Introspective Diffusion Language Models

Yifan Yu[*,1,2], **Yuqing Jian**[*,1], **Junxiong Wang**[1], **Zhongzhu Zhou**[1], **Donglin Zhuang**[1], Xinyu Fang[1], Xiaoxia Wu[1], Qingyang Wu[1], Shuaiwen Leon Song[1], Tri Dao[1,4], Ben Athiwaratkun[1], James Zou[†,1,5], Fan Lai[†,◇,2], Chenfeng Xu[†,◇,1,3]

[1]Together AI [2]University of Illinois Urbana-Champaign
[3]The University of Texas at Austin [4]Princeton University [5]Stanford University

[*]Equal contribution [†]Equal advising [◇]Corresponding author

Abstract

Diffusion language models promise parallel generation, yet still lag behind autoregressive (AR) models in quality. We attribute this gap to a failure of introspective consistency: AR models agree with their own generations, while DLMs often do not. We define the introspective acceptance rate, which measures whether a model accepts its previously generated tokens. This reveals why AR training has a structural advantage: causal masking and logit shifting implicitly enforce introspective consistency. Motivated by this observation, we introduce Introspective Diffusion Language Model (I-DLM), a paradigm that retains diffusion-style parallel decoding while inheriting the introspective consistency of AR training. I-DLM uses a novel introspective strided decoding (ISD) algorithm, which enables the model to verify previously generated tokens while advancing new ones in the same forward pass. From a systems standpoint, we build an I-DLM inference engine on AR-inherited optimizations and further customize it with a stationary-batch scheduler. To the best of our knowledge, I-DLM is the first DLM to match the quality of its same-scale AR counterpart while outperforming prior DLMs in both model quality and practical serving efficiency across 14 benchmarks. It reaches 69.6 on AIME-24 and 45.7 on LiveCodeBench-v6, exceeding LLaDA-2.1-mini (16B) by more than 26 and 15 points, respectively. Beyond quality, I-DLM is designed for the growing demand of large-concurrency serving, delivering about 3× higher throughput than state-of-the-art DLMs.

Code | **Website** | **Models**

1 Introduction

Diffusion language models (DLMs) (Austin et al., 2021; Sahoo et al., 2024; Nie et al., 2025a) offer an appealing alternative to autoregressive (AR) language models: by iteratively refining a block of tokens, they break the sequential bottleneck of next-token decoding and enable parallel generation. Yet this promise remains largely unrealized, as shown in Figure 1. First, a substantial quality gap between DLMs and AR models remains a key barrier to adoption (§2). Second, from an efficiency standpoint, limited systems support for diffusion inference prevents DLMs from translating their theoretical parallelism into practical speedups.

The historical trajectory of DLM development is highly revealing. Across a broad line of work—from early continuous formulations (Li et al., 2022) and uniform-state DLMs (Austin et al., 2023) to discrete diffusion models (Lou et al., 2024); and from fully bidirectional attention (Nie et al., 2025b), to blockwise decoding (Cheng et al., 2025; Wu et al., 2025) and causal-mask decoding (Liu et al., 2025a)—each generation moves closer to discrete AR language models. Increasingly AR-like training signals have also been introduced to improve optimization and quality (Liu et al., 2025b; Gat et al., 2025; Ye et al., 2025; Tian et al.,

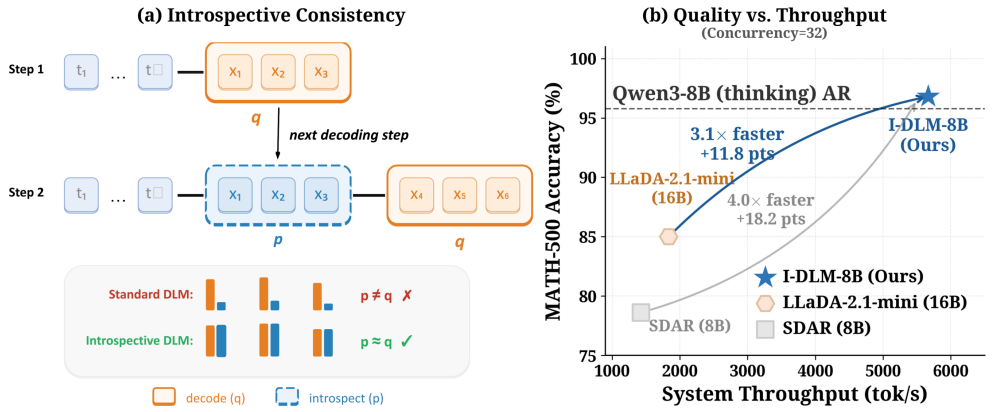


Figure 1: **(a)** Introspective consistency: standard DLMs generate tokens whose distributions q diverge from the model's own next-step predictions p; I-DLM trains generation and introspection to agree ($p \approx q$). **(b)** Quality vs. throughput on MATH-500: I-DLM-8B matches Qwen3-8B (thinking) AR performance while achieving 3.1× higher throughput and +11.8 points over LLaDA-2.1-mini (16B), and 4.0× higher throughput over SDAR (8B).

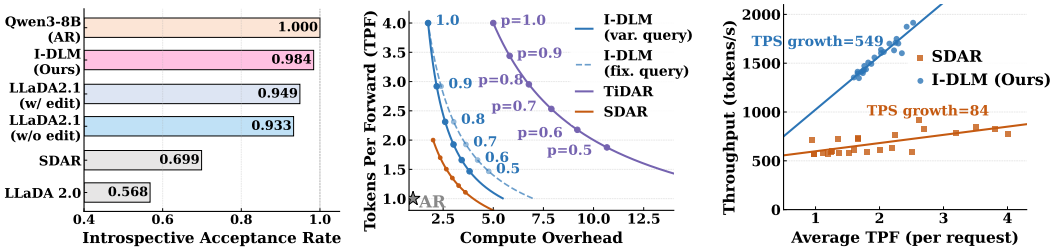
2025). In other words, much of the field has implicitly converged on the same intuition: the path to stronger DLMs should progressively move closer to AR models. In this work, we argue for a different trajectory. Rather than beginning from diffusion and asking how to make it more like AR, we begin from AR and ask: *what is the essential principle that makes AR models so strong, and can it be preserved in a parallel generation paradigm?*

We revisit this question from two perspectives: algorithms and systems. From the algorithmic perspective, this change in viewpoint leads us to re-examine a deceptively simple design at the core of AR modeling: causal masking together with logit shifting. Beyond enabling next-token prediction, this training structure implicitly teaches the model to revisit and validate its previously generated tokens under the same predictive rule. In effect, AR models are trained to agree with what they generate, exposing a fundamental limitation of existing DLMs: although they can generate tokens in parallel, they are generally *not* trained to agree with their own generations (e.g., due to multi-step bidirectional denoising). We formalize this gap through the notion of *introspective acceptance rate*, which measures whether a model internally accepts its previously generated tokens. We find that this property serves as a useful proxy for the degree to which a DLM remains consistent with its own generations.

From the systems perspective, existing DLMs are often optimized for aggressive low-latency decoding, but this comes at the cost of substantially higher computational overhead. While such overhead can be partially hidden in memory-bound regimes, production deployments requiring large-batch inference quickly hit the compute-bound regime. Unfortunately, LLM serving stacks are poorly matched to the multiple-query, multiple-denoising patterns.

Contributions. These algorithmic-system mismatches motivate us to co-design the *Introspective Diffusion Language Model* (I-DLM), a new paradigm that preserves the introspective consistency of AR while retaining parallelism. I-DLM is built on introspective-consistency training, an efficient recipe for converting pretrained AR models into DLMs, and a novel introspective strided decoding (ISD) algorithm that generates N tokens per forward pass while verifying prior tokens against a causal anchor distribution. We show that explicitly enforcing introspective consistency during training is key to substantially closing the quality gap between DLMs and strong same-scale AR models. We make the following contributions:

- **A key insight: introspective consistency is the missing principle in prior DLMs.** We show that diffusion language models do not inherit the *introspective consistency* of AR models: they are not trained to agree with their own generations. This missing property fundamentally limits their ability to realize the full capability of the underlying model.
- **A new training paradigm for high-quality parallel generation.** We introduce *introspective-consistency training*, a simple yet effective recipe for converting pretrained AR models into introspective DLMs (e.g., just using 5B tokens). By explicitly enforcing



(a) **Generation–introspection gap.** Introspective acceptance rate (avg $\min(1, p/q)$) between generation (q) and introspection (p) distributions. AR and Ours achieve near-perfect consistency.

(b) **Compute economics** at stride N=4. p denotes the per-token acceptance probability. ISD achieves high TPF at low overhead.

(c) **Batching efficiency.** TPF vs. batch throughput (tokens/s) at batch size 8. SDAR's throughput barely scales with TPF; I-DLM translates higher TPF into proportionally higher throughput.

Figure 2: **Bottleneck analysis.** Key gaps between DLMs and AR models: (a) existing DLMs exhibit a generation–introspection gap—they can generate tokens but cannot reliably introspect on their own output, as measured by the introspection rate; (b) DLM parallel decoding consumes far more compute per token, collapsing throughput under concurrency; (c) higher TPF does not translate to proportionally higher throughput for existing DLMs.

the model's introspective consistency, our method enables parallel decoding without sacrificing AR-level quality. Unlike prior advances, it requires neither distillation schedules nor masking curricula, yielding a stable and efficient path to high-quality DLMs.

- **A novel single-pass decoding algorithm that unifies generation and verification.** We propose *Introspective Strided Decoding (ISD)*, which simultaneously generates new tokens and revises prior ones within the same forward pass. At [MASK] positions, the model proposes new tokens; at introspection positions, it revisits previous tokens against its causal anchor distribution. This yields outputs that provably match the base AR distribution, without confidence heuristics or separate verification passes.
- **An AR-compatible serving stack for deployable DLMs and self-speculative decoding** We design an inference stack that is directly compatible with existing AR serving systems (e.g., SGLang). Besides, we develop a gated residual adaptation mechanism where LoRA adapters are applied only at mask positions, while verification relies on the base model weights. It supports a continuum between near-lossless and strictly lossless modes.
- **Comprehensive evidence that closes the quality gap while delivering real efficiency gains.** Across 14 benchmarks, we show that I-DLM is the first DLM to match strong same-scale AR quality while substantially outperforming prior DLMs in both capability and serving efficiency. Our results establish a new quality–efficiency frontier.

We have open-sourced models and systems for community research and deployment.

2 Background and Motivation

Despite DLLM's promise of parallel generation, existing DLLMs lag behind AR models in both quality and throughput. We trace these gaps to three inherent limitations of current DLMs (Figure 2). (1) Low introspective consistency: DLMs cannot reliably agree with their generations, making coherent reasoning difficult; (2) Compute inefficiency: both training and inference require substantially more FLOPs in tokens per forward (TPF) than AR models; and (3) Inference engine incompatibility: multi-token, multi-step denoising in DLM inference is poorly aligned with modern AR serving stacks, leading to inefficient execution.

