# DSPEC: EFFICIENT SPECULATIVE DECODING FOR BLOCK-DIFFUSION LANGUAGE MODELS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Speculative decoding accelerates a target language model by letting a lightweight drafter propose several tokens that the target verifies together. For block-diffusion language models, which decode one block of tokens at a time, it offers an alternative to the prevailing approach of lowering a confidence threshold, which trades generation quality for speed. Existing speculative decoders for these models, however, use trajectory-based or autoregressive-style verification, whose draft quality and continuation-state costs both affect acceleration. We present DSPEC, which rests on the observation that a target pass over a fully masked block already predicts every position and thus yields a complete draft. DSPEC uses this draft directly or corrects it with a one-layer head that reads the target's representation at each position. A single folded target pass then verifies the draft while preparing the block's key–value state and the next block's draft, so a fully accepted block costs only one pass. Requests rejected at different positions are recovered jointly in one shared pass. In SGLang, DSPEC achieves 1.6–4.3× the throughput of Fast-dLLM's threshold-1.0 decoding on LLaDA2.0-mini and SDAR-8B at batch sizes 1–128. At batch one, its seven-task mean quality is within about one percentage point of that baseline. On matched hardware, it is roughly twice as fast as the fastest of three existing speculative decoders. Our analysis further shows that longer acceptance does not guarantee higher throughput, so proposal quality and continuation cost must be optimized together.

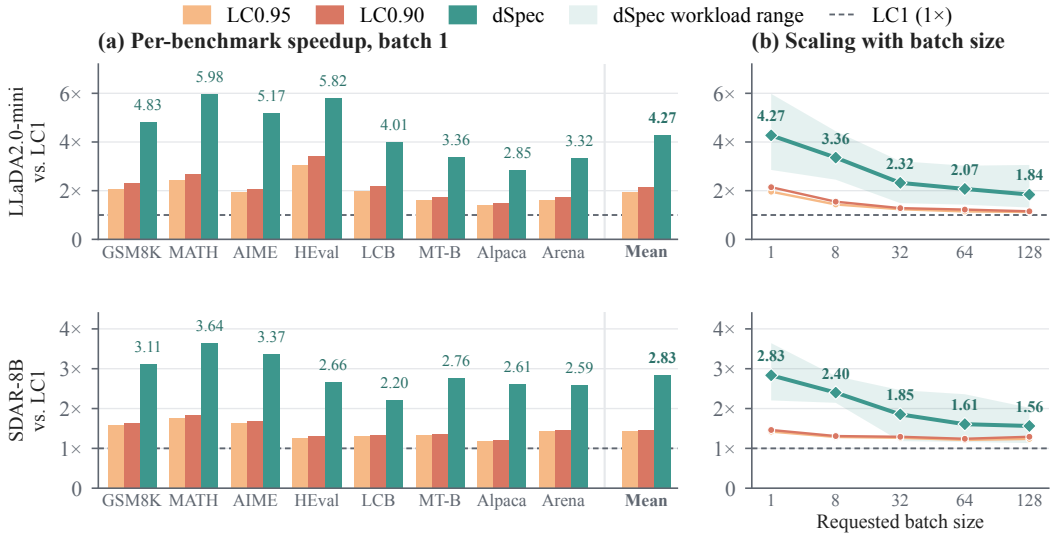


Figure 1: Speedup of DSPEC over confidence-threshold decoding at threshold 1.0 (LC1, dashed) in SGLang. (a) Each workload at batch size 1. (b) Geometric mean at each batch size; the band spans the workloads.

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

# 1 INTRODUCTION

Practical block-diffusion decoders commit predictions whose confidence clears a threshold (Wu et al., 2026b; Cheng et al., 2025; Bie et al., 2025); lowering it raises throughput at the cost of quality. Speculative decoding instead separates inexpensive proposals from target-verified commitments (Leviathan et al., 2023; Chen et al., 2023). For autoregressive targets, verification also produces the accepted prefix's key–value (KV) states. Within a bidirectional block, verification and continuation can use different token contexts, so verification KV does not automatically provide the state needed to continue.

A target pass over a fully masked block already predicts every position. The target's own information thus gives drafting a strong starting point: its predictions supply candidate tokens without a second model (Wu & Zhang, 2025), and the underlying representations are natural inputs for correcting each position. We use every position to form a complete draft and ask how to verify it while preparing the continuation states needed for the next block.

Trajectory verification, used by FreeDave, SSD, and Spiffy (Wu & Zhang, 2025; Gao et al., 2025; Agrawal et al., 2026), checks candidate denoising states against a reference sampler's path. Exact state evaluation preserves that sampler; Spiffy's practical graph layout is exact under block-causal attention or dual caches and near-lossless for full-sequence bidirectional models. Acceptance depends on both token predictions and unmasking order, so correct tokens can still be rejected when proposed in the wrong order.

Autoregressive-style verification, used by S2D2 and SimSD (Han et al., 2026; Cui et al., 2026), uses target-model verification: S2D2 checks self-drafted tokens in block-size-one mode, while SimSD checks a separate drafter's tokens with a temporal attention mask. Their verification layouts do not produce the bidirectional block state needed to continue, so that state is rebuilt after every commitment. No existing method combines complete drafts from the target, verification that also prepares continuation states, and batched recovery of requests that reject at different positions.

DSPEC uses the target's masked-block predictions, called the anchor, directly as a complete Self-Draft, or refines them with an in-position head in Learned Draft. A single *folded* target pass checks the draft against left-to-right (L2R) predictions, builds candidate-block KV, and predicts the next anchor (figure 2). Full acceptance reuses these states; requests rejected at different positions are reconstructed jointly. Shared intermediate representations make verification approximate, and we evaluate its effect on complete generations.

In SGLang (Zheng et al., 2024), DSPEC (Learned Draft) runs 1.56–4.27× faster than threshold-1.0 decoding on LLaDA2.0-mini and SDAR-8B at batch sizes 1–128. At batch one, its seven-task mean quality changes by $+0.23$ and $-1.00$ percentage points, respectively. Our contributions are:

- **Verification with reusable continuation states.** We combine approximate L2R verification, candidate KV construction, and next-block prediction in one target pass. Full acceptance reuses these states; batched recovery reconstructs states for rejected requests with unequal committed lengths. With this execution path, the target's own masked-block predictions provide a competitive complete draft without a learned drafter: Self-Draft achieves about twice FreeDave's throughput on SDAR-8B (sections 3.3, 4.2 and 4.5).
- **In-position refinement of target predictions.** Self-Draft uses the target's masked-block predictions as a complete proposal. Learned Draft optionally applies a one-layer residual head to each position's target representation to refine its existing logits, with acceptance and throughput evaluated separately (sections 3.1, 3.2 and 4.3).
- **Acceptance-dependent gains under serving.** Across two targets and batches 1–128, we quantify the speed–quality trade-off. Learned refinement improves throughput on LLaDA2 but not SDAR; high full-acceptance rates favor state reuse, while low rates make batched recovery more valuable (sections 4.2 and 4.3).

