Fast Decoding for Diffusion LLMs via Training-Free Self-Speculation

Ligong Han[1,2]* **Hao Wang**[1,2] **Han Gao**[3] **Kai Xu**[1,2] **Akash Srivastava**[2,4]
[1]Red Hat AI Innovation [2]MIT-IBM Watson AI Lab [3]Iowa State University [4]Core AI, IBM

Abstract

Block-diffusion language models offer a promising path toward faster-than-autoregressive generation by combining block-wise autoregressive decoding with within-block parallel denoising. However, in the few-step regime needed for practical acceleration, standard confidence-thresholded decoding is often brittle: aggressive thresholds hurt quality, while conservative thresholds require unnecessary denoising steps. Existing approaches that address this issue either require additional training or incur extra test-time compute. We present `S2D2`, a training-free self-speculative decoding framework for block-diffusion language models. Our key observation is that a block-diffusion model becomes autoregressive when the block size is reduced to one, allowing the same pretrained model to act as both drafter and verifier. `S2D2` inserts a speculative verification step into standard block-diffusion decoding and uses lightweight routing policies to decide when verification is worth its cost. This yields a hybrid decoding trajectory in which diffusion proposes tokens in parallel, while the autoregressive mode acts as a local sequence-level critic. Across three mainstream block-diffusion families, `S2D2` consistently improves the accuracy–speed tradeoff over strong confidence-thresholding baselines. On SDAR, we observe up to 4.7× speedup over autoregressive decoding, and up to 1.57× over a tuned dynamic decoding baseline while improving accuracy by up to 4.5 points. On LLaDA2.1-Mini, `S2D2` remains complementary to built-in self-correction, including a conservative setting where it is 4.4× faster than the static baseline with slightly higher accuracy. Code is available at `https://github.com/phymhan/S2D2`.

1 Introduction

Autoregressive (AR) models have driven recent progress in language modeling, especially on reasoning-heavy tasks (Vaswani et al., 2017; Brown et al., 2020; Touvron et al., 2023; Wei et al., 2022; Kojima et al., 2022; Jaech et al., 2024; Guo et al., 2025). Their strict left-to-right generation, however, limits decoding flexibility and inference parallelism. This has motivated diffusion-based language models, which offer a different generation paradigm with potential gains in controllability and speed (Hoogeboom et al., 2021; Shih et al., 2022; Nie et al., 2025; Li et al., 2022; Schiff et al., 2025; Rojas et al., 2026; Labs et al., 2025; Wang et al., 2025; He et al., 2026). Masked diffusion models (Austin et al., 2021a; Sahoo et al., 2024; Shi et al., 2024), first scaled in vision (Chang et al., 2022; Han et al., 2022), have now been extended to language and shown competitive quality (Lou et al., 2023; Gong et al., 2024; Nie et al., 2025; Ye et al., 2025; Chandrasegaran et al., 2025).

Practical acceleration still requires decoding in only a few denoising steps while preserving efficient Transformer inference (e.g., KV caching). Block diffusion (Arriola et al., 2025) combines block-wise AR generation (for cache reuse) with within-block diffusion updates (for parallelism), but few-step decoding remains difficult: the common mean-field, token-factorized parameterization (Xu et al., 2024; Yoo et al., 2025; Zhang et al., 2025; 2026) weakens sequence-level dependencies and can accumulate errors as steps decrease. Prior

*Correspondence to: Ligong Han (`lihan@redhat.com`)

work addresses this with explicit sequence-level modeling. EDLM (Xu et al., 2024), for example, introduces an AR energy model and uses self-normalized importance sampling to steer denoising. While effective, this adds training and inference overhead. We instead target a *speed-first* question: can we exploit AR structure at inference time, without extra training, while keeping block-diffusion parallelism?

To study this question, we introduce S2D2, a *training-free self-speculative decoding* framework for block-diffusion LMs. The key observation is that when block size is reduced to 1, a block-diffusion model becomes autoregressive and can serve as a verifier. We therefore use standard block-diffusion decoding as the drafter and block-size-1 decoding of the same model as the verifier, enabling self-speculation without distillation, auxiliary models, or architectural changes. Since verification adds one extra forward pass, we use lightweight routing policies to invoke it only when worthwhile.

Beyond efficiency, our motivation is also algorithmic. Confidence-threshold decoding can be brittle because acceptance relies on draft confidence alone. Speculative rejection sampling instead uses verifier-normalized acceptance (via the probability ratio), providing a stronger local test for committing drafted tokens. Our goal is not to exactly reproduce block-size-1 AR decoding; rather, we use AR verification as a local sequence-level critic inside a hybrid diffusion trajectory.

Empirically, this simple design is often both *faster* and *more accurate* than strong dynamic confidence-threshold baselines. Across three mainstream block-diffusion families, S2D2 improves the accuracy-speed frontier, especially in large-block regimes where standard diffusion decoding is unstable. AR-ness diagnostics further support the view that our verifier provides a stochastic, greedy AR-guided energy correction. Our contributions are as follows:

- We introduce, to the best of our knowledge, the first *training-free self-speculative decoding* method for block-diffusion language models by reusing the block-size-1 mode of the same model as a sequence-level verifier.
- We develop a practical framework with self-verification masks and lightweight routing policies, enabling plug-and-play acceleration for existing block-diffusion models without additional training.
- Through experiments on five models from three major block-diffusion families, we show that S2D2 often improves accuracy while also being faster than competitive dynamic confidence-threshold baselines.
- We provide analysis connecting S2D2 to AR-guided residual energy correction, interpreting speculative verification as a stochastic, greedy local preference for lower residual energy.

2 Related Work

AR-diffusion hybrid language models. A key challenge for diffusion LMs is combining parallel token updates with efficient Transformer inference. Block diffusion (BD3) (Arriola et al., 2025) is the first successful AR-diffusion hybrid to combine block-wise AR generation, within-block diffusion decoding, and KV caching, making few-step decoding practical. This design underlies recent block-diffusion LMs such as LLaDA 2.x (Bie et al., 2025; 2026) and SDAR (Cheng et al., 2025). Related hybrids include ReFusion (Li et al., 2025), which uses diffusion to plan low-dependency blocks for parallel AR decoding, and Esoteric Language Models (Sahoo et al., 2025), which combine any-order AR modeling with standard AR decoding. We focus on training-free inference-time acceleration for existing block-diffusion.

Speculative decoding and self-speculation. Speculative decoding uses a drafter and verifier with rejection sampling to accelerate generation while preserving the target model distribution (Leviathan et al., 2023; Chen et al., 2023). In AR models, Draft & Verify (Zhang et al., 2024) realizes self-speculation via a weakened version of the same model. In diffusion LMs, ASSD (Guo & Ermon, 2025) verifies arbitrary token subsets via any-subset AR modeling (Shih et al., 2022), but requires specific architectures (e.g., XLNet-style (Yang et al., 2019)) and is not plug-and-play for most pretrained diffusion LMs. SSD for diffusion

LMs (Gao et al., 2025) instead uses hierarchical batching over multiple prefix states. Our approach is speed-first and training-free: we reuse the existing block-size-1 AR mode of block-diffusion models for single-pass verification. Our routing policies are orthogonal and could be combined with batching-based SSD.

Self-correction and sequence-level correction in diffusion LMs. Beyond verifier-based rejection sampling, LLaDA2.1 (Bie et al., 2026) introduces token editing, which supports an "unmask early, correct later" strategy but does not perform verifier-based sequence-level acceptance. Our results show S2D2 is complementary to this mechanism. At a broader level, EDLM (Xu et al., 2024) and density-ratio-based discrete diffusion methods (Lou et al., 2023) also target sequence-level correction, but they rely on additional modeling or extra multi-sample inference. In contrast, we reuse the same pretrained block-diffusion model in AR mode as a local verifier, with no retraining.

3 Background

Block-wise autoregressive diffusion decoding. We use block-wise autoregressive generation with block size B. Given a prompt $x_{1:m}$, decoding proceeds one block at a time: initialize $x^b \leftarrow [\texttt{MASK}]^B$, decode it conditioned on the prompt and finalized blocks, and reuse their KV cache $(\mathbf{K}, \mathbf{V})$ (Algorithm 2). Let $M_t = \{i : x_i^b = [\texttt{MASK}]\}$ denote masked positions at diffusion step t.

Following masked absorbing-state diffusion, the reverse transition from noise level t to $s < t$ is

$$p_\theta(z_s \mid z_t) = q(z_s \mid z_t, x = x_\theta(z_t, t)), \tag{1}$$

where under the SUBS parameterization of MDLM (Sahoo et al., 2024), for a masked position $(z_t = m)$,

$$p_\theta(z_s \mid z_t = m) = \mathrm{Cat}\left(z_s;\ \frac{1-\alpha_s}{1-\alpha_t} m + \frac{\alpha_s - \alpha_t}{1-\alpha_t} x_\theta(z_t, t)\right). \tag{2}$$

Under SUBS, each masked position is independently unmasked with probability $\rho_{t\to s} = \frac{\alpha_s - \alpha_t}{1-\alpha_t}$ and, if unmasked, assigned a token sampled from $x_\theta(z_t, t)$.

In practice, LLaDA (Nie et al., 2025) uses few-step confidence-based decoding instead of directly sampling from (2): a draft pass produces token proposals $\hat{x}$ and confidences p from logits ℓ, then masked positions are accepted by a fixed schedule or dynamic threshold (Algorithm 2). Detailed discussion of this transition from MDLM posterior sampling to LLaDA confidence-based decoding is deferred to Appendix A.2.

Speculative decoding for autoregressive models. Speculative decoding (SD) speeds up autoregressive generation by letting a drafter propose multiple tokens and a verifier check them in parallel (Leviathan et al., 2023; Chen et al., 2023). If the draft assigns probability p_i to proposed token $\hat{x}_i$ and the verifier assigns q_i under the target AR distribution, tokens are scanned left-to-right and accepted with probability $\min\left(1, \frac{q_i}{p_i}\right)$. At the first rejection, we resample from the residual distribution $(P_{\ell^{\text{ver}}} - P_\ell)_+$ and end the speculative segment. The procedure preserves the target AR distribution while often accepting multiple tokens per verifier pass.

4 Method

4.1 Training-Free Self-Speculative Decoding for Block-Diffusion

S2D2 reuses a single block-diffusion model in two roles: standard block-diffusion decoding acts as the *drafter*, and the same model with block size 1 acts as an autoregressive *verifier*. This gives self-speculative decoding without auxiliary models, retraining, or architecture changes (Figure 1). At each denoising step, the drafter proposes tokens $\hat{x}$ with draft probabilities p from logits ℓ. Instead of immediately applying confidence-threshold acceptance, S2D2

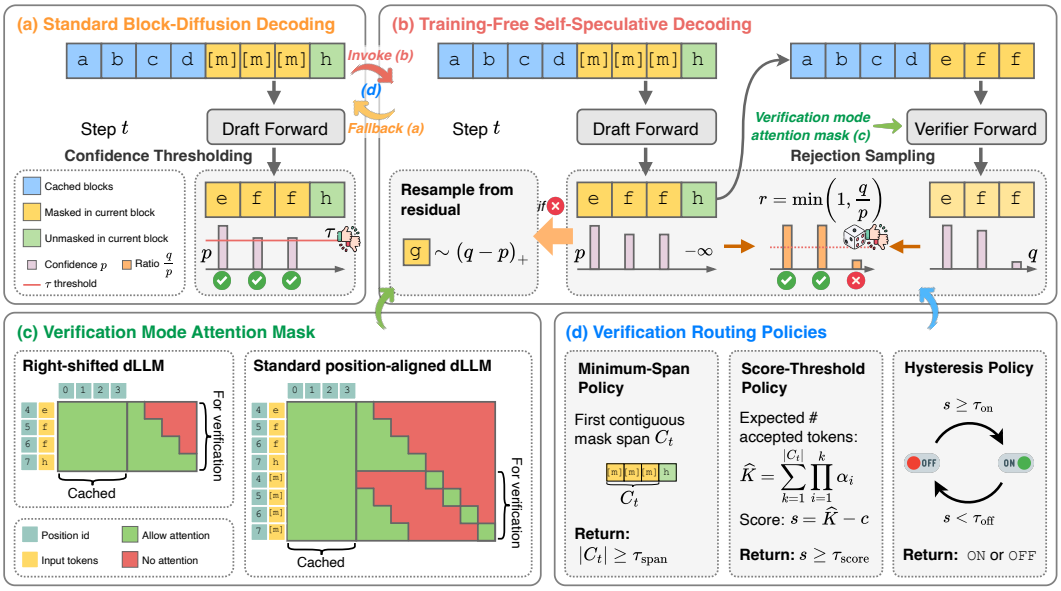


Figure 1: Overview of S2D2. (a) Standard block-diffusion decoding accepts drafted tokens by confidence thresholding. (b) S2D2 inserts a self-speculative verification step: the same model under block-size-1 autoregressive masking verifies the first contiguous masked span, accepts tokens by rejection sampling, and falls back to standard diffusion decoding when verification is not invoked or terminates early. (c) Verification-mode attention masks for right-shifted and standard position-aligned diffusion LLMs; we draw the full-block mask for illustration, though in practice only C_t is verified. (d) Lightweight routing policies decide when verification is worth its additional cost.

optionally verifies the first contiguous masked span C_t: we switch to block-size-1 masking, compute verifier probabilities q on drafted tokens in C_t, and run standard speculative acceptance left-to-right with probability $\min(1, q_i / p_i)$. At the first rejection, we resample from the residual distribution $(P_{\ell^{\text{ver}}} - P_\ell)_+$ and terminate that speculative segment (Algorithm 3).