DLMs lack introspective consistency. We formalize *introspective acceptance rate* α as follows: for each generated token x_k sampled from distribution q_k, we perform a separate forward pass with all tokens revealed to obtain the corresponding causal distribution p_k and compute $\alpha = \frac{1}{L}\sum_k \min(1,\, p_k(x_k)/q_k(x_k))$. For AR models, $p = q$ by construction, yielding $\alpha = 1$ and thus perfect generation-introspection consistency. We evaluate α on IFEval using each model's best configuration from its official release (Figure 2a). SDAR (8B) achieves only 0.699

and LLaDA 2.0-flash (100B) only 0.568, indicating substantial divergence between what these models generate and what they would endorse upon re-examination. LLaDA 2.1 (Bie et al., 2026) is philosophically close to us. Although it is motivated as improving the revision capability of DLMs, we view revision and introspective acceptance as deeply connected: improving a model's ability to revise its own outputs naturally improves its ability to endorse them upon re-examination. LLaDA 2.1 restructures the training and data pipeline for a token-to-token supervision format, whereas our introspective-consistency training is much simpler, jointly training generation and introspection under a unified objective without redesigning the data pipeline (§3.1). Despite this simplicity, I-DLM achieves substantially higher introspective acceptance rates with far greater token efficiency: it matches its AR base model using only 4.5B training tokens, while SDAR needs 54B tokens (12× more) and still yields much worse downstream quality (10.0 vs. 69.6 on AIME-24).

Parallel decoding does not translate to compute efficiency. Existing baselines incur more FLOPs per decoded token, which effectively dilutes the achieved speedup by pushing the kernel to a compute bound, as shown in Figure 2b. We define compute overhead as the ratio of total FLOPs to decode a given number of tokens compared to AR decoding. We provide detailed tokens per forward (TPF) pass and compute overhead analysis given different per-token acceptance rate in Appendix B. Figure 2b shows the tradeoff at stride N=4 for I-DLM, SDAR (block diffusion model), and Tidar (DLM with branched inference methods): at a TPF of ∼2.5, I-DLM incurs only ∼2.5× compute overhead, while TiDAR requires ∼7.8×. In contrast, SDAR's TPF is capped at 2.0. This is because block diffusion (SDAR, LLaDA) requires T denoising steps plus a mandatory KV-commit forward that produces no new tokens, capping TPF at $N/2$ even under ideal acceptance (p=1). Appendix C details why this overhead is hard to eliminate. We further analyze compute efficiency in Appendix B.5.

DLM inference is incompatible with AR serving engines. The modern LLM serving stack (e.g., continuous batching, fused attention kernels, paged KV cache) is highly optimized for AR decoding. DLMs break these assumptions. In AR serving, continuous batching works because all requests advance uniformly. In block diffusion, tokens within a block converge at different rates: some positions pass the confidence threshold early, yet all requests must synchronize at the slowest block, wasting the TPF gain. Figure 2c reflects this: SDAR's TPS growth rate (slope=84) is nearly flat with respect to TPF. In contrast, ISD uses strict causal attention (compatible with AR kernels) and adaptive stride where each step produces at least one quality-guaranteed token via introspection, achieving a TPS growth rate of 549 (§3.3). We show other AR serving mismatch (e.g., attention kernel mismatch) in Appendix D.

We bring up further discussions on related works in Appendix A.

3 Introspective Diffusion Language Model

We present I-DLM to address the aforementioned bottlenecks. Causal attention with logit shift closes the generation–introspection gap and improves training efficiency (§3.1); introspective strided decoding eliminates the compute overhead of iterative denoising (§3.2); and the preserved causal structure enables direct integration into AR serving stacks (§3.3).

3.1 Introspective-Consistency Training

We convert a pretrained AR model into an introspective diffusion model by combining causal attention, logit shift, and an all-masked objective. The training mask is shown in Appendix E. We note that the training method itself is simple, but it is built on a deep and fundamental insight: This design preserves what is most elegant about introspective-consistency and objective–inference alignment while retaining the parallel decoding benefits of diffusion models. We discuss this in the Appendix H

Causal training with logit shift. We apply *strict causal masking* uniformly across both the masked and clean portions: for any query position j and key position i, attention is permitted only when $i \leq j$. Unlike SDAR which uses block-causal attention (bidirectional within blocks), this holds for masked tokens in x_t as well—each `[MASK]` attends only to preceding positions. In the clean region x_0, standard causal attention is used to ensure

generation and introspection operate under the same attention pattern (Section 2) and enable KV cache reuse at inference. We pair this with a logit shift: the hidden state at position i is trained to predict token $i{+}1$ rather than token i. Standard masked diffusion trains hidden$[\mathtt{M}_i] \to$ token$[i]$, which breaks the AR model's inherent logits$[i] \to$ token$[i{+}1]$ mapping and crucially prevents clean positions from providing a meaningful verify signal. Our shifted formulation preserves this mapping: clean positions produce the causal anchor p (the verify distribution), while masked positions produce tokens q (the decode distribution).

All-masked training with auto-balanced loss. Given a clean sequence $x_0 = (x_1, \ldots, x_L)$, we replace *all* tokens with a special [MASK] token to obtain the fully masked input $x_t = (\mathtt{M}, \ldots, \mathtt{M})$. The training input is the concatenation $[x_t \mid x_0]$, where x_t is the all-masked sequence and x_0 provides the clean reference. Unlike standard masked diffusion training, which masks a random fraction r of tokens, wasting $(1{-}r)$ of the compute on unsupervised positions, our all-masked regime ensures that every position contributes a useful training signal, eliminating the supervision dilution identified in Section 2.

We apply a cross-entropy loss with shifted labels separately to the masked and clean regions:

$$\mathcal{L}_{\text{mask}} = -\frac{1}{|\mathcal{S}_t|} \sum_{\ell \in \mathcal{S}_t} \log p_\theta(x_0^{\ell+1} \mid [x_t, x_0]_{\leq \ell}), \quad \mathcal{L}_{\text{clean}} = -\frac{1}{|\mathcal{S}_0|} \sum_{\ell \in \mathcal{S}_0} \log p_\theta(x_0^{\ell+1} \mid [x_t, x_0]_{\leq \ell}), \tag{1}$$

where $\mathcal{S}_t$ and $\mathcal{S}_0$ are the non-padding positions in the masked and clean regions, respectively, and $x_0^{\ell+1}$ is the shifted target (next token). This is the same cross-entropy objective used in AR pretraining—no separate distillation loss or teacher model is required. On clean positions, $\mathcal{L}_{\text{clean}}$ trains the *introspection* pathway: the model learns to produce the causal anchor distribution $p_\theta(x_{i+1} \mid x_{\leq i})$, recovering the exact AR training objective. On masked positions, $\mathcal{L}_{\text{mask}}$ trains the *decode* pathway: the model learns to produce tokens q from [MASK] hidden states, enabling strided generation at stride $N > 1$.

Although both pathways share the same cross-entropy objective, their loss magnitudes can differ substantially: masked positions face a harder prediction task and tend to produce larger losses, especially early in training. A fixed weighting risks the decode pathway dominating the gradient, undermining the introspection pathway that is critical for verification quality. We address this with an auto-balanced loss:

$$\mathcal{L} = \mathcal{L}_{\text{mask}} + \hat{s} \cdot \mathcal{L}_{\text{clean}}, \quad \hat{s} = \frac{\mathcal{L}_{\text{mask}}}{\mathcal{L}_{\text{clean}}}, \tag{2}$$

where $\hat{s}$ is the ratio of loss magnitudes, treated as a fixed scalar at each training step (not differentiated through). This dynamically rescales the clean-position loss to match the magnitude of the masked-position loss, ensuring both pathways receive equal effective gradient magnitude without manual tuning. Because both pathways share the same objective and receive balanced supervision, the model naturally aligns q with p, maximizing the introspective acceptance rate (Figure 2a).

3.2 Introspective Strided Decoding (ISD)

Introspective Strided Decoding (ISD) is the inference counterpart to I-DLM training. Because the model uses strict causal attention, ISD begins with a standard AR prefill over the prompt, then *dynamically* selects the effective stride at each decode step via the p/q acceptance criterion. Unlike block diffusion that commits to a fixed block size, or speculative decoding that requires a separate draft model, the stride in ISD adapts intrinsically based on the model's own self-consistency: easy tokens are accepted in parallel while difficult tokens fall back toward AR-quality generation.

Single-pass stride and introspection. ISD generates tokens in steps (Algorithm 1, Figure 3 bottom): ❶ *Step 1 (bootstrap).* We append N [MASK] tokens to the prefix and run a single forward pass. Due to the logit shift, the last clean position produces a quality-guaranteed token x_1—identical to an AR prediction, requiring no introspection. This token anchors the subsequent introspection chain: at the next step, x_1's hidden state produces the causal anchor p_2 for evaluating $\hat{x}_2$, and so on. The remaining $N{-}1$ [MASK] positions produce

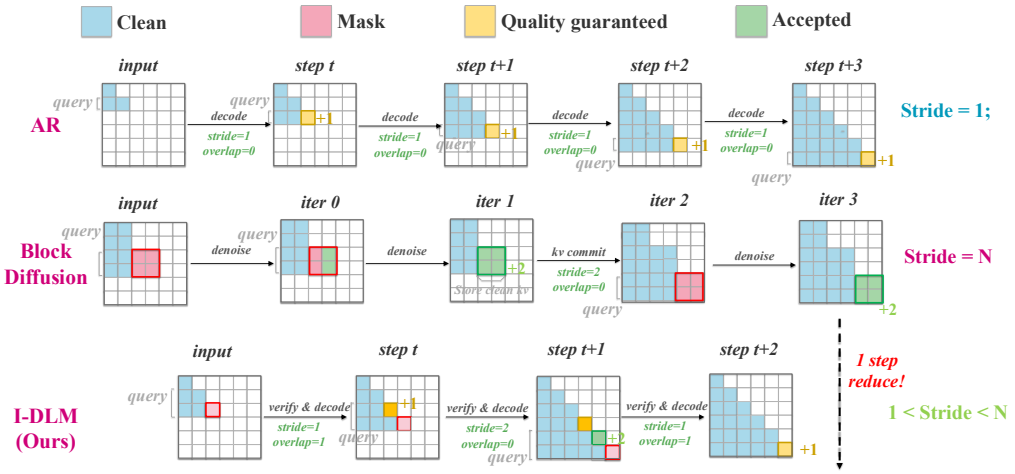


Figure 3: **Comparison of decoding paradigms.** Our I-DLM uses strict causal attention with adaptive stride ($1 < \text{stride} < N$) and is a drop-in replacement within AR serving infrastructure. ISD produces a quality-guaranteed token $\boldsymbol{x}_{i+1}$ together with draft tokens $\hat{\boldsymbol{x}}_{i+2:i+N}$ via introspective strided decoding, $\boldsymbol{x}_{i+1}, \hat{\boldsymbol{x}}_{i+2:i+N} = \pi_{\text{strided}}(c_{1:i}, m_{1:N-1})$; Residual ISD (R-ISD) additionally gates a LoRA residual for bit-for-bit lossless output: $\boldsymbol{x}_{i+1:i+N} = \pi_{\text{ar}}(c_{1:i}, m_{1:N-1}) + \pi_{\text{L}}(m_{1:N-1})$.

strided tokens $\hat{x}_2, \ldots, \hat{x}_N$; and ❷ *Step $t > 1$ (stride + introspection).* We fill in the accepted (or resampled) tokens from the previous step, append N fresh `[MASK]` tokens, and run a *single* forward pass that simultaneously:

- *Introspects* on the previous tokens; the filled-in tokens are now clean, so the logits at each position produce causal anchor distributions p_k (the true AR next-token distributions);
- *Generates* new tokens from the appended `[MASK]` positions.

