# WeDLM: Reconciling Diffusion Language Models with Standard Causal Attention for Fast Inference

**Aiwei Liu**[1,*,†], **Minghua He**[1,2,*,‡], **Shaoxun Zeng**[3], **Sijun Zhang** [1],
**Linhao Zhang**[1], **Chuhan Wu**[1], **Wei Jia**[1], **Yuan Liu**[1], **Xiao Zhou**[1], **Jie Zhou**[1]

[1]WeChat AI, Tencent [2]Peking University [3]Tsinghua University

`https://wedlm.github.io`
`https://github.com/tencent/WeDLM`
🤗 `https://huggingface.co/collections/tencent/wedlm`

## Abstract

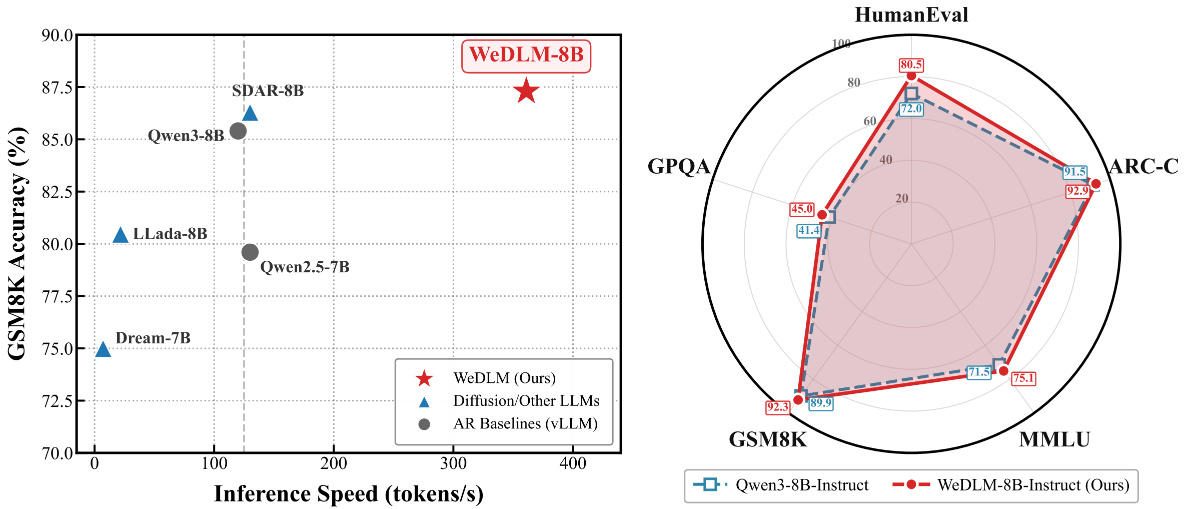
Autoregressive (AR) generation is the standard decoding paradigm for Large Language Models (LLMs), but its token-by-token nature limits parallelism at inference time. Diffusion Language Models (DLLMs) offer parallel decoding by recovering multiple masked tokens per step; however, in practice they often fail to translate this parallelism into deployment speed gains over optimized AR engines (e.g., vLLM). A key reason is that many DLLMs rely on bidirectional attention, which breaks standard prefix KV caching and forces repeated contextualization, undermining efficiency. We propose `WeDLM`, a diffusion decoding framework built entirely on standard causal attention to make parallel generation prefix-cache friendly. The core idea is to let each masked position condition on all currently observed tokens while keeping a strict causal mask, achieved by *Topological Reordering* that moves observed tokens to the physical prefix while preserving their logical positions. Building on this property, we introduce a streaming decoding procedure that continuously commits confident tokens into a growing left-to-right prefix and maintains a fixed parallel workload, avoiding the stop-and-wait behavior common in block diffusion methods. Experiments show that `WeDLM` preserves the quality of strong AR backbones while delivering substantial speedups, approaching 3× on challenging reasoning benchmarks and up to 10× in low-entropy generation regimes; critically, **our comparisons are against AR baselines served by vLLM under matched deployment settings**, demonstrating that diffusion-style decoding can outperform an optimized AR engine in practice.

## 1. Introduction

The autoregressive (AR) generation of Large Language Models (LLMs) is bottlenecked by its step-by-step nature. This sequential decoding underutilizes modern accelerators and often becomes

---

[*] Equal Contribution.
[†] Corresponding to Aiwei Liu: `coveliu@tencent.com`
[‡] Work done during internship at WeChat AI.



(a) Performance-Speed Trade-off

(b) Holistic Capability Comparison

Figure 1 | **Performance and capability overview of `WeDLM-8B`.** (a) **Speed vs. Accuracy:** `WeDLM-8B` achieves a ~3× speedup over the vLLM-optimized AR baseline (`Qwen3-8B`) on GSM8K, while also significantly outperforming prior diffusion models in both inference speed (tps) and accuracy. Dream and LLaDA use the dInfer inference engine [17]; SDAR uses JetEngine. (b) **Holistic Evaluation:** `WeDLM-8B-Instruct` matches or surpasses the strong capabilities of the `Qwen3-8B-Instruct` baseline, showing improvements across several mathematical, coding, and general knowledge benchmarks.

memory bound [8]. Diffusion Language Models (DLLMs) offer an appealing alternative: they recover multiple masked tokens in parallel [33]. Yet in practical deployments, existing DLLMs have not shown a clear speed advantage over highly optimized AR serving engines such as vLLM [14]. A key reason is that AR systems convert algorithmic efficiency into real throughput via native KV caching and mature runtime optimizations (e.g., PagedAttention [14] and CUDA Graphs). This implies that outperforming optimized AR baselines requires more than parallel prediction: a DLLM must be prefix-cache compatible, i.e., it should continuously grow a cache-valid left-to-right prefix so that most computation is reused rather than recomputed.

The main obstacle is prefix-cache incompatibility in prior diffusion designs. Many representative DLLMs (e.g., LLaDA [19] and Dream [30]) employ full bidirectional attention, which intrinsically couples each token's representation to both past and future positions. As a result, KV caching cannot be applied in the standard way: even early predictions are not immediately cache-valid and must be recomputed as later tokens change. Block-wise variants (e.g., SDAR [5] and NBDiff [25]) partially restore prefix reuse by committing completed blocks, but their gains are constrained by two effects. First, bidirectional attention within a block delays cacheability: a token cannot be committed until the entire block is finalized, because its KV state may depend on unresolved suffix positions. Second, diffusion-style resolution can be out of order, which reduces how much of the newly predicted content forms a contiguous left-to-right prefix and therefore limits effective cache reuse. These observations motivate a different design choice: we argue that bidirectional attention is not essential for parallel mask recovery, and that restoring strict causal structure is the most direct path to cache-friendly diffusion decoding.

In this work, we propose **`WeDLM`**, a framework that performs diffusion-style mask recovery entirely under standard causal attention to make parallel decoding compatible with prefix caching. Our key insight is that mask recovery only requires each masked position to access all currently observed tokens; this can be achieved without bidirectional attention via *Topological Reordering*. Specifically, we move observed tokens to the physical front while preserving their logical positions through RoPE position ids [24], so masked tokens can attend to the full observed context under an unmodified causal mask. This causal structure is naturally aligned with prefix caching: once earlier positions are resolved, their KV states depend only on committed context and can be reused immediately. We further introduce *Dual-Stream Masking* to reduce the training–inference gap induced by prefix-conditioned decoding. By constructing a clean *memory stream* alongside a masked *prediction stream* (with shared positional encoding), each prediction block is trained to condition on clean history rather than on potentially noisy intermediate predictions.

For inference, we develop *Streaming Parallel Decoding*, an algorithm explicitly organized around prefix commitment. It combines: (i) a position-aware confidence rule (implemented as a distance-penalized selection) that prioritizes earlier unresolved positions and encourages left-to-right growth; (ii) strict causal attention, which guarantees that newly committed prefix tokens become cache-valid immediately; and (iii) a dynamic sliding window that continuously refills new masked slots as soon as tokens are committed, avoiding the stop-and-wait behavior of block-wise methods. With attention remaining a standard causal mask, each iteration reduces to a small causal prefill over the active window on top of an existing KV cache, enabling direct use of optimized AR inference infrastructure such as FlashAttention [8], PagedAttention [14], and CUDA Graphs without kernel changes.

Experimental results demonstrate that WeDLM efficiently adapts to standard AR backbones. We instantiate WeDLM on both Qwen2.5-7B and Qwen3-8B, utilizing 100B tokens for continued training and 10B tokens for SFT. Across diverse benchmarks, including code generation (MBPP [2], HumanEval [4], HumanEval-plus [15]), math reasoning (GSM8K [7], MATH [12], GPQA [23]), and general knowledge (MMLU [13], ARC [6], HellaSwag [31]), WeDLM not only preserves but often improves upon the capabilities of its base models. Notably, WeDLM-8B achieves an average score of 77.36 on our benchmark suite, surpassing Qwen3-8B-Instruct (75.12) by over 2 points. **Unlike prior works that compare against unoptimized baselines, we benchmark WeDLM directly against the state-of-the-art vLLM engine.** Results show that WeDLM achieves up to 3× end-to-end acceleration on complex reasoning tasks and exceeds 10× speedups on some low-entropy generation scenarios, **demonstrating that diffusion-style decoding can outperform an optimized AR engine in matched, practical inference conditions**.

### Main Contributions

- **Causal Diffusion.** We propose WeDLM, a DLLM framework that performs mask recovery entirely under causal attention via *Topological Reordering*. This design enables seamless initialization from pre-trained AR checkpoints and inherent prefix-cache compatibility—predicted tokens can be cached immediately without waiting for subsequent positions.
- **Streaming Parallel Decoding.** We introduce a decoding strategy designed around prefix-cache compatibility: a distance penalty promotes left-to-right resolution, the causal mask enables immediate caching of predicted prefixes, and a dynamic sliding window continuously refills new masks as finalized tokens are committed—eliminating the stop-and-wait bottleneck of block-wise methods.
- **First DLLM to Outperform Industrial AR Engines.** We demonstrate that WeDLM surpasses optimized vLLM baselines in wall-clock speed, achieving more than 3× speedups on complex reasoning tasks while maintaining generation quality.