Because verification adds one extra forward pass, it is not always worthwhile (e.g., very short candidate spans). S2D2 therefore uses lightweight *verification routing policies* to decide when to verify and when to fall back to standard confidence-based diffusion decoding. Section 4.2 details the verifier mask construction, and Section 4.3 presents routing policies.

4.2 Self-Verification Mode

We need verifier probabilities for drafted tokens under a block-size-1 autoregressive view, while keeping one shared pretrained block-diffusion model. For a drafted span, the verifier score at position i should condition on drafted tokens to its left and keep position i masked; the key challenge is computing all such scores in parallel. For position-aligned diffusion LLMs (e.g., LLaDA and SDAR), we use the standard "2L trick": for a drafted span of length L, concatenate the drafted tokens with an all-[MASK] copy at the same positions, and apply

$$\mathbf{M}_{\text{ver}} = \begin{bmatrix} \mathbf{A}_L & \mathbf{0}_L \\ \mathbf{A}_L^{<} & \mathbf{I}_L \end{bmatrix}, \tag{3}$$

where $\mathbf{A}_L$ is the causal mask, $\mathbf{A}_L^{<}$ its strict lower-triangular part, and $\mathbf{I}_L$ the identity. This yields all drafted-token verifier confidences in one forward pass (Figure 1(c)). For right-shifted models (e.g., Dream and Fast-dLLM v2), the standard causal mask already provides the verifier view.

Our setup is related to ASSD (Guo & Ermon, 2025), but ASSD requires any-subset AR modeling and dedicated architecture/training; S2D2 is training-free and plug-and-play for

existing block-diffusion models, at the cost of verifying only the first contiguous masked span. Even if verification is invoked at every step, S2D2 is still not identical to decoding with block size 1, since drafting and cache updates need not be fully causal under the original block-diffusion attention pattern. Our optional partially causal drafting variant uses

$$\mathbf{M}_{\text{draft}}^{(j)} = \begin{bmatrix} \mathbf{A}_j & \mathbf{0}_{j,B-j} \\ \mathbf{1}_{B-j,j} & \mathbf{1}_{B-j,B-j} \end{bmatrix}, \tag{4}$$

where j is the first masked position in the current block and $x^b_{<j}$ is treated as committed. Here $\mathbf{A}_j \in \{0,1\}^{j\times j}$ is the causal mask on the committed prefix. The four block dimensions are $j \times j$, $j \times (B-j)$, $(B-j) \times j$, and $(B-j) \times (B-j)$, respectively. A visualization is provided in Appendix Figure 4.

4.3 Verification Routing Policies

Verification is useful only when the expected gain from accepting multiple tokens offsets one extra verifier forward pass. We therefore use lightweight routing to decide, at each diffusion step, whether to verify the first contiguous masked span C_t or fall back to confidence-based diffusion decoding (Figure 1(d), Algorithm 4).

Expected accepted prefix length. Let $C_t = (1, \ldots, L)$ after local reindexing. We estimate the expected accepted prefix length as

$$\hat{K} = \sum_{k=1}^{L} \prod_{i=1}^{k} \alpha_i, \tag{5}$$

where $\alpha_i \in [0,1]$ approximates the acceptance probability at position i. We use two proxies: a margin-based form $\alpha_i = \mathbf{1}[m_i \geq \tau_{\text{margin}}]$ (with m_i the top-1 minus top-2 draft probability), and an entropy-based form $\alpha_i = \exp(-\beta \tilde{H}_i)$ with $\tilde{H}_i = H_i / \log V$. Additional estimators are deferred to Appendix A.5.

Routing scores and policies. We map $\hat{K}$ to a verification score using either

$$s = \hat{K} - c \text{ (static)}, \qquad s = \hat{K} - c \cdot N_{\text{hi}} \text{ (dynamic)}, \tag{6}$$

where c is a cost hyperparameter and N_{hi} counts high-confidence tokens in the current block (i.e., $N_{\text{hi}} = |\{i \in M_t : p_i > \tau\}|$ at diffusion step). We then use one of the following policies:

- **Minimum-span:** invoke verification when $|C_t| \geq \tau_{\text{span}}$. Despite its simplicity, this rule is often surprisingly effective and flexible. For example, setting $\tau_{\text{span}} = 1$ always enables verification, setting $\tau_{\text{span}} = B/2$ focuses verification on earlier steps with longer spans, and setting $\tau_{\text{span}} = B$ restricts verification to only the first step of a block.
- **Score-threshold:** invoke verification when $s \geq \tau_{\text{score}}$, using confidence structure rather than span length alone.
- **Hysteresis:** let $h \in \{\text{ON}, \text{OFF}\}$ denote the hysteresis state. If $h = \text{ON}$ and $s < \tau_{\text{off}}$, set $h \leftarrow \text{OFF}$; if $h = \text{OFF}$ and $s \geq \tau_{\text{on}}$, set $h \leftarrow \text{ON}$. Verification is invoked iff $h = \text{ON}$. The motivation is to avoid oscillation between speculative and diffusion modes.
- **Contextual bandit:** we also study a UCB-style contextual bandit router as an additional policy (Auer et al., 2002); details are deferred to Appendix A.6.

4.4 Analysis

We summarize how S2D2 relates to AR-guided residual energy correction. Additional derivations and discussion are deferred to Appendix A.3.

Not equivalent to global autoregressive decoding. S2D2 uses block-size-1 AR mode only as a *local verifier*. Verification is applied only on the first contiguous masked span C_t, can be skipped by routing, and stops at the first rejection; after that, decoding returns to diffusion with residual resampling. Drafting and KV caching are still generally produced under block-diffusion attention, so the overall trajectory is hybrid rather than globally causal.

Connection to AR-guided residual energy correction. EDLM (Xu et al., 2024) defines a residual energy over diffusion proposals as

$$E_\phi(x_0, x_t) \approx -\log p_{\mathrm{AR}}(x_0 \mid x_t) + \log p_\theta(x_0 \mid x_t) - \log Z. \tag{7}$$

Both EDLM and S2D2 exploit AR-vs-diffusion discrepancy, but differently: EDLM uses global multi-sample reweighting, while S2D2 applies online local correction through speculative acceptance and residual resampling.

Remark 1 (Local energy-guided interpretation). Let p_i be the draft probability of sampled token $\hat{x}_i$ and q_i its verifier probability under block-size-1 AR mode. The local residual energy and acceptance form are

$$E_i(\hat{x}_i) := -\log q_i + \log p_i, \qquad \min\left(1, \frac{q_i}{p_i}\right) = \min(1, e^{-E_i(\hat{x}_i)}). \tag{8}$$

Thus, lower-residual-energy drafted tokens are more likely to be accepted, while higher-energy mismatches are corrected via residual resampling.

This also explains the objective difference: EDLM spends extra test-time compute for quality via global reweighting, whereas S2D2 is designed for acceleration and invokes verification only when expected gain likely amortizes the extra verifier pass.

5 Experiments

Experimental setup. Detailed evaluation setup, including prompt templates and task-specific answer/code extraction, is provided in Appendix A.4.

Models. We evaluate S2D2 on five models from three block-diffusion families: SDAR (Cheng et al., 2025) (1.7B/4B/8B), Fast-dLLM v2 (Wu et al., 2025), and LLaDA2.1 (Bie et al., 2026). SDAR and Fast-dLLM v2 are adapted from autoregressive models, whereas LLaDA is trained from scratch. These cover both position-aligned (SDAR, LLaDA) and right-shifted (Fast-dLLM v2) architectures.

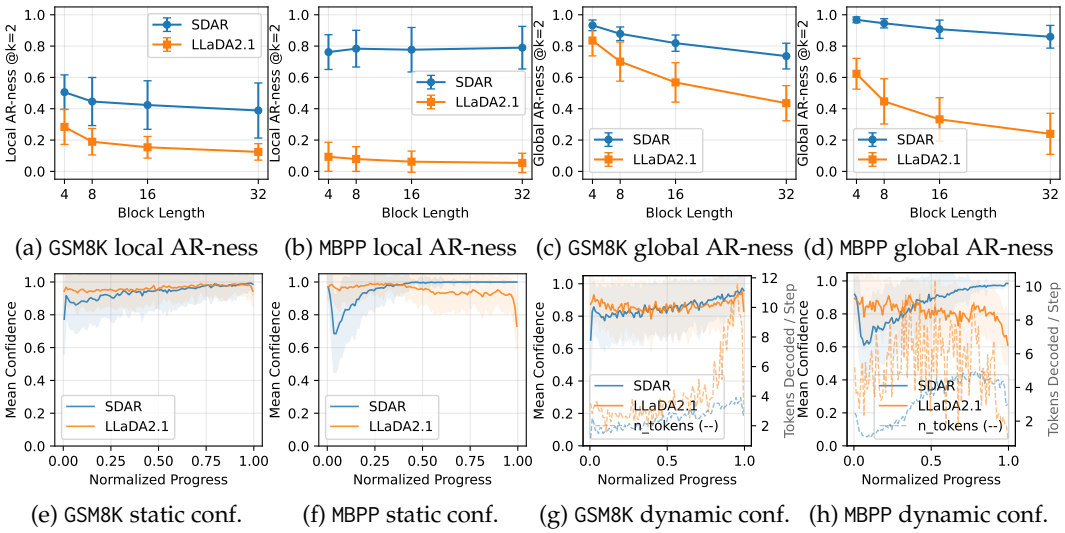
Benchmarks. We report results on **GSM8K** (Cobbe et al., 2021) (math reasoning), **MBPP** (Austin et al., 2021b) and **HumanEval** (Chen, 2021) (code generation), and **IFEval** (Zhou et al., 2023) (instruction following).

Decoding baselines. We compare against standard block-diffusion decoding across block sizes, denoising steps, and static/dynamic confidence schedules. Speedups are reported against the autoregressive baseline (block size 1), except for Fast-dLLM v2 where we use B = 4, SB = 1 because B = 1, SB = 1 is unreliable.

5.1 Main Results

Results on SDAR. Table 1 reports accuracy–speed tradeoffs on SDAR-1.7B/4B/8B across GSM8K, MBPP, HumanEval, and IFEval. Here, B denotes block size. Standard SDAR decoding is most reliable at small B (especially B = 4 or 8), so we use static/dynamic confidence decoding with B = 4 as the diffusion baseline.