# 2 PRELIMINARIES

**Block generation.** Let $T$ be the frozen target, $c$ the committed prefix, and $B$ the next block's length. Bidirectional within-block attention lets visible tokens and masks jointly determine each

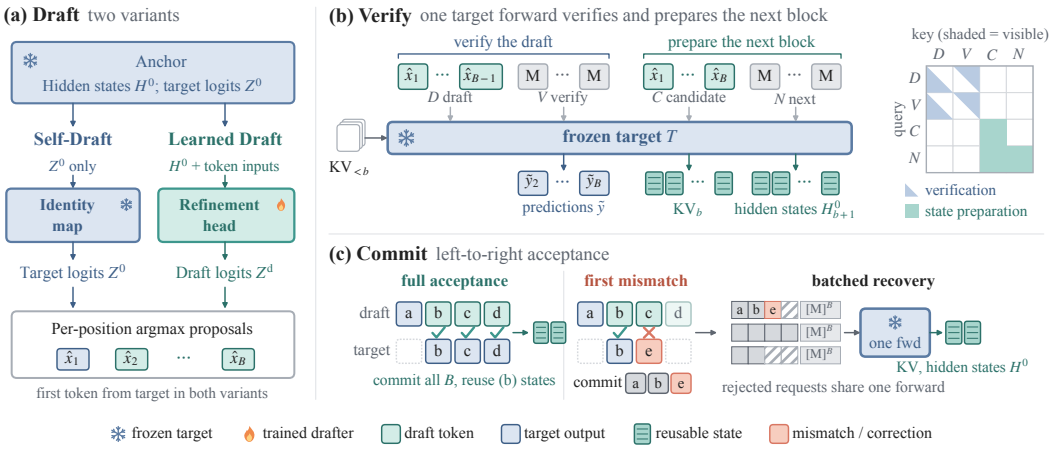


Figure 2: The two DSPEC variants share one decoding path. (a) Self-Draft takes the anchor's argmax and Learned Draft adds the refinement head; (b) one target pass checks the draft in regions $D/V$, where each query sees the preceding draft tokens and the remaining masks, while $C/N$ prepare the candidate KV and the next anchor without reading $D/V$; (c) full acceptance keeps these states, and rejected requests are rebuilt in one batched pass.

position's hidden state. Confidence-threshold decoding repeatedly evaluates this block, committing predictions above a threshold and at least one position per pass (Wu et al., 2026b).

**Proposing and committing.** DSPEC verifies candidate tokens in parallel and accepts a consecutive L2R prefix. At the first mismatch, it commits the preceding candidates and the verifier's replacement, discarding the remaining candidates. This extension of $c$ is a *block step*; continuation requires committed-token KV and next-block predictions.

**Verification and continuation states.** An accepted autoregressive prefix retains its verification context. In a bidirectional block, changing later tokens can change earlier representations. Consequently, neither masked-block KV nor KV from a different verification layout automatically supplies the desired continuation state. Separate execution rebuilds committed KV and next-block predictions after verification. Both DSPEC variants prepare candidate states during verification, reusing them on full acceptance and reconstructing them on rejection.

## 3 METHOD

DSPEC drafts a complete block from the anchor, verifies it and prepares the next block in one folded pass, and jointly reconstructs the states of rejected requests (figure 2). Acceptance extends the output contiguously from left to right (Hou & Kwok, 2026); appendix H covers partially filled blocks.

### 3.1 SELF-DRAFT: DRAFTING FROM THE ANCHOR

A target pass over the prefix $c$ followed by $B$ masks gives the final normalized hidden states $H^0 \in \mathbb{R}^{B\times d}$ and the logits $Z^0$, which together form the anchor:

$$H^0 = T_{\text{hidden}}(c, \texttt{[MASK]}^B), \qquad Z^0 = H^0 W^\top, \tag{1}$$

where $d$ is the target hidden size and $W \in \mathbb{R}^{|\mathcal{V}|\times d}$ is the frozen vocabulary head. DSPEC (Self-Draft) takes the most likely token at every position,

$$\hat{x}_i = \arg\max_v Z^0_{i,v}, \qquad i = 1, \ldots, B, \tag{2}$$

so every position supplies a candidate without a learned pass or confidence-based selection; positions $2, \ldots, B$ are then verified.

### 3.2 Learned Draft: Refining the Anchor

The anchor predicts every position from masks alone, so no prediction sees the tokens proposed earlier in the same block. dSpec (Learned Draft) corrects this with a refinement head that reads, at each position $i$, the anchor state $H_i^0$ and the embedding of an input token $\tilde{x}_i$, where $\tilde{x}_1$ is the last committed token and the other inputs are masks:

$$u_i = W_f \left[\mathrm{RMSNorm}(E(\tilde{x}_i)); \mathrm{RMSNorm}(H_i^0)\right]. \tag{3}$$

Here $E$ is the frozen target embedding table and brackets denote concatenation. One bidirectional Transformer layer maps all $u_i$ to a hidden-space correction, which the frozen vocabulary head turns into a residual on the anchor logits:

$$\Delta H = f_\theta(U), \qquad Z^{\mathrm{d}} = Z^0 + \Delta H W^\top. \tag{4}$$

The first token always comes from the anchor, and the head proposes the remaining positions in parallel:

$$\hat{x}_1 = \arg\max_v Z^0_{1,v}, \qquad \hat{x}_i = \arg\max_v Z^{\mathrm{d}}_{i,v} \quad (i \geq 2). \tag{5}$$

In both variants $\hat{x}_1$ is committed directly, so accepted draft length counts positions $2, \ldots, B$ only and is at most $B - 1$. Appendices B and H give the architecture.

### 3.3 Folded Verification, Commitment, and Recovery

**One pass for checking and continuing.** The target checks positions $2, \ldots, B$ in parallel using two regions of length $B - 1$: $D$ holds the draft tokens as context, and $V$ holds masks whose predictions $\tilde{y}$ are compared with the draft. Shared representations make these predictions an approximation to sequential L2R verification (appendix H). Region $C$ holds the $B$ candidates and builds their KV, while region $N$ holds $B$ masks and predicts the next block, totaling $4B - 2$ queries. Apart from the committed prefix, $C$ reads only itself and $N$ reads only $C$ and $N$, so neither depends on the representations in $D/V$ (tables 30 and 31).

**Acceptance and commitment.** Each draft token is compared with the target's highest-logit prediction, and the first mismatch

$$r = \min\big(\{i \in \{2, \ldots, B\} : \hat{x}_i \neq \tilde{y}_i\} \cup \{B + 1\}\big) \tag{6}$$

determines commitment. Full acceptance ($r = B + 1$) commits the block, retains $C$'s KV, and takes $N$'s prediction as the next anchor, costing one target pass. Otherwise, $\hat{x}_{1:r-1}$ and correction $\tilde{y}_r$ are committed. Bidirectional attention prevents truncating the prepared KV to this prefix; the next block starts after the correction.