## 2. Preliminary

### 2.1. Autoregressive Language Modeling

Given a token sequence $\mathbf{x} = [x_1, x_2, \ldots, x_T]$, a standard autoregressive language model factorizes the joint probability into conditional probabilities:

$$P(\mathbf{x}) = \prod_{t=1}^{T} P(x_t \mid x_{<t}; \theta), \tag{1}$$

where $\theta$ denotes model parameters and $x_{<t} = [x_1, \ldots, x_{t-1}]$ is the preceding context. During decoding, the model generates $x_t$ conditioned on the previously generated prefix. Consequently, generation is sequential: later tokens depend on earlier ones through the conditioning structure in Eq. 1.

### 2.2. Decoupled Positional Representation

Eq. 1 implicitly ties each token's logical position to its physical index (i.e., $p_t = t$). We make this dependency explicit by representing inputs as token–position pairs $(x_t, p_t)$. The model then defines the conditional distribution as a function of both token identities and supplied position ids:

$$P(x_t \mid x_{<t}, p_{\leqslant t}; \theta) = \mathrm{LLM}(x_{\leq t}, p_{\leq t}; \theta), \tag{2}$$

where $\mathrm{LLM}(\cdot)$ denotes the model-induced conditional distribution at position $t$ (implemented by a softmax over the logits at that position). This decoupling is naturally supported by Rotary Positional Embeddings (RoPE) [24], where attention scores are indexed by the supplied logical positions $\mathbf{p}$ rather than by physical indices. Therefore, tokens may be processed in a different physical order while still being referenced by their logical positions; the resulting computation additionally depends on the attention mask, which determines the allowed information flow. We exploit this flexibility in our method.

### 2.3. Masked Diffusion Language Models

Masked Diffusion Language Models (MDLMs) formulate text generation as denoising rather than strictly sequential prediction. Given a clean sequence $\mathbf{x}_0$ of length $L$, a noising process samples a masking ratio $\gamma \in (0, 1]$ and corrupts a random subset of positions $\mathcal{M}$ (with $|\mathcal{M}| = \gamma L$) by replacing them with a special `[MASK]` token, producing a corrupted sequence $\mathbf{x}_\gamma$. The model is trained to reconstruct the original tokens at the masked positions. A commonly used training objective is a weighted cross-entropy loss:

$$\mathcal{L}(\theta) = -\mathbb{E}_{\gamma, \mathbf{x}_0, \mathbf{x}_\gamma}\left[\frac{1}{\gamma}\sum_{i=1}^{L} \mathbf{1}[x_\gamma^{(i)} = \mathbf{M}] \log p_\theta(x_0^{(i)} \mid \mathbf{x}_\gamma)\right], \tag{3}$$

where the factor $1/\gamma$ compensates for varying numbers of masked tokens under different noise levels. For brevity, we omit explicit conditioning on $\gamma$ (or the timestep) in $p_\theta$.

Standard MDLMs typically employ bidirectional attention so that masked positions can aggregate information from all observed tokens. However, this design introduces two limitations. First, bidirectional attention is incompatible with the KV cache mechanism that enables efficient autoregressive decoding. Second, when adapting pre-trained autoregressive models, the bidirectional structure induces an inductive-bias mismatch with causal representations. In this work, we optimize an MDLM-style denoising objective under strictly causal attention, enabled by reordering-based context exposure introduced in §4.

## 3. Motivation and Analysis

Two observations directly shape `WeDLM`'s design. (1) In KV-cached deployment, decoding speed is governed primarily by prefix cacheability rather than per-step parallelism. (2) Mask recovery does not require bidirectional attention; it can be implemented with standard causal attention.

### 3.1. Prefix Cacheability ($p_{\text{cache}}$) as an Inference Metric

Prior DLLMs mostly pursue speed by increasing tokens predicted per forward. In practice, a more inference-critical factor is how many predicted tokens can be converted into a growing, KV-cache-valid prefix. With KV caching, a token is reusable only if its key/value states depend only on earlier context; therefore, only a left-to-right prefix is cacheable. Predicted tokens that do not enter the committed prefix must be recomputed in later forwards, increasing total compute.

We quantify this effect with two indicators. Let $N_{\text{gen}}$ be the number of new tokens finally produced (excluding the initial prefill prompt), and let $N_{\text{fwd}}$ be the total number of token instances processed by the network across all decoding forwards after prefill (including repeated processing due to recomputation). We define the prefix cacheability (cache-hit probability) as

$$p_{\text{cache}} \triangleq \frac{N_{\text{gen}}}{N_{\text{fwd}}} \in (0, 1], \tag{4}$$

which can be interpreted as: during post-prefill decoding, a processed token instance becomes a final, cache-reusable token with probability $p_{\text{cache}}$. Equivalently, the average recomputation factor is $1/p_{\text{cache}}$.

This metric reflects an efficiency dimension that is distinct from per-step parallelism. Fully bidirectional methods (e.g., LLaDA, Dream) may predict many tokens per forward, yet often achieve low $p_{\text{cache}}$ because few predictions are immediately cache-valid. Block-wise methods (e.g., SDAR, NBDiff) improve speed largely by increasing $p_{\text{cache}}$ via partial prefix commitment. Hence, in KV-cached decoding, improving $p_{\text{cache}}$ can match or exceed speedups from increasing per-step parallelism, making it a primary objective for inference-oriented decoding.

**Implication.** In most MDLM-style decoders, $p_{\text{cache}}$ collapses mainly due to two structural issues: (i) bidirectional KV coupling—token representations depend on future (unresolved) tokens, so even early predictions are not cache-valid and must be recomputed; and (ii) out-of-order resolution—later positions are often resolved before earlier ones, disrupting the left-to-right prefix structure that KV caching requires. We address (i) by enforcing standard causal attention in `WeDLM` (see §4), and address (ii) with inference-time mechanisms that bias decoding toward left-to-right commitment (see §5).

### 3.2. Rethinking the Necessity of Bidirectional Attention

Masked diffusion language models (MDLMs) recover masked tokens conditioned on the available (unmasked) context. Standard MDLMs [19, 30] typically adopt bidirectional attention so that each position can aggregate information from all others. While natural, this is not a requirement of the mask-recovery objective itself.

Our key observation is that the information flow needed for mask recovery can be realized under standard causal attention by enforcing two algorithmic principles:

*Design Principles for Causal Mask Recovery*

Let $\mathcal{O}$ denote the set of observed (unmasked) positions and $\mathcal{M}$ the set of masked positions.

(i) **Observed-Context Visibility:** Each masked position should be able to attend to all observed tokens $x_{\mathcal{O}}$, enabling prediction with global observed context.

(ii) **Directed Dependence Among Masks:** Mask-to-mask interactions need not be symmetric. We impose a directed ordering over $\mathcal{M}$ so that each masked position attends only to a subset of masked positions that precede it in the chosen order. This replaces within-step mutual visibility with a causal dependence structure implementable by causal attention.

Principle (i) captures the essential requirement of MDLM-style denoising: masked predictions should be allowed to condition on all observed evidence. Principle (ii) is a modeling choice: we parameterize dependencies within the masked set using a causal factorization under an ordering $\pi$. Concretely, we model the conditional joint as $q_\theta(x_{\mathcal{M}} \mid x_{\mathcal{O}}; \pi) = \prod_{j=1}^{|\mathcal{M}|} q_\theta\big(x_{\pi(j)} \mid x_{\mathcal{O}}, x_{\pi(<j)}\big)$. This directed factorization allows earlier-resolved masked tokens to influence later ones without requiring bidirectional attention.

We do not claim equivalence to bidirectional-attention MDLMs. Rather, the two principles above are sufficient to implement set-conditioned masked prediction using standard causal attention. Whether the directed dependence in Principle (ii) is sufficient in practice is empirical, which we evaluate in §6.

**Design requirement for `WeDLM`.** Combining §3.1 and §3.2, our goal is: keep standard causal attention while ensuring that each masked position can access the full observed context. In the next section, we show how to satisfy this requirement via *Topological Reordering*, and then address the training–inference gap using *Dual-Stream Masking*.

## 4. WeDLM Training: Causal Mask Recovery

This section presents the training framework of `WeDLM`, which reconciles parallel language decoding with standard causal attention. Building on the analysis in §3, our training design targets two requirements: (1) prefix-cache-compatible computation under a strict causal mask, and (2) full observed-context visibility for mask recovery. We first introduce *Topological Reordering* (§4.1), which exposes the entire observed set to masked positions under an unmodified causal mask. We then present *Dual-Stream Masking* (§4.2) to mitigate the training–inference mismatch induced by prefix-conditioned decoding at inference time.

### 4.1. Causal Mask Recovery via Topological Reordering

To enforce the visibility constraint that masked positions attend to the full observed context under standard causal masking, we introduce *Topological Reordering*. Instead of using bidirectional attention masks, we apply a permutation that places all observed tokens before masked tokens in the physical computation order. We decouple physical order from logical text positions (indexed by position embeddings), so that every masked token can attend to all observed tokens using unmodified causal attention.