Crucially, introspection piggybacks on the next stride step at zero additional cost: every step after the first produces both accepted tokens and fresh tokens in one forward pass. This contrasts with SDAR ($T{+}1$ forwards per block), WeDLM (confidence-based streaming without distributional guarantees), and TiDAR (unified draft and introspection but $N(N{+}1)$ token queries per step; our logit shift reduces this to N^2).

Adaptive stride via p/q acceptance. We introspect on each token by comparing it against its causal anchor using the p/q acceptance criterion (Leviathan et al., 2023): token x_k sampled from q_k is accepted with probability $\min(1,\, p_k(x_k)/q_k(x_k))$. When accepted, the token provably follows the causal anchor distribution. On rejection, the token is resampled from the corrected distribution $\text{normalize}(\max(0, p_k - q_k))$ and all subsequent tokens are discarded; the next step becomes a pure stride with no tokens to introspect. When all tokens are accepted, a bonus token is sampled from the final anchor distribution, achieving stride $N{+}1$. This mechanism naturally adapts the effective stride per step without manual tuning. The formal procedure is given in Algorithm 1 in Appendix K.

Lossless ISD with residual adaptation. ISD uses a single model as both proposer and introspector, eliminating the need for a separate draft model to train, maintain, or synchronize. While it can be used as a standalone generative model, its single-forward drafting-and-verification structure also makes it a natural fit for self-speculative decoding.

Inspired by the gated LoRA approach in multi-token prediction (Samragh et al., 2025), we gate the LoRA residual with a per-token binary mask: `[MASK]` positions use base+LoRA weights to produce high-quality strided tokens, while introspection positions use *base-model-only* weights. Critically, because the entire model uses strict causal attention, introspection positions cannot attend to token positions—the causal anchor distribution p is computed from base-only weights over a base-only KV cache, identical to a pure base AR forward pass. This guarantees that ISD introspects against the *exact* base AR distribution, making the output bit-for-bit lossless (Appendix G).

Theoretical speedup analysis. Let p_k denote the acceptance probability of the k-th strided token, and $P_k = \prod_{j=1}^{k} p_j$ the cumulative acceptance probability. For ISD at stride N, the expected tokens per forward pass (TPF) is (see Appendix B for derivation):

$$\text{TPF}_N = \frac{2 + P_1 + P_2 + \cdots + P_{N-2}}{2 - P_{N-1}}. \tag{3}$$

At perfect acceptance ($p_k = 1$), $\text{TPF}_N = N$ recovers the theoretical maximum. At $p_k = 0$, $\text{TPF}_N = 1$ degenerates to AR. In the memory-bound decode regime, forward pass latency is approximately constant regardless of stride size, so wall-clock speedup $\approx$ TPF. With typical acceptance rates of $p \geq 0.85$, stride N=3 achieves TPF $\approx$ 2.3–2.4$\times$ with a compute overhead of only $\sim$2$\times$—meaning ISD translates most of the parallel decoding benefit into real throughput gain with minimal wasted compute. Our evaluations show that I-DLM achieves close-to-optimal speedup empirically (Section 4).

3.3 I-DLM Serving Stack: AR-Compatible Serving

Prior DLMs rely on custom inference pipelines that forgo the optimizations accumulated in AR serving systems (§2). Because I-DLM preserves strict causal attention, we integrate it directly into SGLang as a drop-in extension, inheriting paged KV cache, continuous batching, and tensor parallelism without modification. We illustrate the main system optimization below. Further details are available in §K.1 (ablated in §4.2).

(1) AR-inherited optimizations. Each ISD step maps to SGLang's native *extend* mode, appending $2N-1$ tokens with causal attention. Unlike block diffusion—where requests within a batch must synchronize at the slowest converging block, breaking continuous batching (§2)—ISD produces at least one quality-guaranteed token every step, so all requests advance uniformly and continuous batching works unmodified. This is why I-DLM's throughput scales proportionally with concurrency while block diffusion methods plateau (Figure 2c). The causal structure further enables two key reuses. First, we capture the entire extend forward into a single CUDA graph, replaying it each step with only `input_ids` and attention metadata updated in-place. Second, since our extended sizes are small ($\leq$9 tokens), we replace the three-kernel attention cascade with a single paged-only kernel per layer, eliminating $2L$ redundant launches (Appendix D).

(2) Stationary-batch scheduling. ISD has a strict dependency chain: forward $\rightarrow$ verify $\rightarrow$ trim $\rightarrow$ prepare $\rightarrow$ forward. This prevents the CPU–GPU overlap used in AR serving, making CPU overhead directly additive to step latency. We mitigate this with a stationary-batch decode loop that reuses the batch object across consecutive ISD steps, bypassing the full scheduler rebuild. Within this loop, KV slots are allocated via a single batched scatter, constant metadata is cached, and the ISD-specific KV trim-and-commit cycle frees rejected and MASK positions after each verification. Non-critical I/O is deferred to an overlap window during the next GPU forward.

4 Experiments

Setup: We train two I-DLM variants: **I-DLM-8B** and **I-DLM-32B**, converted from Qwen3-8B and Qwen3-32B (Yang et al., 2025), respectively, using the all-masked causal training recipe (Section 3.1). We provide the training and evaluation setup in Appendix I.

4.1 End-to-End Performance

Table 1 summarizes I-DLM's quality; ablations on different stride settings are in §4.2.

I-DLM surpasses the quality of strongest DLMs with comparable sizes. I-DLM consistently outperforms all existing DLMs, often by large margins. At 8B scale, I-DLM-8B exceeds LLaDA-2.1-mini (16B) across all benchmarks despite using half the parameters, with particularly large gains on reasoning and code tasks (e.g., +26.3 on AIME-24 and +15.3 on LiveCodeBench-v6). Compared to SDAR, which is built on the same Qwen3-8B base,

Table 1: **End-to-end quality.** Accuracy (%) on 14 benchmarks. I-DLM results use ISD (N=4, sampling). Underline: best non-AR result under 30B. †: best non-AR result under 100B.

	LLaDA-2.1 -mini	LLaDA-2.0 -flash	LLaDA-2.1 -flash	SDAR 8B	SDAR 30B-A3B	I-DLM 8B	Qwen3 8B	I-DLM 32B	Qwen3 32B
Params	*16B*	*100B*	*100B*	*8B*	*30B*	*8B*	*8B*	*32B*	*32B*
Knowledge & Reasoning									
ARC-C	90.2	—	—	91.9	93.2	95.8	95.8	96.8†	97.2
MMLU	74.5	—	—	78.6	82.8	82.4	83.5	86.8†	87.2
MMLU-Pro	64.8	74.8	76.6	56.9	61.5	73.1	75.1	79.7†	80.1
GPQA-D	46.0	—	—	40.2	36.7	55.6	58.9	62.1†	64.1
GPQA	53.3	62.3	67.3†	—	—	54.9	55.4	58.7	65.0
Math									
GSM8K	89.0	—	—	91.7	91.4	95.0†	96.0	94.9	94.7
MATH-500	85.0	—	—	78.6	77.8	96.8	95.8	97.6†	97.8
MathBench	84.2	—	—	76.9	79.3	89.1	93.1	95.6†	95.5
AIME-24	43.3	—	—	10.0	16.7	69.6	73.1	83.3†	76.7
AIME-25	43.3	60.0	63.3	10.0	10.8	60.8	65.4	80.0†	80.0
Code									
HumanEval	86.0	—	—	78.7	87.2	93.3	95.1	96.3†	96.3
MBPP	82.1	—	—	72.0	71.6	92.2	93.4	94.6†	95.7
LCB-v6	30.4	42.5	45.4	16.6	21.7	45.7	50.3	57.1†	58.3
Instruction Following									
IFEval	83.2	82.6	83.6	61.4	60.6	84.7	84.7	84.7†	84.5

Table 2: **Extended comparison** on benchmarks commonly reported across diffusion LLM methods. "—" indicates the result is not reported in the original paper.

Method	GSM8K	MMLU	HumanEval	MBPP	IFEval
Qwen3-8B (AR)	96.0	83.5	95.1	93.4	84.7
NBDiff (7B)	91.0	82.9	89.0	87.6	60.8
Jacobi Forcing (7B)	91.4	—	83.5	70.4	—
WeDLM (8B)	90.2	75.5	75.0	67.0	—
LightningRL (8B)	90.3	—	72.6	58.3	—
TiDAR (8B)	80.4	76.6	57.9	65.4	—
DREAM (7B)	81.0	70.6	57.9	58.8	62.5
Fast-dLLM (7B)	78.5	—	43.3	28.2	—
Mercury Coder Small	—	—	90.0	76.6	—
Gemini Diffusion	—	—	89.6	76.0	—
Ours (8B)	**95.0**	**82.4**	**93.3**	**92.2**	**84.7**

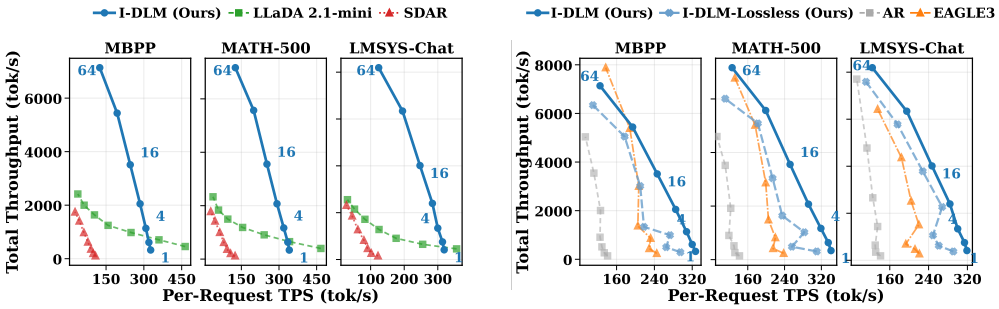
I-DLM improves dramatically (e.g., 69.6 vs. 10.0 on AIME-24). At larger scale, I-DLM-32B continues to outperform substantially larger models, surpassing LLaDA-2.1-flash (100B) by +16.7 on AIME-25 and +11.7 on LiveCodeBench-v6. Moreover, Table 2 shows that our I-DLM even outperforms proprietary DLMs on code generation, such as Mercury Coder Small (93.3 vs. 90.0 on HumanEval) and Gemini Diffusion (92.2 vs. 76.0 on MBPP).

I-DLM achieves comparable quality to AR models.

Beyond outperforming prior DLMs, I-DLM is the first to match the quality of same-scale AR models. As shown in Table 1, I-DLM-8B achieves near-identical performance to Qwen3-8B, matching exactly on ARC-C and IFEval, remaining within $\sim$1 point on MMLU, and even surpassing Qwen3-8B on Math-500. This result validates that introspective consistency is a key missing ingredient in prior DLMs: by aligning generation with verification through logit-shifted causal training, I-DLM recovers AR-level capability.