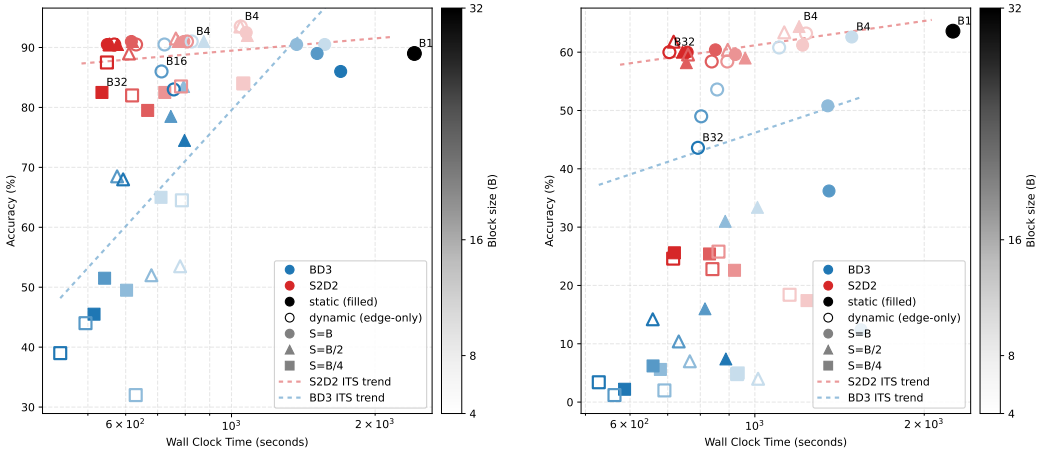
For S2D2, we report two operating points per model: config-A (accuracy-oriented) and config-B (speed-oriented). For SDAR-1.7B, config-A uses B = 4 with minimum-span routing ($\tau_{\text{span}} = 2$), while config-B uses B = 32 with always-on speculative verification and AR caching. For SDAR-4B, config-A uses B = 16 with always-on verification and AR caching, and config-B uses B = 32 with the same strategy. For SDAR-8B, config-A uses B = 4 with score-threshold routing ($\tau_{\text{score}} = 0$, static score with $c = 1$) and AR caching, while config-B uses B = 16 with always-on verification and AR caching. Across most settings, both configs outperform the dynamic confidence-threshold baseline while remaining faster. Config-A usually gives the best overall accuracy–speed balance, whereas config-B gives larger speedups with a modest accuracy tradeoff. A representative highlight is SDAR-1.7B (config-B): S2D2 reaches 4.7× speedup over AR decoding, i.e., about 1.57× over dynamic decoding (4.7/3.1), while improving average accuracy by 4.5 points (52.9 vs. 48.4).



(a) GSM8K local AR-ness (b) MBPP local AR-ness (c) GSM8K global AR-ness (d) MBPP global AR-ness

(e) GSM8K static conf. (f) MBPP static conf. (g) GSM8K dynamic conf. (h) MBPP dynamic conf.

Figure 2: AR-ness (@k, $k = 2$) and decoding-confidence statistics on GSM8K and MBPP. Top row: local and global AR-ness for SDAR-8B-Chat and LLaDA 2.1. Bottom row: normalized decoded-token confidence under static and dynamic diffusion decoding; dashed curves in dynamic decoding indicate the number of decoded tokens per step. Reference accuracies (GSM8K, MBPP): SDAR-8B-Chat, AR $(89.3\%, 64.4\%)$ and diffusion $(89.6\%, 61.0\%)$; LLaDA 2.1, AR $(90.8\%, 65.8\%)$ and diffusion $(90.8\%, 67.8\%)$.



(a) GSM8K (b) MBPP

Figure 3: Accuracy versus wall-clock time for SDAR-8B-Chat on GSM8K and MBPP. ITS denotes inference-time scaling, the ITS trend curve is fitted on points with accuracy $> 30\%$. Across block sizes, denoising steps, and decoding schedules, S2D2 generally achieves a better accuracy–speed frontier than BD3.

Results on Fast-dLLM v2. Table 2 reports results on Fast-dLLM v2 with fixed block size $\mathrm{B} = 32$ and varying sub-block size $\mathrm{SB} \in \{4, 8, 16, 32\}$. (Here, B is block size and SB is sub-block size.) The case $\mathrm{SB} = 32$ corresponds to standard block-diffusion decoding (config-C). Because Fast-dLLM v2 is unreliable at $\mathrm{B} = 1, \mathrm{SB} = 1$, we use $\mathrm{B} = 4, \mathrm{SB} = 1$ as the autoregressive-style baseline.

For the diffusion baseline, we keep the default dynamic-threshold style and fix $\tau = 0.9$, with configs A/B/C at SB $= 4/8/32$; sub-block caching is disabled since it is not lossless here. For S2D2, config-A uses SB $= 4$ with hysteresis routing ($\tau_{\text{on}} = 1$, $\tau_{\text{off}} = -5$) and dynamic score ($c = 1$), config-B uses the same policy at SB $= 16$, and config-C uses minimum-span

Table 1: Main accuracy and speedup results on the SDAR family. Speedups are measured relative to the autoregressive baseline. For each model, we report standard diffusion decoding at the strongest baseline block size and two representative S2D2 settings: config-A for higher accuracy and config-B for higher speed.

Method	GSM8K	MBPP	HumanEval	IFEval	Avg
SDAR-1.7B-Chat					
AR	76.3 (1.0×)	48.4 (1.0×)	62.2 (1.0×)	38.6 (1.0×)	56.4 (1.0×)
Diffusion (static)	**77.9** (1.7×)	13.6 (1.3×)	51.8 (1.9×)	43.3 (2.1×)	46.7 (1.9×)
Diffusion (dynamic)	77.2 (2.7×)	21.6 (1.3×)	51.8 (3.4×)	43.0 (3.8×)	48.4 (3.1×)
S2D2 (config-A)	77.4 (2.3×)	**45.4** (1.5×)	48.8 (2.3×)	**46.2** (2.7×)	**54.4** (2.5×)
S2D2 (config-B)	73.8 (**4.3**×)	44.4 (**2.9**×)	**52.4** (**4.1**×)	41.1 (**5.2**×)	52.9 (**4.7**×)
SDAR-4B-Chat					
AR	88.5 (1.0×)	59.6 (1.0×)	75.6 (1.0×)	54.3 (1.0×)	69.5 (1.0×)
Diffusion (static)	**89.0** (1.4×)	**57.2** (1.5×)	**73.8** (1.3×)	51.0 (1.8×)	67.7 (1.7×)
Diffusion (dynamic)	88.9 (2.7×)	55.6 (2.1×)	**73.8** (1.9×)	51.7 (3.7×)	67.5 (3.1×)
S2D2 (config-A)	87.5 (3.7×)	56.0 (1.3×)	69.5 (3.1×)	**59.5** (4.1×)	**68.1** (3.6×)
S2D2 (config-B)	87.4 (**4.3**×)	57.0 (**2.3**×)	68.3 (**3.3**×)	56.7 (**5.1**×)	67.4 (**4.5**×)
SDAR-8B-Chat					
AR	89.3 (1.0×)	64.4 (1.0×)	75.6 (1.0×)	57.7 (1.0×)	71.7 (1.0×)
Diffusion (static)	**89.6** (1.4×)	61.0 (1.4×)	**79.3** (1.6×)	52.4 (1.6×)	70.6 (1.5×)
Diffusion (dynamic)	89.3 (2.6×)	60.6 (2.1×)	78.0 (2.9×)	54.0 (2.7×)	70.5 (2.6×)
S2D2 (config-A)	**89.6** (2.0×)	**62.0** (2.1×)	78.7 (2.2×)	60.1 (2.2×)	**72.6** (2.1×)
S2D2 (config-B)	88.3 (**3.8**×)	61.4 (**2.8**×)	74.4 (**3.2**×)	**60.9** (**3.9**×)	71.3 (**3.7**×)

Table 2: Main accuracy and speedup results on Fast-dLLM v2. We fix block size $B = 32$ and vary the sub-block size SB. Speedups are measured relative to the autoregressive-style baseline $B = 4, SB = 1$. Config-C corresponds to the normal BD3 without sub-blocks.

Method	GSM8K	MBPP	HumanEval	IFEval	Avg
AR	85.1 (1.0×)	51.8 (1.0×)	32.9 (1.0×)	67.7 (1.0×)	59.4 (1.0×)
Diffusion (config-A)	**84.8** (2.5×)	**50.6** (2.3×)	14.6 (1.4×)	63.1 (1.6×)	53.3 (2.1×)
Diffusion (config-B)	83.2 (3.3×)	47.8 (2.9×)	11.0 (1.5×)	61.6 (1.9×)	50.9 (2.7×)
Diffusion (config-C, SB=32)	78.3 (3.5×)	46.2 (3.6×)	8.5 (1.7×)	59.4 (1.9×)	48.1 (2.9×)
S2D2 (config-A)	83.8 (1.6×)	48.8 (2.0×)	21.3 (1.5×)	62.7 (1.3×)	**54.2** (1.6×)
S2D2 (config-B)	84.2 (3.0×)	45.0 (3.7×)	**22.0** (2.0×)	**64.0** (**1.9**×)	53.8 (2.8×)
S2D2 (config-C, SB=32)	82.0 (**3.9**×)	45.6 (**3.8**×)	19.5 (**2.5**×)	63.2 (1.8×)	52.6 (**3.1**×)

routing with $\tau_{\text{span}} = 8$ at SB $= 32$. Compared with the diffusion baseline, S2D2 consistently improves the frontier: config-A improves accuracy with only minor speed loss, config-B improves both accuracy and speed, and config-C recovers much of the large-SB accuracy drop while still adding speedup. In particular, at config-C (SB $= 32$), S2D2 is about 1.07× faster than dynamic decoding (3.1× vs. 2.9×) with a +4.5-point average accuracy gain.

Results on LLaDA2.1. Table 3 reports preliminary GSM8K/MBPP results on LLaDA2.1-Mini (Bie et al., 2026). Unlike standard block-diffusion models, LLaDA supports token editing: previously unmasked tokens can be revised when confidence exceeds τ_{edit}. This enables an "unmask early, correct later" behavior that is related in spirit to self-speculation, but without rejection sampling.

Table 3: Accuracy and speedup results on LLaDA2.1-Mini. Speedups are measured relative to the AR baseline. We compare the built-in editing-based diffusion decoder against S2D2 under two threshold settings.

	Block-Size = 1 (AR)			$\tau_{\text{mask}} = 1,\ \tau_{\text{edit}} = 0.9$		
Method	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg
Diffusion	90.8 (1.0×)	65.8 (1.0×)	78.3 (1.0×)	90.8 (0.6×)	67.6 (0.5×)	79.2 (0.5×)
	$\tau_{\text{mask}} = 0.7,\ \tau_{\text{edit}} = 0.5$			$\tau_{\text{mask}} = 0.95,\ \tau_{\text{edit}} = 0.9$		
Diffusion	89.6 (**2.7**×)	57.8 (**2.8**×)	73.7 (**2.7**×)	**91.0** (1.7×)	66.4 (1.8×)	78.7 (1.7×)
S2D2	**90.8** (2.2×)	**64.0** (2.1×)	**77.4** (2.1×)	89.8 (**2.1**×)	**68.8** (**2.2**×)	**79.3** (**2.2**×)

We evaluate two settings: quality mode ($\tau_{\text{mask}} = 0.7$, $\tau_{\text{edit}} = 0.5$) and a conservative setting ($\tau_{\text{mask}} = 0.95$, $\tau_{\text{edit}} = 0.9$). In quality mode, S2D2 improves average accuracy over diffusion (77.4% vs. 73.7%) with moderate speed loss. In the conservative setting, S2D2 improves both accuracy (79.3% vs. 78.7%) and speedup (2.2× vs. 1.7×), i.e., about 1.3× faster with +0.6 points. Relative to the static baseline under the same setting (0.5×, 79.2%), this is 4.4× faster with slightly higher accuracy (+0.1); note that this static baseline is itself slower than AR (0.5× vs. 1.0×). Overall, S2D2 appears complementary to LLaDA's built-in self-correction.

5.2 Analysis and Ablation

Decoding behavior analysis. We analyze baseline diffusion behavior using DiffuCoder local/global AR-ness metrics (Gong et al., 2025), where 1 corresponds to exact left-to-right autoregressive decoding, together with decoded-token confidence; for dynamic decoding, we also report tokens decoded per step (Figure 2). For LLaDA2.1, we disable editing by setting $\tau_{\text{edit}} = 1$. Additional plots are in Appendix Figures 5 and 6d.

AR-ness shows a task dependence: for SDAR, it is higher on MBPP than GSM8K, while for LLaDA 2.1 the trend reverses. This matches the relative competitiveness of AR versus diffusion decoding across tasks. Confidence trajectories also differ: SDAR confidence typically rises over decoding, whereas LLaDA often starts high and drops near the end, suggesting stronger AR structure in mathematical reasoning than in code generation.

Inference-time scaling trend. Figure 3 shows accuracy versus wall-clock time on SDAR-8B-Chat across block sizes B, denoising steps S, schedules, and mask-span settings. S2D2 generally stays on a better frontier and exhibits flatter inference-time scaling than standard diffusion decoding, especially at larger block sizes where the baseline degrades.

