

Multi-Block Diffusion Language Models

Yijie Jin¹, Jiajun Xu², Yuxuan Liu¹, Chenkai Xu¹
Jiajun Li³, Kai Yu¹, Pengfei Liu¹, Zhijie Deng^{1,†}

¹Shanghai Jiao Tong University ²Xi'an Jiaotong University ³Huawei
{drewjin0827, zhijied}@sjtu.edu.cn

[†]Corresponding author.

Abstract

Block Diffusion Language Models (BD-LMs) support KV caching and flexible-length generation, but their native Single-Block Diffusion (SingleBD) inference leaves inter-block parallelism unused. We study Multi-Block Diffusion (MultiBD), where a *running-set* of consecutive blocks is refined concurrently. Existing teacher-forced BD-LMs observe only one noisy block conditioned on a clean prefix, while diffusion forcing exposes a long noisy suffix whose state distribution still differs from practical MultiBD inference. We bridge this gap with *Multi-Block Diffusion Language Models* (MBD-LMs), obtained by post-training BD-LMs using *Multi-block Teacher Forcing* (MultiTF). MultiTF trains on bounded *noise-groups* conditioned on clean prefixes, with heterogeneous slot-wise noise patterns that better match inference. An optimized *Block Buffer* further preserves prefix-cache reuse and static input shapes, translating inter-block parallelism into wall-clock acceleration. On math and code benchmarks, MBD-LLaDA2-Mini increases average Tokens Per Forward pass (TPF) from 3.47 to 6.19 while maintaining comparable accuracy (80.0% versus 81.0%). Combined with DMax, it reaches 9.34 TPF with a 1.02-point accuracy drop. Project Page: <https://sjtu-deng-lab.github.io/mbd-lms>

1 Introduction

Diffusion Language Models (DLMs) enable native parallel token refinement (Sahoo et al., 2024; Nie et al., 2025), but fully bidirectional DLMs do not naturally support KV caching or flexible-length generation. Block Diffusion Language Models (BD-LMs) address both limitations with block-causal generation (Arriola et al., 2025; Bie et al., 2025; Cheng et al., 2025). Their native *Single-Block Diffusion* (SingleBD) decodes one noisy block conditioned on a clean cached prefix,

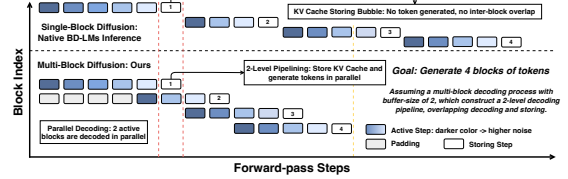


Figure 1: SingleBD processes blocks sequentially and creates KV-cache storing bubbles. MultiBD refines future blocks while completed blocks enter the cache, exposing inter-block parallelism.

preserving intra-block parallelism and KV reuse. However, blocks are decoded sequentially: each must finish and enter the cache before the next starts, leaving the inter-block parallelism in Figure 1 unused.

Discrete Diffusion Forcing (D2F) (Wang et al., 2025) exposes multiple noisy suffix blocks during training. Conditioned on a clean prefix, it assigns monotonically increasing noise ratios to the suffix, enabling MultiBD-style decode-store overlap and inter-block parallelism. Yet a train-inference mismatch remains: processing the entire suffix as one *running-set* is inefficient and empirically ineffective, whereas measured traces maintain a small set (typically two active blocks) with large adjacent noise gaps (Lu et al., 2026). Reliable MultiBD therefore needs training states matching bounded set size and heterogeneous slot-wise noise.

We formulate *Multi-Block Diffusion Language Models* (MBD-LMs) around this bounded *running-set*. The formulation contains TF-trained SingleBD and D2F-style full-suffix states as extremes, while identifying practical MultiBD as the intermediate regime. We then introduce *Multi-block Teacher Forcing* (MultiTF), which post-trains a BD-LM on bounded groups of consecutive noisy blocks. A *Group-Aware Dual-Stream Mask* exposes within-group context while retaining clean earlier groups, and

systematic/random layouts plus a chain-uniform *noise-scheduler* reproduce inference-like set sizes and slot-wise noise gaps.

To serve MBD-LMs efficiently, we introduce a fixed-shape *Block Buffer*. Idle slots activate in place, and completed front blocks enter the KV cache before the buffer slides. This preserves prefix-cache reuse and CUDA Graph replay while decoding overlaps cache writes.

Across math and code benchmarks, MBD-LLaDA2-Mini raises average TPF from 3.47 to 6.19 (+78.4%) at comparable accuracy (80.0% versus 81.0%). With DMax (Chen et al., 2026b), it reaches 9.34 TPF (+47.1%) with a 1.02-point accuracy drop and, on two H100 GPUs, raises throughput from 779 to 927 TPS.

Our contributions are threefold:

- an MBD-LM formulation that unifies SingleBD, full-suffix diffusion forcing, and bounded MultiBD through the *running-set*;
- MultiTF post-training that aligns bounded *noise-groups*, slot-wise noise, and attention states with MultiBD inference; and
- a *Block Buffer* based engine that preserves KV/prefix caching and static-shape execution, converting TPF gains into measured TPS gains.

2 Preliminaries

2.1 Diffusion Language Models

A masked DLM (Sahoo et al., 2024; Nie et al., 2025; Ye et al., 2025) corrupts a clean sequence $\mathbf{x}_0 \in \mathcal{V}^L$ by independently replacing tokens with $[M]$ at rate $t \in [0, 1]$. The model reconstructs only masked positions with weighted cross-entropy

$$\mathcal{L}_{\text{DLM}}(\theta) = -\mathbb{E} \left[\frac{1}{t} \sum_{i \in \mathcal{M}(\mathbf{x}_t)} \log p_\theta(x_t^i | \mathbf{x}_t) \right], \quad (1)$$

where $\mathcal{M}(\mathbf{x}_t) = \{i : x_t^i = [M]\}$. Inference starts from an all-mask sequence and iteratively commits high-confidence predictions.

2.2 Block Diffusion Language Models

A BD-LM (Arriola et al., 2025; Bie et al., 2025) partitions $\mathbf{x}_0 = [\mathbf{b}_1, \dots, \mathbf{b}_K]$ into $K = L/B$ blocks of size B and factorizes

$$p_\theta(\mathbf{x}_0) = \prod_{k=1}^K p_\theta(\mathbf{b}_k | \mathbf{x}_0^{(<k)}), \quad (2)$$

$$\mathbf{x}_0^{(<k)} = [\mathbf{b}_1, \dots, \mathbf{b}_{k-1}].$$

Block-causal attention lets every block attend to itself and clean predecessors, enabling KV caching.

Teacher forcing. We use *Teacher Forcing* (TF) as an operational term for the clean-prefix/one-noisy-block state used by standard BD-LMs, rather than as terminology introduced by one paper; Block Diffusion (Arriola et al., 2025) is a representative instance. Only $\mathbf{b}_k$ is corrupted, and masked tokens are predicted from $\mathbf{x}_0^{(<k)}$ and $\mathbf{b}_{k,t}$:

$$\mathcal{L}_{\text{TF}} = -\mathbb{E} \left[\frac{1}{t} \sum_{i \in \mathcal{M}(\mathbf{b}_{k,t})} \log p_\theta(b_k^i | \mathbf{c}_{k,t}) \right], \quad (3)$$

$$\mathbf{c}_{k,t} = (\mathbf{x}_0^{(<k)}, \mathbf{b}_{k,t}).$$

Consequently, native SingleBD learns no state containing multiple noisy blocks.

Discrete diffusion forcing. D2F (Wang et al., 2025) instead corrupts a block-partitioned suffix with $0 \leq t_1 < \dots < t_K \leq 1$:

$$\mathbf{x}_t^{\text{suf}} = [\mathbf{b}_{1,t_1}, \dots, \mathbf{b}_{K,t_K}], \quad (4)$$

$$\mathbf{b}_{k,t_k} \sim q_{t_k}(\cdot | \mathbf{b}_k).$$

Conditioned on a clean prefix $\mathbf{x}_0^{\text{pre}}$, let $\mathbf{h}_{k,t} = (\mathbf{x}_0^{\text{pre}}, \mathbf{b}_{1,t_1}, \dots, \mathbf{b}_{k,t_k})$. Its noisy-prefix factorization is defined as

$$p_\theta(\mathbf{x}_0^{\text{suf}} | \mathbf{h}_{K,t}) = \prod_{k=1}^K p_\theta(\mathbf{b}_k | \mathbf{h}_{k,t}). \quad (5)$$

D2F exposes multiple noisy blocks, but the long monotonic suffix differs from the bounded, heterogeneous states in Figure 2. Its arbitrary-length fully attended prefix also differs from BD-LM prefix-cache semantics; Appendix C.5 gives the controlled block-causal comparison.

3 Methodology

We first formulate MBD-LMs through their inference state, then align training to that state with MultiTF, and finally implement it with static *Block Buffer* mechanism.

3.1 Why MultiBD Requires Co-Design

Moving from SingleBD to MultiBD creates a coupled modeling-and-systems problem. On the modeling side, a TF-trained BD-LM has only learned one-noisy-block conditionals, whereas a MultiBD step asks it to resolve several blocks at different

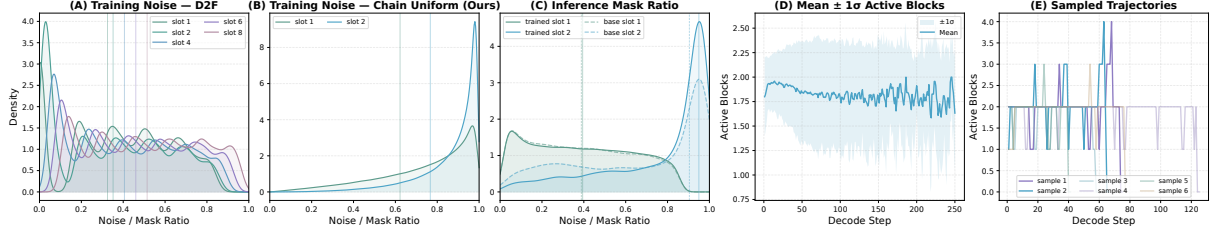


Figure 2: **Train-inference state analysis.** (A) A D2F-style schedule produces weak slot separation. (B) chain-uniform MultiTF creates heterogeneous slot-wise mask ratios. (C) these distributions better match MultiBD inference. (D–E) the active *running-set* is bounded, with mean size near two and occasional expansion.

Table 1: Properties of SingleBD, dynamic MultiBD, and our jointly trained and served MBD-LM design.

Property	SingleBD	Dynamic MultiBD	MBD-LM
Concurrent noisy blocks	no	yes	yes
Fixed physical query shape	yes	no	yes
Stable prefix-cache reuse	yes	fragile	yes
Matched multi-block states	no	partial	yes

stages of refinement. Exposing the full noisy suffix, as in D2F, moves the state in the opposite direction: the number of visible noisy blocks grows with sequence length, while a monotonic schedule weakly separates adjacent slots. The target is instead small, dynamic, and heterogeneous.

On the systems side, a literal implementation would append a block once the tail is clean and delete the cached front. Although correct, changing query length, attention layout, and cache offsets disrupts CUDA Graph replay and complicates shared-prefix caching, motivating the requirements in Table 1.

Our solution therefore fixes the logical inference target first, synthesizes that target during MultiTF post-training, and realizes it with a bounded physical buffer. The same *running-set* abstraction connects all three components: it defines which blocks coexist, determines which training states must be constructed, and specifies which slots the serving engine must maintain.

3.2 Multi-Block Diffusion Language Models

At step s , MultiBD maintains a consecutive *running-set* $\mathcal{R}_s = \{a_s, \dots, c_s\}$. Blocks before a_s have been committed and form the clean cached prefix $\mathbf{x}_0^{(<a_s)} = [\mathbf{b}_1, \dots, \mathbf{b}_{a_s-1}]$; later blocks have not entered decoding. The running-set contains both active noisy blocks and any completed preceding blocks waiting to enter the cache. For every $j \in \mathcal{R}_s$, let $t_{j,s} \in [0, 1]$ denote its current mask ratio and $\mathbf{b}_{j,t_{j,s}}$ its visible state. We set $t_{j,s} = 0$ once

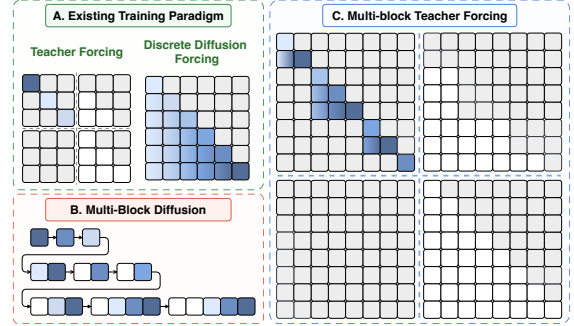


Figure 3: Training-state continuum. TF sees one noisy block; D2F exposes a long monotonic suffix; MultiTF constructs bounded *noise-groups* with heterogeneous slot-wise noise.

a resident block is complete, so the same notation covers noisy and ready-to-cache blocks. Each relative position in $\mathcal{R}_s$ is a *logical slot*.