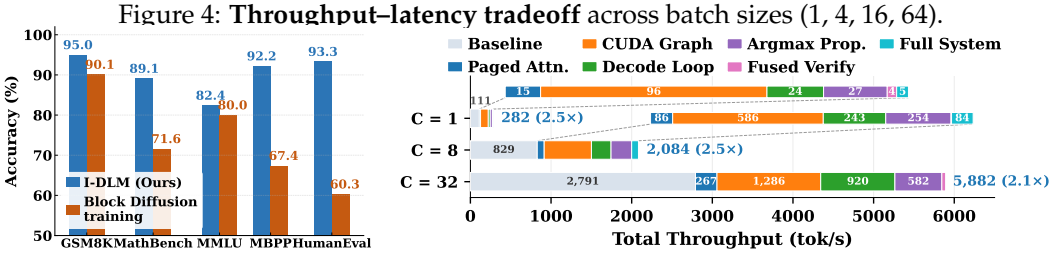
I-DLM delivers superior serving efficiency than existing DLMs. We evaluate end-to-end serving performance on MBPP, MATH-500, and LMSYS-Chat under concurrency levels $C \in \{1, 2, 4, 8, 16, 32, 64\}$ (Appendix I). Figure 4 shows the throughput–latency tradeoff. Across all workloads, I-DLM consistently outperforms prior DLMs at moderate to high concurrency. Starting from $C \geq 4$, I-DLM achieves higher per-request throughput than both SDAR (8B) and LLaDA-2.1-mini (16B), with the advantage widening as concurrency increases. At typical deployment scales (C=16–32), I-DLM delivers 2.2–3.8$\times$ higher throughput than LLaDA-2.1-mini and 3.7–4.5$\times$ over SDAR. Under heavy load (C=64), I-DLM sustains stable per-request throughput ($\sim$125 tok/s), translating to 2.9–4.1$\times$ higher throughput at C=64.

Comparison with speculative decoding for AR models.



(a) **Comparison with DLMs** (b) **Comparison with Spec. Decoding**

Figure 4: **Throughput–latency tradeoff** across batch sizes (1, 4, 16, 64).



(a) **Training ablation.** I-DLM (causal + logit shift) vs. block diffusion (block-causal, no logit shift).

(b) **Systems optimization ablation.** Cumulative throughput at C=1, 8, 32. The full system achieves 2.1–2.5× over the naive baseline.

Figure 5: **Performance breakdown of the training design and systems optimizations.**

Figure 4(b) compares I-DLM against EAGLE3 (Li et al., 2025), a speculative decoding method that relies on an auxiliary draft model on top of the base AR model. I-DLM outperforms EAGLE3 in per-request throughput from C=1 through C=32 across all benchmarks (e.g., 341 vs. 238 tok/s on MATH-500, 319 vs. 221 on LMSYS-Chat, and 327 vs. 245 on MBPP at C=1). Notably, even I-DLM-Lossless—which produces output bit-for-bit identical to the base AR model—exceeds EAGLE3 at most concurrencies (e.g., 310 vs. 238 tok/s at C=1 on MATH-500), despite EAGLE3 requiring a separate draft model. As concurrency increases, I-DLM maintains its advantage over baselines (199 vs. 176 tok/s on MATH-500 and 195 vs. 184 on LMSYS-Chat at C=32).

4.2 Ablation Studies

Ablations of the training design. Figure 5a compares I-DLM's introspective-consistency training (causal attention + logit shift + all-masked objective) against standard block diffusion training (block-causal attention, no logit shift), on the same data budget. The gap is substantial, especially on long-horizon reasoning tasks: code generation drops sharply (HumanEval: 93.3 → 60.3; MBPP: 92.2 → 67.4), and math reasoning degrades significantly (MathBench: 89.1 → 71.6). Yet, knowledge tasks are relatively unaffected (MMLU: 82.4 → 80.0). So introspective consistency improves error accumulation over reasoning chains.

Ablation of the system design. Figure 5b reports efficiency breakdown at C=1, 8, 32. The largest gain comes from CUDA graph capture (+42–76%), which eliminates kernel launch overhead. Decode-loop (stationary-batch scheduling) optimizations (+11–21%) and argmax proposals (+11–15%) further improve throughput by reducing host-side scheduling. Additional gains come from paged-only attention (+10–14%) and kernel fusion (+1–4%).

Impact of stride size. The stride N in ISD controls the parallelism-quality trade-off. Starting from a base model trained at N=3, we extend to larger strides (N=4, 8) via continued training. As shown in Table 3, TPF scales nearly linearly (1.80 → 4.01 from N=2 to N=8), while accuracy remains stable across tasks (e.g., MATH-500 within 94.6–96.8%, MBPP within 88.3–93.4%). These results show that I-DLM sustains quality under increased parallelism.

Adaptive stride via relaxed acceptance. ISD offers a control knob: a loose threshold τ that multiplies the acceptance ratio by $(1+\tau)$, boosting the effective acceptance rate. At τ=0

Table 3: **Impact of stride size.** TPF, TPS (bs=1), and accuracy at stride N on 1×H100.

N	TPF	TPS	MATH-500	MBPP
2	1.80	209.6	96.8	93.4
3	2.48	281.9	95.8	92.8
4	2.96	324.5	96.8	92.2
8	4.01	445.1	94.6	88.3

Table 4: **Relaxed acceptance.** τ trades quality for TPF at N=4.

τ	0	0.1	0.2	0.5	1
HumanEval	93.3	93.3	92.1	91.8	91.2
TPF	2.63	2.62	2.68	2.72	2.73

Table 5: **I-DLM under bidirectional any-order generation.** Bidirectional fine-tuning enables any-order generation while preserving introspective consistency, and introspective editing further improves generation quality.

Model	Properties	Introspective Consistency	Edit Mode	MATH-500	GSM8K	AIME-24@8
SDAR-8B	Any-order	Poor	N/A	78.6	91.7	10.0
LLaDA-2.1-mini (16B)	Any-order	Good	Yes	85.0	89.0	43.3
I-DLM-8B (Bidirectional)	Any-order	Good	No	90.8	87.0	50.8
I-DLM-8B (Bidirectional)	Any-order	Good	Yes	92.8	91.7	62.9
I-DLM-8B	Left-to-right	Best	Yes	96.8	95.0	72.5

(strict), ISD provably matches the AR distribution. Increasing τ relaxes this guarantee for higher TPF without retraining. As shown in Table 4, quality is robust to relaxation: at τ=1.0, HumanEval drops by only 1.6 points (93.3 → 91.2) while TPF increases from 2.63 to 2.73, suggesting that I-DLM's proposals are already well-aligned with the causal anchor.

4.3 I-DLM Beyond Causal Attention: Parallelism and Introspective Consistency

Any-order generation with bidirectional attention has become a standard setting for diffusion language models. However, we argue that neither arbitrary generation order nor bidirectional attention is the defining advantage of a strong DLM. Instead, what matters are two capabilities enabled by diffusion: *parallel generation*, which improves decoding efficiency by generating multiple tokens simultaneously, and *introspective consistency*, which allows the model to revisit and refine its intermediate predictions to improve generation quality. Crucially, flexibility in generation order is not necessarily beneficial (Ni et al., 2026). Our main I-DLM adopts causal attention, but the underlying principles of parallel generation and introspective consistency are not restricted to the causal setting.

We further fine-tune I-DLM-8B with a bidirectional block-attention mask, enabling bidirectional, arbitrary-order generation while preserving parallel decoding. We additionally introduce an *introspective editing* variant that applies the introspective acceptance criterion throughout denoising, adaptively remasking tokens whose predictions exhibit low consistency. This mechanism allows the model to selectively revisit uncertain intermediate decisions and refine them in subsequent denoising steps, extending introspective generation from causal decoding to the bidirectional setting.

As shown in Table 5, I-DLM remains effective under bidirectional any-order generation. Bidirectional I-DLM-8B achieves 90.8/87.0/50.8 on MATH-500, GSM8K, and AIME-24, respectively, and introspective editing further improves these results to 92.8/91.7/62.9. Nevertheless, our main causal I-DLM achieves the strongest performance, reaching 96.8/95.0/72.5 on the same benchmarks. These results show that bidirectional generation is a viable extension of I-DLM and continues to benefit from introspective editing, while any-order generation is not necessarily superior to the causal design (Ni et al., 2026).

5 Conclusion

In this paper, we revisit the DLM design through the lens of autoregressive modeling and identify *introspective consistency* as the missing principle behind their quality gap. Building on this insight, we introduce I-DLM, a new paradigm that unifies parallel generation and self-verification via logit-shifted causal training and introspective strided decoding. Our results show that enforcing consistency enables DLMs to match AR-level quality, substantially outperforming existing DLMs, achieving 2.9–4.1× better throughput on large concurrency.

References

AIME. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces, 2023. URL https://arxiv.org/abs/2107.03006.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, et al. Llada2. 1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

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

Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*, 2025.

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 preprint arXiv:1803.05457*, 2018.

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

Justin Deschenaux and Caglar Gulcehre. Beyond autoregression: Fast llms via self-distillation through time. *arXiv preprint arXiv:2410.21035*, 2024.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

Yonggan Fu, Lexington Whalen, Zhifan Ye, Xin Dong, Shizhe Diao, Jingyu Liu, Chengyue Wu, Hao Zhang, Enze Xie, Song Han, et al. Efficient-dlm: From autoregressive to diffusion language models, and beyond in speed. *arXiv preprint arXiv:2512.14067*, 2025.

Itai Gat, Heli Ben-Hamu, Marton Havasi, Daniel Haziza, Jeremy Reizenstein, Gabriel Synnaeve, David Lopez-Paz, Brian Karrer, and Yaron Lipman. Set block decoding is a language model inference accelerator, 2025. URL https://arxiv.org/abs/2509.04185.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. *arXiv preprint arXiv:2410.17891*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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 preprint arXiv:2103.03874*, 2021.

Lanxiang Hu, Siqi Kou, Yichao Fu, Samyam Rajbhandari, Tajana Rosing, Yuxiong He, Zhijie Deng, and Hao Zhang. Fast and accurate causal parallel decoding using jacobi forcing. *arXiv preprint arXiv:2512.14681*, 2025.

Yanzhe Hu, Yijie Jin, Pengfei Liu, Kai Yu, and Zhijie Deng. Lightningrl: Breaking the accuracy-parallelism trade-off of block-wise dllms via reinforcement learning. *arXiv preprint arXiv:2603.13319*, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. Cllms: Consistency large language models. In *Forty-first International Conference on Machine Learning*, 2024.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

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

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori Hashimoto. Diffusion-LM improves controllable text generation. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=3s9IrEsjLyk.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

Aiwei Liu, Minghua He, Shaoxun Zeng, Sijun Zhang, Linhao Zhang, Chuhan Wu, Wei Jia, Yuan Liu, Xiao Zhou, and Jie Zhou. Wedlm: Reconciling diffusion language models with standard causal attention for fast inference. *arXiv preprint arXiv:2512.22737*, 2025a.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Hongwei Liu, Zilong Zheng, Yuxuan Qiao, Haodong Duan, Zhiwei Fei, Fengzhe Zhou, Wenwei Zhang, Songyang Zhang, Dahua Lin, and Kai Chen. Mathbench: Evaluating the theory and application proficiency of llms with a hierarchical mathematics benchmark. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 6884–6915, 2024b.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. Tidar: Think in diffusion, talk in autoregression. *arXiv preprint arXiv:2511.08923*, 2025b.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion language modeling by estimating the ratios of the data distribution. 2023.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution, 2024. URL https://arxiv.org/abs/2310.16834.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. *arXiv preprint arXiv:2305.09781*, 2023.