**Problem Setup.** Consider a clean sequence $\mathbf{x}_0 = [x_1, x_2, \ldots, x_L]$ with logical positions $\mathbf{p} = [1, 2, \ldots, L]$. We sample a masking ratio $\gamma \in (0, 1]$ and uniformly select a subset of indices

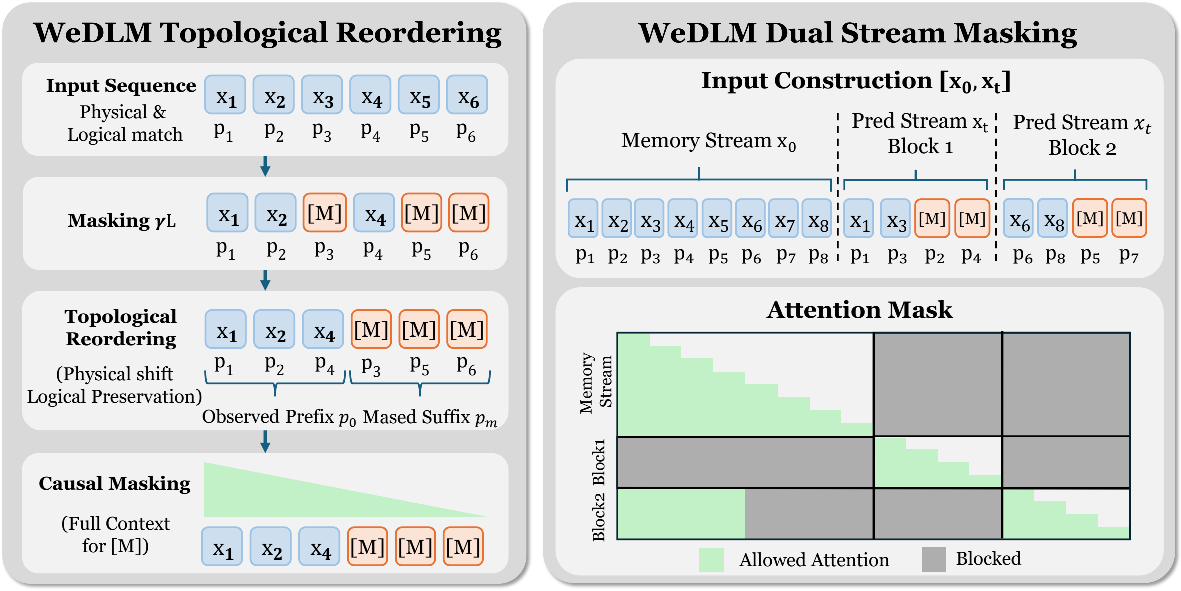


Figure 2 | **Overview of the `WeDLM` training framework. Left:** *Topological Reordering* physically shifts observed tokens to the prefix while preserving logical positions. This grants masked tokens access to the full observed context under standard causal masking. **Right:** *Dual-Stream Masking* concatenates a clean Memory Stream with a masked Prediction Stream. The block-wise attention mask ensures that the Prediction Stream conditions on clean memory history rather than noisy preceding predictions, aligning training dynamics with inference.

$\mathcal{M} \subset \{1,\dots,L\}$ with $|\mathcal{M}| = \gamma L$ to be masked. The remaining indices $\mathcal{O} = \{1,\dots,L\} \setminus \mathcal{M}$ are observed tokens, with $|\mathcal{O}| = N_o$ and $|\mathcal{M}| = N_m$.

**Topological Reordering.** Standard MDLMs use bidirectional attention so that masked tokens can access observed tokens regardless of position. We provide the same observed-context visibility under causal attention via a reordering operation. Specifically, we construct a reordered sequence $\tilde{\mathbf{x}}$ by placing all observed tokens before all masked tokens:

$$\tilde{\mathbf{x}} = [\underbrace{x_{o_1}, x_{o_2}, \dots, x_{o_{N_o}}}_{\text{observed tokens}},\ \underbrace{\texttt{[M]}, \texttt{[M]}, \dots, \texttt{[M]}}_{N_m \text{ mask tokens}}], \tag{5}$$

where $\{o_1, o_2, \dots, o_{N_o}\}$ are observed indices sorted in ascending order and `[M]` is a shared mask token. We preserve logical positions through a reordered position sequence:

$$\tilde{\mathbf{p}} = [\underbrace{o_1, o_2, \dots, o_{N_o}}_{\mathbf{p}_o},\ \underbrace{m_1, m_2, \dots, m_{N_m}}_{\mathbf{p}_m}], \tag{6}$$

where $\{m_1, m_2, \dots, m_{N_m}\}$ are masked indices, also sorted in ascending order.

**Context Awareness under Causal Masking.** Under causal attention, a token at physical index $i$ can only attend to indices $\{1,\dots,i-1\}$. In $\tilde{\mathbf{x}}$, observed tokens occupy $\{1,\dots,N_o\}$ and masked tokens occupy $\{N_o+1,\dots,L\}$; thus every masked token can attend to all observed tokens. Positional encodings (e.g., RoPE) are indexed by logical positions $\tilde{\mathbf{p}}$, so attention scores depend on logical relative offsets rather than physical indices.

**Training Objective.** With $(\tilde{\mathbf{x}}, \tilde{\mathbf{p}})$, we train the model to recover the ground-truth tokens at masked positions. For the $j$-th masked position (physical index $N_o + j$ with logical position $m_j$), the model predicts $x_0^{(m_j)}$ conditioned on the causal prefix. Following Eq. 2, we define:

$$\mathcal{L}(\theta) = -\mathbb{E}_{\gamma,\mathbf{x}_0,\mathcal{M}}\left[\frac{1}{\gamma}\sum_{j=1}^{N_m}\log p_\theta\left(x_0^{(m_j)} \mid \tilde{\mathbf{x}}_{<N_o+j},\ \tilde{\mathbf{p}}_{<N_o+j}\right)\right], \tag{7}$$

where the factor $1/\gamma$ follows the weighting convention in Eq. 3. The key difference from bidirectional MDLMs is that we operate under strictly causal attention: each masked token conditions only on earlier physical positions, yet still accesses the full observed context through topological reordering.

### 4.2. Dual-Stream Masking for Training

The objective in Eq. 7 masks tokens uniformly over the sequence. During inference, however, decoding proceeds in a prefix-conditioned regime: the unresolved tokens predominantly reside in a (block-wise) suffix due to left-to-right progression, inducing a train–inference distribution gap. Related motivations appear in work bridging autoregressive and diffusion objectives [5, 25]. A naive fix—masking only short suffixes—would exclude most tokens from the loss. We therefore propose *Dual-Stream Masking*, which simulates suffix-style decoding while preserving training efficiency.

**Dual-Stream Construction.** Given a clean sequence $\mathbf{x}_0 = [x_1, x_2, \ldots, x_L]$ with positions $\mathbf{p} = [1, 2, \ldots, L]$, we construct two copies: a *memory stream* $\mathbf{x}_o$ and a *prediction stream* $\mathbf{x}_t$, both initially identical to $\mathbf{x}_0$. These streams are concatenated to form the physical input:

$$\mathbf{x}_{\text{input}} = [\ \underbrace{\mathbf{x}_o}_{\text{memory stream}},\ \underbrace{\mathbf{x}_t}_{\text{prediction stream}}\ ]. \tag{8}$$

Critically, both streams share the same position sequence:

$$\mathbf{p}_{\text{input}} = [\underbrace{1, 2, \ldots, L}_{\mathbf{p}_o},\ \underbrace{1, 2, \ldots, L}_{\mathbf{p}_t}]. \tag{9}$$

This places the two streams in the same positional reference frame (e.g., under RoPE), enabling alignment between clean memory tokens and their masked counterparts in the prediction stream. The two streams are distinguished by their physical segment and the attention mask.

**Block-wise Masking and Reordering.** We partition the prediction stream $\mathbf{x}_t$ into $K = \lceil L/B \rceil$ non-overlapping blocks of size $B$ (except possibly the last). For each block $k \in \{1, \ldots, K\}$, we sample a masking ratio $\gamma_k \in (0, 1]$ and apply masking, followed by intra-block topological reordering within $\mathbf{x}_t$ as in §4.1: observed tokens in the block are moved to the front and masked tokens to the back, while logical positions are preserved. The memory stream $\mathbf{x}_o$ remains unmasked throughout and is not reordered.

**Attention Pattern.** The attention mask is designed to match inference-time conditioning: each block in $\mathbf{x}_t$ should rely on clean preceding context rather than noisy predictions. For a token in block $k$ of the prediction stream, its visible context is:

- **Memory stream:** All memory tokens whose logical positions precede block $k$, providing clean context for earlier blocks.

- **Current block:** Tokens within block $k$ of $\mathbf{x}_t$ that precede the current token in the reordered physical sequence (standard causal masking).

Notably, tokens in block $k$ of $\mathbf{x}_t$ cannot attend to previous blocks within $\mathbf{x}_t$; they instead access the corresponding clean history from $\mathbf{x}_o$. This simulates the inference setting where earlier blocks are finalized and used as context.

**Training Objective.** Let $\mathcal{M}_k$ denote the set of masked logical positions within block $k$, and let $\tilde{\mathbf{x}}_t^{(k)}$ denote the reordered block in the prediction stream. We aggregate losses across blocks:

$$\mathcal{L}(\theta) = -\mathbb{E}_{\{\gamma_k\},\mathbf{x}_0}\left[\sum_{k=1}^{K}\frac{1}{\gamma_k}\sum_{j\in\mathcal{M}_k}\log P_\theta\left(x_0^{(j)} \mid \mathbf{x}_o^{(<k)}, \tilde{\mathbf{x}}_t^{(k,<j)}\right)\right], \tag{10}$$

where $\mathbf{x}_o^{(<k)}$ denotes memory tokens whose logical positions precede block $k$, and $\tilde{\mathbf{x}}_t^{(k,<j)}$ denotes tokens in the reordered block $k$ that physically precede the masked position $j$. The factor $1/\gamma_k$ follows the per-block weighting convention.

**Inference Compatibility.** During inference, we discard the memory stream and decode with a standard causal attention mask over a single sequence. This requires no model-architecture changes and is compatible with optimized attention implementations such as FlashAttention [8].

## 5. WeDLM Inference: Streaming Parallel Decoding

In §3, we introduced prefix cacheability $p_{\text{cache}}$ (Eq. 4) as a metric that captures how much of the post-prefill compute becomes reusable prefix KV states. In industrial KV-cached serving, latency is influenced not only by how many tokens are proposed per forward, but also by how effectively these parallel proposals can be committed into a contiguous left-to-right prefix. Here we instantiate this objective into a concrete procedure, Streaming Parallel Decoding, which incrementally commits cache-ready tokens under standard causal attention while continuously refilling new masked positions to maintain steady GPU utilization.