An MBD-LM factorizes a MultiBD step as

$$\begin{aligned}
 p_{\theta}(\mathbf{b}_{\mathcal{R}_s} \mid \mathbf{x}_0^{(<a_s)}, \mathbf{b}_{\mathcal{R}_s, t_s}) \\
 = \prod_{j=a_s}^{c_s} p_{\theta}(\mathbf{b}_j \mid \mathbf{x}_0^{(<a_s)}, \mathbf{b}_{a_s, t_{a_s, s}}, \dots, \mathbf{b}_{j, t_{j, s}}),
 \end{aligned} \tag{6}$$

where $\mathbf{b}_{\mathcal{R}_s}$ denotes the clean target blocks and $\mathbf{b}_{\mathcal{R}_s, t_s}$ their current visible states. Thus each block attends to the cached prefix, itself, and causally preceding slots, but never to a future slot. When the front block stabilizes, it is committed to the KV cache and removed; a future block may then enter at the back.

Figure 3 unifies three regimes. SingleBD is the $|\mathcal{R}_s| = 1$ extreme. A full suffix with monotonic D2F-style noise resembles the opposite extreme, although D2F is a training construction rather than an executable serving state. Practical MultiBD occupies the bounded intermediate regime: $|\mathcal{R}_s| > 1$ exposes inter-block parallelism, but a small upper bound limits per-step work.

Inference-state invariants. A valid MultiBD state satisfies three properties. *Contiguity* requires the blocks in the *running-set* to be consecutive, so a completed front block can enter the KV cache without skipping any prefix block. *Causal visibility* allows each block to attend only to the clean cached prefix, itself, and preceding blocks in the *running-set*, thereby preserving the block-causal factorization. *Boundedness* limits the number of resident blocks, keeping per-step computation controlled while allowing future blocks to be refined before the front block is committed. Accordingly, training must match practical MultiBD states in both *running-set* size and slot-wise noise, rather than simply exposing multiple noisy blocks.

3.3 Multi-block Teacher Forcing

MultiTF approximates the bounded *running-set* states encountered during MultiBD inference by constructing corresponding training-side *noise-groups*. Let $G_{\max}$ denote the maximum noise-group size, and let a *group-layout* $\lambda = (H_1, \dots, H_{|\lambda|})$ partition the block sequence into consecutive noise-groups.

We first construct systematic layouts to provide deterministic coverage of group sizes and relative block positions. For each group size $g \in \{2, \dots, G_{\max}\}$ and boundary offset $h \in \{0, \dots, g-1\}$, the layout $\lambda_{g,h}$ places a group boundary every g blocks. Its q -th noise-group is

$$H_{g,h,q} = \{1+h+qg, \dots, h+(q+1)g\} \cap \{1, \dots, K\}. \quad (7)$$

The intersection handles groups truncated at the sequence boundaries. By enumerating all g offsets, every interior consecutive span of g blocks appears as a noise-group in exactly one systematic layout. Equivalently, each interior block is observed at every group-relative position across the g shifted layouts. This construction therefore covers all supported running-set sizes and alignments up to the upper bound $G_{\max}$.

Systematic layouts use a fixed group size within each layout and thus provide regular but limited boundary patterns. To increase training-state diversity, we additionally construct random layouts by sampling successive group sizes from $\{2, \dots, G_{\max}\}$ until the full block sequence is covered. These layouts introduce mixed group sizes and irregular boundaries while preserving the bounded noise-group structure. The final layout

set combines both components, $\Lambda = \Lambda_{\text{sys}} \cup \Lambda_{\text{rand}}$: systematic layouts provide coverage, whereas random layouts enrich the combinations of group sizes and boundary positions. Complete construction and coverage arguments are provided in Appendices B.1 and A.

Chain-uniform noise-scheduler. For a group $H_m = (j_1, \dots, j_n)$, define

$$\begin{aligned} t_{\text{eff}} &= t_{\text{high}} - \rho \Delta t, \\ \Delta t &= t_{\text{high}} - t_{\text{low}}. \end{aligned} \quad (8)$$

We sample a floor $\ell_0 \sim \mathcal{U}(t_{\text{low}}, t_{\text{eff}})$ and then

$$t_{j_i} \sim \mathcal{U}(\ell_{i-1}, t_{\text{eff}}), \quad \ell_i \leftarrow t_{j_i}. \quad (9)$$

This Markov chain is monotone within a group, yet its conditional intervals differ by slot and can produce large adjacent gaps. The distinction is important: monotonicity alone only orders the slots, whereas MultiBD inference exposes a nearly resolved leading block beside substantially noisier future blocks. Chain-uniform scheduling is designed to reproduce that heterogeneity, as visualized in Figure 2.

Group-Aware Dual-Stream Mask. For layout λ , MultiTF concatenates a corrupted view and a clean view, $\mathbf{X}_\lambda = [\mathbf{x}_t^\lambda; \mathbf{x}_0]$. A noisy query in group H_m attends to clean earlier groups and noisy blocks within H_m ; a clean query follows the original block-causal rule. Writing $g_\lambda(i)$ for the group index, $b(i)$ for the block index, and $\mathcal{N}, \mathcal{C}$ for noisy/clean streams, define $G_{ij} : g_\lambda(i) = g_\lambda(j)$ and $P_{ij} : b(j) \leq b(i)$.

$$A_\lambda(i, j) = \begin{cases} 1, & i \in \mathcal{N}, j \in \mathcal{C}, g_\lambda(j) < g_\lambda(i), \\ 1, & i, j \in \mathcal{N}, G_{ij}, P_{ij}, \\ 1, & i, j \in \mathcal{C}, P_{ij}, \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

Thus later groups are not leaked into the current group, while within-group noisy context matches MultiBD. The clean stream supplies the teacher-forced prefix for every group; the noisy stream supplies the inference-like running-set state. Because clean blocks from the current or later group are masked out, the construction cannot reveal the target through the duplicated clean sequence. Multiple groups can nevertheless contribute masked-token losses in one forward pass, making the layouts computationally useful rather than merely data augmentation.

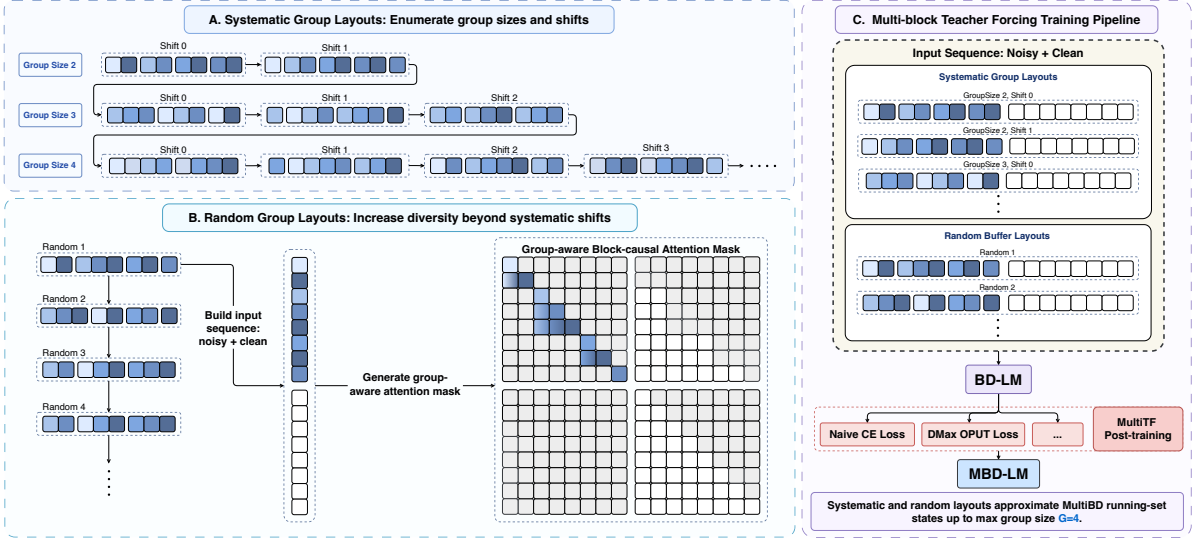


Figure 4: **MultiTF pipeline.** (A) shifted systematic layouts cover group sizes and relative positions; (B) random layouts add diversity, and the Group-Aware Dual-Stream Mask builds noisy/clean views; (C) masked CE plus optional model-specific objectives post-train BD-LMs into MBD-LMs.

Objective. For masked positions $\mathcal{M}_\lambda$, the default objective is

$$\mathcal{L}_{\text{MultiTF}}(\theta) = -\mathbb{E}_{\lambda, t} \left[\frac{1}{|\mathcal{M}_\lambda|} \sum_{i \in \mathcal{M}_\lambda} \ell_{\theta, i} \right], \quad (11)$$

$$\ell_{\theta, i} = \log p_\theta(x_0^i | \mathbf{X}_\lambda, A_\lambda).$$

The normalization gives equal weight to masked positions within a sampled layout. The clean stream acts only as context, while predictions and loss are taken from masked positions in the noisy stream. The complete algorithm and model-specific DMax/SDAR objective variants are illustrated in Appendix B.

3.4 Optimized Multi-Block Diffusion

Why a naive dynamic set is insufficient. An abstract MultiBD implementation would append a block whenever the active tail becomes sufficiently clean and delete the front when completed. Although correct, the changing query length and attention layout prevent stable kernel plans and CUDA Graph replay; Algorithm 4 in Appendix C.1 makes this failure mode explicit.

Fixed-shape Block Buffer. We allocate N_{buf} physical slots,

$$\mathcal{W}_s = \mathcal{R}_s \parallel \mathcal{D}_s, \quad |\mathcal{W}_s| = N_{\text{buf}}, \quad (12)$$

where $\mathcal{D}_s$ is the trailing dummy-slot segment. Resident blocks form $\mathcal{R}_s$ and the remaining slots form

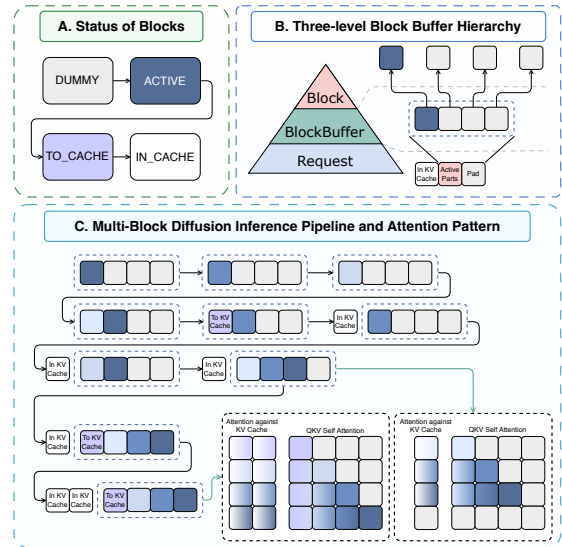


Figure 5: **MultiBD system support.** Blocks transition DUMMY→ACTIVE→TO-CACHE→IN-CACHE. A fixed *Block Buffer* refines active blocks jointly, commits completed prefixes to the KV cache, and slides without changing the physical input shape.

$\mathcal{D}_s$; thus N_{buf} realizes the bounded *running-set* sizes covered by MultiTF. A future block activates through metadata; after the front is cached, the logical window advances without changing shape.

Deterministic state transitions. Each slot follows DUMMY→ACTIVE→TO-CACHE→IN-CACHE, as shown in Figure 5. Within a step, the engine first decides whether the active tail is sufficiently advanced to activate one trailing

dummy slot, then refines every active slot in a single forward pass. A completed block is marked TO-CACHE only when every preceding resident block is already cacheable or cached, preserving prefix order. The engine commits consecutive front blocks, slides the logical window, and appends the same number of dummy slots without reallocating the physical buffer. These transitions modify metadata and cache offsets rather than tensor dimensions.

Cache preservation. Cached prefix keys/values are immutable, while buffer tokens use block-causal extend attention. Hence cache semantics match SingleBD even though several blocks are refined together: a cached block is never recomputed, and active slots cannot modify or bidirectionally attend through the cached prefix. The same separation also permits request-level prefix sharing under paged KV-cache systems. Algorithm 5, state transitions, threshold definitions, and prefix-cache comparisons are in Appendix C. The resulting static execution produces the measured throughput in Table 7.