**Batched recovery.** Rejected requests commit different lengths $r_j$. Rather than issuing one call per length, batched recovery pads every committed segment to $R = \max_j r_j$, appends $B$ masks, and rebuilds the committed KV and the next anchor for all of them in one target pass; valid queries never read padding, and each request's next block is positioned after its own prefix (algorithm 1).

**Why L2R target predictions.** L2R unmasking can match or outperform confidence-based ordering on recent block-diffusion models (Hou & Kwok, 2026). Trajectory verification checks both token predictions and unmasking order (Wu & Zhang, 2025; Gao et al., 2025), potentially rejecting correct tokens proposed in the wrong order. We check predictions conditioned on preceding draft tokens and remaining masks, aligning verification with commitment so better drafts yield longer acceptance.

**Why a shared layout.** Exact L2R verification requires $B-1$ separate partial blocks. Sharing $D/V$ representations fits verification and $C/N$ in one pass; appendix H specifies the approximation.

### 3.4 Training Learned Draft

**Learning from sequential target predictions.** The frozen target generates greedy L2R teacher trajectories with AngelSpec (Liu et al., 2026a), predicting each position from the preceding tokens and the remaining masks, and the refinement head learns to reproduce these predictions

Table 1: Absolute throughput (output tokens/s) and accepted draft length $\tau$ of DSPEC.

| | LLaDA2.0-mini ($B = 32$) | | | | | SDAR-8B ($B = 4$) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Batch | LC1 | LC0.95 | LC0.90 | DSPEC | $\tau$ | LC1 | LC0.95 | LC0.90 | DSPEC | $\tau$ |
| 1 | 39.7 | 75.0 | 81.9 | 161.0 | 6.17 | 37.4 | 55.2 | 56.8 | 102.7 | 2.50 |
| 8 | 146.5 | 205.0 | 219.5 | 466.7 | 6.13 | 113.1 | 152.6 | 156.5 | 263.0 | 2.51 |
| 32 | 410.7 | 498.9 | 518.7 | 881.5 | 6.21 | 241.2 | 313.8 | 323.5 | 441.5 | 2.50 |
| 64 | 586.8 | 669.9 | 703.8 | 1113.1 | 6.28 | 331.0 | 412.3 | 422.6 | 525.3 | 2.51 |
| 128 | 783.7 | 858.2 | 882.2 | 1298.1 | 6.29 | 418.3 | 494.6 | 506.6 | 583.9 | 2.51 |

from the anchor. With teacher representation $H_i^{\text{teacher}}$, the teacher and head distributions are $p_i = \text{softmax}(H_i^{\text{teacher}} W^\top)$ and $q_i = \text{softmax}(Z_i^{\text{d}})$.

**Encouraging consecutive agreement.** An early mismatch rejects everything after it. Following streak- or prefix-level acceptance objectives (Sandler et al., 2025; Whalen et al., 2026), we define our loss using the per-position overlap

$$\alpha_i = \sum_{v \in \mathcal{V}} \min\{p_i(v), q_i(v)\} = 1 - \text{TV}(p_i, q_i), \tag{7}$$

where TV is total variation distance. For a fully valid block, the *Prefix-TV objective* is

$$\mathcal{L}_{\text{prefix-TV}} = 1 - \frac{1}{B} \sum_{k=1}^{B} \prod_{i=1}^{k} \alpha_i, \tag{8}$$

and an improvement at an early position raises every prefix product that contains it. Following the link between overlap and speculative acceptance (Samarin et al., 2026), we use this objective as a surrogate for greedy drafting; appendix H gives the remaining details and the cross-entropy (CE) control.

## 4 EXPERIMENTS

**Setup.** We evaluate LLaDA2.0-mini (LLaDA2, $B = 32$) and SDAR-8B (SDAR, $B = 4$). One-layer heads are trained for one epoch on OpenPerfectBlend (Labonne, n.d.) with Prefix-TV and frozen-target L2R teachers. Training and inference use NVIDIA H20 GPUs; main SGLang evaluations use single-GPU workers, BF16, and no CUDA Graphs. DSPEC denotes Learned Draft unless marked Self-Draft. Baselines use Fast-dLLM's confidence-threshold rule (Wu et al., 2026b) with exact blockwise KV caching at thresholds 1.0, 0.95, 0.90, and 0.5 (LC1, LC0.95, LC0.90, LC0.5); speedups default to LC1.

We measure end-to-end throughput at batches 1–128 on GSM8K, MATH-500, AIME 2025, HumanEval, LiveCodeBench, MT-Bench, Alpaca, and Arena-Hard-v2. The first five supply scores for speed–quality comparisons; separate batch-one quality evaluation adds IFEval and GPQA-Diamond. Appendices A to C give training and evaluation details.

### 4.1 RQ1: DOES DSPEC IMPROVE THE SPEED–QUALITY TRADE-OFF?

We evaluate approximate verification on complete generations, measuring both throughput and task quality. At batch size 1, DSPEC reaches $4.27\times$/$2.83\times$ LC1 throughput on LLaDA2/SDAR (figures 1 and 3), while seven-task mean quality changes by $+0.23$/$-1.00$ percentage points, with variation across tasks (table 2). Speedups remain at least $1.84\times$/$1.56\times$ across batches 1–128, with the largest gains at batch one (absolute throughput in table 1).

Lowering the confidence threshold offers another route to faster decoding, so we compare its speed–quality trade-off with DSPEC. On the five scored speed workloads, DSPEC is about $2\times$ faster than LC0.90 at batch one with similar or higher mean quality (figure 4 and table 11). LC0.5 narrows the

Table 2: Quality across seven benchmarks (%, batch one). Bold marks the best score per target.

| | LLaDA2.0-mini | | | | SDAR-8B | | | |
|---|---|---|---|---|---|---|---|---|
| Task | LC1 | LC0.95 | LC0.5 | DSPEC | LC1 | LC0.95 | LC0.5 | DSPEC |
| GSM8K | 88.70 | 88.60 | 71.42 | **89.44** | 90.37 | 90.37 | 79.61 | **90.60** |
| MATH-500 | **76.80** | 76.40 | 59.60 | 75.53 | 76.60 | 76.20 | 66.60 | **78.80** |
| AIME 2025 | 13.33 | 13.33 | 3.33 | **20.00** | **16.67** | **16.67** | 0.00 | 10.00 |
| HumanEval | 87.80 | **88.41** | 37.80 | 87.20 | 68.29 | 68.29 | 46.95 | **69.51** |
| LiveCodeBench | **40.76** | 40.16 | 5.59 | 39.65 | **24.27** | **24.27** | 13.36 | 22.84 |
| IFEval | **82.07** | 80.71 | 70.49 | 78.62 | 62.78 | **62.97** | 58.96 | 62.72 |
| GPQA-Diamond | 39.39 | **41.75** | 29.29 | 40.07 | 26.26 | 26.26 | **27.27** | 23.74 |
| Avg. | 61.27 | 61.34 | 39.65 | **61.50** | **52.18** | 52.15 | 41.82 | 51.17 |