### 5.1. Inference Requirements for Streaming Decoding

Streaming Parallel Decoding is designed to maximize the rate of prefix commitment. After the initial prefill on a prompt prefix $\mathbf{x}$, the decoder repeatedly (i) predicts multiple masked positions in parallel, and (ii) converts sufficiently confident predictions into a committed prefix whose KV states can be reused by subsequent steps. To achieve a high $p_{\text{cache}}$ in practice, the inference procedure should satisfy the following requirements.

*Inference Requirements for High Prefix Commitment*

(i) **Immediate KV Validity (No Bidirectional Coupling).** Once a token is predicted, its KV representation should depend only on already-committed context; otherwise it cannot be cached immediately and will require recomputation later, lowering $p_{\text{cache}}$.

(ii) **Left-to-Right Commitment (Avoid Out-of-Order Resolution).** Since standard KV caching only supports a contiguous prefix, decoding should prioritize resolving earlier logical positions so that each step extends the committed prefix as much as possible.

(iii) **No Stop-and-Wait (Avoid Pipeline Bubbles).** The algorithm should avoid stalling at block boundaries; it should maintain a steady workload per forward by introducing new masked positions as soon as tokens are committed.

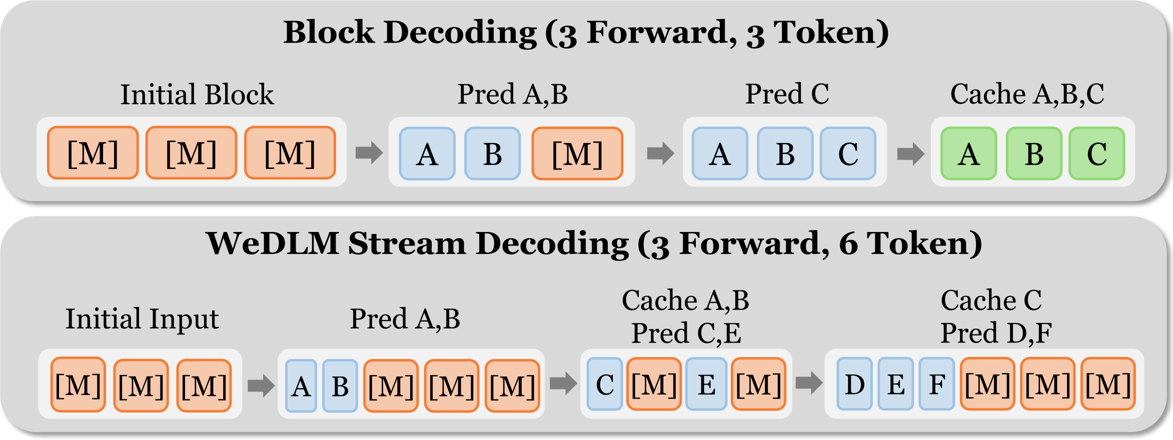


Figure 3 | **Block Decoding vs. WeDLM Streaming Parallel Decoding.** Block decoding suffers from stop-and-wait: bidirectional dependence within a block prevents committing any token until the entire block is finalized. In contrast, `WeDLM` uses standard causal attention with a dynamic sliding window: resolved tokens (e.g., A, B) are immediately cache-ready and committed, while new mask tokens (e.g., C, E) are appended for parallel prediction.

**How `WeDLM` meets these requirements.** Requirement (i) is enabled by `WeDLM`'s standard causal attention: under a strict causal mask, each token's KV state depends only on tokens at earlier physical indices. Combined with inference-time topological reordering that places already-resolved tokens before unresolved masks (while preserving logical positions via position ids), a resolved token becomes cache-valid immediately after it is produced. Requirements (ii)–(iii) are addressed by Streaming Parallel Decoding: we use a position-aware confidence rule that biases resolution toward the left, and a dynamic sliding window with on-the-fly refilling to remove block-boundary synchronization.

### 5.2. Streaming Parallel Decoding

We now describe Streaming Parallel Decoding, an inference strategy that operates entirely under standard causal attention and KV caching. The key idea is to maintain a fixed-size window of $W$ slots, containing a mixture of filled tokens (already predicted but not yet committed) and `[M]` tokens (pending prediction). At each step, we: **(a) reorder** the window so that filled tokens appear before masks (positions preserved via per-token global position ids), **(b) run a causal forward** conditioned on the persistent cache, **(c) commit** the leftmost contiguous filled prefix (now cache-valid), **(d) predict** additional mask slots based on confidence and position, and **(e) refill** the window with new masks to keep parallelism constant.

**Distance Penalty for Left-to-Right Commitment.** To increase the chance that resolved tokens form a long contiguous prefix, we bias mask selection toward earlier positions. Following Ye et al. [30], we use prediction entropy to determine which masks to fill. Let $p_i(\cdot)$ be the predicted distribution at mask slot $i$ with entropy $H_i = -\sum_v p_i(v) \log p_i(v)$. We define a distance-adjusted entropy:

$$\tilde{H}_i = H_i + \lambda \cdot d_i, \tag{11}$$

where $d_i$ is the distance from slot $i$ to the leftmost remaining mask slot in the current window, and $\lambda > 0$ controls the strength of left-to-right preference. SELECTBYENTROPY returns mask indices whose adjusted entropy falls below a threshold $\tau$. This reduces out-of-order resolution patterns and accelerates contiguous prefix growth, improving $p_{\text{cache}}$.

**Algorithm 1** Streaming Parallel Decoding

```
Require: Prompt prefix $\mathbf{x}$, window size $W$, entropy threshold $\tau$, distance penalty $\lambda$
Ensure: Generated sequence $\mathbf{y}$
$\mathbf{y} \leftarrow [\,]$; $(\mathbf{K}, \mathbf{V}) \leftarrow \textsc{Prefill}(\mathbf{x})$
$\mathcal{W} \leftarrow [\texttt{[M]}]^{W}$                    ▷ Each slot carries a fixed global position id
while $\mathcal{W} \neq \emptyset$ do
        ▷ Reorder & Forward: filled tokens placed before masks (logical positions preserved)
    $\mathcal{W} \leftarrow [\mathcal{W}_{\text{filled}}; \mathcal{W}_{\text{mask}}]$
    $(\boldsymbol{\ell}, \mathbf{K}_{\mathcal{W}}, \mathbf{V}_{\mathcal{W}}) \leftarrow \textsc{Forward}(\mathcal{W}, \mathbf{K}, \mathbf{V})$
        ▷ Commit: commit the leftmost contiguous filled prefix (cache-valid under causal mask)
    $n \leftarrow \min\{i : \mathcal{W}[i] = \texttt{[M]}\}$ or $|\mathcal{W}|$ if none
    Append $\mathcal{W}[0{:}n]$ to $\mathbf{y}$
    Extend $(\mathbf{K}, \mathbf{V})$ with $(\mathbf{K}_{\mathcal{W}}[0{:}n], \mathbf{V}_{\mathcal{W}}[0{:}n])$
    $\mathcal{W} \leftarrow \mathcal{W}[n{:}]$
        ▷ Predict: fill a subset of masks based on confidence and position bias
    $\mathcal{F} \leftarrow \textsc{SelectByEntropy}(\boldsymbol{\ell}_{\text{mask}}, \tau, \lambda)$
    $\mathcal{W}[i] \leftarrow \textsc{Sample}(\boldsymbol{\ell}_i)$ for $i \in \mathcal{F}$
        ▷ Refill: append new masks to maintain constant parallelism
    $\mathcal{W} \leftarrow [\mathcal{W}; [\texttt{[M]}]^{n}]$
end while
return $\mathbf{y}$
```

**Immediate Caching under Standard Causal Attention.** Under `WeDLM`'s strict causal mask, the KV representation of a token depends only on the physical prefix. After reordering, when filled tokens occupy the leftmost slots of $\mathcal{W}$, they attend only to the persistent cached prefix and earlier filled slots. Therefore, the leftmost contiguous filled prefix is immediately cache-valid and can be committed to $(\mathbf{K}, \mathbf{V})$ without waiting for other mask slots to be resolved. This directly addresses the bidirectional KV-coupling issue highlighted in §3.

**Dynamic Sliding Window to Eliminate Stop-and-Wait.** Block-wise decoders must wait until an entire block becomes final before committing, creating pipeline bubbles. In contrast, streaming maintains a fixed window size $W$. At each step, committed tokens are removed and an equal number of new `[M]` slots are appended. This on-the-fly refill keeps the amount of work per forward approximately constant, maintaining computational saturation and avoiding block-boundary synchronization.

**Compatibility with Efficient Inference Systems.** Streaming Parallel Decoding derives its efficiency from operating entirely under standard causal attention. Each decoding step reduces to a causal forward over the current window conditioned on the cached prefix—effectively a small prefill—which is natively supported by FlashAttention [8], PagedAttention [14], and CUDA Graphs without kernel modification. This design enables direct deployment on industrial AR inference infrastructure.

## 6. Experiments

This section evaluates `WeDLM` across multiple dimensions. We first describe the experimental setup (§6.1), then present results on generation quality (§6.3) and inference efficiency (§6.4), followed by ablation studies (§6.5).