4 Experiments

4.1 Experimental Setup

Models and training. We evaluate MultiTF on LLaDA2.0 (Bie et al., 2025) and SDAR (Cheng et al., 2025) backbones, including DMax-enhanced variants (Chen et al., 2026b). LLaDA2-Mini is a 16B-parameter mixture-of-experts model with approximately 1B active parameters per token (16B-A1B), whereas SDAR-8B-Chat is an 8B dense model. For each backbone, MultiTF uses systematic shifted and random *group-layouts* to approximate bounded MultiBD *running-set* states; the resulting checkpoints are denoted MBD-*. Training-free MultiBD instead applies the same inference engine to the original BD-LM without post-training.

Evaluation protocol. We evaluate all models and variants on GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021), MBPP+, and HumanEval+ (Austin et al., 2021; Chen et al., 2021; Liu et al., 2023) with greedy decoding (temperature 0). All reported transitions are deterministic: thresholds are fixed before evaluation, and block activation and commitment use no sampling. Accuracy is exact match for math and pass@1 for code. We report Tokens Per Forward pass (TPF)

and Accuracy Under Parallelism (AUP), the trapezoidal area under the accuracy–TPF curve following d3LLM (Qian et al., 2026). Training, inference, and hardware details appear in Appendix E.

4.2 Main Results

Parallelism–quality trade-off. Table 2 shows that MultiTF increases inter-block parallelism while maintaining comparable generation quality. MBD-LLaDA2-Mini raises average TPF from 3.47 to 6.19 (+78.4%) and average AUP from 247 to 449, while average accuracy remains comparable (80.0% versus 81.0%). The gain is consistent across all four tasks: TPF reaches 5.55–7.85 rather than 2.27–5.53 under SingleBD. With DMax, average TPF rises from 6.35 to 9.34 (+47.1%) and AUP from 460 to 661, with a 1.02-point average accuracy decrease. Thus DMax’s token-level self-correction and MultiBD’s inter-block parallelism are compatible rather than redundant.

Effect of train–inference alignment. Training-free MultiBD exposes parallelism already latent in a BD-LM, but its unseen multi-block states can reduce quality. On LLaDA2-Mini, training-free MultiBD increases TPF from 3.47 to 4.41 but lowers accuracy from 80.0% to 78.6%. MultiTF then raises accuracy to 81.0% and TPF to 6.19, improving both sides of the trade-off relative to the training-free configuration. On LLaDA2-Mini-DMax, MultiTF similarly improves average accuracy from 78.2% to 78.6% while retaining approximately 9.3 TPF. These comparisons indicate that inference-side scheduling alone is useful, but matched post-training is what makes aggressive MultiBD reliable.

Generalization across backbones and block sizes. The gains are not confined to the LLaDA2 MoE architecture. On the dense SDAR-8B-Chat-b32 model, MultiTF raises average TPF from 2.54 to 4.46 (+75.6%) and average accuracy from 69.0% to 69.7%. With block size four, MBD-SDAR-8B-Chat-b4 reaches 2.42 TPF versus 1.25 for SingleBD while retaining comparable average accuracy (75.3% versus 75.6%). The smaller block exposes less intra-block work, so the relative benefit of concurrent blocks is particularly clear.

Training-free transfer to additional checkpoints. Table 3 further isolates the inference-side mechanism on LLaDA2 checkpoints that receive no MultiTF post-training. On LLaDA2-

Table 2: **Math and code results.** SingleBD is native single-block inference; training-free MultiBD adds multi-block inference without post-training; MBD-* uses MultiTF post-training and MultiBD inference. MultiTF substantially increases parallelism while maintaining comparable quality.

Model	GSM8K		MATH500		MBPP+		HumanEval+		Average		
	Acc.	TPF	Acc.	TPF	Acc.	TPF	Acc.	TPF	Acc.	TPF	AUP
<i>LLaDA2-Mini-DMax (buffer 2, block 32)</i>											
SingleBD	91.9	5.70	76.8	6.13	72.2	6.14	77.4	7.44	79.6	6.35	460
MultiBD (training-free)	89.8	8.76	73.8	9.08	72.2	8.44	76.8	11.0	78.2	9.31	652
MBD-LLaDA2-Mini-DMax	91.7	8.95	75.0	9.31	70.1	8.34	77.4	10.8	78.6	9.34	661
<i>LLaDA2-Mini (buffer 2, block 32)</i>											
SingleBD	91.9	2.27	74.2	2.83	75.7	3.25	78.1	5.53	80.0	3.47	247
MultiBD (training-free)	92.7	2.76	73.6	3.53	72.5	3.97	75.6	7.37	78.6	4.41	302
MBD-LLaDA2-Mini	92.0	5.55	79.2	6.02	72.5	5.35	80.5	7.85	81.0	6.19	449
<i>SDAR-8B-Chat-b32 (buffer 4, block 32)</i>											
SingleBD	90.1	2.52	65.6	3.81	52.7	1.83	67.7	2.00	69.0	2.54	142
MultiBD (training-free)	89.0	2.78	60.6	5.06	52.1	1.97	65.9	2.24	66.9	3.01	156
MBD-SDAR-8B-Chat-b32	89.2	3.08	68.0	5.08	59.0	4.87	62.8	4.82	69.7	4.46	210
<i>SDAR-8B-Chat-b4 (buffer 4, block 4)</i>											
SingleBD	91.1	1.33	72.8	1.46	64.8	1.13	73.7	1.07	75.6	1.25	85.5
MultiBD (training-free)	90.5	2.39	70.6	2.68	65.8	1.55	74.4	1.47	75.3	2.00	130
MBD-SDAR-8B-Chat-b4	91.8	2.28	72.4	2.52	64.3	2.62	72.6	2.24	75.3	2.42	149

Table 3: Training-free MultiBD transfer to additional LLaDA2 checkpoints on math benchmarks. Each cell reports Accuracy/TPF.

Inference	GSM8K	MATH500	Average
<i>LLaDA2-Mini-CAP</i>			
SingleBD	91.7/3.08	77.8/3.71	84.8/3.40
MultiBD	91.2/ 4.00	77.2/ 4.94	84.2/ 4.47
<i>LLaDA2.1-Mini</i>			
SingleBD	93.0/4.12	81.4/4.87	87.2/4.50
MultiBD	92.3/ 5.80	81.0/ 7.20	86.6/ 6.50

Mini-CAP, MultiBD raises average TPF from 3.40 to 4.47 (+31.5%) while average accuracy changes from 84.8% to 84.2%. On LLaDA2.1-Mini, average TPF rises from 4.50 to 6.50 (+44.4%) with the same 0.6-point accuracy decrease; the gain appears on both GSM8K (4.12 to 5.80 TPF) and MATH500 (4.87 to 7.20 TPF). Thus, MultiBD transfers across LLaDA2 versions without post-training, while the controlled experiments below motivate MultiTF for more aggressive states.

4.3 Controlled Training-State Comparison

Table 4 isolates training-state construction: MultiTF and the D2F-style monotonic baseline use the *same* backbone, data, objective, training budget, and inference configuration. MultiTF raises average accuracy from 74.5% to 78.6% and TPF from 9.05 to 9.35. The accuracy gains hold on every benchmark, led by HumanEval+ (70.1% to 77.4%) and MATH500 (70.4% to 75.0%), while

Table 4: Controlled D2F-style comparison. Each cell is Accuracy/TPF.

Training states	GSM	MATH	MBPP+	HE+	Avg.
D2F-style	88.3/9.01	70.4/9.27	69.3/7.63	70.1/10.3	74.5/9.05
MultiTF	91.7/8.95	75.0/9.31	70.1/8.34	77.4/10.8	78.6/9.35

Table 5: MultiTF component ablations averaged over HumanEval+ and GSM8K with LLaDA2-Mini-DMax.

Configuration	Acc ↑	TPF ↑	AUP ↑
SingleBD (Native)	84.67	6.57	536.89
<i>noise-group layouts construction</i>			
+ systematic layouts	83.22	9.71	<u>774.03</u>
+ random layouts	82.72	9.42	747.46
systematic + random layouts (ours)	<u>84.59</u>	9.87	805.34
<i>block-level noise-scheduler</i>			
D2F-style monotonic scheduler	79.34	8.76	657.74
random scheduler	83.14	9.70	771.74
sorted-uniform scheduler	81.28	<u>9.73</u>	748.73
chain-uniform scheduler (ours)	<u>84.59</u>	9.87	805.34

TPF is comparable on GSM8K and slightly higher elsewhere. This comparison shows that merely exposing multiple noisy suffix blocks is insufficient: reliable MultiBD training must also match the bounded running-set size and heterogeneous noise levels across blocks.

4.4 Ablations and Boundary Coherence

Group-layout construction. Table 5 shows complementary roles: systematic layouts cover bounded set sizes and group-relative positions,

Table 6: Effect of tail activation on block-boundary coherence. Aggressive activation ($\tau_{\text{add}} = 0.1$) slightly increases TPF but sharply reduces code accuracy. Each cell reports Accuracy/TPF.

τ_{add}	HumanEval+	MBPP+
0.1	39.0/11.5	47.1/9.21
0.9 (ours)	77.4/10.8	70.1/8.34

while random layouts add irregular boundaries. Alone they yield 83.2% and 82.7% accuracy, below SingleBD’s 84.7%; together they recover 84.6% while reaching 9.87 TPF and the best AUP.

Noise scheduling. Chain-uniform matches MultiBD’s *heterogeneous slot-wise noise gaps*, not merely monotonicity. D2F-style and random yield 79.3% and 83.1% accuracy. Sorted-uniform reaches 9.73 TPF but only 81.3% accuracy. Despite its monotonic schedule, the 3.3-point gap from chain-uniform shows that slot-conditional gaps—not ordering—are decisive.

Table 6 stress-tests block activation. The aggressive setting ($\tau_{\text{add}} = 0.1$) raises TPF slightly but collapses code accuracy; the default $\tau_{\text{add}} = 0.9$ preserves block-boundary coherence. Appendix D.1 gives syntax, runtime, and incorrect-output failures.

4.5 Train–Inference Alignment Analysis

Figure 2 directly compares the states induced by training schedulers with states measured during MultiBD decoding. The analysis separates two factors that a full noisy suffix obscures: the marginal mask-ratio distribution of each logical slot and the number of active blocks simultaneously exist in the running-set.

Slot-wise noise gaps. The D2F-style monotonic construction in Figure 2(A) yields heavily overlapping slot distributions. In actual MultiBD traces, the first slot is often nearly complete when a much noisier trailing slot is activated, creating a substantially larger adjacent gap. Chain-uniform sampling produces this separation in Figure 2(B), and the resulting training distributions more closely resemble the inference-time states in panel (C). This distributional evidence explains why the two monotone schedulers in Table 5 behave differently: monotonicity alone is insufficient, and the relative noise gaps between adjacent slots must also match MultiBD inference.

Table 7: Single-sample throughput on two H100 GPUs (TP=2). Gains are relative to each model’s SingleBD counterpart.

Model	TPF	ms/step	TPS	Gain
LLaDA2-Mini	3.47	7.07	517	–
MBD-LLaDA2-Mini	6.19	8.78	746	+44.2%
LLaDA2-Mini-DMax	6.35	9.02	779	–
MBD-LLaDA2-Mini-DMax	9.34	11.2	927	+19.0%

Bounded active set. Panels (D–E) show that MultiBD usually maintains about two active blocks, with occasional expansion to three or four. The active count changes over a trajectory as leading blocks stabilize and trailing blocks enter, but it remains far smaller than the full remaining suffix. This supports using bounded *noise-groups* and varying group-relative positions in MultiTF. Together, the slot and count analyses show that train–inference alignment requires matching both *which* states coexist and *how many* are processed, rather than only making future noisy blocks visible.

4.6 Wall-Clock Throughput

TPF is a model-level measure of useful progress, whereas TPS also includes the cost of processing a static physical buffer. Let $Q_s = N_{\text{buf}}B$ be the number of query tokens processed at step s and define the token-efficiency factor

$$\eta_{\text{tok}}(s) = \frac{\text{TPF}_s}{Q_s}, \quad \text{TPS} = \frac{\text{TPF}}{T_{\text{step}}} = \frac{\eta_{\text{tok}} Q_s}{T_{\text{step}}}. \quad (13)$$

A larger buffer amortizes weight, prefix-KV, and launch costs, but extra slots raise T_{step} ; useful-token gains must exceed that overhead.