Table 3: DSPEC/LC0.5 speedup (geometric mean) and quality (%, arithmetic mean) on GSM8K, MATH-500, HumanEval, and LiveCodeBench; three-round averages.

| | LLaDA2.0-mini | | | SDAR-8B | | |
|---|---|---|---|---|---|---|
| | | Quality | | | Quality | |
| Batch | Speedup | LC0.5 | DSPEC | Speedup | LC0.5 | DSPEC |
| 1 | 1.06× | 43.7 | 72.9 | 1.63× | 51.9 | 65.5 |
| 8 | 1.23× | 44.1 | 72.6 | 1.58× | 52.2 | 63.7 |
| 32 | 1.02× | 43.1 | 72.0 | 1.38× | 51.6 | 64.7 |
| 64 | 1.03× | 43.9 | 72.4 | 1.35× | 52.9 | 64.6 |
| 128 | 1.03× | 42.4 | 72.3 | 1.19× | 51.4 | 64.6 |

speed gap but incurs large quality losses and remains slower at every tested batch (table 3). Thus, threshold relaxation does not reproduce DSPEC's speed–quality trade-off in these configurations.

### 4.2 RQ2: When Do State Reuse and Batched Recovery Help?

With the drafter and verification rule fixed, we add folding and then batched recovery to separate execution, which constructs continuation states after verification (figure 5). Ratios use separate execution at the same target and batch size as the baseline.

SDAR's 76% full-acceptance rate enables frequent state reuse: folding alone gives 1.31× throughput at batch one. LLaDA2 accepts only 2.5% in full, so most steps require reconstruction. At batch 32,

| | batch | GSM8K | MATH | AIME | HEval | LCB | MT-B | Alpaca | Arena | **Mean** |
|---|---|---|---|---|---|---|---|---|---|---|
| LLaDA2.0-mini | 1 | 4.83 | 5.98 | 5.17 | 5.82 | 4.01 | 3.36 | 2.85 | 3.32 | **4.27** |
| | 8 | 3.74 | 4.44 | 2.99 | 4.39 | 3.44 | 3.31 | 2.65 | 2.45 | **3.36** |
| | 32 | 2.41 | 2.78 | 1.88 | 3.21 | 2.71 | 2.50 | 2.08 | 1.49 | **2.32** |
| | 64 | 2.55 | 2.26 | 1.87 | 3.03 | 2.03 | 1.90 | 1.90 | 1.42 | **2.07** |
| | 128 | 2.09 | 2.05 | 1.87 | 3.06 | 1.71 | 1.64 | 1.46 | 1.30 | **1.84** |
| SDAR-8B | 1 | 3.11 | 3.64 | 3.37 | 2.66 | 2.20 | 2.76 | 2.61 | 2.59 | **2.83** |
| | 8 | 2.81 | 2.81 | 2.55 | 2.21 | 2.26 | 2.14 | 2.26 | 2.26 | **2.40** |
| | 32 | 2.18 | 1.92 | 1.18 | 1.42 | 2.28 | 2.19 | 2.46 | 1.61 | **1.85** |
| | 64 | 2.36 | 1.66 | 1.16 | 1.47 | 2.16 | 1.29 | 1.79 | 1.32 | **1.61** |
| | 128 | 2.00 | 1.42 | 1.17 | 1.82 | 1.81 | 2.02 | 1.42 | 1.13 | **1.56** |

Figure 3: Speedup of DSPEC over LC1 on each workload at each batch size. The last column is the eight-workload geometric mean plotted in figure 1.

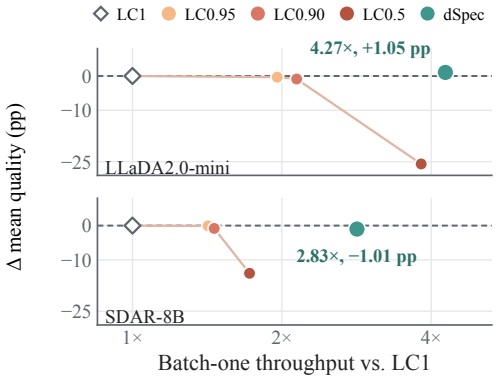


Figure 4: Speed and mean quality on the five scored speed workloads at batch size 1, relative to LC1. Lowering the threshold trades quality for speed, whereas DSPEC reaches higher speed at about the same quality.

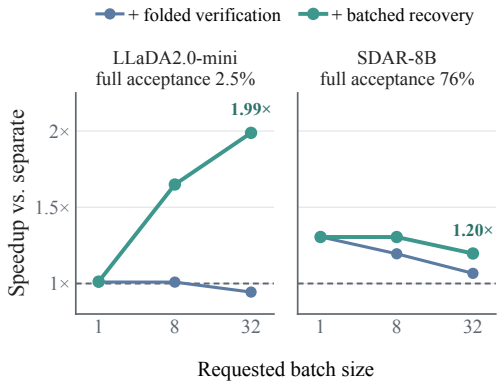


Figure 5: Throughput after adding folded verification, then batched recovery, to separate execution. Panel titles give each target's full-acceptance rate, which decides which step helps.

Table 4: Drafting choices at batch one, as accepted draft length and speedup of the candidate over the control. Hidden / Delta has no timing; *earlier training schedule.

| Comparison | Target | Head | Accepted length | Speedup |
|---|---|---|---|---|
| Self-Draft → Learned Draft | LLaDA2.0-mini | 1 epoch | 5.57 → 6.16 | 1.050 ± 0.005 |
| | SDAR-8B | 1 epoch | 2.48 → 2.50 | 0.954 ± 0.004 |
| Hidden → Delta | LLaDA2.0-mini | 10k | 5.15 → 6.04 | – |
| CE → Prefix-TV | LLaDA2.0-mini | 10k | 5.78 → 6.00 | 1.033 ± 0.002 |
| AR → Parallel | LLaDA2.0-mini | 10k* | 7.53 → 6.07 | 1.863 ± 0.005 |

folding alone gives $0.94\times$, but adding batched recovery yields $1.99\times$. Preparing candidate states alone therefore provides little benefit when most requests reject.

At batch one, where no requests can be combined during recovery, LLaDA2 gains only $1.01\times$ (appendix G). Folding therefore benefits frequent full acceptance, while batched recovery addresses reconstruction across concurrent rejected requests.

### 4.3 RQ3: When Does Better Drafting Improve Throughput?

**When learned refinement pays off.** Self-Draft already supplies proposals from the target's predictions. Learned refinement is optional: its additional commitments must compensate for its execution cost. Let $C$ denote mean committed output tokens per proposal step, including corrections, and $T$ mean total step time, for Self-Draft ($s$) and Learned Draft ($l$) (appendix E). Pooled throughput $C/T$ rises exactly when

$$\frac{C_l - C_s}{C_s} > \frac{T_l - T_s}{T_s}. \tag{9}$$

At batch one, LLaDA2's 7.5% increase in $C$ exceeds its 2.2% increase in $T$, improving throughput. SDAR's 0.2% gain in $C$ falls short of its 4.5% increase in $T$, favoring Self-Draft (table 4). This target-dependent pattern holds at batches 8 and 32 (appendix E).