Table 1 | Main results on generation quality across diverse benchmarks for **Base models**. We compare our `WeDLM` against autoregressive (AR) baselines and recent diffusion language models (DLLMs). The columns for our model are highlighted in blue. Best results in each row are **bolded**.

| **Benchmark** | **AR Baseline** | | **DLLM Baseline** | | **WeDLM (Ours)** | |
|---|---|---|---|---|---|---|
| | *Base Model* | | *Base Model* | | *Base Model* | |
| | Qwen2.5-7B | Qwen3-8B | LLaDA-8B | Dream-7B | `WeDLM-7B` | `WeDLM-8B` |
| *General Reasoning* | | | | | | |
| ARC-C (0-shot) | 89.93 | 92.66 | 81.14 | 88.40 | 90.70 | **92.92** |
| ARC-E (0-shot) | 96.55 | 97.13 | 92.00 | 96.21 | 96.13 | **97.14** |
| HellaSwag (10-shot) | 80.20 | 85.27 | **85.34** | 78.05 | 85.11 | 84.55 |
| MMLU (5-shot) | 71.62 | 74.03 | 64.61 | 70.64 | 71.93 | **75.46** |
| *Math & Science* | | | | | | |
| GSM8K (3-shot) | 79.23 | 85.97 | 71.80 | 75.97 | 84.76 | **90.20** |
| MATH (4-shot) | 43.40 | 50.80 | 28.00 | 38.00 | 48.20 | **53.60** |
| GPQA-Diamond (5-shot) | 33.70 | 37.00 | 29.80 | 25.76 | 36.87 | **42.42** |
| *Code Generation* | | | | | | |
| MBPP (3-shot) | 65.30 | **70.94** | 41.99 | 56.47 | 61.81 | 67.00 |
| HumanEval (4-shot) | 59.14 | 68.90 | 31.71 | 20.12 | 68.90 | **75.00** |
| HumanEval-plus (4-shot) | 53.05 | 63.40 | 28.05 | 19.51 | 64.00 | **68.90** |
| **Average** | 67.21 | 72.61 | 55.44 | 56.91 | 70.84 | **74.72** |

### 6.1. Training Details

We initialize `WeDLM` from pre-trained autoregressive models in the Qwen family. Specifically, we use Qwen2.5-7B [22] and Qwen3-8B [28] as our base models. These models provide strong foundations with well-established performance across diverse tasks. We perform continued pretraining on 100B tokens to adapt the base models to the `WeDLM` framework. The learning rate starts at $3 \times 10^{-6}$ and gradually decays to $3 \times 10^{-7}$ following a cosine schedule. For the Dual-Stream Masking strategy described in §4.2, we set the block size $B = 32$. To efficiently handle the irregular attention patterns introduced by topological reordering, we employ Magi Attention [32], which accelerates computation over non-rectangular attention masks without requiring custom CUDA kernels. To preserve the original autoregressive capabilities, we incorporate an auxiliary AR loss during training. This loss is computed on the same sequences using standard next-token prediction, ensuring that the model retains its causal language modeling ability while learning the masked diffusion objective. After pretraining, we perform supervised fine-tuning (SFT) to improve instruction-following capabilities. We use 10K internal instruction-response pairs for this stage. The learning rate is set to $3 \times 10^{-6}$ with a cosine decay schedule. The resulting models are denoted as `WeDLM-7B` (based on Qwen2.5-7B) and `WeDLM-8B` (based on Qwen3-8B).

### 6.2. Evaluation Setup

We evaluate `WeDLM` on a diverse set of benchmarks spanning reasoning, knowledge, and code generation. For knowledge and commonsense reasoning, we use ARC-Challenge [6] (0-shot), GPQA [23] (5-shot), HellaSwag [31] (10-shot), and MMLU [13] (5-shot). For mathematical

Table 2 | Main results on generation quality across diverse benchmarks for **Instruct models**. We compare our `WeDLM` against autoregressive (AR) baselines and recent diffusion language models (DLLMs). The columns for our model are highlighted in blue. Best results in each row are **bolded**.

| **Benchmark** | **AR Baseline**<br>*Instruct Model* | | **DLLM Baseline**<br>*Instruct Model* | | | **WeDLM (Ours)**<br>*Instruct Model* | |
|---|---|---|---|---|---|---|---|
| | Qwen2.5-7B | Qwen3-8B | LLaDA-8B | Dream-7B | SDAR-8B | `WeDLM-7B` | `WeDLM-8B` |
| *General Reasoning* | | | | | | | |
| ARC-C (0-shot) | 86.09 | 91.47 | 85.92 | 87.20 | 91.13 | 89.59 | **92.92** |
| ARC-E (0-shot) | 93.27 | 96.17 | 94.32 | 93.27 | 97.01 | 96.09 | **97.43** |
| HellaSwag (10-shot) | 87.59 | 86.13 | 78.55 | 62.00 | **92.12** | 84.75 | 82.94 |
| MMLU (5-shot) | 71.98 | 71.52 | 63.70 | 64.19 | 73.61 | 70.52 | **75.14** |
| *Math & Science* | | | | | | | |
| GSM8K (3-shot) | 89.91 | 89.91 | 80.59 | 79.00 | 91.66 | 87.57 | **92.27** |
| MATH (4-shot) | 45.00 | **69.60** | 34.20 | 41.00 | 43.40 | 55.40 | 64.80 |
| GPQA-Diamond (5-shot) | 27.10 | 41.41 | 25.25 | 35.86 | 38.38 | 33.84 | **44.95** |
| *Code Generation* | | | | | | | |
| MBPP (3-shot) | 63.66 | 68.37 | 36.24 | 58.52 | 67.97 | 63.66 | **70.53** |
| HumanEval (4-shot) | 76.22 | 71.95 | 36.59 | 57.32 | 76.83 | 75.00 | **80.49** |
| HumanEval-plus (4-shot) | 70.12 | 64.63 | 32.32 | 51.22 | 70.12 | 71.34 | **73.78** |
| **Average** | 71.09 | 75.12 | 56.77 | 62.96 | 74.22 | 72.78 | **77.53** |

reasoning, we evaluate on GSM8K [7] (3-shot) and MATH [12] (4-shot). For code generation, we use MBPP [2] (3-shot) and HumanEval [4] (4-shot). For generative tasks (GSM8K, GPQA, MBPP, HumanEval, and MATH), we set the maximum generation length to 512 tokens and use a sampling temperature of 0.1. For inference, to ensure fair comparison, the results in Tables 1 and 2 are obtained with a unified step-wise decoding scheme: at each step, all methods (including our model and diffusion baselines) generate one token by selecting the position with the lowest entropy; unless otherwise specified, we use a window size $W = 6$ and a distance-based penalty coefficient $\lambda = 0.10$ (see Eq. 11) to balance generation quality and speed. We compare `WeDLM` against both autoregressive baselines and recent diffusion language models. The autoregressive baselines include Qwen2.5-7B and Qwen3-8B, which serve as the base models for our method. For diffusion models, we compare against LLaDA-8B [19], Dream-7B [30], and SDAR-8B [5]. To ensure fair comparison, each model uses its recommended inference framework: LLaDA and Dream use dInfer, SDAR uses JetEngine, and the Qwen models use vLLM [14]. Our `WeDLM` models are also served via vLLM, demonstrating seamless compatibility with industrial inference systems.

### 6.3. Performance Evaluation

Tables 1 and 2 report generation quality for `WeDLM` under base and instruct settings, respectively. Across both settings, the main trend is consistent: `WeDLM` not only preserves but often improves upon the capabilities of its underlying autoregressive (AR) checkpoints, while maintaining a large margin over prior diffusion language models.

On **base models** (Table 1), `WeDLM-7B` achieves an average score of 70.84, improving over Qwen2.5-7B (67.21) by 3.6 points, and `WeDLM-8B` reaches 74.72, exceeding Qwen3-8B (72.61) by 2.1 points. The gains concentrate on reasoning-heavy tasks: on GSM8K, `WeDLM-7B` improves

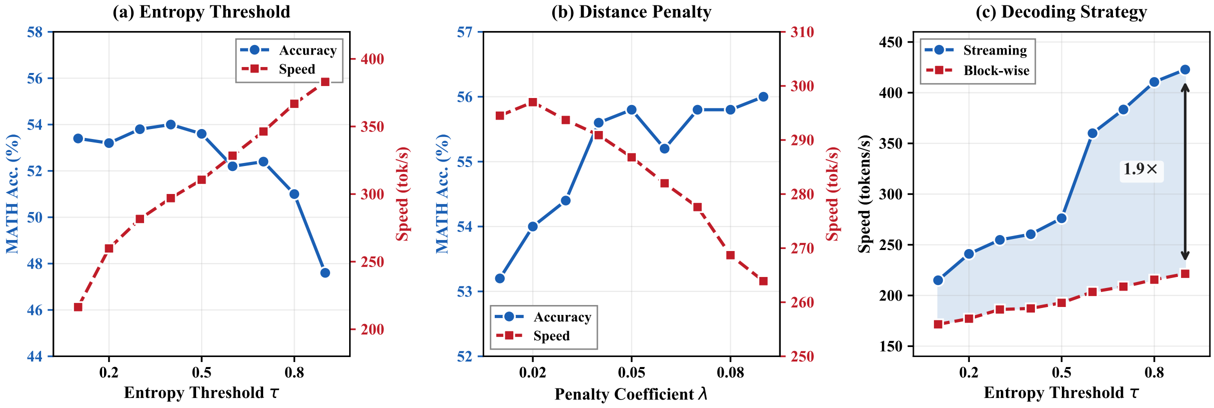


Figure 4 | Ablation studies on inference hyperparameters. (a) Effect of entropy threshold $\tau$ on MATH accuracy and generation speed, revealing a quality-speed trade-off with optimal range $\tau \in [0.3, 0.5]$. (b) Effect of distance penalty coefficient $\lambda$, showing that prioritizing left-positioned tokens improves accuracy with minimal speed cost. (c) Comparison of Streaming Parallel Decoding versus block-wise decoding across entropy thresholds; streaming achieves up to 1.9× speedup by enabling immediate prefix commitment.

by 5.5 points (84.76 vs. 79.23) and `WeDLM-8B` by 4.2 points (90.20 vs. 85.97). We observe similar improvements on MATH (+4.8 and +2.8 points) and GPQA-Diamond (+3.2 and +5.4 points). For code, `WeDLM` shows notable gains on HumanEval (+9.8 for 7B and +6.1 for 8B), while MBPP is the only benchmark with a consistent drop (about 3–4 points), suggesting sensitivity to domain or prompt-format differences.