Table 7 uses the final rebuttal measurements. For LLaDA2-Mini, $1.78 \times$ TPF divided by $1.24 \times$ step cost predicts the measured $1.44 \times$ throughput. For DMax, $1.47/1.24 = 1.18 \times$ predicts the measured $1.19 \times$. Both configurations therefore convert TPF into measured wall-clock speedup. Buffer overhead remains: TPS gains lag TPF gains. The comparison is instructive: TPF should not be reported as a proxy for speed; the $1.78 \times$ and $1.47 \times$ TPF gains become $1.44 \times$ and $1.19 \times$ TPS gains after accounting for step latency. Per-benchmark TPS, compute and KV-traffic terms, and the complete roofline model derivation of MultiBD are in Appendix D.3.

5 Related Work

5.1 Diffusion Language Models

Diffusion Language Models (DLMs) generate text through iterative denoising and enable parallel token refinement as an alternative to autoregressive generation. Representative models include LLaDA (Nie et al., 2025), Dream (Ye et al., 2025), and LLaDA2.x (Bie et al., 2025, 2026), which improve scaling, initialization, and editable refinement. However, fully bidirectional DLMs are difficult to serve efficiently because they do not naturally support KV caching or flexible-length generation, both of which are essential for practical deployment and on-policy reinforcement learning.

Block Diffusion Language Models (BD-LMs) introduce block-causal generation (Arriola et al., 2025; Bie et al., 2025; Cheng et al., 2025). Their native *Single-Block Diffusion* (SingleBD) decodes one noisy block from a clean cached prefix, enabling KV caching and intra-block parallelism, yet processes blocks sequentially. We study *Multi-Block Diffusion* (MultiBD), a broader regime in which a bounded *running-set* of consecutive blocks is refined concurrently.

5.2 Efficient DLM Inference and Training

Efficient DLMs have been studied through distillation, scheduling, caching, and parallel decoding. D2F (Wang et al., 2025) introduces noisy-block visibility during training and demonstrates the potential of MultiBD-style pipelined decoding. DMax (Chen et al., 2026b), d3LLM (Qian et al., 2026), LightningRL (Hu et al., 2026), CForce (Ren et al., 2026), and dParallel (Chen et al., 2025) improve the accuracy-parallelism trade-off through training objectives or decoding schedules. Fast-dLLM (Wu et al., 2025) and LoPA (Xu et al., 2025) accelerate inference through caching and lookahead parallelism.

Speculative decoding instead follows a draft-verify paradigm: a lightweight model proposes tokens and a separate target model verifies them in parallel (Leviathan et al., 2023). DFlash (Chen et al., 2026a) uses a block-diffusion model as the parallel drafter, while DSpark (Cheng et al., 2026) combines semi-autoregressive drafting with confidence-scheduled target verification. These methods accelerate an autoregressive target through acceptance of speculative drafts. By contrast, MultiTF and MultiBD directly train and serve the BD-LM itself, without a separate draft-

verify pair; the two directions are therefore complementary rather than interchangeable.

5.3 Positioning of Multi-Block Parallelism

Existing acceleration methods operate at different levels. DMax and dParallel control token commitment within each denoising step; Fast-dLLM reduces repeated computation through cache reuse; LoPA speculates over alternative continuations; and d3LLM or LightningRL reshapes the denoising trajectory. MultiBD instead targets *inter-block* concurrency by refining later blocks before the leading block is fully committed. It is therefore complementary to token-level methods, as illustrated by the additive TPF gain of MBD-LLaDA2-Mini-DMax in Table 2.

D2F is the closest prior because it exposes noisy suffix blocks and motivates pipelined decoding. However, its full suffix is neither a bounded nor fixed-cost inference state, and its prefix-full attention is not naturally compatible with the immutable block-causal prefix used for cache sharing. Its monotonic suffix also does not capture the heterogeneous slot-wise noise gaps observed during MultiBD inference. MBD-LMs instead combine a bounded *running-set*, matched MultiTF *noise-groups*, and a static *Block Buffer*. Table 4 isolates the training-state benefit.

6 Conclusion

We introduced *Multi-Block Diffusion Language Models* (MBD-LMs), framing practical MultiBD as concurrent refinement of a bounded *running-set* between SingleBD and full-suffix constructions. MultiTF aligns post-training with this state through bounded *noise-groups*, systematic and random layouts, chain-uniform scheduling, and the *Group-Aware Dual-Stream Mask*, while the *Block Buffer* preserves prefix caching and static-shape execution at inference. On LLaDA2-Mini, MBD-LMs raise average TPF from 3.47 to 6.19 at comparable accuracy; combined with DMax, they reach 9.34 TPF and 927 TPS on two H100 GPUs. Training-free transfer to LLaDA2-Mini-CAP and LLaDA2.1-Mini further shows that the inference mechanism is not checkpoint-specific. The controlled D2F-style comparison shows that reliable MultiBD requires bounded, heterogeneous training states, while the throughput measurements demonstrate the importance of systems support for converting TPF gains into realized TPS.

Limitations

Evaluation scope. Our experiments emphasize verifiable math and code tasks, where exact-match and execution-based metrics are reliable. Open-ended generation, dialogue, multilingual generation, and long-form coherence remain to be evaluated with appropriate human and automatic protocols. We cover 8B dense and 16B-A1B MoE backbones, but not substantially larger models or high-concurrency production traffic.

Configuration sensitivity. MultiBD introduces block size, buffer size, and activation, completion, and token-update thresholds. We fix these values before evaluation, but the best configuration can depend on the model, task, context length, and hardware. Automated tuning could improve results but would make cross-system comparisons less direct.

Serving and kernel complexity. The *Block Buffer* makes the physical shape static, but adds state transitions, slot bookkeeping, cache movement, and failure-recovery logic to a serving stack. Specialized kernels and CUDA Graphs must be maintained across model architectures, compiler versions, and devices. These engineering costs may outweigh the benefit in small deployments.

TPF versus realized throughput. TPF measures useful decoding parallelism, not wall-clock speed. Resident and dummy slots increase per-step work, so a TPF gain does not translate one-to-one into TPS; the gap grows when execution is compute-bound or the buffer is under-utilized. Thus, our two-H100 results should not be extrapolated across hardware or batching.

Likelihood-based evaluation. As with other masked and block-diffusion language models, exact sequence likelihood and conventional perplexity are not directly available from the iterative denoising process. This limits likelihood-based comparison with autoregressive models and complicates calibration analysis.

Reinforcement learning. MultiBD reinforcement learning is promising but introduces a larger state space than SingleBD: rewards depend jointly on tokens, block activation, and cache transitions. Credit assignment across concurrent blocks and delayed block-boundary failures is difficult, and naive optimization may trade correctness for aggressive parallelism. Developing stable objectives

and unbiased estimators for this setting is left to future work.

Acknowledgments

This work was supported by the Shanghai Key Technology R&D Program “New Generation of Information Technology” (No. 25511103700), the National Natural Science Foundation of China (Nos. 62306176 and 92470118), the CCF-ALIMAMA TECH Kangaroo Fund (No. CCF-ALIMAMA OF 2025010), and Ant Group.

References

- Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. [Block diffusion: Interpolating between autoregressive and diffusion language models](#). In *International Conference on Learning Representations (ICLR)*. Oral Presentation.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. [Program synthesis with large language models](#). *arXiv preprint arXiv:2108.07732*.
- Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, and 31 others. 2026. [LLaDA2.1: Speeding up text diffusion via token editing](#). *arXiv preprint arXiv:2602.08676*.
- Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, and 12 others. 2025. [LLaDA2.0: Scaling up diffusion language models to 100b](#). *arXiv preprint arXiv:2512.15745*.
- Nicolas Boizard, Hippolyte Gisserot-Boukhlef, Kevin El-Haddad, Céline Hudelot, and Pierre Colombo. 2025. [When does reasoning matter? a controlled study of reasoning’s contribution to model performance](#). *arXiv preprint arXiv:2509.22193*.
- Jian Chen, Yesheng Liang, and Zhijian Liu. 2026a. [DFlash: Block diffusion for flash speculative decoding](#). *arXiv preprint arXiv:2602.06036*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela

- Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#). *arXiv preprint arXiv:2107.03374*.
- Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. 2025. [dParallel: Learnable parallel decoding for dLLMs](#). *arXiv preprint arXiv:2509.26488*.
- Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. 2026b. [DMax: Aggressive parallel decoding for dLLMs](#). *arXiv preprint arXiv:2604.08302*.
- Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and Bowen Zhou. 2025. [SDAR: A synergistic diffusion-autoregression paradigm for scalable sequence generation](#). *arXiv preprint arXiv:2510.06303*.
- Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, and 14 others. 2026. [DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation](#). *arXiv preprint arXiv:2607.05147*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. [Training verifiers to solve math word problems](#). *arXiv preprint arXiv:2110.14168*.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. [Measuring mathematical problem solving with the math dataset](#). *arXiv preprint arXiv:2103.03874*.
- Yanzhe Hu, Yijie Jin, Pengfei Liu, Kai Yu, and Zhijie Deng. 2026. [LightningRL: Breaking the accuracy-parallelism trade-off of block-wise dLLMs via reinforcement learning](#). *arXiv preprint arXiv:2603.13319*.
- jtatman. 2025. [Python code dataset 500k](#). Hugging Face dataset.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. [Fast inference from transformers via speculative decoding](#). In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.
- Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. [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.
- Guanxi Lu, Hao Mark Chen, Yuto Karashima, Zhi-can Wang, Daichi Fujiki, and Hongxiang Fan. 2026. [AdaBlock-dLLM: Semantic-aware diffusion LLM inference via adaptive block size](#). *arXiv preprint arXiv:2509.26432*.
- Qianli Ma, Yaowei Zheng, Zhelun Shi, Zhongkai Zhao, Bin Jia, Ziyue Huang, Zhiqi Lin, Youjie Li, Jiacheng Yang, Yanghua Peng, Zhi Zhang, and Xin Liu. 2025. [VeOmni: Scaling any modality model training with model-centric distributed recipe zoo](#). *arXiv preprint arXiv:2508.02317*.
- Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. [Large language diffusion models](#). *arXiv preprint arXiv:2502.09992*.
- Yu-Yang Qian, Junda Su, Lanxiang Hu, Peiyuan Zhang, Zhijie Deng, Peng Zhao, and Hao Zhang. 2026. [d3LLM: Ultra-fast diffusion LLM using pseudo-trajectory distillation](#). *arXiv preprint arXiv:2601.07568*.
- Yuji Ren, Chenkai Xu, Zhuocheng Gong, Jianguo Li, and Zhijie Deng. 2026. [CFORCE: Boosting parallel decoding for dLLMs via consistency forcing](#). *Preprint, arXiv:2608.13925*.
- Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. [Simple and effective masked diffusion language models](#). *arXiv preprint arXiv:2406.07524*.
- Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. 2025. [Diffusion LLMs can do faster-than-AR inference via discrete diffusion forcing](#). *arXiv preprint arXiv:2508.09192*.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. 2025. [Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding](#). *arXiv preprint arXiv:2505.22618*.
- Chenkai Xu, Yijie Jin, Jiajun Li, Yi Tu, Guoping Long, Dandan Tu, Mingcong Song, Hongjie Si, Tianqi Hou, Junchi Yan, and Zhijie Deng. 2025. [LoPA: Scaling dLLM inference via lookahead parallel decoding](#). *arXiv preprint arXiv:2512.16229*.
- Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025. [Dream 7B: Diffusion large language models](#). *arXiv preprint arXiv:2508.15487*.

A Theoretical View of MultiTF

This appendix provides a simple theoretical view of Multi-block Teacher Forcing (MultiTF). The goal is not to prove that MultiTF directly improves downstream accuracy. Instead, we show that MultiTF can be interpreted as a coverage-based surrogate for the ideal MultiBD training objective, and that its approximation gap is controlled by the mismatch in *running-set* coverage and inter-block noise-ratio distributions.

Ideal MultiBD objective. Let $\mathcal{R} = \{a, \dots, c\}$ denote a consecutive MultiBD *running-set* with size $|\mathcal{R}| \leq G_{\max}$, where $G_{\max}$ is the maximum *noise-group* size used in MultiTF. For a clean sequence $\mathbf{x}_0$, noise ratios $\mathbf{t}$, and model θ , define the state loss as follows

$$\ell_{\theta}(\mathbf{x}_0, \mathcal{R}, \mathbf{t}) = -\frac{1}{|\mathcal{M}_{\mathcal{R}}|} \sum_{i \in \mathcal{M}_{\mathcal{R}}} \ell_{\theta, i}, \quad (14)$$

$$\ell_{\theta, i} = \log p_{\theta} \left(x_0^i \mid \mathbf{x}_0^{(<a)}, \mathbf{b}_{\mathcal{R}, \mathbf{t}} \right),$$

where $\mathbf{x}_0^{(<a)}$ is the clean prefix before $\mathcal{R}$, $\mathbf{b}_{\mathcal{R}, \mathbf{t}}$ denotes the noisy blocks inside $\mathcal{R}$, and $\mathcal{M}_{\mathcal{R}}$ denotes masked positions in the *running-set*.