**Refinement inputs and objective.** On LLaDA2, using target logits $Z^0$ as a residual base (Delta, equation (4)) raises accepted draft length by 17.2% over prediction from the same hidden states alone, supporting refinement of the target's existing prediction. Replacing CE with Prefix-TV raises

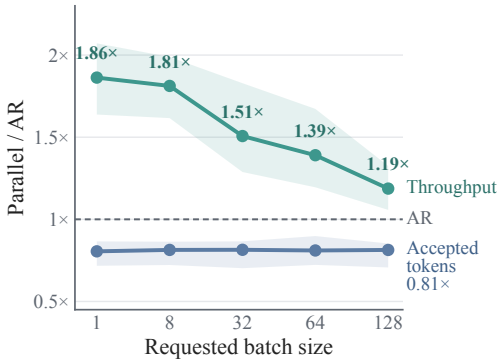


Figure 6: Parallel drafting relative to autoregressive drafting on LLaDA2. Parallel drafting accepts shorter drafts yet runs faster at every batch size.

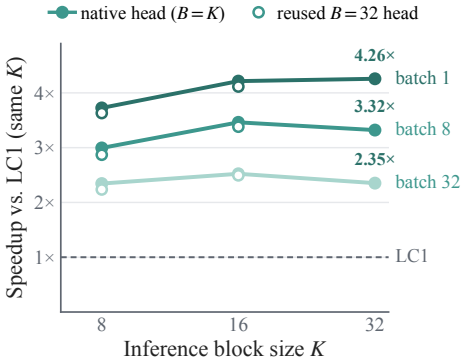


Figure 7: Speedup over LC1 at each inference block size $K$ on LLaDA2. Filled markers use a head trained at $B = K$, and hollow markers reuse the $B = 32$ head.

Table 5: SDAR-8B in HF on two H20s at batch one: eager (left), CUDA Graphs (right). Speed and quality aggregate GSM8K, MATH-500, HumanEval, LiveCodeBench, and IFEval. Speedup uses Native in the same mode; quality is in %.

| Eager | | | CUDA Graphs | | |
|---|---|---|---|---|---|
| Method | Speedup | Quality | Method | Speedup | Quality |
| Native | 1.00× | 63.92 | Native | 1.00× | 63.96 |
| FreeDave | 1.62× | 64.04 | SimSD | 1.25× | 60.11 |
| S2D2 | 1.34× | 59.33 | | | |
| DSPEC (Self-Draft) | 3.25× | 64.80 | DSPEC (Self-Draft) | 3.14× | 64.80 |
| DSPEC (Learned Draft) | 3.12× | 64.24 | DSPEC (Learned Draft) | 3.10× | 64.24 |

throughput by 1.8–3.3%, supporting the objective's emphasis on prefix acceptance (table 4 and appendix E).

**Why parallel drafting.** Autoregressive drafting provides a further test of the acceptance–cost trade-off. On LLaDA2 at batch one, autoregressive drafting accepts 24% longer prefixes, yet parallel drafting is 1.86× faster and remains faster through batch 128 (figure 6). Sequential drafting's extra acceptance does not cover its execution cost: the relevant quantity is committed tokens per unit time.

### 4.4 RQ4: Can One Drafter Serve Shorter Inference Blocks?

Different batches favor different inference block sizes (figure 7). To test reuse without retraining, we run LLaDA2's $B = 32$ head at $K = 8$ and $K = 16$ against heads trained at those widths, fixing inference width and batch size in each comparison.

The reused head retains approximately 95–99% of the short-block heads' throughput with similar accepted lengths (table 24). A single head therefore serves these shorter widths with little throughput loss. Appendix F reports workload- and configuration-dependent quality alongside throughput.

### 4.5 RQ5: How Does dSpec Compare with Other Speculative Methods?

We compare FreeDave (Wu & Zhang, 2025), S2D2 (Han et al., 2026), and SimSD (Cui et al., 2026) on SDAR-8B in HF at batch one, using two H20s per configuration. We reproduce the authors' code

with its supported execution modes: eager for FreeDave and S2D2, CUDA Graphs (CG) for SimSD. Table 5 groups results by mode; speedups use the same-mode Native baseline, which commits one highest-confidence token per step (settings in appendix I).

Self-Draft has the highest throughput and mean quality in each mode: approximately $2\times$ FreeDave's throughput in eager execution, and $3.14\times$ Native with CG versus SimSD's $1.25\times$.

Both FreeDave and Self-Draft use target predictions, but FreeDave checks a confidence-ordered path across multiple candidate states, whereas Self-Draft verifies a complete block against L2R predictions in one folded pass. This comparison supports using target predictions directly for competitive speculation without a learned drafter in this setting. The execution ablation in section 4.2 separately examines state reuse. Learned Draft is slightly slower in both modes, consistent with SDAR's limited refinement benefit in section 4.3.

## 5 Related Work

**Drafting from target information.** Autoregressive speculative decoding verifies proposals in a causal target pass (Leviathan et al., 2023; Chen et al., 2023). Target features support feature prediction in EAGLE (Li et al., 2024), fused-feature token prediction in EAGLE-3 (Li et al., 2025), and conditioned diffusion drafting in DFlash (Chen et al., 2026). Domino adds prefix-dependent causal logit corrections to a parallel drafter (Huang et al., 2026); DFlash 2 combines local convolutions with a path selector over per-position top-$k$ candidates (Inco AI, 2026). These drafters use prefix features; DSPEC uses representations and predictions at every draft position for Self-Draft or in-position refinement. Our objective builds on distillation (Zhou et al., 2024), streak- or prefix-level acceptance objectives (Sandler et al., 2025; Whalen et al., 2026), and distribution overlap (Samarin et al., 2026). Reasoning-step speculation and confidence/cost-based verification scheduling in SpecCoT, DSpark, and D-cut (Shi et al., 2025; Cheng et al., 2026; Liu et al., 2026b) complement draft formation.

**Parallel decoding and state reuse.** Block Diffusion combines autoregressive blocks with within-block diffusion and committed-block caching (Arriola et al., 2025); SDAR and LLaDA2.0 scale this design (Cheng et al., 2025; Bie et al., 2025). Fast-dLLM and Fast-dLLM v2 commit by confidence threshold (Wu et al., 2026b;a), EB-Sampler and APD adapt commitment width (Ben-Hamu et al., 2025; Israel et al., 2025), and dKV-Cache and dInfer accelerate caching and execution (Ma et al., 2025a;b). DSPEC separates proposing from committing and jointly executes verification and continuation-state preparation.