Zanlin Ni, Shenzhi Wang, Yang Yue, Tianyu Yu, Weilin Zhao, Yeguo Hua, Tianyi Chen, Jun Song, Cheng Yu, Bo Zheng, et al. The flexibility trap: Rethinking the value of arbitrary order in diffusion language models. *arXiv preprint arXiv:2601.15165*, 2026.

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 preprint arXiv:2502.09992*, 2025a.

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, 2025b. URL https://arxiv.org/abs/2502.09992.

Augustus Odena, Charles Sutton, David Martin Dohan, Ellen Jiang, Henryk Michalewski, Jacob Austin, Maarten Paul Bosma, Maxwell Nye, Michael Terry, and Quoc V Le. Program synthesis with large language models. *n/a, page n/a, n/a. N/a*, 2021.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First conference on language modeling*, 2024.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential. *arXiv preprint arXiv:2507.11851*, 2025.

Yuchuan Tian, Yuchen Liang, Shuo Zhang, Yingte Shu, Guangwen Yang, Wei He, Sibo Fang, Tianyu Guo, Kai Han, Chao Xu, et al. From next-token to next-block: A principled adaptation path for diffusion llms. *arXiv preprint arXiv:2512.06776*, 2025.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

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 preprint arXiv:2505.22618*, 2025.

Shutong Wu and Jiawei Zhang. Free draft-and-verification: Toward lossless parallel decoding for diffusion large language models. *arXiv preprint arXiv:2510.00294*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

A Detailed Related work

Diffusion Language Models. Masked diffusion language models corrupt text by replacing tokens with [MASK] and train a model to reverse the process (Austin et al., 2021; Sahoo et al., 2024; Lou et al., 2023). LLaDA (Nie et al., 2025a) scaled this paradigm to 8B parameters, LLaDA 2.0 (Bie et al., 2025) to 100B via mixture-of-experts, and LLaDA 2.1 (Bie et al., 2026) introduced token editing with confidence-based decoding. DREAM (Ye et al., 2025) introduced the logit shift technique, aligning the diffusion objective with the AR model's logits$[i] \to$ token$[i{+}1]$ mapping. Block Diffusion (Arriola et al., 2025) generates fixed-size blocks autoregressively while denoising tokens within each block.

Converting pretrained AR models into diffusion models is a more data-efficient alternative. Gong et al. (2024) showed that fine-tuning with a masked diffusion objective reduces training cost significantly. Self-Distillation Through Time (Deschenaux & Gulcehre, 2024) and Efficient-DLM (Fu et al., 2025) further streamline the pipeline. SDAR (Cheng et al., 2025) converts AR models via full-model training on $\sim$50B tokens for block-parallel generation, and NBDiff (Tian et al., 2025) extends this with causal prefix constraints. TiDAR (Liu et al., 2025b) proposes a sequence-level hybrid that drafts via diffusion and verifies autoregressively.

Unlike these approaches, I-DLM combines strict causal attention with logit-shifted prediction from MASK positions—by maximally respecting the pretrained AR model's attention and prediction patterns, our conversion is far more data-efficient and closes the quality gap to the base AR model.

Speculative decoding and multi-token prediction for LLMs. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) accelerates AR models by drafting tokens with a fast model and verifying them in parallel, provably preserving the target distribution. This guarantee relies on the target model having a well-trained verify distribution p—a property that AR models possess inherently but standard DLLMs lack due to mask-only training. Extensions include Medusa (Cai et al., 2024) (multiple prediction heads), the EAGLE family (Li et al., 2024; 2025) (feature-level drafting), and SpecInfer (Miao et al., 2023) (tree-structured verification). Multi-token prediction (MTP) trains models to predict multiple future tokens simultaneously (Gloeckle et al., 2024). Samragh et al. (2025) proposed gated sparse expert LoRA for MTP—the closest prior work to our conditional LoRA, though designed for multi-head prediction rather than diffusion verification. Consistency LLMs (Kou et al., 2024) achieve acceleration via Jacobi iteration. ISD differs from these by using a single model with training-time stride capability to both achieve quality guarantee and deliver high latency and throughput gains under high concurrency.

DLLM-specific decoding. FastDLLM (Wu et al., 2025) enables confidence-aware parallel decoding with KV cache reuse. Jacobi Forcing (Hu et al., 2025) distills diffusion models for fewer-step convergence but degrades at larger batch sizes. Free Draft-and-Verification (Wu & Zhang, 2025) explores self-speculative approaches. WeDLM (Liu et al., 2025a) reconciles diffusion with causal attention for KV cache reuse. These methods rely on confidence-based acceptance or iterative denoising, which either lacks formal quality guarantees or incurs high compute overhead; our ISD provides provable AR-quality output in a single stride–introspection cycle.

Despite these advances, fundamental gaps remain between DLLMs and AR models in training scalability, inference alignment, compute efficiency, and infrastructure compatibility. We analyze these gaps quantitatively in the next section.

B TPF and Compute Overhead Analysis

We analyze the theoretical tokens per forward pass (TPF) and compute overhead for three parallel decoding paradigms: ISD (ours), block diffusion (SDAR), and branched self-speculative decoding (TiDAR). Throughout, N is the block/stride size and p is the uniform per-token acceptance probability ($P_k = p^k$). Compute overhead (OH) is the ratio of total query tokens to output tokens; AR has OH $= 1$ by definition.

B.1 ISD (Ours)

ISD alternates between an *NP step* (propose only: append N masks, produce 1 free token via logit shift + $N-1$ speculative proposals, finalize nothing) and *P steps* (introspect previous proposals + propose new ones). A renewal cycle is one NP step followed by a geometric chain of P steps ending on the first rejection.

Each P step has probability p^{N-1} of all-pass (finalizing N tokens, continuing the chain) and $1 - p^{N-1}$ of rejection. On rejection at position k, we finalize $k+1$ tokens (1 free + $k-1$ accepted + 1 resampled).

TPF. Expected tokens and forwards per cycle:

$$\mathbb{E}[\text{tokens}] = \frac{2 + p + \cdots + p^{N-2}}{1 - p^{N-1}}, \qquad \mathbb{E}[\text{forwards}] = \frac{2 - p^{N-1}}{1 - p^{N-1}}. \tag{4}$$

$$\text{TPF}_{\text{ISD}} = \frac{2 + p + p^2 + \cdots + p^{N-2}}{2 - p^{N-1}}. \tag{5}$$

At $p = 1$: TPF $= N$; at $p = 0$: TPF $= 1$ (AR).

Overhead. The P step processes $2N-1$ query tokens ($N-1$ filled proposals + N masks). The NP step processes N queries (variable) or $2N-1$ queries (fixed, padded to match P).

Variable query:

$$\text{OH}_{\text{var}} = \frac{3N - 1 - Np^{N-1}}{2 + p + \cdots + p^{N-2}}. \tag{6}$$

Fixed query (SGLang/vLLM, both steps use $2N-1$):

$$\text{OH}_{\text{fix}} = \frac{(2N-1)(2 - p^{N-1})}{2 + p + \cdots + p^{N-2}}. \tag{7}$$

The gap vanishes as $p \to 1$ and is irrelevant in the memory-bound regime where wall-clock speedup $\approx$ TPF.

B.2 Block Diffusion (SDAR)

SDAR generates a block of N tokens via iterative denoising with a force schedule, then commits the KV cache in a separate forward pass. At each denoising step, $H \sim \text{Binomial}(R, p)$ tokens pass the confidence threshold out of R remaining; at least $\max(H, 1)$ are committed. Let $\mathbb{E}[S \mid N]$ be the expected denoising steps for N tokens (computed recursively). Total forwards = $\mathbb{E}[S \mid N] + 1$ (denoising + KV commit), each processing all N tokens.

$$\text{TPF}_{\text{SDAR}} = \frac{N}{\mathbb{E}[S \mid N] + 1}, \qquad \text{OH}_{\text{SDAR}} = \mathbb{E}[S \mid N] + 1. \tag{8}$$

Note TPF $\times$ OH $= N$ always. Even at $p = 1$ (1 denoising step + 1 KV commit = 2 forwards), $\text{TPF}_{\text{SDAR}} \leq N/2$—the mandatory KV commit caps throughput at half the theoretical maximum.

B.3 Branched Self-Speculative Decoding (TiDAR)

TiDAR uses a single forward to both verify N draft tokens and pre-draft N branches of N masks each, covering all possible rejection outcomes. The input is N verify tokens + N^2 branch masks = $N(N{+}1)$ queries per forward, always exactly 1 forward per cycle.

$$\text{TPF}_{\text{TiDAR}} = 1 + p + p^2 + \cdots + p^{N-1} = \frac{1-p^N}{1-p}. \tag{9}$$

$$\text{OH}_{\text{TiDAR}} = \frac{N(N{+}1)}{1 + p + \cdots + p^{N-1}}. \tag{10}$$

TiDAR achieves the highest TPF (no NP recovery step), but the N^2 branch masks are structurally wasteful: only 1 of N branches is selected per forward, so $(N{-}1) \cdot N$ mask queries are always discarded. Even at $p = 1$, OH $= N{+}1$ and efficiency $= N/(N{+}1) < 1$—TiDAR can never be FLOP-efficient.

B.4 Summary

Table 6: TPF and overhead formulas (uniform p, stride N).

Method	TPF	Overhead	Queries/fwd
ISD (var.)	$\frac{2+p+\cdots+p^{N-2}}{2-p^{N-1}}$	$\frac{3N-1-Np^{N-1}}{2+p+\cdots+p^{N-2}}$	N or $2N{-}1$
ISD (fix.)	(same)	$\frac{(2N-1)(2-p^{N-1})}{2+p+\cdots+p^{N-2}}$	$2N{-}1$
SDAR	$\frac{N}{\mathbb{E}[S\|N]+1}$	$\mathbb{E}[S\|N] + 1$	N
TiDAR	$\frac{1-p^N}{1-p}$	$\frac{N(N+1)(1-p)}{1-p^N}$	$N(N{+}1)$

B.5 Compute Efficiency

We define *compute efficiency* as TPF/OH. Since all methods use more queries per forward than AR (OH > 1), they always consume more total FLOPs. Efficiency measures whether the extra computation is worthwhile: efficiency > 1 means the TPF gain outweighs the overhead cost (each additional FLOP produces more than one additional token), while efficiency < 1 means the overhead grows faster than the speedup.

Figure 6 plots compute efficiency against acceptance rate p for $N{=}4$. ISD is the only method that crosses the break-even line: the variable-query regime reaches efficiency > 1 at $p \approx 0.83$, and fixed-query at $p \approx 0.86$. At our empirically observed acceptance rates ($p \geq 0.85$), ISD achieves 1.08–2.29$\times$ efficiency, meaning the TPF speedup more than compensates for the extra compute per forward.

In contrast, TiDAR's efficiency is capped at $N/(N{+}1) = 0.80$ even at $p = 1$, due to the N^2 branch masks of which only N are ever used. SDAR reaches efficiency $= 1$ only at $p = 1$ (the degenerate case of 1 denoising step), and is below 0.72 at practical acceptance rates. This distinction is critical for throughput-sensitive batch serving, where compute per token determines cost.