Let $p_{\text{inf}}(\mathcal{R}, \mathbf{t})$ be the inference-time distribution of MultiBD states, and let $q_{\text{MultiTF}}(\mathcal{R}, \mathbf{t})$ be the training-state distribution induced by MultiTF *group-layouts* and the chain-uniform *noise-scheduler*. The ideal MultiBD objective is

$$\mathcal{L}_{\text{MultiBD}}^*(\theta) = \mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{(\mathcal{R}, \mathbf{t}) \sim p_{\text{inf}}} [\ell_{\theta}(\mathbf{x}_0, \mathcal{R}, \mathbf{t})], \quad (15)$$

while MultiTF minimizes the surrogate objective

$$\mathcal{L}_{\text{MultiTF}}(\theta) = \mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{(\mathcal{R}, \mathbf{t}) \sim q_{\text{MultiTF}}} [\ell_{\theta}(\mathbf{x}_0, \mathcal{R}, \mathbf{t})]. \quad (16)$$

Systematic shifts cover bounded *running-sets*.

Assume the sequence is padded so that boundary effects can be ignored. For a fixed *noise-group* size $g \in \{2, \dots, G_{\max}\}$, MultiTF constructs g shifted layouts. Then every consecutive *running-set* $\mathcal{R} = \{a, \dots, a + g - 1\}$ appears as one *noise-group* in exactly one shifted layout for that g .

Proof. For a fixed g , each shifted layout places group boundaries every g blocks with a different offset. For a *running-set* starting at block a , choosing the shift $h = (a - 1) \bmod g$ aligns a group boundary with a , so $\{a, \dots, a + g - 1\}$ appears as one *noise-group*. The shift is unique modulo g , so the *running-set* appears once among all of the g shifted layouts.

Thus, systematic layouts cover all consecutive *running-sets* with size between 2 and $G_{\max}$. Equivalently, for each fixed g , every block appears once at every group-relative logical slot across the g shifts. Random layouts do not change this support guarantee, but add additional samples with non-regular *noise-group-size* combinations.

Objective mismatch bound. We next bound the gap between the ideal MultiBD objective and the MultiTF surrogate objective. Let $p_{\mathcal{R}}$ and $q_{\mathcal{R}}$ be the marginal distributions over *running-sets* under p_{inf} and q_{MultiTF} , respectively.

We assume:

A1. Bounded MultiBD states. The inference distribution p_{inf} is supported on consecutive *running-sets* with $2 \leq |\mathcal{R}| \leq G_{\max}$.

A2. Bounded loss. For all $\theta, \mathbf{x}_0, \mathcal{R}, \mathbf{t}$,

$$0 \leq \ell_{\theta}(\mathbf{x}_0, \mathcal{R}, \mathbf{t}) \leq M.$$

A3. Lipschitz dependence on noise ratios.

For every $\theta, \mathbf{x}_0, \mathcal{R}$, the state loss is L_t -Lipschitz in the noise-ratio vector:

$$|\ell_{\theta}(\mathbf{x}_0, \mathcal{R}, \mathbf{t}) - \ell_{\theta}(\mathbf{x}_0, \mathcal{R}, \mathbf{t}')| \leq L_t \|\mathbf{t} - \mathbf{t}'\|_1. \quad (17)$$

Here $\text{TV}(p, q) = \frac{1}{2} \sum_x |p(x) - q(x)|$ denotes the total variation distance between two discrete distributions.

Define the *running-set* distribution mismatch as

$$\delta_{\mathcal{R}} = \text{TV}(p_{\mathcal{R}}, q_{\mathcal{R}}),$$

and assume the conditional noise-ratio mismatch satisfies

$$W_1(p_{\text{inf}}(\mathbf{t} \mid \mathcal{R}), q_{\text{MultiTF}}(\mathbf{t} \mid \mathcal{R})) \leq \delta_t$$

for every *running-set* $\mathcal{R}$, where W_1 is the Wasserstein-1 distance under the ℓ_1 metric.

Under these assumptions, for any model θ ,

$$|\mathcal{L}_{\text{MultiBD}}^*(\theta) - \mathcal{L}_{\text{MultiTF}}(\theta)| \leq M\delta_{\mathcal{R}} + L_t\delta_t. \quad (18)$$

Proof. For clarity, omit the outer expectation over $\mathbf{x}_0$. Define $\mu_p(\mathcal{R}) = \mathbb{E}_{p(\mathbf{t} \mid \mathcal{R})}[\ell_{\theta}]$ and $\mu_q(\mathcal{R}) = \mathbb{E}_{q(\mathbf{t} \mid \mathcal{R})}[\ell_{\theta}]$. Adding and subtracting $\mathbb{E}_{q_{\mathcal{R}}}[\mu_p]$ gives

$$\begin{aligned} |\mathbb{E}_{p_{\mathcal{R}}}[\mu_p] - \mathbb{E}_{q_{\mathcal{R}}}[\mu_q]| &\leq |\mathbb{E}_{p_{\mathcal{R}}}[\mu_p] - \mathbb{E}_{q_{\mathcal{R}}}[\mu_p]| \\ &\quad + |\mathbb{E}_{q_{\mathcal{R}}}[\mu_p - \mu_q]|. \end{aligned} \quad (19)$$

The first term is bounded by $M\text{TV}(p_{\mathcal{R}}, q_{\mathcal{R}}) = M\delta_{\mathcal{R}}$, since the loss is bounded in $[0, M]$. The second term is bounded by $L_t\delta_t$ by the Lipschitz assumption and the definition of W_1 . Combining the two terms gives Eq. 18.

Excess target risk. Let $\hat{\theta}$ be a model whose MultiTF objective is within ϵ_{opt} of the best model in a hypothesis class Θ :

$$\mathcal{L}_{\text{MultiTF}}(\hat{\theta}) \leq \min_{\theta \in \Theta} \mathcal{L}_{\text{MultiTF}}(\theta) + \epsilon_{\text{opt}}.$$

Then

$$\begin{aligned} \mathcal{L}_{\text{MultiBD}}^*(\hat{\theta}) - \min_{\theta \in \Theta} \mathcal{L}_{\text{MultiBD}}^*(\theta) \\ \leq 2(M\delta_{\mathcal{R}} + L_t\delta_t) + \epsilon_{\text{opt}}. \end{aligned} \quad (20)$$

This bound shows that reducing *running-set* distribution mismatch $\delta_{\mathcal{R}}$ and noise-ratio mismatch δ_t directly tightens the gap between MultiTF training and ideal MultiBD inference. Systematic shifts reduce support mismatch by covering bounded consecutive *running-sets* up to size G_{max} , random layouts add distributional diversity, and the chain-uniform *noise-scheduler* reduces noise-ratio mismatch by producing heterogeneous slot-wise noise gaps. Therefore, MultiTF can be viewed as a coverage-based surrogate for the ideal objective of our proposed MBD-LM.

B MultiTF Training Implementation Details

This appendix provides implementation details for Multi-block Teacher Forcing (MultiTF), which post-trains BD-LMs into MBD-LMs. The terminology follows Section 3.3: training-side structures are called *noise-groups*, *group-layouts*, and *noise-schedulers*, while inference-side structures are called *Block Buffers* and *slots*. We use G_{max} for the maximum *noise-group* size, Λ for the set of *group-layouts*, λ for one *group-layout*, and H_m for one *noise-group*. We use VeOmni (Ma et al., 2025) as the training framework. SDAR models are post-trained on reasoning/code data from prior studies (Boizard et al., 2025; jtatman, 2025); LLaDA2.x and DMax-enhanced models are post-trained on the corresponding reasoning/code mixtures used by their base recipes.

B.1 Group-Layout Construction

MultiTF constructs a *group-layout* set

$$\Lambda = \Lambda_{\text{sys}} \cup \Lambda_{\text{rand}},$$

where Λ_{sys} contains systematic shifted layouts and Λ_{rand} contains random layouts. Each *group-layout* $\lambda = (H_1, \dots, H_{|\lambda|})$ partitions the block sequence $[\mathbf{b}_1, \dots, \mathbf{b}_K]$ into consecutive *noise-groups*. Each *noise-group* $H_m = \{a_m, \dots, c_m\}$ has the same consecutive-block form as a possible MultiBD *running-set*.

Algorithm 1 Multi-block Teacher Forcing

Require: Clean sequence $\mathbf{x}_0$; block size B ; maximum *noise-group* size G_{max} ; noise bounds $t_{\text{low}}, t_{\text{high}}$; margin ρ ; random-layout count N_{rand} .

- 1: Partition $\mathbf{x}_0$ into blocks $[\mathbf{b}_1, \dots, \mathbf{b}_K]$.
- 2: Generate systematic layouts over every size $g \in \{2, \dots, G_{\text{max}}\}$ and shift $h \in \{0, \dots, g-1\}$.
- 3: Generate N_{rand} random layouts and let Λ be the union of both layout sets.
- 4: $t_{\text{eff}} \leftarrow t_{\text{high}} - \rho(t_{\text{high}} - t_{\text{low}})$; $\mathcal{J} \leftarrow 0$.
- 5: **for** each $\lambda \in \Lambda$ **do**
- 6: Initialize $\mathbf{x}_t^\lambda \leftarrow \mathbf{x}_0$.
- 7: **for** each *noise-group* $H_m = (j_1, \dots, j_{n_m}) \in \lambda$ **do**
- 8: Sample $\ell \sim \mathcal{U}(t_{\text{low}}, t_{\text{eff}})$.
- 9: **for** $i = 1$ **to** n_m **do**
- 10: Sample $t_{j_i} \sim \mathcal{U}(\ell, t_{\text{eff}})$; $\ell \leftarrow t_{j_i}$.
- 11: Replace $\lfloor Bt_{j_i} \rfloor$ random tokens of $\mathbf{b}_{j_i}$ with $[\mathbf{M}]$.
- 12: **end for**
- 13: **end for**
- 14: $\mathbf{X}_\lambda \leftarrow [\mathbf{x}_t^\lambda; \mathbf{x}_0]$; construct Group-Aware Dual-Stream Mask $\mathbf{A}_\lambda$.
- 15: Evaluate the model on $(\mathbf{X}_\lambda, \mathbf{A}_\lambda)$ and add masked CE $\mathcal{J}_\lambda$ to $\mathcal{J}$.
- 16: **end for**
- 17: **return** $\mathcal{J}/|\Lambda|$.

Systematic layouts. For each *noise-group* size $g \in \{2, \dots, G_{\text{max}}\}$ and shift $h \in \{0, \dots, g-1\}$, MultiTF constructs a shifted layout $\lambda_{g,h}$ by placing group boundaries every g blocks with offset h . Formally, define the boundary set

$$\begin{aligned} \mathcal{B}_{g,h} &= \text{sort}(\{1, K+1\} \cup \mathcal{C}_{g,h}), \\ \mathcal{C}_{g,h} &= \{1+h+qg : q \in \mathbb{Z}, \chi(q)\}, \\ \chi(q) &: 1 < 1+h+qg < K+1. \end{aligned}$$

Let $\mathcal{B}_{g,h} = (r_1, \dots, r_{n_{g,h}+1})$ after sorting. The q -th *noise-group* in $\lambda_{g,h}$ is

$$H_{g,h,q} = \{r_q, \dots, r_{q+1}-1\}, \quad q = 1, \dots, n_{g,h}.$$

Boundary *noise-groups* can be shorter than g , while interior *noise-groups* have size g . The systematic layout set is

$$\Lambda_{\text{sys}} = \{\lambda_{g,h} : 2 \leq g \leq G_{\text{max}}, 0 \leq h < g\}.$$

Ignoring boundary effects, every consecutive *running-set* $\{a, \dots, a+g-1\}$ of length g appears as one *noise-group* in exactly one shifted layout by choosing $h = (a-1) \bmod g$. Equivalently, for each fixed g , every block appears once at every group-relative position across the g shifts. The number of systematic layouts is therefore

$$|\Lambda_{\text{sys}}| = \sum_{g=2}^{G_{\text{max}}} g = \frac{(G_{\text{max}}+2)(G_{\text{max}}-1)}{2}. \quad (21)$$

Random layouts. Systematic layouts provide structured coverage but are regular by construction. To increase layout diversity, MultiTF further samples N_{rand} random layouts. For each random layout, we sequentially draw group sizes

$$g_m \sim \text{Uniform}\{2, \dots, G_{\text{max}}\}$$

and form consecutive groups

$$H_m = \{a_m, \dots, \min(a_m + g_m - 1, K)\},$$

$$a_{m+1} = \min(a_m + g_m, K + 1),$$

until the full block sequence is covered. These random layouts add non-regular *noise-group*-size combinations and boundary patterns without replacing the coverage guarantee of systematic layouts. The total number of layout variants per clean sequence is

$$|\Lambda| = \frac{(G_{\text{max}} + 2)(G_{\text{max}} - 1)}{2} + N_{\text{rand}}. \quad (22)$$

All layouts are batched as independent input sequences during post-training. This increases the effective number of training states per clean sample, but also increases training cost; exact settings are reported in Table 11. A theoretical coverage view is provided in Appendix A.