**Verifying diffusion proposals.** FreeDave, SSD, and Spiffy verify denoising states against a reference sampler's path (Wu & Zhang, 2025; Gao et al., 2025; Agrawal et al., 2026). Trajectory-level speculation studies exactness and drift (Pan et al., 2026); PSD combines parallel unmasking with multi-depth speculation (Sun et al., 2026). SimSD checks a separate diffusion drafter with a temporal mask (Cui et al., 2026), S2D2 checks self-drafted tokens in the target's block-size-one mode (Han et al., 2026), and other designs learn drafting and verification or add causal components (Fang et al., 2026; Campbell et al., 2026). DSPEC uses complete anchor drafts, prepares continuation states during verification, and jointly recovers requests with unequal committed lengths.

## 6 Conclusion

DSPEC uses the anchor as a complete draft, verifies it in a folded pass that also prepares the next block, and jointly reconstructs the states of rejected requests. Across two targets and batches 1–128, it achieves $1.56$–$4.27\times$ the throughput of threshold-1.0 decoding. At batch one, its seven-task mean quality is within about one percentage point of that baseline. Refinement improves throughput only when its gain in committed tokens exceeds its added step time, and the benefit of state reuse or recovery depends on the acceptance regime.

Future work includes lightweight within-block dependencies for parallel drafting (Cheng et al., 2026), subject to the cost criterion in equation (9). We evaluate targets up to 16B; larger models such as LLaDA2.0-flash (Bie et al., 2025) require profiling how model scale and input-dependent expert routing affect drafting, verification, and recovery costs.

## Reproducibility Statement

We will make the code and data for training, inference, and evaluation publicly available. Appendices A to C and H describe the experimental setup, checkpoints, metrics, and implementation details.

## AI Use Statement

Large language models (LLMs) were used to polish the language of the manuscript, to discuss options for experimental design, to assist with figure design, and to check the consistency of the reported numbers with the experimental records. All technical content, experimental design, and scientific claims are the work of the authors.

## References

Sudhanshu Agrawal, Risheek Garrepalli, Raghavv Goel, Christopher Lott, Fatih Porikli, and Mingu Lee. Structuring the future: Diffusion LLM speculative decoding via calibrated draft graphs. *arXiv preprint arXiv:2509.18085*, 2026. URL https://arxiv.org/abs/2509.18085v4. Version 4, revised June 2026; first released in 2025.

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. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2503.09573.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *arXiv preprint arXiv:2505.24857*, 2025. URL https://arxiv.org/abs/2505.24857.

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, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. LLaDA2.0: Scaling up diffusion language models to 100B. *arXiv preprint arXiv:2512.15745*, 2025. URL https://arxiv.org/abs/2512.15745.

Andrew Campbell, Valentin De Bortoli, Jiaxin Shi, and Arnaud Doucet. Self-speculative masked diffusions. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/pdf?id=ogMTEtHO6M.

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. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.06036.

Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and Bowen Zhou. SDAR: A synergistic diffusion-autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*, 2025. URL https://arxiv.org/abs/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, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026. URL https://arxiv.org/abs/2607.05147.

Junxia Cui, Haotian Ye, Runchu Tian, Hongcan Guo, Jinya Jiang, Haoru Li, Chaojie Ren, Yiming Huang, Kaijie Zhu, Zhongkai Yu, Kun Zhou, and Jingbo Shang. SimSD: Simple speculative decoding in diffusion language models. *arXiv preprint arXiv:2606.02544*, 2026. URL https://arxiv.org/abs/2606.02544.

DeepSeek-AI. DeepSpec: Evaluation dataset conversion. GitHub source code, 2026a. URL https://github.com/deepseek-ai/DeepSpec/blob/main/eval_datasets/convert_eval_datasets_to_jsonl.py. Accessed September 26, 2026.

DeepSeek-AI. DeepSpec: Evaluation task configuration. GitHub source code, 2026b. URL https://github.com/deepseek-ai/DeepSpec/blob/main/eval.py. Accessed September 26, 2026.

Xun Fang, Yunchen Li, Hang Yuan, and Zhou Yu. Factorization-error-free discrete diffusion language model via speculative decoding. *arXiv preprint arXiv:2605.14305*, 2026. URL https://arxiv.org/abs/2605.14305.

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. URL https://arxiv.org/abs/2510.04147.

Ligong Han, Hao Wang, Han Gao, Kai Xu, and Akash Srivastava. S2D2: Fast decoding for diffusion LLMs via training-free self-speculation. *arXiv preprint arXiv:2603.25702*, 2026. URL https://arxiv.org/abs/2603.25702.

Kai Syun Hou and James Kwok. Rethinking the generation order of block diffusion language models. *arXiv preprint arXiv:2607.24306*, 2026. URL https://arxiv.org/abs/2607.24306.

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding. *arXiv preprint arXiv:2605.29707*, 2026. URL https://arxiv.org/abs/2605.29707.

Inco AI. DFlash 2: Keep drafting parallel. Technical blog, August 2026. URL https://inco.ai/blog/dflash2/. Published August 18, 2026; accessed September 26, 2026.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion LLMs via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025. URL https://arxiv.org/abs/2506.00413.

Maxime Labonne. Open-PerfectBlend. Hugging Face dataset, n.d. URL https://huggingface.co/datasets/mlabonne/open-perfectblend. Accessed September 26, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 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*, pp. 19274–19286, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948, 2024. URL https://proceedings.mlr.press/v235/li24bt.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/c7b5a35ea98b62512a869c19ea7b03cb-Abstract-Conference.html.

Hong Liu, Rui Cen, Junhan Shi, Guangshuo Qin, Jiebin Zhang, Tianyu Liu, Runzhi Fan, Guoliang Zhao, Ruobing Xie, Kai Zhang, Song Liu, Guanghua Yu, and Jianchen Zhu. AngelSpec: Towards real-world high performance inference with speculative decoding. *arXiv preprint arXiv:2607.25852*, 2026a. URL https://arxiv.org/abs/2607.25852.

Tianyu Liu, Yuhao Shen, Rui Cen, Junhan Shi, Jiebin Zhang, Guangshuo Qin, Hong Liu, Song Liu, Guanghua Yu, and Jianchen Zhu. D-cut: Adaptive verification depth pruning for batched speculative decoding. *arXiv preprint arXiv:2607.14647*, 2026b. URL https://arxiv.org/abs/2607.14647.

LiveCodeBench Contributors. LiveCodeBench: Official evaluation repository. GitHub repository, n.d. URL https://github.com/LiveCodeBench/LiveCodeBench. Accessed September 26, 2026.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dKV-Cache: The cache for diffusion language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. URL https://proceedings.nips.cc/paper_files/paper/2025/hash/db0ee27cb50dd9087b133f6e7d28a90e-Abstract-Conference.html.

Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, Xinyuan Zhang, Zhen Tao, Haibo Feng, Ziyun Jiang, Ying Xu, Zenan Huang, Yihong Zhuang, Haokai Xu, Jiaqi Hu, Zhenzhong Lan, Junbo Zhao, Jianguo Li, and Da Zheng. dInfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv:2510.08666*, 2025b. URL https://arxiv.org/abs/2510.08666.