C Why Block Diffusion Requires a Separate KV Commit Pass

Our SDAR inference uses SGLang's native DLLM support[1]. In this section we explain why block diffusion methods (SDAR, LLaDA) require a mandatory KV commit forward pass after denoising, and why this overhead is difficult to eliminate.

[1] https://github.com/sgl-project/sglang/pull/19044

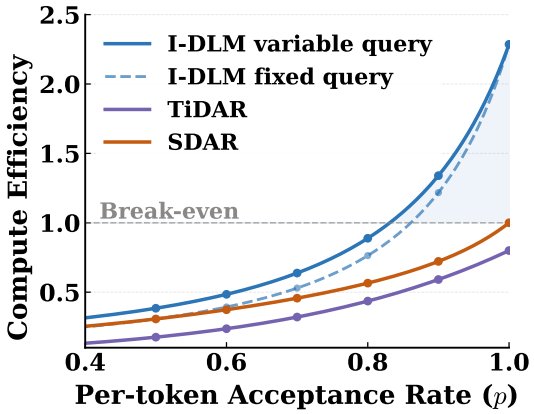


Figure 6: **Compute efficiency** (TPF / Overhead) vs. acceptance rate at stride N=4. Efficiency > 1 means the TPF gain outweighs the extra compute cost. ISD is the only method above the break-even line at practical acceptance rates.

After T denoising steps produce N final tokens, a separate forward pass with `store_kv=True` must write these tokens into the KV cache so that subsequent blocks can attend to them. This pass produces zero new tokens and is mandatory even at p=1, capping TPF at $N/2$.

A natural optimization would fuse commit with the next block's denoising in one forward: $[\underbrace{t_1, \ldots, t_N}_{\text{commit}}, \underbrace{\texttt{M}, \ldots, \texttt{M}}_{\text{denoise}}]$. However, this requires *per-position mixed attention masks*: committed tokens need bidirectional attention among themselves, while MASK tokens need block-causal attention. Current attention kernels (FlashAttention, FlashInfer) support only a single global `causal` flag and cannot mix bidirectional and causal attention within one forward pass. Implementing this would require custom attention kernels, doubled query size ($2N$), and complex batching logic for concurrent requests at different stages. SGLang's codebase confirms this: `dllm_is_commit` and `dllm_needs_commit` flags are defined but never activated—the optimization was considered but abandoned.

Because I-DLM uses strict causal attention throughout, no commit pass is needed. Each ISD step is a standard extend operation that incrementally commits KV entries as in AR decoding—no mixed masks, no custom kernels.

D Attention Kernel Overhead

Block diffusion uses block-causal attention, which differs from the strict token-level causal mask that AR serving kernels are optimized for. In SGLang, the standard extend path uses a cascade of three attention kernels per layer: (1) ragged attention among new tokens, (2) paged attention against the cached prefix, and (3) a merge kernel to combine the two via log-sum-exp renormalization. This cascade is optimized for large prefills but wasteful for DLLM decode steps that process only N=4–5 tokens, where the ragged kernel's advantage vanishes and the $3\times$ kernel launch overhead dominates.

Because I-DLM uses strict causal attention with small extended sizes, it can bypass the cascade and use a single paged attention kernel per layer, reducing kernel launches from $3L$ to L (where L is the number of layers). Figure 7 shows the resulting forward latency comparison across concurrency levels. At low concurrency (C=1), the cascade overhead is modest (+4%), but it grows to +20% at C=64 as the additional kernel launch overhead compounds with batching.

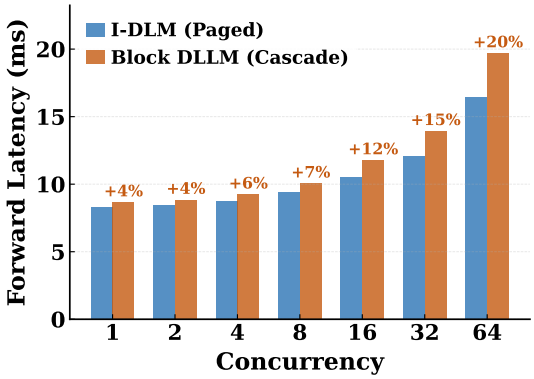


Figure 7: **Attention kernel forward latency** at varying concurrency. I-DLM (Paged, single kernel) vs. Block DLLM (Cascade, three kernels). The cascade overhead grows from +4% at C=1 to +20% at C=64.

E Attention Mask Structure

Figure 8 visualizes the attention mask used during I-DLM training, contrasted with SDAR's block-causal mask. The input sequence is the concatenation $[x_t \mid x_0]$, where x_t is the all-masked (noisy) region and x_0 is the clean reference. The mask is composed of three components:

- **Noisy self-attention (M_{noisy}):** Self-attention within the noisy region x_t. In our setting (`use_regular_causal=True`), this is *causal within each block*: position j in block b attends only to positions $i \leq j$ in the same block. SDAR instead uses bidirectional attention within blocks.
- **Cross-attention (M_{cross}):** Cross-attention from noisy tokens to clean tokens. Each noisy block b attends to clean tokens from all *preceding* blocks ($b' < b$), providing conditional context from the clean reference.
- **Clean self-attention (M_{clean}):** Self-attention within the clean region x_0. In our setting, this is *strict token-level causal* ($q \geq kv$), preserving the AR attention pattern exactly. SDAR instead uses block-causal attention here.

The final mask is $M = M_{\text{noisy}} \cup M_{\text{cross}} \cup M_{\text{clean}}$.

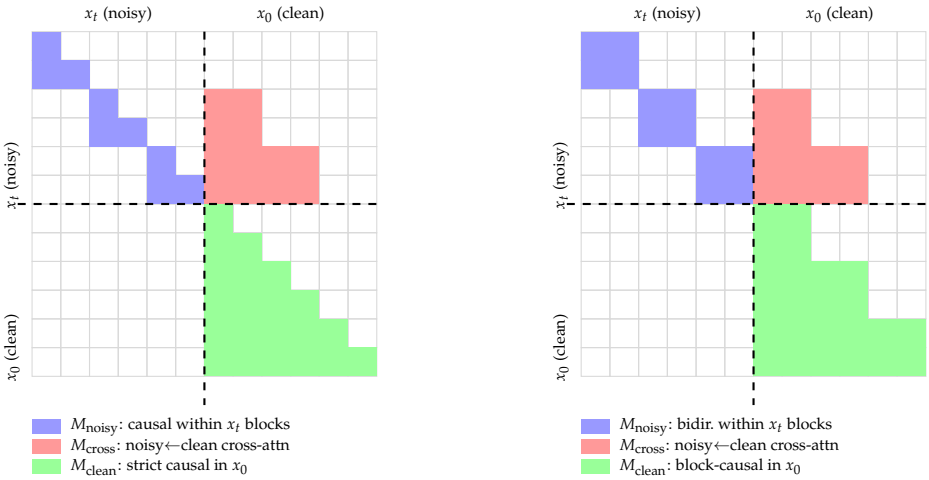
F ISD Step-by-Step Illustration

Figure 9 provides a detailed step-by-step illustration of Introspective Strided Decoding at stride N=3, showing both the all-accept and rejection cases.

G Lossless ISD with Gated LoRA

Figure 10 illustrates the gated LoRA mechanism used in Residual ISD (R-ISD). During each forward pass, token positions are partitioned into two types based on their input: `[MASK]` positions (proposals) activate the LoRA residual, while clean positions (introspection) use base-model-only weights. Because of strict causal attention, introspection positions cannot attend to any `[MASK]` position—their KV cache entries are computed entirely from base weights over clean tokens.

Per-position computation. Table 7 summarizes the computation at each position type. The gated LoRA mechanism applies a per-token binary mask $\mathbf{m}$: each linear layer computes $h_j \leftarrow Wx_j + \mathbf{1}_{[j \in \texttt{M}]} \cdot BA\, x_j$, where $\mathbf{1}_{[j \in \texttt{M}]}$ is 1 for `[MASK]` positions and 0 otherwise.



(a) **I-DLM (Ours).** Strict causal within x_t blocks and strict causal in x_0.

(b) **SDAR (Block Diffusion).** Bidirectional within x_t blocks and block-causal in x_0.

Figure 8: **Attention mask comparison** for block size N=2, sequence length L=6. Input is $[x_t \mid x_0]$ (noisy $\|$ clean). Rows are query positions; columns are key positions. Our I-DLM (left) uses strict causal attention everywhere, preserving AR compatibility. SDAR (right) uses bidirectional attention within noisy blocks and block-causal attention in the clean region. The three mask components—M_{noisy} (noisy self-attention), M_{cross} (noisy←clean cross-attention), M_{clean} (clean self-attention)—are color-coded.

Table 7: **Per-position breakdown in R-ISD.** Introspect and exact positions use base-only weights, producing the AR-identical causal anchor p. Propose positions add the LoRA residual to produce aligned proposals q.

Position	Weights	Output	Role
$\hat{x}_k$ (introspect)	Wx (base only)	p_k (causal anchor)	Produces p for p/q acceptance
x_1 (exact)	Wx (base only)	p (AR-identical)	Quality-guaranteed token
$\texttt{[M]}_k$ (propose)	$Wx + BAx$ (base+LoRA)	q_k (proposal)	Strided proposal aligned with p

The overall R-ISD pipeline can be expressed compactly as:

$$\underbrace{x_{i+1}}_{\substack{\text{from } h[c_i] \\ \text{(base only)}}} \;,\; \underbrace{\hat{x}_{i+2}, \ldots, \hat{x}_{i+N}}_{\substack{\text{from } h[\texttt{M}_{1:N-1}] \\ \text{(base+LoRA)}}} = \pi_\theta(c_{1:i}, \texttt{M}_{1:N-1}) \tag{11}$$

where π_θ denotes the model with gated LoRA: $W \to W + \mathbf{m} \odot BA$. With LoRA active on `[MASK]` positions, all proposals become higher quality, increasing the introspective acceptance rate while the causal anchor p remains the exact base AR distribution.

H Discussion about introspective-consistency training

The central insight of this work is that the success of AR models is not solely due to next-token prediction, but rather due to a deeper structural property: AR training unifies generation and introspection in one forward pass. This gives AR models an intrinsic form of self-verification. Existing diffusion language models miss this property.