B.2 Chain-uniform Noise-Scheduler

For each *noise-group* $H_m = (j_1, \dots, j_{n_m})$, MultiTF applies the chain-uniform *noise-scheduler* used in Algorithm 1. We first define an effective upper bound

$$t_{\text{eff}} = t_{\text{high}} - \rho(t_{\text{high}} - t_{\text{low}}), \quad (23)$$

where ρ is the noise-transition margin ratio, corresponding to `noise_transition_margin_ratio` in the implementation. This parameter is independent of the random *noise-scheduler* power-law bias γ_{rand} , which is used only for the random *noise-scheduler* ablation.

For each group, a group-level floor ℓ is first sampled from the lower part of the noise range. Then each block samples its mask ratio from the interval between the current floor and the effective upper bound, and the sampled ratio becomes the floor for the next block:

$$\ell_0 \sim \mathcal{U}(t_{\text{low}}, t_{\text{eff}}),$$

$$t_{j_i} \sim \mathcal{U}(\ell_{i-1}, t_{\text{eff}}), \quad (24)$$

$$\ell_i \leftarrow t_{j_i}, \quad i = 1, \dots, n_m.$$

This construction produces monotonic but randomized slot-wise mask ratios inside each *noise-group*. Compared with the fixed-step D2F schedule over a long noisy sequence, the resulting groups have larger and more variable block-level noise-ratio gaps, matching the heterogeneous active blocks observed during MultiBD inference.

For each block with mask ratio t_{j_i} , MultiTF replaces $\lfloor B \cdot t_{j_i} \rfloor$ randomly selected token positions in $\mathbf{b}_{j_i}$ with $[\mathbf{M}]$. For a layout λ , the resulting noisy sequence is denoted as $\mathbf{x}_t^\lambda$.

B.3 Group-Aware Dual-Stream Mask

Following the TF-style construction of Block Diffusion, MultiTF concatenates the noisy and clean sequences into the input sequence

$$\mathbf{X}_\lambda = [\mathbf{x}_t^\lambda; \mathbf{x}_0]. \quad (25)$$

The attention mask has the block form

$$\mathbf{A}_\lambda = \begin{bmatrix} \mathbf{M}_{\text{GD}} & \mathbf{M}_{\text{GOC}} \\ 0 & \mathbf{M}_{\text{BC}} \end{bmatrix}, \quad (26)$$

where $\mathbf{M}_{\text{GD}}$ is the group-aware diagonal mask on the noisy part, $\mathbf{M}_{\text{GOC}}$ is the group-aware offset-causal mask from noisy tokens to clean tokens, and $\mathbf{M}_{\text{BC}}$ is the standard block-causal mask on the clean part.

Let $\mathcal{N}_\lambda$ and $\mathcal{C}$ denote token positions in the noisy and clean parts, respectively. Let $g(i)$ be the *noise-group* index of token i , $\beta(i)$ be its block index, and $\alpha(i)$ be the first block index of the *noise-group* containing i . The three masks are indicators of the following conditions:

$$\begin{aligned} \text{GD}_{ij} &: i, j \in \mathcal{N}_\lambda, g(i) = g(j), \beta(j) \leq \beta(i), \\ \text{GOC}_{ij} &: i \in \mathcal{N}_\lambda, j \in \mathcal{C}, \beta(j) < \alpha(i), \\ \text{BC}_{ij} &: i, j \in \mathcal{C}, \beta(j) \leq \beta(i). \end{aligned} \quad (27)$$

That is, $[\mathbf{M}_u]_{ij} = \mathbf{1}[u_{ij}]$ for $u \in \{\text{GD}, \text{GOC}, \text{BC}\}$; all other entries are zero. Thus, noisy tokens can attend to same-*noise-group* noisy tokens from the same or preceding blocks, each *noise-group* can condition on clean prefix blocks before it, and clean tokens never attend to noisy tokens. This implements the visibility pattern required by Equation 6 without information leakage.

B.4 MultiTF Objective and Model-specific Training Recipes

MultiTF defines the training-state construction: the layout λ , the noisy sequence $\mathbf{x}_t^\lambda$, the clean

Algorithm 2 DMax OPUT Self-Denoising Branch

Require: Model θ ; input sequence $\mathbf{X}_\lambda = [\mathbf{x}_t^\lambda; \mathbf{x}_0]$; noisy length N ; mask token id m .

- 1: Run a no-gradient forward pass on the noisy part: $\mathbf{L} \leftarrow \theta(\mathbf{X}_\lambda)_{:N}$.
- 2: Compute argmax predictions $\hat{\mathbf{x}} \leftarrow \arg \max \mathbf{L}$.
- 3: Replace masked positions in $\mathbf{x}_t^\lambda$ with $\hat{\mathbf{x}}$.
- 4: **return** the partially self-denoised input sequence.

sequence $\mathbf{x}_0$, and the *Group-Aware Dual-Stream Mask* $\mathbf{A}_\lambda$. Different base BD-LMs can reuse the same MultiTF input sequences while keeping their own model-specific training recipes.

B.4.1 Default MultiTF CE Objective

The default MultiTF objective is masked-token cross-entropy on masked positions in the noisy part of $\mathbf{X}_\lambda$. Let

$$\mathcal{M}_\lambda = \{i : \mathbf{x}_t^\lambda[i] = [\mathbf{M}]\} \quad (28)$$

denote the masked positions. The objective is

$$\mathcal{L}_{\text{MultiTF}}(\theta) = -\mathbb{E}_{\lambda, \mathbf{t}, \mathbf{x}_0} \left[\frac{1}{|\mathcal{M}_\lambda|} \sum_{i \in \mathcal{M}_\lambda} \ell_{\theta, i} \right], \quad (29)$$

$$\ell_{\theta, i} = \log p_\theta(x_0^i | \mathbf{X}_\lambda, \mathbf{A}_\lambda).$$

This objective is used for BD-LMs whose original training recipe is standard masked-token CE.

B.4.2 DMax-enhanced Models: OPUT Self-denoising

For DMax-enhanced models, we keep the same MultiTF input sequences and add the DMax OPUT self-denoising branch. For each MultiTF input sequence, OPUT forms two branches. The standard branch computes the training loss on the original noisy input sequence. The self-denoising branch first runs a no-gradient forward pass, replaces masked positions in the noisy part with the model’s argmax predictions, and then computes the loss on this partially self-denoised input. Gradients flow only through the second forward pass of the self-denoising branch. This exposes the model to partially self-generated states while keeping the MultiTF layout and attention-mask construction unchanged. The procedure is summarized in Algorithm 2.

B.4.3 SDAR Models: Block-wise Noise-weighted CE

For SDAR models, we also reuse the same MultiTF input sequences and *Group-Aware Dual-Stream Masks*. The difference lies in the loss

normalization. Instead of computing one global masked-token CE over all masked positions, SDAR applies a block-wise noise-weighted CE, where the loss of each block is normalized by the mask ratio applied to that block.

Let $\mathcal{B}_k$ denote token positions of block k in the noisy part, and let

$$\mathcal{M}_{\lambda, k} = \mathcal{M}_\lambda \cap \mathcal{B}_k \quad (30)$$

be the masked positions in block k under layout λ . Let $t_{\lambda, k}$ denote the mask ratio assigned to block k . The SDAR-style MultiTF objective is

$$\mathcal{L}_{\text{MultiTF}}^{\text{SDAR}}(\theta) = -\mathbb{E}_{\lambda, \mathbf{t}, \mathbf{x}_0} \left[\frac{1}{K} \sum_{k=1}^K \ell_{\lambda, k} \right], \quad (31)$$

$$\ell_{\lambda, k} = \frac{1}{\max(t_{\lambda, k}, \epsilon)} \sum_{i \in \mathcal{M}_{\lambda, k}} \ell_{\theta, i},$$

$$\ell_{\theta, i} = \log p_\theta(x_0^i | \mathbf{X}_\lambda, \mathbf{A}_\lambda),$$

where ϵ is a small constant used for numerical stability. This block-wise normalization extends the diffusion loss to the full block sequence while preserving the per-block noise weighting used by SDAR. It differs from Equation 29, which normalizes the loss globally over all masked positions in the noisy part.

B.5 Sorted-uniform Scheduler Baseline

The sorted-uniform *noise-scheduler* is a baseline for constructing monotonic block-level noise within each *noise-group*. For a *noise-group* $H_m = (j_1, \dots, j_{n_m})$, it independently samples n_m mask ratios from a uniform distribution and then sorts them in ascending order before assigning them to the blocks in the *noise-group*, as summarized in Algorithm 3:

$$u_1, \dots, u_{n_m} \stackrel{\text{i.i.d.}}{\sim} \mathcal{U}(t_{\text{low}}, t_{\text{high}}),$$

$$u_{(1)} \leq \dots \leq u_{(n_m)},$$

$$t_{j_i} = u_{(i)}, \quad i = 1, \dots, n_m.$$

This produces a monotonic noise pattern similar in spirit to D2F. However, unlike the chain-uniform *noise-scheduler* in Appendix B.2, the gaps between adjacent slots are only induced by order statistics of uniformly sampled values and are not explicitly encouraged to be large.

Algorithm 3 Sorted-uniform Block-level Noise-Scheduler

Require: Noise-group $H_m = (j_1, \dots, j_{n_m})$; noise bounds $t_{\text{low}}, t_{\text{high}}$.

- 1: **for** $i \leftarrow 1$ **to** n_m **do**
- 2: Sample $u_i \sim \mathcal{U}(t_{\text{low}}, t_{\text{high}})$.
- 3: **end for**
- 4: Sort sampled ratios: $u_{(1)} \leq \dots \leq u_{(n_m)}$.
- 5: **for** $i \leftarrow 1$ **to** n_m **do**
- 6: Assign $t_{j_i} \leftarrow u_{(i)}$.
- 7: **end for**
- 8: **return** block-level mask ratios $\{t_{j_i}\}_{i=1}^{n_m}$.

C MultiBD Inference Implementation Details

This appendix expands the optimized MultiBD inference algorithm introduced in Section 3.4. The main design goal is to execute the MultiBD *running-set* in Equation 6 with a static physical input shape, while preserving prefix cache reuse.

C.1 A dynamic *running-set* prevents static-shape execution.

A direct implementation of MultiBD maintains a dynamic *running-set* in addition to the committed prefix cache. When the latest active block reaches an add-block threshold, the decoder appends a fully masked future block to the *running-set*. When the front active block is completed, the decoder writes it into the KV cache and removes it from the *running-set*. This dynamic procedure exposes inter-block parallelism, but the number of active tokens changes across decoding steps and across requests. As shown in Figure 6a(2), such shape variation is unfriendly to CUDA Graph capture and replay.

C.2 A fixed *Block Buffer* implements MultiBD states.

Optimized MultiBD replaces dynamic appending with a fixed-size *Block Buffer*. The *Block Buffer* contains N_{buf} physical block slots. At each decoding step, active slots represent the logical *running-set* $\mathcal{R}_s$, while dummy slots reserve capacity for future blocks. Adding a future block therefore activates an existing dummy slot instead of extending the physical input sequence. When the front active block is completed, it is committed to the KV cache, removed from the *running-set*, and the *Block Buffer* slides forward by replacing the consumed slot with a new dummy slot at the tail. This realizes MultiBD while keeping the number of processed buffer tokens fixed at $N_{\text{buf}} \cdot B$.

Algorithm 4 Naive MultiBD with a Dynamic Running-Set

Require: Model θ ; block size B ; thresholds $\tau_{\text{add}}, \tau_{\text{semi}}$, τ_{M2T} .

// Initialize dynamic MultiBD state

- 1: Initialize prefix KV cache $\mathcal{K} \leftarrow \emptyset$ and dynamic *running-set* $\mathcal{Y} \leftarrow \emptyset$.
- 2: Append one fully masked ACTIVE block to $\mathcal{Y}$.
- 3: **while** generation is not complete **do**
 // Grow the running-set dynamically
 4: **if** the latest active block has progress $> \tau_{\text{add}}$ and EOS has not appeared **then**
 Append a fully masked future block to $\mathcal{Y}$.
 end if
 // Decode all blocks in the current running-set
 7: Run θ on $\mathcal{Y}$ with prefix cache $\mathcal{K}$.
 8: **for** each active block $b \in \mathcal{Y}$ **do**
 9: Accept masked positions with confidence $> \tau_{\text{M2T}}$.
 10: **if** the previous active block is semi-complete and no token is accepted **then**
 11: Accept the highest-confidence masked position.
 12: **end if**
 13: **if** b is fully decoded **then**
 14: Mark b as TO-CACHE.
 15: **end if**
 16: **end for**
 // Commit completed prefix blocks
 17: **while** the front block of $\mathcal{Y}$ is TO-CACHE **do**
 18: Write the front block into $\mathcal{K}$ and remove it from $\mathcal{Y}$.
 19: **end while**
20: **end while**
21: **return** generated tokens.