Tianxiang Pan, Baitao Gong, Mo Guang, Hongwei Yong, Tianpeng Jiang, Yaqian Li, Zheng Cao, and Kaiwen Long. Trajectory-level speculative decoding for diffusion language models. *arXiv preprint arXiv:2608.27514*, 2026. URL https://arxiv.org/abs/2608.27514.

Alexander Samarin, Sergei Krutikov, Anton Shevtsov, Sergei Skvortsov, Filipp Fisin, and Alexander Golubev. LK losses: Direct acceptance rate optimization for speculative decoding. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.23881.

Jameson Sandler, Jacob K. Christopher, Thomas Hartvigsen, and Ferdinando Fioretto. SpecDiff-2: Scaling diffusion drafter alignment for faster speculative decoding. *arXiv preprint arXiv:2511.00606*, 2025. URL https://arxiv.org/abs/2511.00606.

SGLang Contributors. Diffusion language models: SGLang documentation. GitHub documentation, 2026. URL https://github.com/sgl-project/sglang/blob/main/docs/docs/supported-models/diffusion_language_models.mdx. Accessed September 26, 2026.

Junhan Shi, Yijia Zhu, Zhenning Shi, Dan Zhao, Qing Li, and Yong Jiang. SpecCoT: Accelerating chain-of-thought reasoning through speculative exploration. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 24405–24415, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.1326. URL https://aclanthology.org/2025.findings-emnlp.1326/.

Shengyin Sun, Yiming Li, Renxi Liu, Xinqi Li, Hui-Ling Zhen, Weizhe Lin, Chen Chen, Xianzhi Yu, Mingxuan Yuan, and Chen Ma. PSD: Pushing the pareto frontier of diffusion LLMs via parallel speculative decoding. *arXiv preprint arXiv:2605.15609*, 2026. URL https://arxiv.org/abs/2605.15609.

Lexington Whalen, Yuki Ito, and Ryo Sakamoto. Teaching diffusion to speculate left-to-right. *arXiv preprint arXiv:2606.11552*, 2026. URL https://arxiv.org/abs/2606.11552.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dLLM v2: Efficient block-diffusion LLM. In *International Conference on Learning Representations*, 2026a. URL https://openreview.net/pdf?id=1NZ3DHF9nT.

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. In *International Conference on Learning Representations*, 2026b. URL https://openreview.net/pdf?id=3Z3Is6hnOT.

Shutong Wu and Jiawei Zhang. Free draft-and-verification: Toward lossless parallel decoding for diffusion large language models. *arXiv preprint arXiv:2510.00294*, 2025. URL https://arxiv.org/abs/2510.00294v3. Version 3, revised February 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.08461.

# A Detailed Experimental Setup

**Hardware and decoding.** In the main experiments, each inference worker runs on one NVIDIA H20 (96GB-class memory), with TP=1 and BF16; multiple workers only shard the evaluation. The targets are SDAR-8B-Chat ($B = 4$) and LLaDA2.0-mini ($B = 32$). Main inference uses greedy decoding, EOS stopping, a 2,048-token output cap, a context setting of 12,288, and the native SDAR or non-thinking LLaDA2 chat template. The full quality suite uses 16,384 output tokens for GPQA-Diamond. A separate long-budget suite sets the cap to 16,384 on MATH-500, AIME 2025, and LiveCodeBench.

**Baselines and execution.** SGLang LowConfidence thresholds 1.0, 0.95, 0.90, and 0.5 are denoted LC1, LC0.95, LC0.90, and LC0.5 (SGLang Contributors, 2026). LC1 is not strict left-to-right decoding. Main SGLang methods use eager execution, and DSpec fuses the vocabulary projection. The single-GPU HF comparisons with SimSD (Cui et al., 2026) match eager or CUDA Graph execution within each row; appendix I reports a separate two-GPU study.

**Workloads.** The speed suite follows the DeepSpec workload convention (DeepSeek-AI, 2026b;a) and contains 2,774 inputs per method and round. GSM8K, MATH-500, AIME 2025, HumanEval, and LiveCodeBench yield objective task scores (1,694 examples in total) used for speed–quality comparisons. MT-Bench uses first-turn prompts; MT-Bench, Alpaca, and Arena-Hard-v2 have no judge scores in this evaluation. The full quality suite contains 3,807 examples. LiveCodeBench uses release_v6 and its evaluation harness (LiveCodeBench Contributors, n.d.), with frozen question dates from 2023-05-07 to 2025-04-06.

Table 6: Evaluation manifests and output budgets. A dash denotes a workload outside the corresponding suite. Speed workloads use a 2k output cap.

| Workload | Speed $n$ | Full quality $n$ | Quality cap | Metric |
|---|---|---|---|---|
| GSM8K | 500 | 1,319 | 2k | Answer accuracy |
| MATH-500 | 500 | 500 | 2k / 16k | Answer accuracy |
| AIME 2025 | 30 | 30 | 2k / 16k | Answer accuracy |
| HumanEval | 164 | 164 | 2k | Greedy pass@1 |
| LiveCodeBench | 500 | 1,055 | 2k / 16k | Greedy pass@1 |
| MT-Bench (first turn) | 80 | – | – | Unscored |
| Alpaca | 500 | – | – | Unscored |
| Arena-Hard-v2 | 500 | – | – | Unscored |
| IFEval | – | 541 | 2k | Prompt-strict |
| GPQA-Diamond | – | 198 | 16k | Answer accuracy |
| Total | 2,774 | 3,807 | | |

**Timing and repeated measurements.** We synchronize and time the wall-clock `engine.generate` call, including prefill, decode, and invocation overhead. Loading, warmup, and scoring are excluded. A batch's wall time is counted once even when shared by multiple requests. Main, LC0.90, and LC0.5 runs use one initial warmup batch with 64 generated tokens per request and clear the cache before each timed batch. Paired ablations and draft-variant/system runs additionally use one full same-input pre-run per batch followed by cache clearing. Ratios stay within their matched run rather than dividing absolute times across warmup protocols. LC0.5 uses identical input hashes, shard order, and actual batch sizes, but different nodes and driver environments.

Rounds use seeds 980406, 42, and 2026, with greedy decoding on the same questions. Small workloads and tail batches can have fewer active requests than the requested batch. Request-level latency quantiles are distinct from batch wall times.

## B Checkpoints and Training Schedules

Training uses NVIDIA H20 GPUs, with separate workers for head optimization and frozen-target teacher generation. The main Learned Draft heads complete one epoch on OpenPerfectBlend (Labonne, n.d.), with global batch size 64, context 4,096, generation length 256, seed 42, and training EOS early stopping disabled. The optimizer is AdamW, with 3% warmup, peak learning rate $5\times10^{-4}$, initial/minimum learning rate $5\times10^{-5}$, and weight decay 0.01. Main heads use the Parallel Delta Drafter and Prefix-TV objective. The one-epoch checkpoints correspond to 22,000 training steps; ablation checkpoints are identified by their training step counts.