Under standard block-causal masking, supervision is applied only to noisy tokens, while clean tokens are left unused as training targets. This is inefficient in computation, but more critically, it teaches the model to denoise without teaching it to introspect. Simply adding extra causal supervision does not solve the problem, because block-causal decoding and next-token prediction still remain disjoint objectives. The fact that prior methods such as

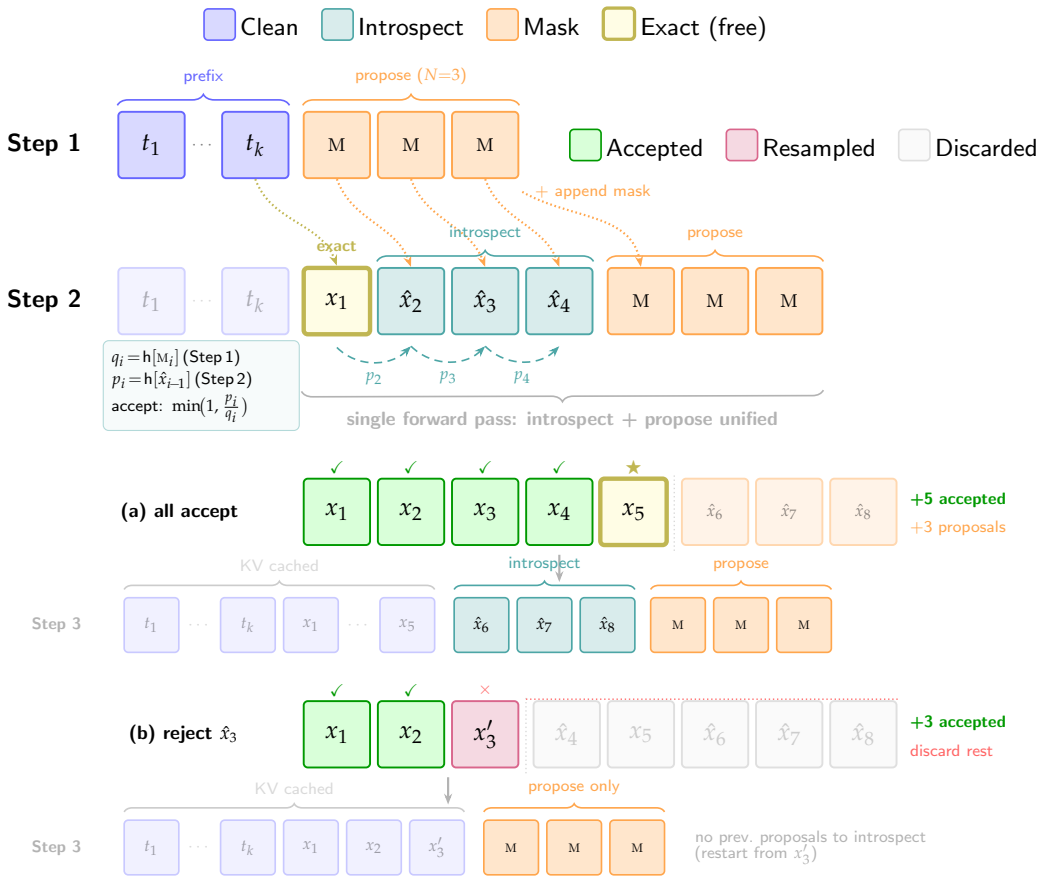


Figure 9: **Detailed ISD illustration at stride N=3. Step 1**: Bootstrap—append 3 [MASK] tokens, producing x_1 (exact) and proposals $\hat{x}_2, \hat{x}_3, \hat{x}_4$. **Step 2**: Single forward pass that introspects on previous proposals (computing causal anchors p_k) while generating new proposals. **(a) All accept**: 4 tokens accepted + bonus x_5; Step 3 introspects on new proposals. **(b) Reject** $\hat{x}_3$: x_1, x_2 accepted, x'_3 resampled, rest discarded; Step 3 is a pure propose step.

NBDiff require causal prefilling followed by block-causal decoding further reveals this gap: generation and introspection are still separated rather than unified.

Our method is built around resolving exactly this mismatch. We introduce a simple masking strategy that explicitly trains the model to generate and introspect within the same forward pass, thereby importing the core structural advantage of AR into diffusion-style language modeling. This design preserves what is most elegant about AR—joint supervision, objective–inference alignment, and self-consistency—while retaining the parallel decoding benefits of diffusion models. Just as importantly, it avoids the systems complications caused by switching between masks at inference time, leading to a cleaner foundation for both modeling and serving.

I Training, Serving, and Evaluation Details

I.1 Training Setup

Base models and data. We train two I-DLM variants: **I-DLM-8B** and **I-DLM-32B**, converted from Qwen3-8B and Qwen3-32B (Yang et al., 2025), respectively, using the introspective-consistency training recipe described in Section 3.1. Our training codebase builds on the open-source SDAR framework(https://github.com/JetAstra/SDAR). The training data

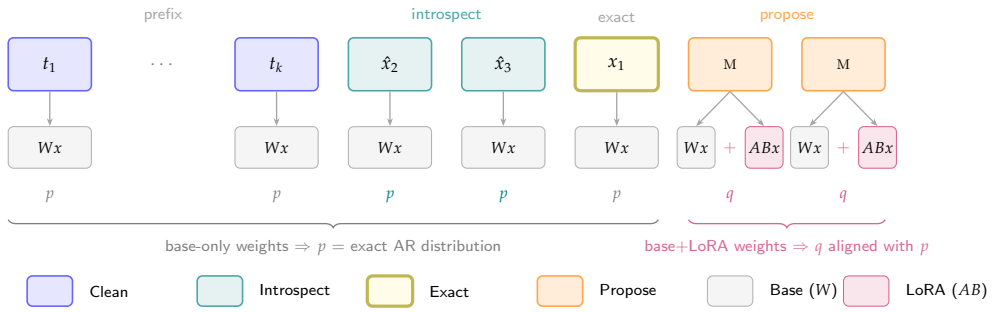


Figure 10: **Gated LoRA in Residual ISD (R-ISD).** During a single forward pass, [MASK] (propose) positions compute $Wx + ABx$ using base+LoRA weights, producing proposal distributions q. Clean and introspect positions compute Wx using base-only weights, producing the causal anchor distribution p—identical to a pure base AR forward pass. Because of causal attention, introspection positions never attend to [MASK] positions, so p is computed entirely from base-only KV entries. This makes the output bit-for-bit lossless with respect to the base AR model.

consists of **4.5B tokens** of responses generated across reasoning datasets. All training is performed on **8 H100 GPUs**.

Training schedule. For I-DLM-8B, we train for **2 epochs** with a stride curriculum: the first epoch trains with stride N=2, and the second epoch trains with stride N=3. During the first two stride expansions (N=1 $\rightarrow$ 2 and N=2 $\rightarrow$ 3), we use a fixed low scale of 0.2 for the cross-entropy loss on clean tokens to speed up masked token learning. Starting from the N=3 $\rightarrow$ 4 expansion, we switch to the auto-balanced loss scaling described in Eq. 2. For I-DLM-32B, we train for **1 epoch** with stride N=2 with LoRA using rank 1024 and a fixed scale of 0.2. We use N=4 for evaluation without additional full training as well.

Hyperparameters. We use full fine-tuning with DeepSpeed ZeRO Stage 2. The learning rate is 1×10^{-5} with cosine decay and a warmup ratio of 0.03. We use a per-device batch size of 1 with gradient accumulation steps of 4, yielding an effective batch size of 32 across 8 GPUs. The maximum sequence length is 4096 tokens. Training uses bf16 mixed precision.

LoRA training. For lossless ISD (R-ISD), we additionally train LoRA adapters with **rank 128** on the same data using the same hyperparameters with a learning rate of 2e-4. The inference follows the gated residual design described in Section 3.2.

I.2 Evaluation Setup

Baselines. We compare against two classes of state-of-the-art methods. *(i) Diffusion LLMs:* LLaDA-2.1-mini (16B)(Bie et al., 2026), LLaDA-2.0-flash (100B)(Bie et al., 2025), LLaDA-2.1-flash (100B)(Bie et al., 2026), SDAR (8B, 30B-A3B)(Cheng et al., 2025), NBDiff (7B)(Tian et al., 2025), DREAM (7B)Ye et al. (2025), WeDLM (8B)Liu et al. (2025a), LightningRL (8B), TiDAR (8B), Jacobi Forcing (7B)(Hu et al., 2026), Fast-dLLM (7B)(Wu et al., 2025), Mercury Coder Small(Labs et al., 2025), and Gemini Diffusion; and *(ii) Speculative decoding:* EAGLE-3 (Li et al., 2025). We use our I-DLM AR counterparts, Qwen3-8B and Qwen3-32B, as baselines.

Benchmarks. We evaluate on 14 benchmarks covering four domains: (i) *Knowledge and Reasoning*: ARC-C(Clark et al., 2018), MMLU(Hendrycks et al., 2020), MMLU-Pro(Wang et al., 2024), GPQA-D(Rein et al., 2024), GPQA(Rein et al., 2024); (ii) *Math Reasoning*: GSM8K(Cobbe et al., 2021), MATH-500(Hendrycks et al., 2021), MathBench(Liu et al., 2024b), AIME-24(AIME), AIME-25(AIME); (iii) *Code Generation*: HumanEval(Chen et al., 2021), MBPP(Odena et al., 2021), LiveCodeBench-v6(Jain et al., 2024); and (iv) *Instruction Following*: IFEval(Zhou et al., 2023). All with thinking mode enabled.

Metrics. For quality, we report accuracy on each benchmark, where baseline results are taken from their original papers, where available, explicitly representing their best performance;

for LLaDA-2.1-mini, we additionally run missing benchmarks using the best configurations reported in the paper. For efficiency, we report *request-level tokens per second* (latency) and *server-level tokens per second* (throughput) under varying concurrency levels.

I.3 Hardware

All experiments are conducted on NVIDIA H100 80GB SXM GPUs with NVLink interconnect. We use CUDA 12.9 with FlashInfer for attention computation. CUDA graphs are enabled for all configurations.

I.4 Serving Configuration

Table 8 summarizes the serving configurations used across all models.

Table 8: **Serving configurations.** All models are served with SGLang on H100 GPUs.

Model	TP	#Servers	Dtype	DLLM Config	LoRA
8B models (1 GPU per server)					
Qwen3-8B (AR)	1	8	bf16	—	—
I-DLM-8B (N=3)	1	8	bf16	blockN3	—
I-DLM-8B (N=5)	1	8	bf16	blockN5	—
I-DLM-8B (Lossless)	1	8	bf16	blockN3+LoRA	r=128
EAGLE-3	1	8	bf16	—	—
32B models (2 GPUs per server)					
Qwen3-32B (AR)	2	4	bf16	—	—
I-DLM-32B (N=3)	2	4	bf16	blockN3	—
I-DLM-32B (Lossless)	2	4	bf16	blockN3+LoRA	r=1024

ISD algorithm configuration. Table 9 lists the ISD algorithm parameters used for each stride configuration.

Table 9: **ISD algorithm configurations.** "blockNk" denotes stride N=k with block_size = $2k-1$.

Config	N	block_size	temperature	top_k	top_p	spec_verify
blockN3	3	5	1.0	50	0.95	true
blockN5	5	9	1.0	50	0.95	true
blockN3+LoRA	3	5	1.0	50	0.95	true

Speculative decoding baselines. Table 10 details the configurations for speculative decoding baselines. All baselines use Qwen3-8B as the target model and are served with SGLang.

Table 10: **Speculative decoding baseline configurations.** "Steps" and "topk" are EAGLE-3 draft parameters

Method	Draft Model	Draft Params	Notes
EAGLE-3	Tengyunw/qwen3_8b_eagle3	steps=3, topk=1, d=4	3 specs verified per step

SGLang server parameters. Common parameters across all configurations: `mem-fraction-static=0.85`, `attention-backend=flashinfer`, `disable-radix-cache=true` (required for DLLM KV trim). CUDA graph capture is enabled by default; the DLLM extend forward (all $2N-1$ tokens) is captured into a single graph per batch size, with attention metadata updated via in-place tensor writes before each replay.