On **instruct models** (Table 2), `WeDLM` remains competitive with, and in many cases surpasses, its AR instruct baselines. `WeDLM-7B` improves over Qwen2.5-7B on ARC-C (+4.0), ARC-E (+2.9), MATH (+9.2), and GPQA-Diamond (+8.8), but underperforms on GSM8K (-3.3) and HumanEval (-3.1). `WeDLM-8B` shows the strongest overall results, reaching an average of 77.53, which is +2.4 over Qwen3-8B (75.12). It delivers consistent gains on reasoning and code, including MMLU (+3.6), GPQA-Diamond (+3.5), HumanEval (+8.5), and HumanEval-plus (+9.2), while remaining close on GSM8K (+2.4) and MBPP (+2.2). These results indicate that the diffusion-style training objective and parallel decoding do not conflict with instruction tuning, and can even amplify it when starting from a strong instruct checkpoint.

Compared to **diffusion baselines**, `WeDLM` maintains a clear advantage in both settings. On base models, LLaDA-8B and Dream-7B average 55.44 and 56.91, which is 15–19 points below `WeDLM`. On instruct models, diffusion baselines improve but still lag behind: the best diffusion instruct baseline (SDAR-8B) averages 74.22, while `WeDLM-8B` reaches 77.53. The gap is most visible on code generation and higher-difficulty reasoning (e.g., HumanEval and GPQA-Diamond), where `WeDLM-8B` sets the best overall scores among the compared models.

### 6.4. Speed Evaluation

**Hyperparameter Sensitivity.** Figure 4 examines the key inference hyperparameters using `WeDLM-7B-Instruct`. The entropy threshold $\tau$ (Figure 4a) controls unmasking confidence: lower values yield higher accuracy but slower generation. Performance remains stable (~53–54%) for $\tau \leqslant 0.5$, then degrades sharply at higher thresholds as low-confidence predictions propagate errors. We recommend $\tau \in [0.3, 0.6]$ for balanced operation. The distance penalty $\lambda$ (Figure 4b) biases selection toward left-positioned tokens. From the perspective of prefix

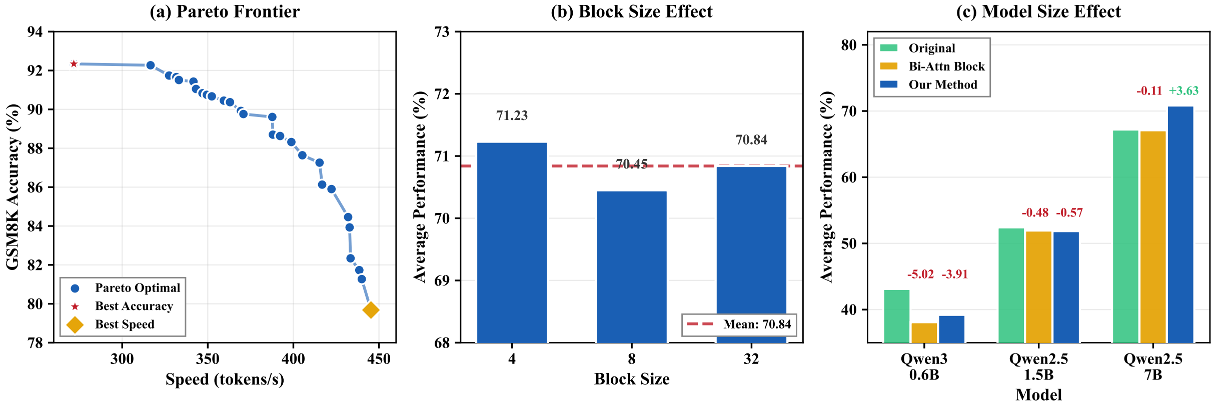


Figure 5 | **Ablation studies.** (a) Pareto frontier on GSM8K showing quality-speed trade-offs across hyperparameter configurations; conservative settings achieve 92.3% accuracy at 1.97× speedup while aggressive settings reach 3.2× acceleration. (b) Block size effect during continued pretraining shows stable performance across $B \in \{4, 8, 32\}$. (c) Attention design and model scale: we compare bidirectional attention within blocks (Bi-Attn Block) against our causal design (Our Method) across model sizes; larger models benefit more from causal adaptation, while bidirectional intra-block attention consistently underperforms.

cacheability (§3.1), prioritizing earlier positions directly increases $p_{\text{cache}}$: tokens resolved earlier in the sequence are more likely to form a contiguous committed prefix, whose KV states become immediately reusable. Increasing $\lambda$ from 0.01 to 0.05 improves accuracy by 2.6 points with only 3% speed reduction, confirming that left-to-right resolution not only accelerates caching but also reduces error accumulation from out-of-order predictions.

**Streaming vs. Block-wise Decoding: A $p_{\text{cache}}$ Perspective.** Figure 4(c) demonstrates that Streaming Parallel Decoding consistently outperforms block-wise decoding across all entropy thresholds. At $\tau = 0.9$, streaming achieves 1.9× speedup (423 vs. 221 tokens/s). This gap can be understood through the lens of prefix cacheability (Eq. 4): block-wise methods must wait until an entire block is finalized before any token becomes cache-valid, yielding lower $p_{\text{cache}}$ due to synchronization overhead. In contrast, streaming commits tokens as soon as they form a contiguous left-to-right prefix, maximizing $p_{\text{cache}}$ by converting each resolved token into a cache-reusable state without delay.

**Quality-Speed Pareto Frontier.** Figure 5(a) presents the Pareto optimal configurations on GSM8K using `WeDLM-8B-Instruct`, spanning accuracy from 79.7% to 92.3% at speeds of 272–445 tokens/s. The frontier reveals a smooth trade-off: conservative settings ($\tau = 0.2$, $\lambda = 0.01$) preserve near-baseline accuracy (92.3%) at 1.97× speedup, while aggressive settings ($\tau = 0.9$, $\lambda = 0.01$) achieve 3.2× acceleration with accuracy above 79%. This flexibility allows practitioners to select operating points based on task-specific requirements.

### 6.5. Ablation Studies

**Block Size.** Figure 5(b) examines the effect of block size $B$ during continued pretraining. Performance remains virtually identical across $B \in \{4, 8, 32\}$, with average scores within a 0.8-point range (70.45–71.23), demonstrating that `WeDLM` is insensitive to block size. This flexibility favors larger block sizes in practice: libraries such as Magi Attention incur higher overhead for smaller blocks, and models trained with larger $B$ naturally support any smaller window size at inference time without retraining, providing greater deployment flexibility.

**Causal vs. Bidirectional Intra-Block Attention.** Figure 5(c) compares our fully causal design against a variant that uses bidirectional attention within each prediction block (while remaining causal across blocks). This relaxes Principle (ii) from §3.2 by allowing mutual visibility among masked tokens inside a block. Overall, causal intra-block attention achieves higher average performance than the bidirectional variant, indicating that the directed factorization is already sufficient for AR-initialized models. Moreover, bidirectional intra-block attention fundamentally limits $p_{\text{cache}}$—tokens cannot be committed until the entire block resolves—whereas our causal design enables immediate per-token caching.

**Base Model Initialization.** Figure 5(c) also investigates how model scale affects adaptation to the `WeDLM` framework. Smaller models (0.6B, 1.5B) experience slight performance degradation (−3.9 and −0.6 points for our method), while larger models (7B) show consistent improvements (+3.6 points). Notably, the improvement magnitude correlates monotonically with the base model's original capability: stronger AR checkpoints adapt more readily to the diffusion objective. This trend hints at a potential scaling law for AR-to-diffusion adaptation, where the benefit of diffusion training increases predictably with model capacity. We leave systematic verification of this hypothesis to future work, but these results already suggest that 7B+ scale models are the recommended starting point for `WeDLM` deployment.

### 6.6. Case Study

To better understand the performance characteristics of `WeDLM`, we analyze its generation behavior across different task modalities. The decoding speed is strongly correlated with the entropy of the output distribution, as shown in the representative cases in Appendix A:

- **Low Entropy (Sequential Patterns):** As shown in Figure 6, the model achieves a peak throughput of **1673.3 tokens/s** on a simple counting task. The deterministic nature of the sequence yields extremely low entropy, allowing the model to speculate and accept many tokens per step.
- **Medium Entropy (Structured Reasoning):** Figure 7 demonstrates a mathematical derivation task. Despite requiring logic, the syntactic structure of the solution is relatively predictable, maintaining a high speed of **745.2 tokens/s**.
- **High Entropy (Open-ended Generation):** In Figure 8, where the model explains Quantum Physics, the speed drops to **197.8 tokens/s**. The high semantic diversity and lexical uncertainty in open-ended text reduce the confidence of speculative tokens, limiting the effective parallel block size.

These results highlight a significant performance disparity: while low-entropy tasks achieve over 8× speedup, high-entropy generation sees diminishing returns. Although this variance partially reflects the intrinsic uncertainty of natural language, it exposes a limitation of the current framework in handling high-perplexity scenarios. Closing this gap—potentially through more robust acceptance mechanisms or dynamic entropy calibration—remains a critical direction for future work to ensure consistent acceleration across all domains.

## 7. Related Work

**Discrete Diffusion Language Models.** Discrete diffusion models learn to iteratively denoise corrupted sequences, enabling parallel generation and bidirectional context modeling. RADD [20] simplified the framework by deriving a time-independent formulation of the concrete score, eliminating the need for time embeddings and enabling efficient caching. Nie et al.

[18] established scaling laws showing that while masked diffusion models require approximately 16× more compute to match autoregressive (AR) perplexity, they exhibit similar scaling trends. LLaDA [19] was the first to scale masked diffusion to 8B parameters, demonstrating competitive performance with AR models. LLaDA-MoE [35] further showed that sparse mixture-of-experts integrates effectively with masked diffusion, matching dense model performance with 1/6 active parameters. Recent work has also demonstrated that diffusion language models can be enhanced through reinforcement learning to improve reasoning capabilities [21, 26, 34].

