respect to decoding randomness.

Controlled attribution. To separate the effects of additional trajectory supervision, skeleton conditioning, and cross-block decoding, we perform controlled comparisons on the same fixed GSM8K subset. Matched-data LLaDA-SFT, trained on the same reasoning trajectories without skeleton conditioning, improves accuracy only from 67.6% to 68.9%. With AR-generated skeleton conditioning and the fine-tuned executor, intra-block decoding reaches 76.6% accuracy with 273 minutes of inference time. Replacing intra-block execution with cross-block decoding yields 74.8% accuracy in 128 minutes. Thus, cross-block decoding reduces latency by 53.1% while changing accuracy by 1.8 percentage points, showing that its primary contribution is to improve the accuracy–latency trade-off.

Skeleton quality audit. We manually inspect 100 skeletons generated by the training-data construction pipeline. Among them, 91 contain no identifiable substantive logical or calculation error, while 99 satisfy the required structural format. The remaining imperfect skeletons further motivate robust skeleton conditioning and Drop-AR.

Evaluation implementation. Evaluation is implemented using `lm-evaluation-harness` with a custom `llada_dist` wrapper for LLaDA-based models. The autoregressive planner and diffusion executor are executed sequentially on 4 NVIDIA H100 GPUs, so simultaneous planner–executor memory partitioning is not required. The memory measurements reported in the efficiency analysis correspond to the diffusion stage.

Cases where our method succeeds while the baseline fails

Question:
A tomato plant has 100 tomatoes. Jane picks 1/4 of that number for use in their house. After a week, she goes back and picks 20 more tomatoes, and the following week picks twice that number. What's the total number of fruits remaining on the tomato plant?
Ground Truth: 15
Planner Output:
Summary:Total 4 steps: Step 1 - calculate 1/4 of 100; Step 2 - subtract 1/4 of 100 from the total; Step 3 - subtract 20 tomatoes; Step 4 - subtract twice 20 tomatoes; Steps: Step 1: Calculate 1/4 of 100 to find how many tomatoes Jane picks initially. Step 2: Subtract the number of tomatoes picked in Step 1 from the initial total of 100. Step 3: Subtract 20 tomatoes picked in the first week from the result in Step 2. Step 4: Subtract twice 20 tomatoes (40 tomatoes) picked in the second week from the result in Step 3.
CBD Output:
Step 1: Calculate 1/4 of 100 to find how many tomatoes Jane picks initially.To find 1/4 of 100, we divide 100 by 4:100 ÷ 4 = 25.So, Jane picks 25 tomatoes initially. Step 2: Subtract the number of tomatoes picked in Step 1 from the initial total of 100.After picking 25 tomatoes, we subtract 25 from the initial total of 100:100 - 25 = 75 Step 3: Subtract 20 tomatoes picked in the first week from the result in Step 2.Jane picks 20 more tomatoes in the first week, so we subtract 20 from the remaining 75 tomatoes:75 - 20 = 55. Step 4: Subtract twice 20 tomatoes (40 tomatoes) picked in the second week from the result in Step 3.

Twice 20 tomatoes is 40 tomatoes. We subtract 40 from the remaining 55 tomatoes:

55 - 40 = 15

15

Baseline Output:
When she picks 1/4 of the number of tomatoes, she goes home with 1/4*100 = «1/4*100=25»25 tomatoes.
The total number of tomatoes remaining on the plant will be 100-25 = «100-25=75»75
The next week she picks 20 more tomatoes, a total of 20+20 = «20+20=40»40 tomatoes.
The total number of tomatoes remaining on the plant will be 75-40 = «75-40=35»35
The next week she picks twice the number she picked the previous week, that is 20*2 = «20*2=40»40 tomatoes.
The total number of tomatoes remaining on the plant will be 35-40 = «35-40=5»5
5

Cases where both methods achieve correct results

Question:
"Madison takes her dog to the dog park. Counting Madison's dog, half the dogs have spots and 1/5 have pointy ears. If 15 dogs have spots, how many have pointy ears?"
Ground Truth: 6
Planner Output:
Summary: Total 3 steps: Step 1 - calculate total number of dogs; Step 2 - determine fraction of dogs with pointy ears; Step 3 - calculate number of dogs with pointy ears; Steps: Step 1: Use the information that half the dogs have spots and 15 dogs have spots to calculate the total number of dogs. Step 2: Recognize that 1/5 of the total number of dogs have pointy ears. Step 3: Multiply the total number of dogs by 1/5 to find the number of dogs with pointy ears.
CBD Output:
Step 1: Use the information that half the dogs have spots and 15 dogs have spots to calculate the total number of dogs.Since half the dogs have spots and 15 dogs have spots, we can find the total number of dogs by multiplying 15 by 2. 15 * 2 = 30. Step 2: Recognize that 1/5 of the total number of dogs have pointy ears.From the given information, we know that 1/5 of the dogs have pointy ears. Since 1/5 of the total number of dogs have pointy ears, we can multiply the total number of dogs by 1/5 to find the number of dogs with pointy ears. Step 3: Multiply the total number of dogs by 1/5 to find the number of dogs with pointy ears.To find the number of dogs with pointy ears, we multiply the total number of dogs (30) by 1/5. This can be calculated as:30 * 1/5 = 6. So, there are 6 dogs with pointy ears.#### 6

Baseline Output:
"There are 15 * 2 = «15*2=30»30 dogs at the dog park.\nThere are 30 / 5 = «30/5=6»6 dogs with pointy ears.\n\n#### 6"

Figure 5: **Qualitative comparison of structured diffusion decoding versus autoregressive baseline.** Left: The baseline produces an incorrect answer due to flawed arithmetic reasoning, while our method remains correct by following an explicit multi-step plan generated by the planner and executed via conditional block diffusion (CBD). Right: When both methods arrive at the correct answer, our approach demonstrates more structured and interpretable reasoning, decomposing the problem into explicit steps, whereas the baseline compresses reasoning into shorter implicit computations.