Additional ablations. Further ablations on token acceptance estimators, routing policies, and rejection-ratio tempering are provided in Appendix A.5, Appendix A.7, and Appendix A.8.

6 Conclusion

We introduced S2D2, a training-free self-speculative decoding framework that reuses a single pretrained block-diffusion model in two modes: standard block-diffusion decoding as the drafter and block-size-1 autoregressive decoding as the verifier. Across multiple block-diffusion families, this plug-and-play design consistently improves the accuracy-speed tradeoff, often delivering both higher accuracy and lower latency than strong dynamic confidence-thresholding baselines. Our analysis suggests speculative verification acts as a local sequence-level correction mechanism that can be viewed as a stochastic, greedy form of autoregressive energy correction, and we hope this perspective encourages further training-free inference-time methods for diffusion language models.

References

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.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

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, 2021a.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b.

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.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Keshigeyan Chandrasegaran, Armin W. Thomas, Jerome Ku, Federico Berto, Jae Myung Kim, Garyk Brixi, Eric Nguyen, Stefano Massaroli, and Michael Poli. Rnd1: Simple, scalable ar-to-diffusion conversion. Oct 2025. URL https://www.radicalnumerics.ai/blog/rnd1.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11315–11325, 2022.

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. 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.

Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. *arXiv preprint arXiv:2209.14687*, 2022.

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.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Yifeng Gao, Ziang Ji, Yuxuan Wang, Biqing Qi, Hanlin Xu, and Linfeng Zhang. Self speculative decoding for diffusion large language models. *arXiv preprint arXiv:2510.04147*, 2025.

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.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Gabe Guo and Stefano Ermon. Reviving any-subset autoregressive models with principled parallel sampling and speculative decoding. *arXiv preprint arXiv:2504.20456*, 2025.

Ligong Han, Jian Ren, Hsin-Ying Lee, Francesco Barbieri, Kyle Olszewski, Shervin Minaee, Dimitris Metaxas, and Sergey Tulyakov. Show me what and tell me how: Video synthesis via multimodal conditioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3615–3625, 2022.

Xiaoxiao He, Quan Dao, Ligong Han, Song Wen, Minhao Bai, Di Liu, Han Zhang, Felix Juefei-Xu, Chaowei Tan, Bo Liu, Martin Renqiang Min, Kang Li, Faez Ahmed, Akash Srivastava, Hongdong Li, Junzhou Huang, and Dimitris N. Metaxas. Dice: Discrete inversion enabling controllable editing for masked generative models. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 762–772, March 2026.

Emiel Hoogeboom, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. *arXiv preprint arXiv:2110.02037*, 2021.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

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.

Jia-Nan Li, Jian Guan, Wei Wu, and Chongxuan Li. Refusion: A diffusion large language model with parallel autoregressive decoding. *arXiv preprint arXiv:2512.13586*, 2025.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in neural information processing systems*, 35:4328–4343, 2022.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

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*, 2025.

Kevin Rojas, Ye He, Chieh-Hsin Lai, Yuhta Takida, Yuki Mitsufuji, and Molei Tao. Improving classifier-free guidance in masked diffusion: Low-dim theoretical insights with high-dim impact. In *The Fourteenth International Conference on Learning Representations*, 2026.

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

Subham Sekhar Sahoo, Zhihan Yang, Yash Akhauri, Johnna Liu, Deepansha Singh, Zhoujun Cheng, Zhengzhong Liu, Eric Xing, John Thickstun, and Arash Vahdat. Esoteric language models, 2025. URL https://arxiv.org/abs/2506.01928.

Yair Schiff, Subham Sekhar Sahoo, Hao Phung, Guanghan Wang, Sam Boshar, Hugo Dallatorre, Bernardo P de Almeida, Alexander M Rush, Thomas PIERROT, and Volodymyr Kuleshov. Simple guidance mechanisms for discrete diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

Andy Shih, Dorsa Sadigh, and Stefano Ermon. Training and inference on any-order autoregressive models the right way. *Advances in Neural Information Processing Systems*, 35: 2762–2775, 2022.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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, 2025. URL https://arxiv.org/abs/2505.22618.

Minkai Xu, Tomas Geffner, Karsten Kreis, Weili Nie, Yilun Xu, Jure Leskovec, Stefano Ermon, and Arash Vahdat. Energy-based diffusion language models for text generation. *arXiv preprint arXiv:2410.21357*, 2024.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

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.

Jaehoon Yoo, Wonjung Kim, and Seunghoon Hong. Redi: Rectified discrete flow. *arXiv preprint arXiv:2507.15897*, 2025.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, 2024.

Tunyu Zhang, Xinxi Zhang, Ligong Han, Haizhou Shi, Xiaoxiao He, Zhuowei Li, Hao Wang, Kai Xu, Akash Srivastava, Vladimir Pavlovic, et al. T3d: Few-step diffusion language models via trajectory self-distillation with direct discriminative optimization. *arXiv preprint arXiv:2602.12262*, 2026.

Yichi Zhang, Alex Schwing, and Zhizhen Zhao. Variational masked diffusion models. *arXiv preprint arXiv:2510.23606*, 2025.

Kaiwen Zheng, Yongxin Chen, Huayu Chen, Guande He, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Direct discriminative optimization: Your likelihood-based visual generative model is secretly a gan discriminator. *arXiv preprint arXiv:2503.01103*, 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 Appendix

A.1 Algorithm details

Algorithms 1–4 form the full decoding pipeline. Algorithm 1 is the shared outer loop for block-wise autoregressive decoding, used by both standard diffusion decoding and S2D2. Within this loop, Algorithm 2 defines dynamic confidence-thresholded diffusion decoding; static decoding is recovered by setting $\tau = 1$. Algorithm 3 is our self-speculative block sampler, which adds verifier-based acceptance on top of the same block framework. Algorithm 4 specifies the routing procedure DoVerify used by Algorithm 3 to decide when verification is invoked.

Algorithm 1 Block-wise autoregressive decoding

Require: Prompt $x_{1:m}$, model x_θ, block size B, block sampler SampleBlock
Ensure: Generated sequence x

$x \leftarrow x_{1:m}$, $\mathbf{K}, \mathbf{V} \leftarrow \emptyset$
while generation not finished **do**
  Initialize a new block $x^b \leftarrow [\texttt{MASK}]^B$
  $x^b \leftarrow \text{SampleBlock}(x_\theta, x^b, \mathbf{K}, \mathbf{V})$
  $\emptyset, \mathbf{K}^b, \mathbf{V}^b \leftarrow x_\theta(x^b)$
  $x \leftarrow x \oplus x^b$
  $(\mathbf{K}, \mathbf{V}) \leftarrow (\mathbf{K} \oplus \mathbf{K}^b,\ \mathbf{V} \oplus \mathbf{V}^b)$
end while
return x

Algorithm 2 SampleBlock for standard BD3

Require: Model x_θ, masked block x^b, KV cache $(\mathbf{K}, \mathbf{V})$, diffusion steps T, confidence threshold τ
Ensure: Decoded block x^b

for $t = 1$ to T **do**
  if x^b has no [MASK] **then**
    break
  end if
  $\ell \leftarrow x_\theta(x^b; \mathbf{K}, \mathbf{V})$ {draft forward}
  $(\hat{x}, p) \leftarrow \text{SampleFromLogits}(\ell)$
  $M_t \leftarrow \{i : x^b_i = \texttt{[MASK]}\}$
  $S_t \leftarrow \{i \in M_t : p_i > \tau\}$
  $S_t \leftarrow S_t \cup \{\arg\max_{i \in M_t} p_i\}$
  $x^b_{S_t} \leftarrow \hat{x}_{S_t}$
end for
return x^b

Optional AR-like drafting/caching mask. Figure 4 visualizes the optional partially causal drafting/caching mask used to make the trajectory more AR-like. The mask definition is given in Eq. 4 in Section 4.2.

A.2 From MDLM posterior sampling to LLaDA confidence-based decoding

For absorbing-state discrete diffusion, the reverse transition under the SUBS parameterization of MDLM (Sahoo et al., 2024) is

$$p_\theta(z_s \mid z_t) = q(z_s \mid z_t, x = x_\theta(z_t, t)). \tag{9}$$

For a masked position $z_t = m$, Eq. (2) can be equivalently interpreted as a two-stage procedure: first, sample whether the position remains masked or becomes visible with

Algorithm 3 SAMPLEBLOCK for S2D2

Require: Model x_θ, masked block x^b, KV cache $(\mathbf{K}, \mathbf{V})$, diffusion steps T, confidence threshold τ
Ensure: Decoded block x^b

for $t = 1$ to T **do**
  if x^b has no `[MASK]` **then**
    break
  end if
  $\ell \leftarrow x_\theta(x^b; \mathbf{K}, \mathbf{V})$ {draft forward}
  $(\hat{x}, p) \leftarrow \text{SAMPLEFROMLOGITS}(\ell)$
  $M_t \leftarrow \{i : x_i^b = \texttt{[MASK]}\}$
  $C_t \leftarrow$ first contiguous mask span in M_t
  if $\text{DOVERIFY}(x^b, \ell, p, M_t, C_t)$ **then**
    $\tilde{x}^b \leftarrow x^b, \quad \tilde{x}^b_{C_t} \leftarrow \hat{x}_{C_t}$
    $\ell^{\text{ver}} \leftarrow x_\theta^{\text{AR}}(\tilde{x}^b; \mathbf{K}, \mathbf{V})$ {verifier forward}
    $(\_, q) \leftarrow \text{SCOREFROMLOGITS}(\ell^{\text{ver}}, \hat{x}_{C_t})$
    $S_t \leftarrow \emptyset$
    for $i \in C_t$ from left to right **do**
      Draw $r \sim \mathcal{U}[0, 1]$
      if $r < \min(1, q_i / p_i)$ **then**
        $S_t \leftarrow S_t \cup \{i\}$
      else
        Resample $\tilde{x}_i \sim (P_{\ell^{\text{ver}}} - P_\ell)_+$
        $S_t \leftarrow S_t \cup \{i\}$
        $\hat{x}_i \leftarrow \tilde{x}_i$
        break
      end if
    end for
    $x^b_{S_t} \leftarrow \hat{x}_{S_t}$
  else
    $S_t \leftarrow \{i \in M_t : p_i > \tau\}$
    $S_t \leftarrow S_t \cup \{\arg\max_{i \in M_t} p_i\}$
    $x^b_{S_t} \leftarrow \hat{x}_{S_t}$
  end if
end for
return x^b

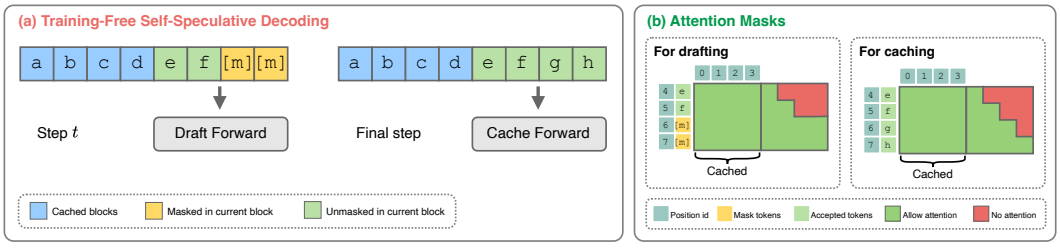


Figure 4: Drafting and caching with autoregressive attention masks.

probability

$$\rho_{t \to s} = \frac{\alpha_s - \alpha_t}{1 - \alpha_t}, \tag{10}$$

and second, if the position is unmasked, sample its token from the model prediction $x_\theta(z_t, t)$. Thus, exact reverse sampling factorizes across masked positions.

Algorithm 4 DoVerify: verification policies

Require: Block x^b, draft logits ℓ, draft confidence p, masked positions M_t, first contiguous mask span C_t, thresholds $\tau_{\text{span}}, \tau_{\text{score}}, \tau_{\text{on}}, \tau_{\text{off}}$, hysteresis state h
Ensure: Boolean decision $d \in \{\texttt{True}, \texttt{False}\}$

Minimum-Span Policy

return $(|C_t| \geq \tau_{\text{span}})$

Score-Threshold Policy

$s \leftarrow \textsc{ComputeVerifyScore}(x^b, \ell, p, M_t, C_t)$
return $(s \geq \tau_{\text{score}})$

Hysteresis Policy

$s \leftarrow \textsc{ComputeVerifyScore}(x^b, \ell, p, M_t, C_t)$
if $h = \texttt{ON}$ and $s < \tau_{\text{off}}$ **then**
  $h \leftarrow \texttt{OFF}$
else if $h = \texttt{OFF}$ and $s \geq \tau_{\text{on}}$ **then**
  $h \leftarrow \texttt{ON}$
end if
return $(h = \texttt{ON})$

In contrast, LLaDA decoding (Nie et al., 2025) typically operates in a few-step regime and does not directly sample from this factorized posterior. Instead, after a draft forward pass, it selects which positions to unmask using token confidence, either according to a fixed schedule or a dynamic threshold. This can be viewed as replacing independent Bernoulli reveal decisions with a confidence-prioritized selection rule.