**Adapting Autoregressive Models to Diffusion.** Given the substantial investment in pretrained AR models, recent works explore efficient adaptation strategies. DiffuLLaMA [11] introduced the shift operation to preserve AR's next-token prediction structure and attention mask annealing for gradual transition to bidirectional attention. Dream 7B [30] proposed context-adaptive noise rescheduling to weight loss by contextual information density, achieving strong performance with only 0.6T tokens from Qwen2.5. Dream-Coder [27] extended this approach to code generation, revealing emergent non-linear generation patterns such as sketch-first reasoning.

**Block Diffusion and Inference Acceleration.** Block diffusion methods apply diffusion within fixed-size blocks while maintaining AR dependencies across blocks. BD3-LM [1] introduced vectorized training and clipped noise schedules to address gradient variance. NBDiff [25] viewed AR as block diffusion with block size 1 and proposed gradual block growth with context-causal attention for smooth adaptation. SDAR [5] demonstrated lightweight AR-to-block-diffusion adaptation with dynamic confidence-based truncation, preserving model capabilities while enabling parallel decoding. Efficient-DLM [9] proposed block-wise attention that remains causal across blocks while enabling bidirectional modeling within each block, combined with position-dependent masking for effective AR-to-dLM conversion. SDLM [16] proposed adaptive-length speculative decoding using longest-prefix decoding within diffusion blocks. SBD [10] unified next-token and masked-token prediction within a single architecture, leveraging entropy-bounded samplers for flexible parallel decoding. LLaDA2.0 [3] further demonstrated the scalability of block diffusion models to 100B parameters.

**Permutation and Reordering.** XLNet [29] studies permutation language modeling, i.e., training an autoregressive objective under random factorization orders with permutation-dependent masking (via two-stream attention) to avoid information leakage. `WeDLM` is different in both goal and mechanism: we focus on inference-time acceleration with diffusion-style parallel decoding while keeping standard causal attention. Our Topological Reordering simply moves currently observed tokens to the physical prefix so masked tokens can attend to them under an unmodified lower-triangular mask, preserving logical positions (e.g., via RoPE position ids) and remaining KV-cache friendly.

## 8. Conclusion

We introduced `WeDLM`, a diffusion-style decoding framework that is explicitly optimized for prefix-cacheable generation under standard causal attention. Our analysis highlights that, in KV-cached decoding, the dominant efficiency driver is not merely "tokens predicted per forward", but the rate at which predictions become a contiguous left-to-right prefix and therefore reusable, which we formalize via prefix cacheability $p_{\text{cache}}$ (Eq. 4). This viewpoint also clarifies why out-of-order resolution and bidirectional KV coupling are fundamentally misaligned with fast decoding: both reduce the fraction of computation that can be amortized by caching.

`WeDLM` addresses this mismatch by enforcing a causal dependency structure throughout training and inference. Topological Reordering exposes the full observed context to masked positions while preserving the strict causal mask, making each newly committed token immediately cache-valid. Building on this property, Streaming Parallel Decoding biases acceptance toward earlier logical positions and continuously refills a fixed window, converting parallel proposals into prefix growth with minimal recomputation. Empirically, `WeDLM` retains (and often improves) the capabilities of strong AR backbones while delivering substantial inference acceleration under matched, cache-enabled decoding.

More broadly, our results suggest that prefix-cacheability should be treated as a first-class design objective for parallel text generation. Since the optimal reuse pattern is inherently close to prefix order, future diffusion language models should be constructed as more effective multi-token prediction (MTP) mechanisms: generating many tokens per iteration is only beneficial insofar as those tokens can be quickly promoted into a cache-valid prefix under a causal computation graph. In this sense, causal diffusion provides a principled route to reconcile diffusion-style parallelism with the algorithmic structure required for efficient cached decoding.

## References

[1] M. Arriola, A. Gokaslan, J. T. Chiu, Z. Yang, Z. Qi, J. Han, S. S. Sahoo, and V. Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. arXiv preprint arXiv:2503.09573, 2025.

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

[3] T. Bie, M. Cao, K. Chen, L. Du, M. Gong, Z. Gong, Y. Gu, J. Hu, Z. Huang, Z. Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. arXiv preprint arXiv:2512.15745, 2025.

[4] M. Chen. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

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

[6] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. arXiv preprint arXiv:1803.05457, 2018.

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

[8] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in neural information processing systems, 35: 16344–16359, 2022.

[9] Y. Fu, L. Whalen, Z. Ye, X. Dong, S. Diao, J. Liu, C. Wu, H. Zhang, E. Xie, S. Han, et al. Efficient-dlm: From autoregressive to diffusion language models, and beyond in speed. arXiv preprint arXiv:2512.14067, 2025.

[10] I. Gat, H. Ben-Hamu, M. Havasi, D. Haziza, J. Reizenstein, G. Synnaeve, D. Lopez-Paz, B. Karrer, and Y. Lipman. Set block decoding is a language model inference accelerator. arXiv preprint arXiv:2509.04185, 2025.

[11] S. Gong, S. Agarwal, Y. Zhang, J. Ye, L. Zheng, M. Li, C. An, P. Zhao, W. Bi, J. Han, et al. Scaling diffusion language models via adaptation from autoregressive models. arXiv preprint arXiv:2410.17891, 2024.

[12] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300, 2020.

[13] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300, 2020.

[14] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL `https://arxiv.org/abs/2309.06180`.

[15] J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. Advances in Neural Information Processing Systems, 36:21558–21572, 2023.

[16] Y. Liu, Y. Cao, H. Li, G. Luo, Z. Chen, W. Wang, X. Liang, B. Qi, L. Wu, C. Tian, et al. Sequential diffusion language models. arXiv preprint arXiv:2509.24007, 2025.

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

[18] S. Nie, F. Zhu, C. Du, T. Pang, Q. Liu, G. Zeng, M. Lin, and C. Li. Scaling up masked diffusion models on text. arXiv preprint arXiv:2410.18514, 2024.

[19] S. Nie, F. Zhu, Z. You, X. Zhang, J. Ou, J. Hu, J. Zhou, Y. Lin, J.-R. Wen, and C. Li. Large language diffusion models. arXiv preprint arXiv:2502.09992, 2025.

[20] J. Ou, S. Nie, K. Xue, F. Zhu, J. Sun, Z. Li, and C. Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. arXiv preprint arXiv:2406.03736, 2024.

[21] L. Pan, S. Tao, Y. Zhai, Z. Fu, L. Fang, M. He, L. Zhang, Z. Liu, B. Ding, A. Liu, et al. d-treerpo: Towards more reliable policy optimization for diffusion language models. arXiv preprint arXiv:2512.09675, 2025.

[22] Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

[23] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In First Conference on Language Modeling, 2024.

[24] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. Neurocomputing, 568:127063, 2024.

[25] Y. Tian, Y. Liang, J. Sun, S. Zhang, G. Yang, Y. Shu, S. Fang, T. Guo, K. Han, C. Xu, et al. From next-token to next-block: A principled adaptation path for diffusion llms. arXiv preprint arXiv:2512.06776, 2025.

[26] G. Wang, Y. Schiff, G. Turok, and V. Kuleshov. d2: Improved techniques for training reasoning diffusion language models. arXiv preprint arXiv:2509.21474, 2025.

[27] Z. Xie, J. Ye, L. Zheng, J. Gao, J. Dong, Z. Wu, X. Zhao, S. Gong, X. Jiang, Z. Li, et al. Dreamcoder 7b: An open diffusion language model for code. arXiv preprint arXiv:2509.01142, 2025.

[28] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

[29] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. Advances in neural information processing systems, 32, 2019.

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

[31] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? arXiv preprint arXiv:1905.07830, 2019.

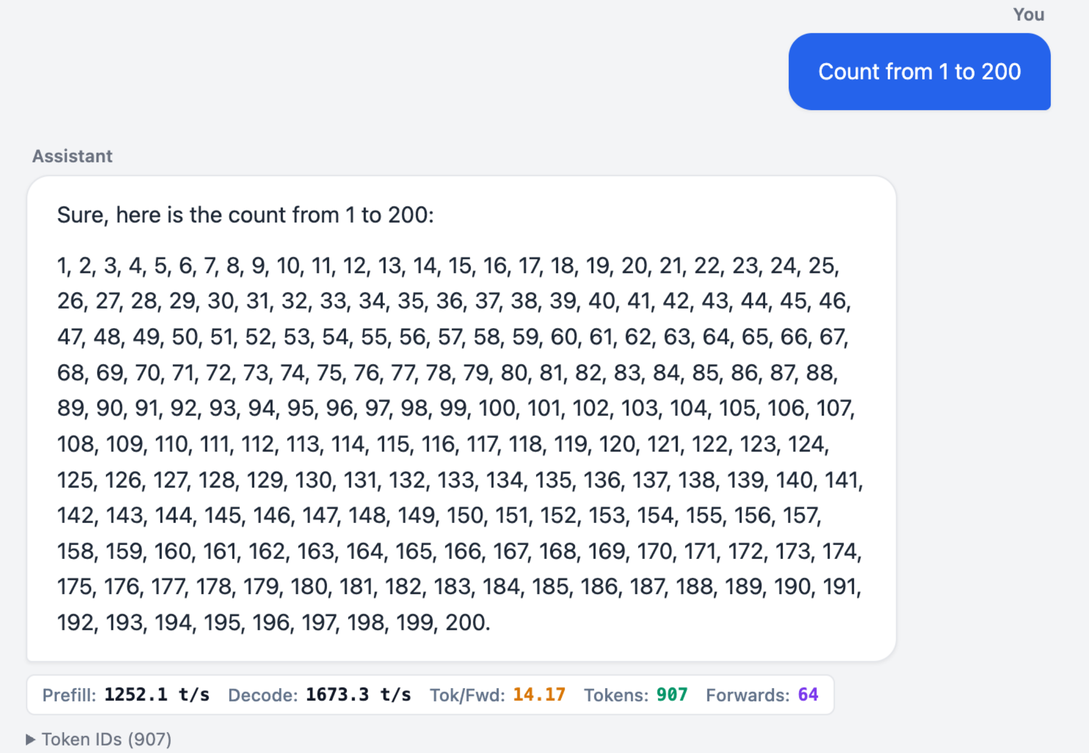
[32] T. Zewei and H. Yunpeng. Magiattention: A distributed attention towards linear scalability for ultra-long context, heterogeneous mask training. `https://github.com/SandAI-org/MagiAttention/`, 2025.