C.3 Block states advance the fixed *Block Buffer*.

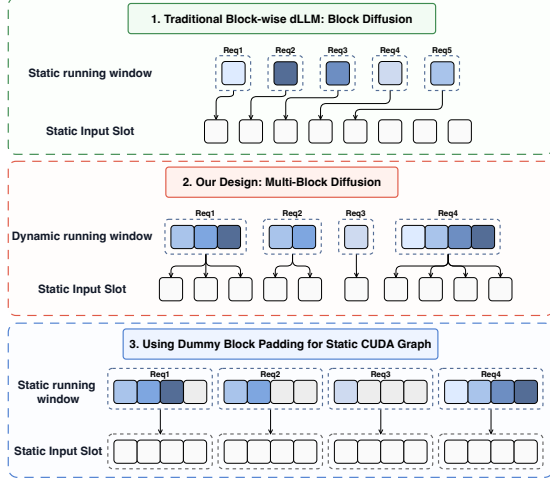
Each physical slot in the *Block Buffer* follows the transition

$$\text{DUMMY} \rightarrow \text{ACTIVE} \rightarrow \text{TO-CACHE} \rightarrow \text{IN-CACHE}.$$

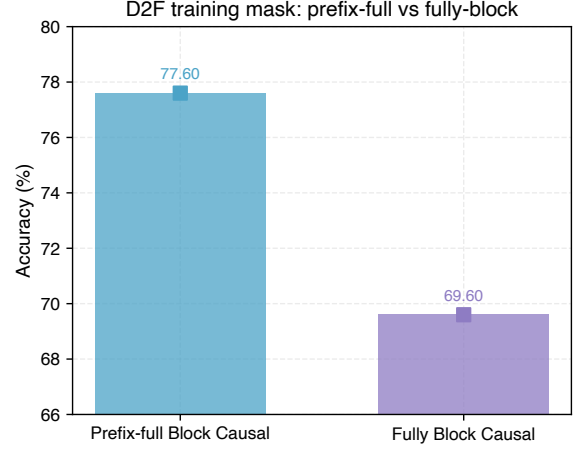
A DUMMY slot is an idle placeholder that preserves the static buffer shape. An ACTIVE slot participates in the current MultiBD forward pass. A TO-CACHE block has completed decoding and is ready to be committed. An IN-CACHE block has been written into the prefix KV cache and no longer belongs to the active part of the *running-set*. These state transitions implement the logical evolution of $\mathcal{R}_s$ without changing the input shape.

C.4 Thresholds control activation and token updates.

MultiBD uses separate thresholds for block activation, fallback progress, and token updates. The add-block threshold τ_{add} controls when a future



(a) CUDA Graph compatibility across decoding designs. (1) SingleBD uses a fixed single active block but exposes no inter-block parallelism. (2) Naive MultiBD appends future blocks dynamically, making the *running-set* length change over time. (3) Optimized MultiBD maps the logical *running-set* into a fixed-size *Block Buffer* with dummy slots, keeping tensor shapes static for CUDA Graph capture and replay.



(b) Making D2F fully block-causal hurts accuracy. Prefix-full attention gives D2F stronger noisy-prefix visibility but is not naturally compatible with prefix KV caching. Directly replacing it with a fully block-causal mask improves cache compatibility but drops accuracy from 77.60% to 69.60%.

Figure 6: Static-shape execution and prefix-cache compatibility analyses. Left: optimized MultiBD keeps tensor shapes static through a fixed-size *Block Buffer*, enabling CUDA Graph capture and replay. Right: making D2F fully block-causal improves cache compatibility but substantially hurts accuracy.

block can enter the fixed *Block Buffer*. The stability threshold τ_{stable} prevents premature activation when the current latest active block is still unstable. The semi-completion threshold τ_{semi} allows later active blocks to use the top-1 context of a preceding block once it has made sufficient progress, even before it is fully cached. The M2T threshold τ_{M2T} controls mask-to-token acceptance, and the T2T threshold τ_{T2T} controls token-to-token revision for models that support T2T updates.

This separation is important because M2T and T2T updates have different reliability profiles. M2T introduces new content into an active block, while T2T overwrites tentative content before commitment. Using separate thresholds stabilizes concurrent block refinement and reduces error propagation across the *running-set*.

C.5 Prefix Caching and Fully Block-Causal D2F

Native D2F is not directly prefix-cache compatible. Prefix caching is a key advantage of BD-LMs. In SingleBD, completed blocks form an immutable clean prefix, so their KV states can be stored and directly reused in later steps. As shown in Figure 7(1), only the current noisy block requires repeated computation. By contrast, native

D2F uses prefix-full attention. Future noisy blocks condition on a prefix-full context, where prefix states are not organized as immutable block-causal prefix pages in the standard BD-LM cache. As illustrated in Figure 7(2), their KV states cannot be reused in the same way as SingleBD prefix blocks.

Fully block-causal D2F variant. To isolate the prefix-caching issue, we construct a fully block-causal D2F variant. Let the full clean sequence be partitioned into BD-LM blocks:

$$\mathbf{x}_0 = [\mathbf{b}_1, \dots, \mathbf{b}_K], \quad \mathbf{b}_k \in \mathcal{V}^B.$$

Suppose native D2F uses a token-level clean prefix

$$\mathbf{x}_0^{\text{pre}} = (x_0^1, \dots, x_0^P),$$

where P can be arbitrary and need not be divisible by B . Let

$$a = \left\lfloor \frac{P}{B} \right\rfloor + 1, \quad r = P - (a - 1)B$$

denote the first block that contains suffix tokens and the number of prefix tokens inside this boundary block, respectively. Then $\mathbf{b}_1, \dots, \mathbf{b}_{a-1}$ are complete clean prefix blocks, while $\mathbf{b}_a$ may contain both prefix tokens and suffix tokens.

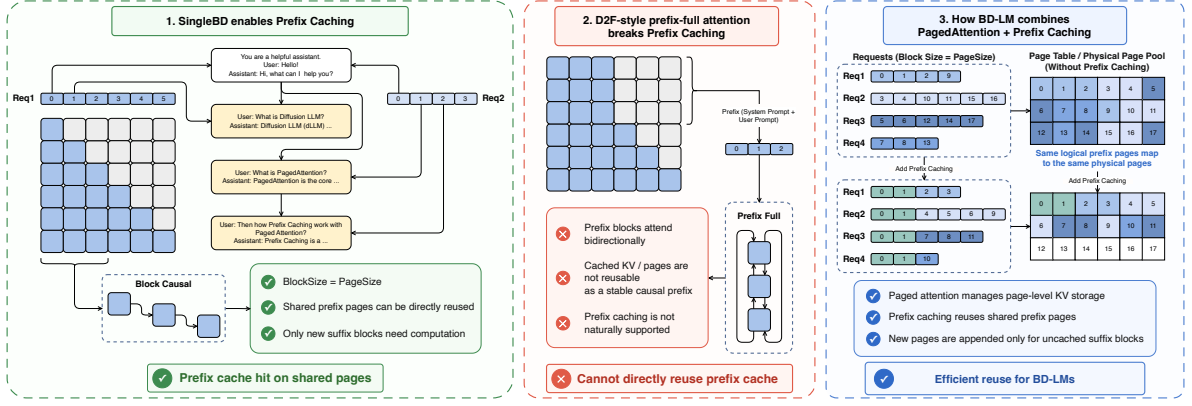


Figure 7: Prefix caching in block-causal BD-LMs. (1) SingleBD keeps completed blocks as an immutable clean prefix, enabling direct KV-cache reuse. (2) D2F-style prefix-full attention breaks this cache semantics because noisy prefix blocks are not reusable as stable causal prefix pages. (3) *Block Buffer* MultiBD separates cached prefix blocks from active *Block Buffer* slots, enabling prefix KV reuse while refining multiple active blocks.

We use $\mathbf{b}_a$ as the first noisy block of the D2F-style suffix, rather than inserting padding tokens. For the boundary block, only its suffix positions are noised and included in the loss:

$$\mathcal{I}_a = \{r + 1, \dots, B\}.$$

For later blocks $j > a$, all positions belong to the suffix blocks:

$$\mathcal{I}_j = \{1, \dots, B\}.$$

We then apply a monotonic D2F-style *noise-scheduler* to the valid suffix positions of blocks $a, \dots, K$:

$$0 \leq t_a < t_{a+1} < \dots < t_K \leq 1.$$

Let $\bar{\mathbf{b}}_{j,t_j}$ denote the partially corrupted block, where positions in $\mathcal{I}_j$ are corrupted by $q_{t_j}(\cdot | \mathbf{b}_j)$ and positions outside $\mathcal{I}_j$ are kept clean. For the boundary block, this means that the prefix part of $\mathbf{b}_a$ remains clean, while the suffix part is noised.

The fully block-causal D2F variant factorizes the suffix as

$$\begin{aligned} \mathbf{c}_{j,t} &= (\mathbf{x}_0^{(<a)}, \bar{\mathbf{b}}_{a,t_a}, \dots, \bar{\mathbf{b}}_{j,t_j}), \\ p_\theta(\mathbf{x}_0^{\text{sup}} | \mathbf{c}_{K,t}) &= \prod_{j=a}^K p_\theta(\mathbf{b}_j^{\mathcal{I}_j} | \mathbf{c}_{j,t}), \end{aligned} \quad (32)$$

where $\mathbf{x}_0^{(<a)} = [\mathbf{b}_1, \dots, \mathbf{b}_{a-1}]$ is the block-causal clean prefix and $\mathbf{b}_j^{\mathcal{I}_j}$ denotes the suffix positions of block j . The loss is computed only on masked positions within $\mathcal{I}_j$.

Compared with native D2F, this variant changes the training-state construction by replacing prefix-full attention with a fully block-causal attention.

Equivalently, it completes the arbitrary token-level prefix to the next block boundary using real continuation tokens from the training sequence, and treats the boundary block as the first block in the noisy suffix. This construction is the training-side counterpart of the extreme MBD-LM state discussed in Section 3.2, where the *running-set* covers all suffix blocks and follows a monotonic D2F-style *noise-scheduler*.

Fully block-causal D2F is not a sufficient fix.

Although the fully block-causal variant improves cache compatibility, it substantially hurts accuracy. As shown in Figure 6b, changing D2F from prefix-full attention to fully block-causal attention drops accuracy from 77.60% to 69.60%. This suggests that D2F relies on stronger prefix-full visibility, and cache compatibility cannot be obtained by simply restricting the attention mask. This result further motivates MultiTF, which keeps the block-causal cached-prefix interface while training on bounded *noise-groups* that better match MultiBD.

Block Buffer MultiBD preserves cache semantics.

Our *Block Buffer* MultiBD design preserves prefix caching by construction. Committed blocks become immutable IN-CACHE prefix context and are represented only through cached KV states. Active blocks remain inside the *Block Buffer* and are recomputed during iterative refinement, while future DUMMY slots remain invisible until activated. As shown in Figure 7(3), this separates cached prefix blocks from active *Block Buffer* slots, enabling prefix KV reuse while still refining multiple active blocks in parallel.

Algorithm 5 Optimized MultiBD with a Fixed Block Buffer

Require: Model θ ; block size B ; buffer size N_{buf} ; thresholds τ_{add} , τ_{semi} , τ_{stable} , τ_{M2T} , and optional τ_{T2T} .
// Initialize fixed Block Buffer

- 1: Initialize prefix KV cache $\mathcal{K}$ and a fixed *Block Buffer* $\mathcal{W}$ with N_{buf} slots.
- 2: Set $\mathcal{W}[0]$ to a fully masked ACTIVE block and all remaining slots to DUMMY.
- 3: **while** generation is not complete **do**
// Activate future blocks without changing shape
- 4: Let $\mathcal{R}$ be the non-DUMMY resident blocks in $\mathcal{W}$.
- 5: Let b_{last} be the last ACTIVE block in $\mathcal{R}$.
- 6: **if** b_{last} satisfies progress $> \tau_{\text{add}}$ and stability $> \tau_{\text{stable}}$ **then**
- 7: Activate the first trailing DUMMY slot if one exists.
- 8: **end if**
// Decode the static Block Buffer
- 9: Run θ on the static $N_{\text{buf}} \cdot B$ *Block Buffer* tokens with prefix cache $\mathcal{K}$.
- 10: **for** each ACTIVE block $b \in \mathcal{W}$ **do**
- 11: Accept masked positions with confidence $> \tau_{\text{M2T}}$.
- 12: **if** no masked position is accepted and the preceding active block is semi-complete **then**
- 13: Accept the highest-confidence masked position in b .
- 14: **end if**
- 15: **if** T2T revision is enabled **then**
- 16: Revise eligible filled but uncommitted positions with confidence $> \tau_{\text{T2T}}$.
- 17: **end if**
- 18: **if** b is complete and all preceding resident blocks are cached or ready-to-cache **then**
- 19: Mark b as TO-CACHE.
- 20: **end if**
- 21: **end for**
// Commit prefix blocks and slide the buffer
- 22: **while** the front slot of $\mathcal{W}$ is TO-CACHE **do**
- 23: Write the front block into $\mathcal{K}$; its state becomes IN-CACHE.
- 24: Pop the front slot and append a new DUMMY slot at the tail.
- 25: **end while**
- 26: **end while**
- 27: **return** generated tokens.