We find this interpretation useful for understanding why confidence-based decoding often works well in practice. First, the exact reverse posterior in Eq. (2) is conditionally factorized across positions, which introduces a **mean-field** approximation at the block level. In the few-step regime, confidence-based selection partially mitigates this approximation by introducing dependence among positions through competitive reveal decisions: instead of revealing positions independently, the decoder preferentially commits to the most reliable tokens first. Second, posterior sampling in diffusion can be viewed as approximating an intractable integral using Monte Carlo samples, as discussed in **diffusion posterior sampling (DPS)** (Chung et al., 2022). From this perspective, practical decoding effectively uses a single sample to approximate the reverse update, and choosing the highest-confidence proposals can reduce the approximation error of this one-sample estimator. Dynamic confidence thresholding strengthens this effect by adaptively revealing as many positions as possible once their confidence exceeds a preset threshold, yielding substantial acceleration while usually preserving generation quality.

A.3 Connection to residual energy

We further clarify the connection between S2D2 and AR-guided residual energy correction.

Residual energy and speculative acceptance. EDLM defines a residual energy over diffusion proposals of the form

$$\begin{aligned} E_\phi(x_0, x_t) &\approx -\log q(x_0 \mid x_t) + \log p_\theta(x_0 \mid x_t) - \log Z \\ &= -\log q(x_0) + \log p_\theta(x_0 \mid x_t) + \big(\log q(\bar{x}_0) - \log Z\big), \end{aligned} \tag{11}$$

where $q(x_0)$ denotes the autoregressive verification density and $p_\theta(x_0 \mid x_t)$ is the diffusion proposal distribution (Xu et al., 2024). Ignoring the additive constant $\log q(\bar{x}_0) - \log Z$, (where $\bar{x}_0$ is the unmasked tokens in x_t) this is exactly the negative log density ratio:

$$E_\phi(x_0, x_t) = -\log \frac{q(x_0)}{p_\theta(x_0 \mid x_t)} \quad \text{up to an additive constant.} \tag{12}$$

In our setting, for a drafted token $\hat{x}_i$, let p_i denote the draft probability under the diffusion forward pass and let q_i denote the verifier probability under the block-size-1 autoregressive mode. Then the local residual energy

$$E_i(\hat{x}_i) := -\log q_i + \log p_i = -\log \frac{q_i}{p_i} \tag{13}$$

is exactly the same discrepancy used in speculative decoding, since

$$\min\left(1, \frac{q_i}{p_i}\right) = \min(1, e^{-E_i(\hat{x}_i)}). \tag{14}$$

Thus, S2D2 and EDLM rely on the same AR-vs-diffusion discrepancy, but use it differently: EDLM uses it for global importance reweighting, while S2D2 uses it online through speculative acceptance and residual resampling.

NCE and a JSD-style interpretation. EDLM fits its AR-parameterized residual model using a noise-contrastive objective. Writing the EDLM NCE loss explicitly,

$$\begin{aligned}\mathcal{L}_{\text{NCE}}(\phi;\theta) = &- \mathbb{E}_{x_+ \sim q(\hat{x}_0|x_t,x_0)}\left[\log \sigma\left(\log \frac{p_\phi(x_+)}{p_\theta(x_+ \mid x_t)}\right)\right] \\ &- \mathbb{E}_{x_- \sim p_\theta(\hat{x}_0|x_t)}\left[\log\left(1 - \sigma\left(\log \frac{p_\phi(x_-)}{p_\theta(x_- \mid x_t)}\right)\right)\right].\end{aligned} \tag{15}$$

This has the same binary likelihood-ratio form as Direct Discriminative Optimization (DDO) (Zheng et al., 2025): positive samples are drawn from the true data posterior, while negative samples are drawn from the diffusion proposals. Under sufficient model capacity, the optimum is attained when the learned AR residual model matches the posterior. In this sense, the NCE objective admits a JSD-style interpretation through its binary likelihood-ratio form.

Relation to block-diffusion training. This perspective is complementary to standard diffusion training. Block-diffusion models are trained by optimizing the usual ELBO, which matches the reverse conditional to the posterior through a KL objective. In particular, the block-size-1 mode is not a separately trained energy model, but an extreme autoregressive conditional induced by the same diffusion model family. This does not imply that it exactly recovers the EDLM AR residual model. However, since both EDLM and block-diffusion training are ultimately driven by approximating the same posterior object from different variational viewpoints, the block-size-1 autoregressive mode provides a natural in-family proxy for an AR energy model.

A stochastic greedy interpretation. Finally, Eq. (14) suggests an optimization-style interpretation of speculative decoding. Since the acceptance probability is monotone in $-E_i(\hat{x}_i)$, drafted tokens with lower residual energy are more likely to be accepted, while higher-energy mismatches are more likely to be rejected and corrected by residual resampling. Therefore, although speculative decoding is not equivalent to exact energy minimization, it can be viewed heuristically as a stochastic, greedy local procedure that prefers lower-energy proposals. Under this view, S2D2 may be interpreted as an online local approximation to AR-guided energy correction.

A.4 Evaluation setup details

By default, we use `lm-eval-harness` (Gao et al., 2024). We switch to our own scripts when task formatting or extraction robustness is critical.

GSM8K. For all models, we use our own script for more robust formatting handling and answer extraction than the default harness path. The system prompt is:

```
Solve the following math problem concisely and clearly and put your final
answer within \boxed{}.
```

MBPP and HumanEval. For LLaDA 2.1 on MBPP, we find the model does not reliably follow the `[BEGIN]` function-completion format expected by `lm-eval-harness`, which can lead to near-zero/zero measured accuracy if used directly. We therefore use our own script with more robust code extraction. The system prompt is:

```
You are an expert Python programmer. Write only the function code, no
explanations.
```

For SDAR, we also evaluate HumanEval with our own script for the same formatting/extraction reason.

Summary of evaluation backends. Our own scripts are used for: GSM8K (all models), MBPP (SDAR and LLaDA2.1-Mini), and HumanEval (SDAR). Other settings use `lm-eval-harness` (Gao et al., 2024). (By default, `lm-eval-harness` uses few-shot prompting on MBPP.)

Ablation subset. For ablations, we use the same fixed 200-sample subset of GSM8K across compared methods.

A.5 Additional ablations on token acceptance estimators

We further ablate several lightweight estimators for the accepted prefix length predictor in Eq. (5). In addition to the two estimators used in the main text, we test four additional variants and a random baseline. Results are shown in Table 4.

Setup. We evaluate these estimators on SDAR-8B-Chat using 50 samples from GSM8K and 50 from MBPP. We invoke speculative decoding at every denoising step and compare the estimated accepted prefix length $\hat{K}$ against the actual accepted prefix length under verification. We report signed error, error standard deviation, and mean absolute error (MAE).

Findings. Hard margin thresholding gives the most accurate estimates. In particular, $\alpha_i = \mathbf{1}[m_i \geq \tau_{\text{margin}}]$ with $\tau_{\text{margin}} = 0.1$ achieves the lowest MAE in Table 4. However, margin-based estimators are more sensitive to calibration, and the best threshold can vary across models and sampling settings. Moreover, better estimation accuracy does not necessarily imply better downstream decoding accuracy. Although hard margin thresholding is the most accurate estimator in Table 4, the soft entropy-based estimator yields higher average task accuracy across block sizes in Tables 7 and 8. For this reason, we use the soft entropy-based estimator, $\alpha_i = \exp(-\beta \tilde{H}_i)$, in the main experiments. It is simpler to use, less tied to model-specific calibration, and empirically more effective in our routing setup. Table 4 also suggests that more accurate, better-calibrated estimators may yield further gains.

A.6 Contextual-bandit routing policy

As an additional adaptive routing baseline, we implement a simple contextual-bandit policy based on upper confidence bounds (UCB). At each decoding decision, the policy chooses between two actions, $a \in \{0, 1\}$, where $a = 0$ denotes standard diffusion decoding without verification and $a = 1$ denotes invoking speculative verification.

Let b_t denote the context bucket at decision step t. The UCB policy selects

$$a_t = \arg\max_a \left(\hat{\mu}_{a,b_t} + \beta \sqrt{\frac{\log t}{n_{a,b_t}}} \right), \tag{16}$$

where $\hat{\mu}_{a,b}$ is the empirical mean reward of action a in context bucket b, $n_{a,b}$ is the number of times that action has been taken in that bucket, and β controls exploration. The empirical mean reward is computed as

$$\hat{\mu}_{a,b} = \frac{1}{n_{a,b}} \sum_{s:\, a_s = a,\, b_s = b} r_s, \qquad n_{a,b} = \#\{s : a_s = a,\ b_s = b\}. \tag{17}$$

Table 4: Token acceptance estimator ablation. We report signed error (Mean Error), error standard deviation (Std Error), and mean absolute error (MAE) for predicting accepted prefix length. Lower absolute error is better.

Config	Mean Error	Std Error	MAE
Random baseline $\alpha_i \sim \mathcal{U}[0,1]$			
–	-8.014	10.293	8.129
Soft entropy estimator $\alpha_i = \exp(-\beta \tilde{H}_i)$			
$\beta = 0.5$	-0.445	8.598	5.843
$\beta = 0.75$	-1.956	8.872	5.279
$\beta = 1.0$	-2.861	9.031	5.084
$\beta = 1.25$	-3.471	9.129	5.059
$\beta = 1.5$	-3.686	8.969	4.901
Confidence-power estimator $\alpha_i = p_i^{\gamma_{\text{conf}}}$			
$\gamma_{\text{conf}} = 0.5$	-4.107	8.220	4.660
$\gamma_{\text{conf}} = 0.75$	-4.838	8.543	5.113
$\gamma_{\text{conf}} = 1.0$	-5.372	8.822	5.564
$\gamma_{\text{conf}} = 1.25$	-5.528	8.787	5.654
$\gamma_{\text{conf}} = 1.5$	-5.724	8.824	5.825
Rényi-2 entropy estimator $\alpha_i = \sum_v p_i(v)^2$			
–	-5.870	8.922	5.961
Hard-entropy threshold $\alpha_i = \mathbf{1}[\tilde{H}_i < \tau_{\text{ent}}]$			
$\tau_{\text{ent}} = 0.01$	-7.631	9.489	7.634
$\tau_{\text{ent}} = 0.02$	-7.271	9.425	7.276
$\tau_{\text{ent}} = 0.05$	-6.505	9.305	6.535
$\tau_{\text{ent}} = 0.1$	-5.516	9.260	5.732
$\tau_{\text{ent}} = 0.2$	-3.333	8.931	4.835
Hard-margin threshold $\alpha_i = \mathbf{1}[m_i \geq \tau_{\text{margin}}]$			
$\tau_{\text{margin}} = 0.01$	4.240	5.871	4.367
$\tau_{\text{margin}} = 0.02$	2.593	4.359	2.868
$\tau_{\text{margin}} = 0.05$	0.953	2.898	1.599
$\tau_{\text{margin}} = 0.1$	0.065	2.476	**1.245**
$\tau_{\text{margin}} = 0.2$	-0.823	2.829	1.371

Following our implementation, the reward is defined as

$$r = \frac{\texttt{decoded\_this\_step}}{\texttt{time\_cost}}, \qquad \texttt{time\_cost} = \begin{cases} 2, & \text{if verification is invoked,} \\ 1, & \text{otherwise.} \end{cases} \tag{18}$$

We use a simple discretized context tuple consisting of: (i) span-length bin for the first contiguous masked span, linearly partitioned from 1 to block size; (ii) decoding-progress bin, measured by the fraction of already unmasked tokens in the current block; and (iii) entropy bin, computed from normalized token entropy. By default, we use two bins for each dimension. We report the ablation results for different context tuples and exploration coefficients in Table 10. In our experiments, this contextual-bandit policy serves as a representative simple training-free reinforcement learning baseline. Although adaptive, it is more involved than the threshold-based policies and was not the best-performing routing strategy in our setting.