[33] L. Zhang, L. Fang, C. Duan, M. He, L. Pan, P. Xiao, S. Huang, Y. Zhai, X. Hu, P. S. Yu, et al. A survey on parallel text generation: From parallel decoding to diffusion language models. arXiv preprint arXiv:2508.08712, 2025.

[34] S. Zhao, D. Gupta, Q. Zheng, and A. Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. arXiv preprint arXiv:2504.12216, 2025.

[35] F. Zhu, Z. You, Y. Xing, Z. Huang, L. Liu, Y. Zhuang, G. Lu, K. Wang, X. Wang, L. Wei, et al. Llada-moe: A sparse moe diffusion language model. arXiv preprint arXiv:2509.24389, 2025.

# A. Additional Qualitative Results



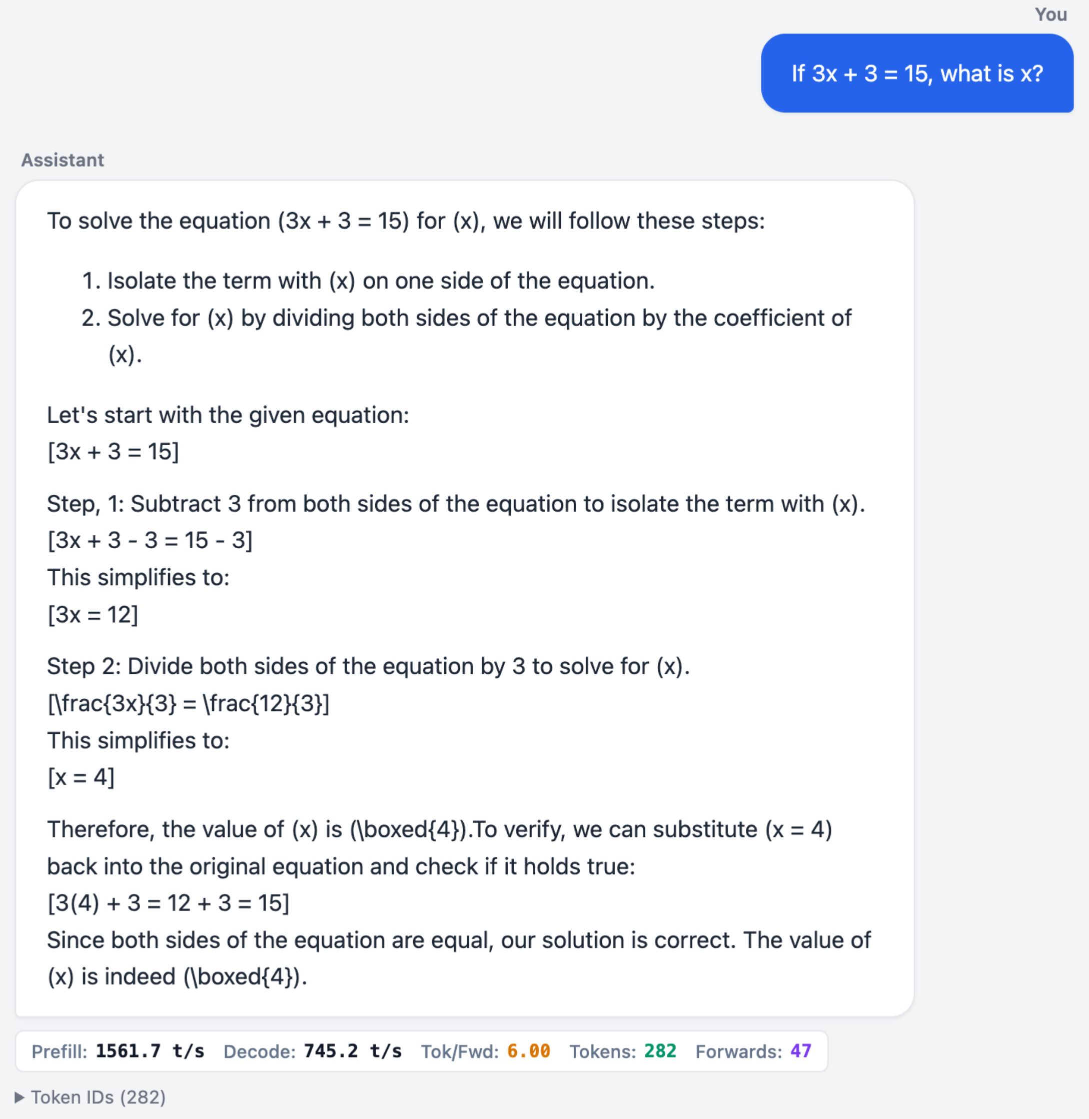
Count from 1 to 200

Sure, here is the count from 1 to 200:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200.

Prefill: `1252.1 t/s` Decode: `1673.3 t/s` Tok/Fwd: `14.17` Tokens: `907` Forwards: `64`

Figure 6 | **Low Entropy Case:** A simple counting task from 1 to 200. Due to the highly predictable deterministic pattern, `WeDLM` achieves a decoding speed of **1673.3 tokens/s**.



If 3x + 3 = 15, what is x?

To solve the equation (3x + 3 = 15) for (x), we will follow these steps:

1. Isolate the term with (x) on one side of the equation.
2. Solve for (x) by dividing both sides of the equation by the coefficient of (x).

Let's start with the given equation:
[3x + 3 = 15]

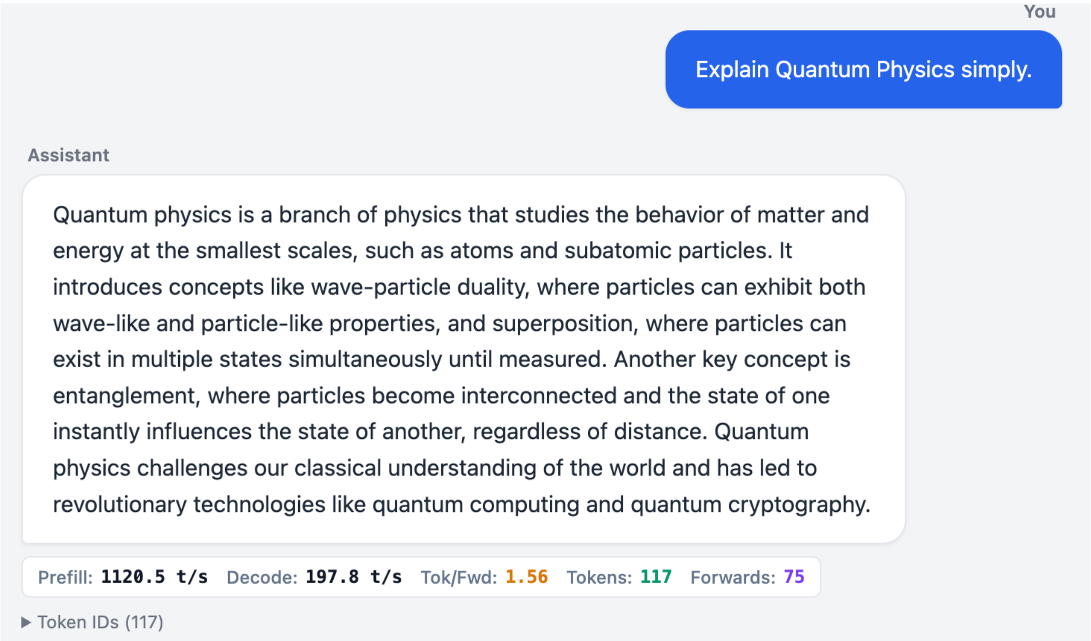
Step, 1: Subtract 3 from both sides of the equation to isolate the term with (x).
[3x + 3 - 3 = 15 - 3]
This simplifies to:
[3x = 12]

Step 2: Divide both sides of the equation by 3 to solve for (x).
[\frac{3x}{3} = \frac{12}{3}]
This simplifies to:
[x = 4]

Therefore, the value of (x) is (\boxed{4}).To verify, we can substitute (x = 4) back into the original equation and check if it holds true:
[3(4) + 3 = 12 + 3 = 15]
Since both sides of the equation are equal, our solution is correct. The value of (x) is indeed (\boxed{4}).

Prefill: **1561.7 t/s** Decode: **745.2 t/s** Tok/Fwd: **6.00** Tokens: **282** Forwards: **47**

Figure 7 | **Medium Entropy Case:** A mathematical reasoning task solving a linear equation. The structured nature of the step-by-step derivation allows for significant parallel decoding, resulting in **745.2 tokens/s**.



Explain Quantum Physics simply.

Quantum physics is a branch of physics that studies the behavior of matter and energy at the smallest scales, such as atoms and subatomic particles. It introduces concepts like wave-particle duality, where particles can exhibit both wave-like and particle-like properties, and superposition, where particles can exist in multiple states simultaneously until measured. Another key concept is entanglement, where particles become interconnected and the state of one instantly influences the state of another, regardless of distance. Quantum physics challenges our classical understanding of the world and has led to revolutionary technologies like quantum computing and quantum cryptography.

Prefill: 1120.5 t/s Decode: 197.8 t/s Tok/Fwd: 1.56 Tokens: 117 Forwards: 75

Figure 8 | **High Entropy Case:** An open-ended knowledge explanation (Quantum Physics). The high semantic diversity and need for precise lexical selection reduce the effective parallel block size, resulting in a speed of **197.8 tokens/s**.