D Additional Rebuttal Experiments and Analysis

D.1 Block-Boundary Failure Taxonomy

Table 8 decomposes failures under the aggressive $\tau_{\text{add}} = 0.1$ setting from Table 6. We classify each non-passing program by its first observed outcome: a syntax/parsing failure, a runtime failure after successful parsing, or an executable but incorrect output. Syntax errors dominate on both datasets, consistent with the hypothesis that activating a future block before the preceding block stabilizes disrupts cross-block program structure.

For MBPP+, one sample exhibited a worker/evaluator inconsistency and is reported

separately rather than assigned to a semantic failure category. The default $\tau_{\text{add}} = 0.9$ is fixed across these code evaluations; no per-instance threshold selection is performed.

D.2 Per-Benchmark Throughput

Table 9 gives the per-benchmark measurements underlying Table 7. TPS is rounded to the nearest integer to avoid implying unsupported measurement precision; relative gains in the main paper are computed from the unrounded measurements.

D.3 Roofline View of MultiBD

At decoding step s , let P_s be the cached-prefix length and let

$$Q_s = |\mathcal{W}_s|B = N_{\text{buf}}B \quad (33)$$

be the number of physical tokens processed by the fixed *Block Buffer* $\mathcal{W}_s$. SingleBD has $N_{\text{buf}} = 1$, whereas MultiBD processes active, resident, and dummy slots to preserve a static shape. The useful-token fraction is

$$\eta_{\text{tok}}(s) = \frac{\text{TPF}_s}{Q_s}, \quad \text{TPS} = \frac{\eta_{\text{tok}}Q_s}{T_{\text{step}}}. \quad (34)$$

Thus, additional buffer work improves throughput only when the useful-token gain exceeds the extra step cost.

For N_{layer} transformer layers, hidden size d , feed-forward size d_{ff} , and vocabulary $\mathcal{V}$, an extend-attention step has approximate compute

$$\begin{aligned} \mathcal{F}_{\text{step}} &= \Theta(\mathcal{F}_{\text{proj}} + \mathcal{F}_{\text{attn}} + \mathcal{F}_{\text{vocab}}), \\ \mathcal{F}_{\text{proj}} &= N_{\text{layer}}Q_s(d^2 + dd_{\text{ff}}), \\ \mathcal{F}_{\text{attn}} &= N_{\text{layer}}d(Q_sP_s + Q_s^2), \\ \mathcal{F}_{\text{vocab}} &= Q_sd|\mathcal{V}|. \end{aligned} \quad (35)$$

The terms respectively represent projections/feed-forward layers, attention to the prefix and current buffer, and the language-model head. With s_{dtype} bytes per element, weight traffic and KV traffic scale as

$$\mathcal{M}_W = \Theta(N_{\text{layer}}s_{\text{dtype}}(d^2 + dd_{\text{ff}})), \quad (36)$$

$$\mathcal{M}_{\text{KV}}(Q_s, P_s) = \Theta(N_{\text{layer}}s_{\text{dtype}}d\kappa_s), \quad (37)$$

$$\kappa_s = \rho(Q_s)(P_s + Q_s) + Q_s, \quad (38)$$

where $\rho(Q_s)$ represents repeated KV reads induced by query tiling. The roofline approximation

Table 8: Failure taxonomy for aggressive block activation ($\tau_{\text{add}} = 0.1$). Dataset percentages use all samples; parenthesized percentages use only classified failures. Counts and thresholds are exact.

Benchmark	n	Pass	Syntax	Runtime	Incorrect output	Evaluator issue
HumanEval+	164	64 (39.0%)	74 (45.1%; 74.0%)	15 (9.1%; 15.0%)	11 (6.7%; 11.0%)	0
MBPP+	378	178 (47.1%)	127 (33.6%; 63.5%)	28 (7.4%; 14.0%)	44 (11.6%; 22.0%)	1

Table 9: Full throughput results for single-sample decoding on two H100 GPUs with tensor parallelism 2.

Model	TPF	ms/step	GSM8K TPS	MATH500 TPS	MBPP+ TPS	HumanEval+ TPS	Avg. TPS
LLaDA2-Mini	3.47	7.07	344	403	496	825	517
MBD-LLaDA2-Mini	6.19	8.78	688	708	647	941	746
LLaDA2-Mini-DMax	6.35	9.02	701	731	755	932	779
MBD-LLaDA2-Mini-DMax	9.34	11.2	835	851	897	1124	927

is defined as

$$T_{\text{roof}} = \max\left(\frac{\mathcal{F}_{\text{step}}}{\Pi_{\text{eff}}}, \frac{\mathcal{M}_W + \mathcal{M}_{\text{KV}}}{\mathcal{B}_{\text{HBM}}}\right), \quad (39)$$

$$T_{\text{step}} \approx T_{\text{roof}} + T_{\text{comm}}(Q_s) + T_{\text{launch}}.$$

Here Π_{eff} and $\mathcal{B}_{\text{HBM}}$ are effective compute and HBM bandwidth. When $P_s \gg Q_s$, attention arithmetic intensity satisfies

$$\text{AI}_{\text{attn}} \approx \frac{dQ_s P_s}{s_{\text{dtype}} \rho(Q_s) P_s d} = \Theta\left(\frac{Q_s}{s_{\text{dtype}} \rho(Q_s)}\right). \quad (40)$$

Increasing Q_s can therefore amortize prefix reads, weights, and launch overhead, but resident and dummy slots reduce η_{tok} . This balance explains why the measured TPF gains of $1.78\times$ and $1.47\times$ become TPS gains of $1.44\times$ and $1.19\times$.

E Experimental Details

This appendix reports the inference and MultiTF post-training hyperparameters used in our experiments. “—” indicates that the corresponding hyperparameter is not applicable. SingleBD (Native) denotes the original single-block inference of each BD-LM; MultiBD (training-free) denotes MultiBD inference without post-training; MBD-* denotes the corresponding MultiTF-trained model.

Table 10: Inference hyperparameters for all evaluated configurations. τ_{add} controls when a future block is activated; τ_{semi} controls semi-completion or fallback progress; τ_{stable} controls activation stability; τ_{M2T} and τ_{T2T} are confidence thresholds for mask-to-token filling and token-to-token revision.

Configuration	Task	Buffer	Block	Max Len	Max New	Max NFE	τ_{add}	τ_{semi}	τ_{stable}	τ_{M2T}	τ_{T2T}
LLaDA2-Mini-DMax											
SingleBD (Native)	Math	1	32	4096	4096	1024	—	—	—	0.50	—
SingleBD (Native)	Code	1	32	4096	4096	1024	—	—	—	0.65	—
MultiBD (training-free)	Math	2	32	4096	4096	1024	0.10	0.90	0.50	0.50	—
MultiBD (training-free)	Code	2	32	4096	4096	1024	0.90	0.90	0.50	0.65	—
MBD-LLaDA2-Mini-DMax	Math	2	32	4096	4096	1024	0.10	0.90	0.50	0.50	—
MBD-LLaDA2-Mini-DMax	Code	2	32	4096	4096	1024	0.90	0.90	0.50	0.65	—
LLaDA2-Mini											
SingleBD (Native)	Math	1	32	4096	4096	1024	—	—	—	0.95	—
SingleBD (Native)	Code	1	32	4096	4096	1024	—	—	—	0.95	—
MultiBD (training-free)	Math	2	32	4096	4096	1024	0.10	0.90	—	0.95	—
MultiBD (training-free)	Code	2	32	4096	4096	1024	0.90	0.90	—	0.95	—
MBD-LLaDA2-Mini	Math	2	32	4096	4096	1024	0.10	0.90	—	0.95	—
MBD-LLaDA2-Mini	Code	2	32	4096	4096	1024	0.90	0.90	—	0.95	—
SDAR-8B-Chat-b32											
SingleBD (Native)	Math	1	32	4096	4096	1024	—	—	—	0.95	—
SingleBD (Native)	Code	1	32	4096	4096	1024	—	—	—	0.95	—
MultiBD (training-free)	Math	4	32	4096	4096	1024	0.10	0.90	—	0.95	—
MultiBD (training-free)	Code	4	32	4096	4096	1024	0.90	0.90	—	0.95	—
MBD-SDAR-8B-Chat-b32	Math	4	32	4096	4096	1024	0.10	0.90	—	0.95	—
MBD-SDAR-8B-Chat-b32	Code	4	32	4096	4096	1024	0.90	0.90	—	0.95	—
SDAR-8B-Chat-b4											
SingleBD (Native)	Math	1	4	4096	4096	1024	—	—	—	0.95	—
SingleBD (Native)	Code	1	4	4096	4096	1024	—	—	—	0.95	—
MultiBD (training-free)	Math	4	4	4096	4096	1024	0.10	0.25	—	0.95	—
MultiBD (training-free)	Code	4	4	4096	4096	1024	0.75	0.75	—	0.95	—
MBD-SDAR-8B-Chat-b4	Math	4	4	4096	4096	1024	0.10	0.25	—	0.95	—
MBD-SDAR-8B-Chat-b4	Code	4	4	4096	4096	1024	0.75	0.75	—	0.95	—
LLaDA2-Mini-CAP											
SingleBD (Native)	Math	1	32	4096	4096	1024	—	—	—	0.95	—
SingleBD (Native)	Code	1	32	4096	4096	1024	—	—	—	0.95	—
MultiBD (training-free)	Math	2	32	4096	4096	1024	0.10	0.90	—	0.95	—
MultiBD (training-free)	Code	2	32	4096	4096	1024	0.90	0.90	—	0.95	—
LLaDA2.1-Mini											
SingleBD (Native)	Math	1	32	4096	4096	1024	—	—	—	0.70	0.50
SingleBD (Native)	Code	1	32	4096	4096	1024	—	—	—	0.70	0.50
MultiBD (training-free)	Math	2	32	4096	4096	1024	0.10	0.90	—	0.70	0.50
MultiBD (training-free)	Code	2	32	4096	4096	1024	0.90	0.90	—	0.70	0.50

Table 11: MultiTF post-training hyperparameters. t_{low} and t_{high} denote the mask-ratio range; ρ is the margin ratio used to determine the effective upper bound t_{eff} ; N_{rand} is the number of random *group-layouts* per sample. The random-scheduler ablation uses a separate power-law bias γ_{rand} , which is independent of ρ and is not used in the chain-uniform scheduler.

Target Model	Task	Objective	Data	Seq Len	Block	Max Group	t_{low}	t_{high}	ρ	N_{rand}	Steps
MBD-LLaDA2-Mini-DMax	Math	MultiTF + DMax OPUT	60k	2048	32	2	0.001	1.00	ρ_{cfg}	0	15000
MBD-LLaDA2-Mini-DMax	Code	MultiTF + DMax OPUT	60k	2048	32	2	0.001	1.00	ρ_{cfg}	2	4000
MBD-LLaDA2-Mini	Math	MultiTF CE	60k	2048	32	2	0.001	1.00	ρ_{cfg}	0	15000
MBD-LLaDA2-Mini	Code	MultiTF CE	60k	2048	32	2	0.001	1.00	ρ_{cfg}	0	6500
MBD-SDAR-8B-Chat-b32	Math	MultiTF CE	20k	2048	32	4	0.001	1.00	ρ_{cfg}	3	3125
MBD-SDAR-8B-Chat-b32	Code	MultiTF CE	10k	2048	32	4	0.001	1.00	ρ_{cfg}	3	1670
MBD-SDAR-8B-Chat-b4	Math	MultiTF CE	20k	2048	4	4	0.001	1.00	ρ_{cfg}	2	1250
MBD-SDAR-8B-Chat-b4	Code	MultiTF CE	10k	2048	4	4	0.001	1.00	ρ_{cfg}	2	200