A.7 Ablation on routing policies

We provide a comprehensive ablation of routing policies on SDAR-8B-Chat over GSM8K and MBPP, using 200 samples from each dataset. We evaluate block sizes $B \in \{4, 8, 16, 32\}$. The corresponding BD3 baseline (Arriola et al., 2025) results are reported in Table 5.

Minimum-span policy. Table 6 reports the ablation of the minimum-span policy, where verification is invoked whenever the first contiguous masked span satisfies $|C_t| \geq \tau_{\text{span}}$. We sweep $\tau_{\text{span}} \in \{1, 2, 4, \ldots, B-1\}$. Figure 3 further plots the cases $\tau_{\text{span}} \in \{1, 2\}$ together with different denoising steps S, showing that `S2D2` generally lies in the upper-left region of the accuracy–speed tradeoff.

Score-threshold policy. For the score-threshold policy, we study two choices of accepted-prefix estimator. Table 7 uses the soft entropy-based estimator ($\beta = 1$), while Table 8 uses the hard confidence-margin estimator ($\tau_{\text{margin}} = 0.05$). Across most settings, both score-based variants outperform the BD3 baseline in accuracy under the same block size, with the gains being especially pronounced at larger block sizes B = 16 and B = 32.

Comparing the two estimators, the entropy-based estimator consistently achieves higher downstream accuracy than the margin-based estimator across all block sizes. This is notable because, as shown in Table 4, the hard margin estimator is more accurate at predicting accepted prefix length.

Hysteresis policy. Table 9 reports the ablation of the hysteresis-based routing strategy. Compared with the plain score-threshold policy, hysteresis introduces temporal consistency by using different thresholds for turning verification on and off.

Contextual-bandit policy. Finally, Table 10 reports the UCB-style contextual-bandit routing policy. More implementation details are given in Appendix A.6.

Table 5: Baseline BD3 results with static and dynamic confidence-based unmasking.

Config	**B=1** (AR)			**B=4**			**B=8**			**B=16**			**B=32**		
	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg
Static	89.0 (1.0×)	63.6 (1.0×)	76.3 (1.0×)	90.5 (1.5×)	**62.6** (1.5×)	**76.6** (1.5×)	90.5 (1.8×)	50.8 (1.7×)	70.7 (1.7×)	**89.0** (1.6×)	36.2 (1.7×)	62.6 (1.6×)	**86.0** (1.4×)	12.4 (1.5×)	49.2 (1.4×)
Dynamic	–	–	–	**91.0** (**2.9**×)	60.8 (**2.0**×)	75.9 (**2.4**×)	90.5 (**3.3**×)	**53.6** (**2.6**×)	**72.0** (**3.0**×)	86.0 (**3.4**×)	**49.0** (**2.8**×)	**67.5** (**3.1**×)	83.0 (**3.2**×)	**43.6** (**2.8**×)	**63.3** (**3.0**×)

Table 6: Ablation of the minimum-span policy. We vary the minimum contiguous mask-span threshold τ_{span}.

τ_{span}	**B=4**			**B=8**			**B=16**			**B=32**		
	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg
1	90.0 (1.9×)	**61.8** (1.9×)	75.9 (1.9×)	90.0 (2.9×)	**61.2** (2.5×)	75.6 (2.7×)	90.0 (3.5×)	**61.4** (**3.0**×)	**75.7** (3.2×)	89.5 (4.7×)	59.8 (3.2×)	74.6 (3.8×)
2	91.5 (**2.1**×)	59.6 (**2.0**×)	75.6 (**2.0**×)	**92.0** (3.2×)	60.6 (2.5×)	**76.3** (2.8×)	90.0 (3.9×)	61.0 (2.9×)	75.5 (3.3×)	**93.0** (**4.8**×)	58.6 (**3.4**×)	75.8 (**4.0**×)
3	**93.0** (1.9×)	61.2 (1.9×)	**77.1** (1.9×)	–	–	–	–	–	–	–	–	–
4	–	–	–	91.5 (**3.3**×)	60.4 (**2.8**×)	76.0 (**3.0**×)	**91.5** (4.1×)	59.4 (**3.0**×)	75.4 (**3.5**×)	91.5 (**4.8**×)	58.8 (3.1×)	75.2 (3.8×)
7	–	–	–	91.5 (3.2×)	58.8 (2.6×)	75.2 (2.9×)	–	–	–	–	–	–
8	–	–	–	–	–	–	90.0 (**4.3**×)	58.0 (2.8×)	74.0 (3.4×)	92.0 (4.7×)	**61.0** (3.2×)	**76.5** (3.8×)
15	–	–	–	–	–	–	88.5 (4.1×)	56.8 (2.7×)	72.6 (3.3×)	–	–	–
16	–	–	–	–	–	–	–	–	–	85.5 (4.1×)	55.2 (2.9×)	70.3 (3.4×)
31	–	–	–	–	–	–	–	–	–	84.5 (3.8×)	50.4 (2.8×)	67.5 (3.2×)

A.8 Ablation on rejection-sampling ratio tempering

Ratio tempering. Table 11 studies tempering of the rejection-sampling ratio, replacing q_i/p_i with $(q_i/p_i)^\gamma$. Smaller γ makes acceptance more aggressive, while larger γ makes it more conservative. We observe that a slightly larger value, e.g. $\gamma = 1.25$, can give a small accuracy improvement in some settings, especially for config-B, but typically with a minor speed degradation. Overall, the default choice $\gamma = 1$ already provides a good tradeoff and is used in the main experiments.

Table 7: Ablation of score-threshold policies with **entropy**-based token acceptance estimator. Static score uses $s = \hat{K} - c$; dynamic score uses $s = \hat{K} - c \cdot N_{\text{hi}}$.

τ_{score}	c	B=4			B=8			B=16			B=32		
		GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg
Static score $s = \hat{K} - c$													
-5	1	90.0 (2.0×)	62.0 (1.9×)	76.0 (2.0×)	92.5 (3.3×)	**62.6** (2.6×)	**77.6** (2.9×)	90.0 (3.6×)	59.0 (2.9×)	74.5 (3.2×)	89.5 (**4.7**×)	59.8 (**3.4**×)	74.6 (**4.0**×)
	2	93.0 (2.1×)	62.0 (2.0×)	77.5 (2.1×)	91.5 (2.8×)	**62.6** (2.4×)	77.0 (2.6×)	91.0 (4.0×)	59.8 (3.2×)	75.4 (3.6×)	90.0 (4.2×)	57.0 (3.2×)	73.5 (3.7×)
	4	93.0 (2.3×)	61.8 (2.0×)	77.4 (2.1×)	91.5 (3.0×)	61.2 (2.5×)	76.4 (2.8×)	90.5 (**4.2**×)	61.4 (3.0×)	76.0 (3.5×)	90.0 (3.9×)	59.8 (3.0×)	74.9 (3.4×)
-1	1	90.0 (2.2×)	62.0 (2.0×)	76.0 (2.1×)	90.0 (3.0×)	61.2 (2.5×)	75.6 (2.7×)	90.5 (**4.2**×)	61.4 (3.2×)	76.0 (**3.7**×)	90.0 (4.4×)	59.8 (3.1×)	74.9 (3.7×)
	2	92.0 (2.0×)	62.2 (1.9×)	77.1 (2.0×)	90.5 (3.3×)	61.8 (2.8×)	76.2 (3.0×)	89.5 (3.8×)	59.4 (3.0×)	74.4 (3.3×)	92.0 (4.7×)	**60.8** (3.4×)	76.4 (4.0×)
	4	89.5 (2.0×)	**63.8** (2.0×)	76.6 (2.0×)	**94.0** (3.4×)	60.6 (2.8×)	77.3 (3.1×)	91.0 (3.6×)	**62.4** (2.8×)	**76.7** (3.2×)	87.5 (4.5×)	58.2 (3.3×)	72.8 (3.9×)
0	1	92.0 (2.3×)	62.2 (2.0×)	77.1 (2.1×)	88.5 (3.3×)	61.8 (2.7×)	75.2 (3.0×)	90.5 (3.6×)	59.4 (2.9×)	75.0 (3.2×)	**93.0** (4.5×)	**60.8** (3.4×)	**76.9** (3.9×)
	2	**93.5** (2.3×)	62.4 (2.1×)	**78.0** (2.2×)	92.0 (3.2×)	58.6 (2.4×)	75.3 (2.8×)	**93.0** (**4.2**×)	58.2 (3.1×)	75.6 (3.6×)	90.5 (4.3×)	59.4 (3.1×)	75.0 (3.6×)
	4	90.0 (2.5×)	59.4 (2.2×)	74.7 (2.3×)	91.5 (3.3×)	59.8 (2.5×)	75.6 (2.9×)	90.5 (4.1×)	58.8 (3.2×)	74.6 (3.6×)	91.5 (4.2×)	56.2 (3.0×)	73.9 (3.5×)
1	1	90.5 (2.3×)	62.4 (2.0×)	76.4 (2.2×)	92.0 (3.2×)	57.2 (2.7×)	74.6 (2.9×)	91.0 (3.9×)	58.2 (3.2×)	74.6 (3.5×)	90.5 (4.3×)	59.4 (3.1×)	75.0 (3.6×)
	2	89.5 (2.1×)	61.2 (1.9×)	75.4 (2.0×)	91.0 (3.5×)	61.6 (2.8×)	76.3 (3.1×)	91.0 (3.8×)	**62.4** (3.0×)	**76.7** (3.4×)	89.5 (4.7×)	60.2 (3.3×)	74.8 (3.9×)
	4	90.5 (2.2×)	59.4 (2.0×)	75.0 (2.1×)	91.0 (3.5×)	56.8 (2.9×)	73.9 (3.2×)	90.0 (3.9×)	53.4 (2.9×)	71.7 (3.3×)	87.5 (3.6×)	55.2 (3.2×)	71.4 (3.4×)
5	1	90.5 (2.8×)	60.6 (1.9×)	75.6 (2.2×)	93.0 (3.3×)	56.2 (2.9×)	74.6 (3.1×)	90.5 (3.7×)	53.8 (2.9×)	72.2 (3.2×)	89.5 (3.5×)	53.0 (3.0×)	71.2 (3.2×)
	2	91.0 (**2.9**×)	60.6 (2.2×)	75.8 (2.5×)	91.5 (2.9×)	55.0 (2.6×)	73.2 (2.8×)	86.5 (3.7×)	51.6 (2.9×)	69.0 (3.3×)	88.0 (3.2×)	48.6 (2.6×)	68.3 (2.9×)
	4	91.0 (**2.9**×)	60.6 (2.2×)	75.8 (2.5×)	90.0 (3.2×)	53.4 (2.7×)	71.7 (2.9×)	88.0 (3.4×)	49.4 (2.8×)	68.7 (3.1×)	83.5 (2.9×)	44.8 (2.6×)	64.1 (2.7×)
Dynamic score $s = \hat{K} - c \cdot N_{\text{hi}}$													
-5	1	92.0 (2.0×)	62.0 (1.8×)	77.0 (1.9×)	**92.5** (3.0×)	**62.6** (2.5×)	**77.6** (2.8×)	90.0 (3.8×)	59.0 (2.9×)	74.5 (3.3×)	91.0 (4.5×)	58.2 (3.4×)	74.6 (3.9×)
-1	1	89.0 (2.2×)	62.0 (2.0×)	75.5 (2.1×)	91.5 (2.8×)	61.2 (2.4×)	76.4 (2.6×)	90.0 (3.9×)	**60.0** (3.2×)	75.0 (3.5×)	**92.0** (4.1×)	58.4 (3.0×)	**75.2** (3.5×)
0	1	**93.5** (2.4×)	60.6 (2.1×)	77.0 (2.3×)	90.5 (**3.8**×)	59.6 (**2.8**×)	75.0 (**3.3**×)	90.5 (**4.1**×)	**60.0** (2.9×)	**75.2** (3.4×)	89.0 (**4.8**×)	**59.4** (**3.5**×)	74.2 (**4.1**×)
1	1	92.5 (**2.9**×)	**62.6** (**2.4**×)	**77.6** (**2.6**×)	86.0 (3.4×)	57.6 (2.6×)	71.8 (3.0×)	**92.0** (3.7×)	58.6 (**3.3**×)	74.3 (**3.7**×)	91.5 (4.3×)	58.6 (3.4×)	75.0 (3.8×)
5	1	90.5 (2.7×)	60.6 (2.0×)	75.6 (2.3×)	89.5 (3.3×)	53.2 (2.7×)	71.4 (3.0×)	87.0 (3.3×)	49.6 (2.8×)	68.3 (3.0×)	84.5 (3.1×)	46.8 (3.0×)	65.6 (3.0×)

Table 8: Ablation of score-threshold policies with **margin**-based token acceptance estimator. Static score uses $s = \hat{K} - c$; dynamic score uses $s = \hat{K} - c \cdot N_{\text{hi}}$.