I.5 Evaluation Configuration

Table 11 details the evaluation settings for each benchmark.

Table 11: **Evaluation configurations per benchmark.** All benchmarks use the Qwen3 chat template with thinking mode enabled. "max_tokens" controls the maximum generation length including the `<think>` block.

Category	Benchmark	#Problems	max_tokens	Metric / Extraction
Knowledge	ARC-C	1,172	32,768	Accuracy; choice letter
	MMLU	14,042	32,768	Accuracy; ABCD extraction
	MMLU-Pro	12,032	32,768	Accuracy; ABCD extraction
	GPQA-Diamond	198	32,768	Accuracy; ABCD extraction
	GPQA	448	32,768	Accuracy; ABCD extraction
Math	GSM8K	1,319	32,768	Accuracy; `\boxed{}`
	MATH-500	500	32,768	Accuracy; `\boxed{}`
	MathBench	3,709	32,768	Accuracy; numerical
	AIME-24	30	32,768	Accuracy; `\boxed{}`
	AIME-25	30	32,768	Accuracy; `\boxed{}`
Code	HumanEval	164	32,768	pass@1; code execution
	MBPP (sanitized)	257	32,768	pass@1; code execution
	LCB-v6	175	32,768	pass@1; code execution
Instruction	IFEval	541	32,768	Prompt-strict accuracy

Sampling parameters. For quality evaluation, all models use temperature t=1.0, top-k=50, top-p=0.95 (matching the ISD algorithm configuration). For the AR baseline (Qwen3-8B/32B), the same sampling parameters are used to ensure a fair comparison. For each benchmark, results are averaged over three runs.

Answer extraction. For math benchmarks (GSM8K, MATH-500, AIME), we extract the final `\boxed{...}` answer after stripping the `<think>...</think>` block. For multiple-choice benchmarks (ARC-C, MMLU, MMLU-Pro, GPQA), we extract the answer letter using the pattern `ANSWER: [A-D]`. For code benchmarks (HumanEval, MBPP, LCB), we extract the last Python code block containing a function definition and execute it against the provided test cases using a sandboxed subprocess with a 10-second timeout. For IFEval, we use the official Google evaluator after stripping thinking blocks.

Baseline reproduction. Results for LLaDA-2.1-mini are reproduced using SGLang with the official configuration (`block_size=4`, `threshold=0.95`, `edit_threshold=0.9`). Results for SDAR are reproduced using SGLang's DLLM serving mode. Results for EAGLE-3 are reproduced using their respective SGLang integrations. All other baseline results are taken from their original publications.

I.6 Throughput Benchmark Configuration

For latency and throughput measurements (Section 4), we use genai-bench[2] with the following settings:

I.7 Infrastructure Ablation Configuration

The ablation study in Section 4.2 isolates the contribution of each serving optimization by disabling them individually via environment variables. Table 13 lists the ablation toggles.

The ablation is cumulative: starting from the naive configuration (all optimizations disabled), we add each optimization one at a time and measure throughput at C=1, C=8, and C=32 on a single H100.

[2] https://github.com/sgl-project/genai-bench

Table 12: **Throughput benchmark configuration.**

Parameter	Value
Concurrency levels	$C \in \{1, 2, 4, 8, 16, 32, 48, 64\}$
Request arrival	Burst mode (all requests submitted simultaneously)
Output length	2,048 tokens (fixed)
Measurement	Total tokens / wall-clock time (first to last completion)
Warmup	5 requests discarded before measurement
GPU	1× H100 (TP=1) for 8B; 2× H100 (TP=2) for 32B

Table 13: **Infrastructure ablation toggles.** Each toggle disables one optimization to measure its isolated contribution.

Toggle	Disables	Effect
`--disable-cuda-graph`	CUDA graph capture/replay	Forces eager model forward
`NO_DECODE_LOOP=1`	Stationary-batch decode loop	Falls back to full scheduler pipeline
`NO_ARGMAX_PROPOSALS=1`	Argmax for draft positions	Uses full sampling for specs
`NO_FUSED_VERIFY=1`	Fused Triton verify kernel	Uses 7+ separate kernel launches
`NO_DEFERRED_STREAM=1`	Deferred stream_output	Runs on critical path
`DLLM_USE_RAGGED=1`	Paged-only attention	Forces cascade (3 kernels/layer)

J Additional Results

J.1 Peak Throughput on Different Hardware

Table 14 reports peak per-request TPS under favorable conditions (low concurrency, long generation), using different model variants and stride configurations. The base I-DLM-8B is trained at N=3; the N=4 checkpoint is obtained via stride extension training from I-DLM-8B, and N=8 is further extended from N=4. The LoRA variant uses a rank-128 adapter with segment-gated conditional activation (R-ISD).

Table 14: **Peak per-request TPS** at various stride, hardware, and model configurations. All models are 8B scale.

Hardware	TP	N	Model Variant	Benchmark	TPS
B200	2	8	I-DLM-8B (ext. training N=8)	GSM8K	925
H100	2	8	I-DLM-8B (ext. training N=8)	MBPP	685
H100	1	4	I-DLM-8B (ext. training N=4)	MATH-500	341
H100	1	4	I-DLM-8B (ext. training N=4)	AIME	314
H100	1	4	I-DLM-8B (ext. training N=4)	ShareGPT	319
H100	1	3	I-DLM-8B	MATH-500	272
H100	1	3	I-DLM-8B R-ISD (LoRA r=128)	MATH-500	240

K ISD algorithm

K.1 Other system optimization

The verification step is fused into a single Triton kernel with online softmax and Gumbel-max correction; the common accept path (∼78% of positions) returns after one streaming pass, skipping correction entirely. Since ISD's p/q criterion guarantees output correctness regardless of proposal quality, we use argmax for proposals to maximize acceptance rate without affecting output diversity. For lossless R-ISD (Appendix G), we implement segment-gated LoRA where a per-token binary mask gates the LoRA residual within the CUDA graph, with cuBLAS replacing segmented GEMV at small token counts and a dedicated CUDA stream overlapping the LoRA shrink with base projections.

Algorithm 1 Introspective Strided Decoding (one iteration)

Require: Prefix tokens $x_{1:L}$, stride N, model M with logit shift
Ensure: Accepted new tokens appended to prefix

// Stride: $N-1$ masks produce N tokens (logit shift gives +1 free)
Construct input $[x_{1:L}, \texttt{MASK}_1, \ldots, \texttt{MASK}_{N-1}]$ ($N-1$ masks)
Run $M \to \text{logits}_{\text{stride}}[1{:}L+N-1]$
Sample $x_{L+1} \sim \text{softmax}(\text{logits}_{\text{stride}}[L])$ {quality-guaranteed (AR prediction)}
for $k = 2$ **to** N **do**
  Sample $x_{L+k} \sim q_k \triangleq \text{softmax}(\text{logits}_{\text{stride}}[L+k-1])$ {strided proposal}
end for
// Introspect: verify proposals against causal anchors
// (Only introspection shown; in practice fused with next proposal into one $2N-1$ token pass)
Construct input $[x_{1:L}, x_{L+1}, \ldots, x_{L+N}]$
Run $M \to \text{logits}_{\text{anchor}}[1{:}L+N]$
$p_k \leftarrow \text{softmax}(\text{logits}_{\text{anchor}}[L+k-1])$ for $k = 1, \ldots, N$ {causal anchor distributions}
// Introspection with adaptive stride
$n_{\text{accepted}} \leftarrow 1$ {x_{L+1} always accepted (quality-guaranteed)}
for $k = 2$ **to** N **do**
  Draw $r \sim \text{Uniform}(0, 1)$
  if $r < \min\big(1,\ p_k(x_{L+k}) \,/\, q_k(x_{L+k})\big)$ **then**
    Accept x_{L+k}; $n_{\text{accepted}} \leftarrow n_{\text{accepted}} + 1$
  else
    Resample $x_{L+k} \sim \text{normalize}\big(\max(0,\ p_k - q_k)\big)$
    $n_{\text{accepted}} \leftarrow n_{\text{accepted}} + 1$; **break**
  end if
end for
// Bonus token if all proposals accepted
if $n_{\text{accepted}} = N$ **then**
  Sample $x_{L+N+1} \sim \text{softmax}(\text{logits}_{\text{anchor}}[L+N])$
  $n_{\text{accepted}} \leftarrow n_{\text{accepted}} + 1$
end if

Append $x_{L+1:L+n_{\text{accepted}}}$ to prefix; commit KV cache

Table 15: **Throughput comparison with AR parallel-generation methods on MATH-500.** We report end-to-end generation throughput (tokens/s) under different request concurrency levels.

Concurrency	Qwen3-8B	I-DLM-8B	Trained MTP	Lookahead Decoding
1	142	**341**	242	186
2	132	**335**	232	217
4	130	**319**	222	198
8	119	**292**	208	176
16	123	**252**	184	129
32	111	**199**	157	81

L Additional ablations

L.1 Additional Efficiency Evaluation

Comparison with AR parallel-generation methods. We further compare I-DLM with representative AR-side parallel-generation methods, including trained multi-token prediction (MTP) (Liu et al., 2024a) and training-free lookahead decoding (Fu et al., 2024). As shown in Tables 15 and 16, I-DLM consistently achieves higher end-to-end throughput across all concurrency levels. On MATH-500, I-DLM achieves 341 and 199 tokens/s at concurrency 1 and 32, respectively, compared with 242/157 for trained MTP and 186/81 for lookahead decoding; similar trends hold on MBPP. These results show that I-DLM's parallel generation translates into practical serving gains beyond representative AR-side acceleration methods.

Table 16: **Throughput comparison with AR parallel-generation methods on MBPP.** We report end-to-end generation throughput (tokens/s) under different request concurrency levels.

Concurrency	Qwen3-8B	I-DLM-8B	Trained MTP	Lookahead Decoding
1	141	**327**	228	192
2	134	**320**	220	215
4	129	**308**	205	204
8	125	**285**	193	173
16	126	**246**	163	121
32	112	**194**	134	75

Table 17: **Robustness to long generation lengths.** We report average tokens per forward pass (TPF) across different generated sequence lengths on mathematical reasoning and code generation tasks.

Generation Length	AIME 2024/2025	LiveCodeBench v6
0–4K	2.91	2.63
4–8K	2.81	2.61
8–16K	2.73	2.50
15–24K	2.61	2.51
24–28K	2.58	2.53

Robustness to long sequences. We further examine whether the parallelism of I-DLM degrades with increasing generation or context length. As shown in Table 17, TPF decreases only gradually from 2.91 to 2.58 on AIME 2024/2025 as generation length increases to 28K tokens, while remaining between 2.50 and 2.63 on LiveCodeBench v6. Moreover, Table 18 shows that I-DLM maintains TPF between 2.13 and 2.42 with contexts up to 64K tokens while preserving 100% retrieval accuracy. These results demonstrate that I-DLM retains substantial decoding parallelism under both long generations and long input contexts.

Table 18: **Robustness to long input contexts.** We report TPF and retrieval accuracy on the 8-needle retrieval task across context lengths up to 64K tokens.

Context Length	TPF	Accuracy (%)
0–8K	2.42	100
8–16K	2.20	100
16–32K	2.28	100
32–48K	2.13	100
48–64K	2.32	100