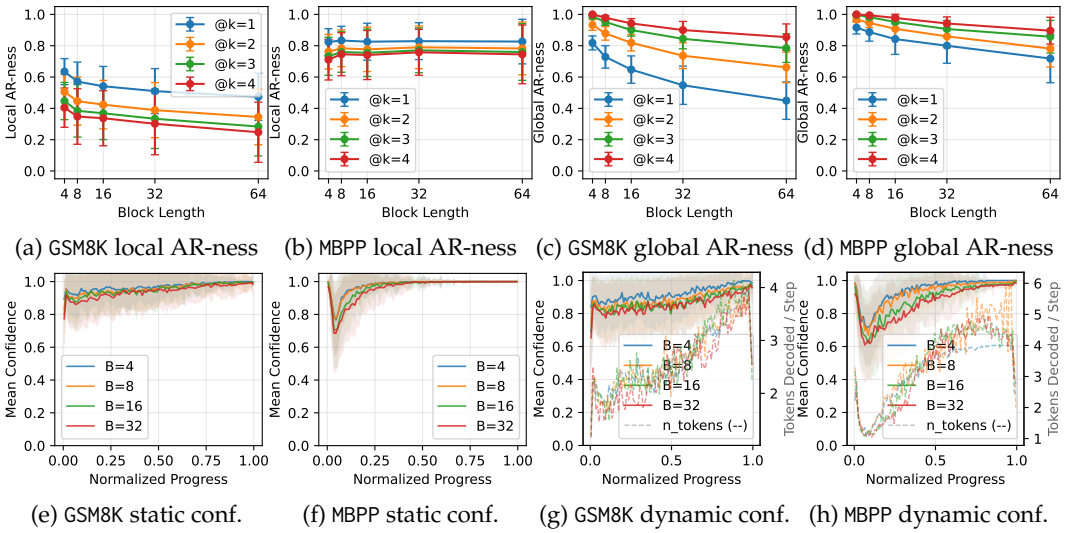
τ_{score}	c	B=4			B=8			B=16			B=32		
		GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg
Static score $s = \hat{K} - c$													
-5	1	93.0 (2.2×)	59.6 (2.1×)	76.3 (2.1×)	90.5 (3.1×)	56.2 (2.7×)	73.4 (2.9×)	89.5 (3.4×)	57.0 (3.1×)	73.2 (3.3×)	89.5 (3.7×)	57.2 (**3.6**×)	73.4 (3.6×)
	2	93.0 (2.2×)	59.6 (1.9×)	76.3 (2.1×)	90.5 (3.1×)	56.2 (2.7×)	73.4 (2.9×)	90.5 (3.5×)	57.0 (3.2×)	73.8 (3.3×)	89.0 (3.7×)	55.6 (3.2×)	72.3 (3.4×)
	4	93.0 (2.2×)	**62.0** (2.0×)	**77.5** (2.1×)	89.0 (3.1×)	56.6 (2.9×)	72.8 (3.0×)	89.5 (3.6×)	55.2 (3.2×)	72.4 (3.4×)	89.5 (3.7×)	55.6 (3.3×)	72.5 (3.5×)
-1	1	93.0 (2.2×)	61.8 (2.0×)	77.4 (2.1×)	91.0 (3.1×)	**62.6** (2.4×)	**76.8** (2.8×)	90.5 (4.1×)	61.4 (3.1×)	**76.0** (3.6×)	**92.5** (4.3×)	59.8 (3.1×)	**76.2** (3.6×)
	2	88.0 (2.3×)	57.8 (1.9×)	72.9 (2.0×)	92.5 (3.1×)	56.0 (2.2×)	74.2 (2.6×)	89.5 (4.0×)	58.2 (3.0×)	73.8 (3.5×)	92.0 (4.0×)	55.6 (3.0×)	73.8 (3.4×)
	4	92.0 (2.2×)	61.2 (2.1×)	76.6 (2.1×)	89.5 (3.3×)	61.0 (2.6×)	75.2 (2.9×)	89.0 (**4.2**×)	59.2 (3.0×)	74.1 (3.5×)	88.0 (4.4×)	**61.0** (3.2×)	74.5 (3.8×)
0	1	89.5 (2.1×)	59.8 (2.0×)	74.6 (2.1×)	89.0 (3.3×)	54.4 (2.8×)	71.7 (3.0×)	90.0 (3.5×)	54.4 (3.0×)	72.2 (3.2×)	85.5 (3.6×)	55.0 (3.4×)	70.2 (3.5×)
	2	90.5 (2.3×)	59.2 (2.1×)	74.8 (2.2×)	90.5 (3.4×)	55.6 (2.9×)	73.0 (3.2×)	86.0 (3.7×)	55.8 (3.1×)	70.9 (3.4×)	85.5 (3.6×)	52.8 (3.4×)	69.2 (3.5×)
	4	92.0 (2.3×)	60.0 (2.2×)	76.0 (2.2×)	93.0 (3.5×)	57.0 (**3.2**×)	75.0 (3.3×)	88.0 (3.7×)	54.8 (3.2×)	71.4 (3.5×)	87.5 (3.7×)	50.4 (3.5×)	69.0 (3.6×)
1	1	90.0 (2.3×)	59.2 (**2.3**×)	74.6 (2.3×)	89.5 (3.2×)	54.4 (2.5×)	72.0 (2.8×)	88.0 (3.9×)	57.6 (3.3×)	72.8 (3.6×)	85.0 (3.3×)	53.0 (3.3×)	69.0 (3.3×)
	2	**93.5** (2.4×)	60.2 (2.2×)	76.8 (2.3×)	93.0 (3.3×)	56.2 (2.8×)	74.6 (3.0×)	87.0 (4.1×)	54.2 (3.3×)	70.6 (**3.7**×)	88.0 (3.3×)	57.0 (3.2×)	72.5 (3.2×)
	4	91.0 (**3.0**×)	60.6 (**2.3**×)	75.8 (**2.6**×)	90.5 (3.3×)	55.6 (2.8×)	73.0 (3.0×)	88.5 (3.7×)	54.2 (**3.5**×)	71.4 (3.6×)	86.0 (3.5×)	49.6 (3.1×)	67.8 (3.3×)
5	1	90.5 (2.7×)	61.4 (2.0×)	76.0 (2.3×)	**93.5** (3.5×)	57.2 (3.0×)	75.4 (**3.3**×)	89.5 (3.5×)	50.6 (2.9×)	70.0 (3.2×)	84.5 (3.5×)	51.2 (3.2×)	67.8 (3.4×)
	2	91.0 (2.5×)	61.4 (2.0×)	76.2 (2.2×)	90.5 (**3.6**×)	56.2 (3.0×)	73.4 (**3.3**×)	86.5 (3.3×)	53.2 (2.9×)	69.8 (3.1×)	86.0 (3.3×)	48.8 (2.9×)	67.4 (3.1×)
	4	90.5 (2.7×)	61.4 (2.1×)	76.0 (2.4×)	89.5 (3.4×)	53.4 (2.8×)	71.5 (3.1×)	89.5 (3.5×)	51.6 (2.8×)	70.5 (3.1×)	86.0 (3.4×)	50.4 (3.0×)	68.2 (3.2×)
Dynamic score $s = \hat{K} - c \cdot N_{\text{hi}}$													
-5	1	90.0 (1.9×)	**62.0** (1.9×)	76.0 (1.9×)	90.0 (3.2×)	62.2 (2.7×)	76.1 (2.9×)	90.0 (3.9×)	**61.8** (2.6×)	**75.9** (3.1×)	88.5 (**4.6**×)	**61.0** (3.3×)	**74.8** (**3.9**×)
-1	1	92.5 (2.1×)	60.2 (2.0×)	**76.4** (2.0×)	**93.5** (2.8×)	61.4 (2.3×)	**77.5** (2.6×)	89.0 (4.0×)	59.6 (3.0×)	74.3 (3.5×)	90.5 (4.1×)	58.8 (2.9×)	74.6 (3.4×)
0	1	93.0 (2.1×)	57.6 (1.9×)	75.3 (2.0×)	86.5 (3.3×)	59.0 (2.9×)	72.8 (3.1×)	88.5 (3.7×)	56.8 (3.1×)	72.6 (3.4×)	88.5 (3.5×)	56.0 (**3.6**×)	72.2 (3.6×)
1	1	**93.5** (**2.9**×)	58.6 (**2.3**×)	76.0 (**2.6**×)	90.0 (3.3×)	56.2 (2.8×)	73.1 (3.1×)	**92.0** (3.7×)	54.8 (**3.3**×)	73.4 (**3.7**×)	87.0 (3.6×)	52.2 (3.3×)	69.6 (3.4×)
5	1	90.5 (2.6×)	60.6 (2.0×)	75.6 (2.3×)	92.0 (**3.6**×)	55.0 (**3.0**×)	73.5 (**3.3**×)	85.5 (3.6×)	51.8 (3.1×)	68.7 (3.3×)	85.0 (3.4×)	51.2 (3.3×)	68.1 (3.3×)

Table 9: Ablation of hysteresis policies with **entropy**-based token acceptance estimator. We use dynamic score only, with fixed $c = 1$.

τ_{on}	τ_{off}	B=4			B=8			B=16			B=32		
		GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg
0	-5	90.5 (1.8×)	**62.4** (1.9×)	76.4 (1.8×)	**91.5** (3.2×)	**63.6** (2.7×)	**77.6** (3.0×)	89.0 (4.0×)	60.2 (2.6×)	74.6 (3.1×)	91.0 (4.3×)	59.2 (3.0×)	75.1 (3.5×)
	-1	**93.0** (2.1×)	**62.4** (1.9×)	**77.7** (2.0×)	89.5 (3.2×)	**63.6** (2.6×)	76.6 (2.9×)	91.5 (**4.1**×)	60.2 (3.0×)	75.8 (3.5×)	91.0 (4.0×)	**59.4** (3.1×)	**75.2** (3.5×)
1	-5	91.5 (2.1×)	**62.4** (**2.0**×)	77.0 (2.0×)	89.0 (3.3×)	59.8 (2.7×)	74.4 (3.0×)	91.5 (3.8×)	59.4 (2.9×)	75.4 (3.3×)	90.5 (4.2×)	55.8 (**3.2**×)	73.2 (**3.7**×)
	-1	91.5 (1.9×)	**62.4** (1.9×)	77.0 (1.9×)	88.5 (3.1×)	59.8 (2.6×)	74.2 (2.9×)	91.0 (**4.1**×)	**61.4** (**3.2**×)	**76.2** (**3.6**×)	**91.5** (**4.4**×)	57.4 (3.1×)	74.4 (3.6×)
5	-5	91.0 (**2.5**×)	60.6 (**2.0**×)	75.8 (**2.3**×)	89.5 (**3.4**×)	53.6 (**2.8**×)	71.5 (**3.1**×)	92.5 (3.7×)	50.4 (2.9×)	71.5 (3.3×)	87.5 (3.7×)	51.6 (3.1×)	69.5 (3.4×)
	-1	90.5 (2.4×)	60.6 (**2.0**×)	75.8 (**2.3**×)	91.0 (3.3×)	55.4 (**2.8**×)	73.2 (3.0×)	**93.0** (3.3×)	51.6 (2.8×)	72.3 (3.1×)	88.5 (3.8×)	50.2 (3.0×)	69.3 (3.4×)

Table 10: Ablation of contextual-bandit UCB policies. The context tuple (a, b, c) denotes the numbers of bins for mask span length, decoding progress, and entropy, respectively. We report results for different UCB exploration coefficients β.

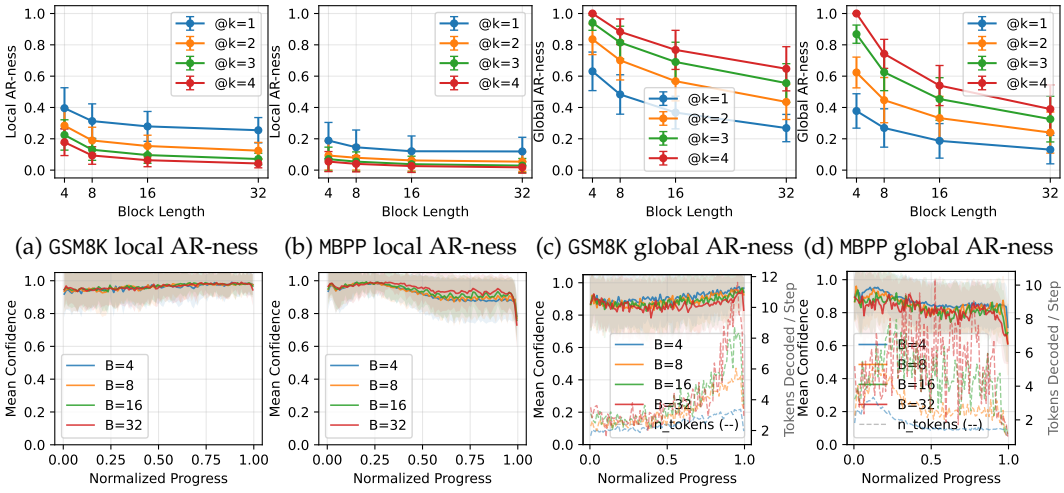
β	B=4			B=8			B=16			B=32		
	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg
Tuple $(1,2,2)$												
0.2	**92.5** (2.2×)	53.4 (1.9×)	73.0 (2.1×)	87.0 (3.4×)	49.0 (2.5×)	68.0 (2.9×)	87.0 (3.4×)	50.2 (2.7×)	68.6 (3.0×)	83.5 (4.2×)	52.6 (**3.2**×)	68.0 (3.6×)
0.5	91.5 (2.2×)	55.4 (2.0×)	73.5 (2.1×)	**89.5** (**3.5**×)	51.6 (**2.7**×)	70.5 (**3.1**×)	89.5 (3.6×)	52.0 (2.8×)	**70.8** (3.2×)	82.5 (3.8×)	49.6 (2.8×)	66.0 (3.2×)
1	**92.5** (2.3×)	55.0 (2.0×)	73.8 (2.2×)	85.5 (3.4×)	47.6 (**2.7**×)	66.5 (3.0×)	86.0 (3.9×)	50.8 (2.9×)	68.4 (3.3×)	81.5 (3.5×)	48.6 (2.9×)	65.0 (3.2×)
2	91.5 (2.3×)	53.6 (1.8×)	72.5 (2.0×)	87.5 (2.7×)	49.8 (2.4×)	68.7 (2.5×)	87.5 (3.8×)	48.0 (2.9×)	67.8 (3.3×)	83.0 (3.8×)	50.0 (3.0×)	66.5 (3.3×)
5	89.5 (2.3×)	46.0 (2.0×)	67.8 (2.1×)	84.0 (2.8×)	44.6 (2.3×)	64.3 (2.5×)	85.0 (3.5×)	47.4 (2.7×)	66.2 (3.1×)	82.5 (3.7×)	46.8 (2.9×)	64.6 (3.3×)
Tuple $(2,1,2)$												
0.2	90.0 (**2.5**×)	57.0 (2.1×)	73.5 (**2.3**×)	85.0 (3.1×)	55.2 (2.6×)	70.1 (2.8×)	85.0 (3.9×)	53.2 (**3.1**×)	69.1 (3.4×)	83.5 (3.8×)	52.8 (3.0×)	68.2 (3.4×)
0.5	90.5 (**2.5**×)	**61.4** (2.1×)	**76.0** (**2.3**×)	88.5 (3.1×)	55.0 (2.6×)	**71.8** (2.9×)	88.0 (**4.0**×)	50.2 (**3.1**×)	69.1 (**3.5**×)	85.5 (4.0×)	48.6 (2.6×)	67.0 (3.2×)
1	89.5 (2.4×)	52.2 (2.0×)	70.9 (2.2×)	84.0 (2.9×)	53.8 (2.5×)	68.9 (2.7×)	84.5 (3.8×)	52.2 (3.0×)	68.3 (3.4×)	82.5 (3.6×)	50.2 (2.8×)	66.3 (3.2×)
2	90.0 (2.2×)	50.4 (2.0×)	70.2 (2.1×)	86.0 (2.9×)	50.2 (2.6×)	68.1 (2.8×)	88.0 (3.7×)	50.6 (2.8×)	69.3 (3.2×)	81.5 (4.0×)	48.6 (2.9×)	65.0 (3.4×)
5	88.5 (**2.5**×)	52.4 (2.0×)	70.5 (2.2×)	83.0 (2.9×)	45.2 (2.3×)	64.1 (2.6×)	87.0 (3.7×)	46.6 (2.9×)	66.8 (3.2×)	86.0 (4.0×)	46.8 (2.9×)	66.4 (3.4×)
Tuple $(2,2,1)$												
0.2	91.0 (**2.5**×)	56.6 (**2.2**×)	73.8 (**2.3**×)	87.5 (3.2×)	**55.6** (2.6×)	71.5 (2.9×)	84.5 (3.5×)	52.8 (2.7×)	68.7 (3.1×)	**87.5** (4.1×)	49.8 (3.0×)	68.7 (3.5×)
0.5	90.0 (2.2×)	57.2 (**2.2**×)	73.6 (2.2×)	89.0 (3.2×)	53.4 (**2.7**×)	71.2 (2.9×)	85.5 (3.3×)	53.2 (2.8×)	69.3 (3.0×)	85.0 (**4.4**×)	51.6 (3.0×)	68.3 (3.6×)
1	91.0 (2.2×)	53.2 (2.0×)	72.1 (2.1×)	85.0 (3.3×)	51.8 (**2.7**×)	68.4 (3.0×)	84.5 (3.6×)	51.4 (2.7×)	68.0 (3.1×)	85.5 (4.2×)	50.0 (3.1×)	67.8 (3.6×)
2	88.5 (2.1×)	53.0 (1.9×)	70.8 (2.0×)	85.0 (3.2×)	49.2 (2.6×)	67.1 (2.9×)	84.5 (3.5×)	50.2 (2.8×)	67.3 (3.1×)	86.5 (4.1×)	51.6 (2.9×)	69.0 (3.4×)
5	90.0 (2.1×)	51.8 (1.8×)	70.9 (1.9×)	87.0 (3.1×)	45.8 (2.6×)	66.4 (2.8×)	84.5 (3.6×)	47.8 (2.8×)	66.1 (3.2×)	82.5 (3.9×)	46.0 (3.0×)	64.2 (3.4×)
Tuple $(2,2,2)$												
0.2	91.0 (**2.5**×)	56.2 (2.1×)	73.6 (**2.3**×)	87.5 (3.0×)	54.8 (2.4×)	71.2 (2.7×)	84.0 (3.6×)	51.4 (3.0×)	67.7 (3.3×)	84.5 (3.9×)	49.8 (2.9×)	67.2 (3.3×)
0.5	**92.5** (2.4×)	57.2 (2.1×)	74.8 (2.2×)	87.5 (3.0×)	53.4 (2.5×)	70.5 (2.7×)	87.5 (3.6×)	53.2 (**3.1**×)	70.3 (3.4×)	83.5 (3.8×)	53.8 (2.9×)	68.7 (3.3×)
1	91.5 (2.5×)	53.2 (**2.2**×)	72.4 (**2.3**×)	86.0 (3.1×)	51.0 (2.5×)	68.5 (2.8×)	86.0 (3.9×)	51.4 (3.0×)	68.7 (3.4×)	86.5 (3.8×)	50.0 (2.7×)	68.2 (3.2×)
2	89.5 (2.4×)	50.6 (2.0×)	70.0 (2.2×)	84.5 (3.1×)	52.4 (2.2×)	68.5 (2.6×)	89.5 (3.6×)	51.4 (2.9×)	70.5 (3.2×)	86.5 (4.1×)	51.6 (2.9×)	69.0 (3.4×)
5	90.0 (2.2×)	51.8 (2.0×)	70.9 (2.1×)	85.0 (3.0×)	45.8 (2.4×)	65.4 (2.7×)	84.0 (3.4×)	47.8 (2.6×)	65.9 (2.9×)	81.5 (3.9×)	46.0 (3.1×)	63.7 (3.4×)
Tuple $(4,4,4)$												
0.2	88.0 (2.3×)	53.4 (2.0×)	70.7 (2.1×)	**89.5** (3.3×)	51.8 (2.6×)	70.7 (2.9×)	87.5 (3.6×)	53.4 (2.8×)	70.5 (3.2×)	85.5 (4.0×)	51.8 (3.1×)	68.7 (3.5×)
0.5	88.5 (2.1×)	54.8 (1.9×)	71.7 (2.0×)	86.0 (3.2×)	52.0 (2.6×)	69.0 (2.9×)	85.0 (3.6×)	53.6 (2.8×)	69.3 (3.1×)	83.0 (4.2×)	**54.6** (3.1×)	68.8 (3.6×)
1	89.5 (2.2×)	52.4 (1.9×)	71.0 (2.1×)	86.0 (3.2×)	51.4 (2.5×)	68.7 (2.8×)	88.5 (3.5×)	52.8 (2.9×)	70.7 (3.1×)	87.0 (4.1×)	53.8 (3.1×)	**70.4** (3.6×)
2	88.5 (2.2×)	51.2 (1.9×)	69.8 (2.0×)	86.5 (3.1×)	49.0 (2.6×)	67.8 (2.8×)	**90.0** (3.6×)	**54.8** (2.8×)	68.4 (3.1×)	84.5 (4.3×)	52.6 (**3.2**×)	68.5 (**3.7**×)
5	89.5 (2.1×)	51.0 (2.0×)	70.2 (2.0×)	83.0 (3.0×)	52.0 (2.5×)	67.5 (2.7×)	89.0 (3.6×)	52.8 (2.8×)	70.9 (3.2×)	82.5 (4.0×)	52.6 (3.0×)	67.5 (3.4×)



(a) GSM8K local AR-ness (b) MBPP local AR-ness (c) GSM8K global AR-ness (d) MBPP global AR-ness

(e) GSM8K static conf. (f) MBPP static conf. (g) GSM8K dynamic conf. (h) MBPP dynamic conf.

Figure 5: AR-ness and decoding confidence statistics for SDAR-8B-Chat. Top row: local/global AR-ness on GSM8K and MBPP. Bottom row: normalized confidence statistics under static and dynamic decoding on GSM8K and MBPP.



(a) GSM8K local AR-ness (b) MBPP local AR-ness (c) GSM8K global AR-ness (d) MBPP global AR-ness

(e) GSM8K static conf. (f) MBPP static conf. (g) GSM8K dynamic conf. (h) MBPP dynamic conf.

Figure 6: AR-ness and decoding confidence statistics for LLaDA-2.1-Mini. Top row: local/global AR-ness on GSM8K and MBPP. Bottom row: normalized confidence statistics under static and dynamic decoding on GSM8K and MBPP.

Table 11: Ablation of rejection-sampling ratio tempering for SSD on SDAR-8B-Chat. We vary the tempering factor γ in the acceptance ratio $(q_i/p_i)^\gamma$ and report accuracy (speedup relative to γ=1).

γ	Config-A			Config-B		
	GSM8K	MBPP	Avg	GSM8K	MBPP	Avg
0.5	89.5 (1.02×)	60.0 (1.06×)	74.8 (1.03×)	88.5 (1.02×)	58.0 (1.00×)	73.2 (1.01×)
0.75	89.5 (0.99×)	62.0 (1.06×)	75.8 (1.01×)	**90.5** (0.99×)	58.5 (0.95×)	74.5 (0.98×)
1.0	89.5 (1.00×)	**64.0** (1.00×)	**76.8** (1.00×)	89.5 (1.00×)	**60.5** (1.00×)	75.0 (1.00×)
1.25	89.5 (1.00×)	60.5 (1.07×)	75.0 (1.02×)	**90.5** (0.93×)	**60.5** (0.96×)	**75.5** (0.94×)
1.5	**90.5** (1.00×)	60.5 (0.96×)	75.5 (0.99×)	**90.5** (1.01×)	60.0 (0.96×)	75.2 (1.00×)