Table 7: Checkpoint groups. All comparisons are paired within the indicated group; the earlier AR schedule is not a main-schedule control.

| Use | Target | Budget | Configuration |
|---|---|---|---|
| Main / variant / system | SDAR / LLaDA2 | 1 epoch | Delta, $B = 4$ / 32 |
| Architecture | LLaDA2 | 10k steps | Main schedule; Hidden / Delta |
| Loss | LLaDA2 | 10k steps | Main schedule; CE / Prefix-TV |
| Block transfer | LLaDA2 | 10k steps | Main schedule; $B = 8, 16, 32$ |
| Drafting mode | LLaDA2 | 10k steps | Earlier schedule; AR / Parallel |
| HF eager | SDAR | 10k steps | Native SimSD comparison |
| HF CUDA Graph | SDAR | 1 epoch | SimSD comparison |

The AR–Parallel comparison uses a matched earlier training schedule. The single-GPU HF comparisons use a 10k-step drafter for eager execution and a one-epoch drafter for CUDA Graphs. In the block-transfer study, $B$ denotes training block size and $K$ denotes inference width. Changing inference $K$ changes both the target's decoding block and proposal width; the native versus long-trained comparison fixes $K$.

Table 8: Learned Draft dimensions; Self-Draft has no trainable module. Trainable counts exclude shared frozen target embeddings and vocabulary projection.

| | SDAR-8B | LLaDA2.0-mini |
|---|---|---|
| Training block size | 4 | 32 |
| Hidden size | 4096 | 2048 |
| Query / KV heads | 32 / 8 | 16 / 4 |
| FFN width | 12288 | 5120 |
| Trainable parameters | 226.51M | 50.34M |

**Evaluation matrix.** Main speed measurements use three rounds at requested batches 1/8/32/64/128; full quality and 16k evaluations use three rounds at batch one. Self-Draft/Learned Draft and system experiments use three rounds at batches 1/8/32; the matched AR/Parallel study covers 1/8/32/64/128; CE/Prefix-TV uses three rounds at 1/8. Hidden/Delta uses one formal round of 2,774 inputs at batch one. Block-size evaluation uses one round at 1/8/32, including 528 method–task evaluations across the full sweep. HF eager and CUDA Graph comparisons each use three rounds at batch one.

## C Metric Definitions

For method $m$, workload $d$, requested batch $b$, and round $r$, let $O_j$ be the total output-token count and $t_j$ the wall time of measured batch $j$, counting each batch once. Define

$$\mathrm{TPS}_{m,d,b,r} = \frac{\sum_j O_j}{\sum_j t_j}, \quad S_{m,d,b,r} = \frac{\mathrm{TPS}_{m,d,b,r}}{\mathrm{TPS}_{\mathrm{base},d,b,r}}, \quad G_{m,b,r} = \exp\left(\frac{1}{8}\sum_{d=1}^{8} \log S_{m,d,b,r}\right). \quad (10)$$

We report the arithmetic mean of three $G_{m,b,r}$ values and their sample SD (denominator two), summarizing timing and numerical variation across evaluation rounds on the same questions. Figure 1 normalizes each method to LC1 within each domain and round. Figure 3 shows DSPEC/ LC1 for each workload and batch. Table 9 reports each method relative to LC1. For the long-budget suite, the geometric mean covers its three tasks only. Pooled absolute tok/s divides all output tokens by all unique batch wall time; its ratio need not equal a workload geometric mean.

Accepted draft length is total accepted draft tokens divided by total proposals. Acceptance proportion divides by total proposed draft tokens instead; full-acceptance rate is the fraction of completely accepted proposals. Both DSPEC variants exclude the forced first token and correction from accepted drafts, giving maximum draft lengths 3 and 31 for $K = 4$ and 32. SimSD's $K = 4$ convention has no identical forced-token exclusion, so its acceptance statistics are not a common-denominator comparison. Committed-token counters include forced/correction tokens before EOS/budget trimming; throughput uses final output-token counts.

Mean task quality (the macro average) first averages each task over rounds, then weights tasks equally. Full-suite macro mixes the disclosed 2k and GPQA 16k budgets. Mathematics uses answer accuracy; code uses one greedy completion and execution-based pass@1; IFEval uses prompt-strict scoring. GPQA headline scores use a fixed post-hoc final-line format normalizer, applied identically to all methods. The original exact-format parser accepts a final nonempty line matching `Answer: A` (A–D, case-insensitive). For extraction failures, normalization additionally accepts balanced Markdown wrappers and one terminal period or exclamation mark. It rejects ambiguous choices and trailing explanations, does not search earlier reasoning, and does not consult the answer key during extraction. All 198 questions remain in the denominator. Exact-format scores for the SGLang study are retained in table 14.

## D SUPPLEMENTARY PERFORMANCE AND QUALITY

Table 9: End-to-end throughput relative to LC1: three-round mean ± sample SD of eight-workload geometric means.

| Target | Batch | LC0.95 | LC0.90 | LC0.5 | DSPEC |
|---|---|---|---|---|---|
| LLaDA2.0-mini | 1 | $1.958 \pm 0.006$ | $2.143 \pm 0.005$ | $3.818 \pm 0.037$ | $4.272 \pm 0.019$ |
| | 8 | $1.433 \pm 0.004$ | $1.547 \pm 0.006$ | $2.610 \pm 0.011$ | $3.358 \pm 0.003$ |
| | 32 | $1.236 \pm 0.002$ | $1.282 \pm 0.001$ | $2.114 \pm 0.012$ | $2.321 \pm 0.005$ |
| | 64 | $1.147 \pm 0.002$ | $1.223 \pm 0.004$ | $1.932 \pm 0.015$ | $2.072 \pm 0.002$ |
| | 128 | $1.125 \pm 0.005$ | $1.152 \pm 0.005$ | $1.668 \pm 0.038$ | $1.840 \pm 0.006$ |
| SDAR-8B | 1 | $1.421 \pm 0.003$ | $1.461 \pm 0.006$ | $1.722 \pm 0.010$ | $2.835 \pm 0.007$ |
| | 8 | $1.286 \pm 0.002$ | $1.309 \pm 0.014$ | $1.522 \pm 0.009$ | $2.400 \pm 0.007$ |
| | 32 | $1.265 \pm 0.001$ | $1.291 \pm 0.001$ | $1.416 \pm 0.007$ | $1.853 \pm 0.007$ |
| | 64 | $1.212 \pm 0.005$ | $1.241 \pm 0.003$ | $1.257 \pm 0.012$ | $1.607 \pm 0.003$ |
| | 128 | $1.238 \pm 0.004$ | $1.291 \pm 0.006$ | $1.351 \pm 0.002$ | $1.562 \pm 0.010$ |

The SimSD quality table uses the labeled speed subset, rather than the full quality manifests. Both matched single-GPU comparisons improve the five-task mean; the table also reports each task separately. At the 2k output cap, LC0.90 is scored on this five-task subset; the separate 16k evaluation covers all three long-budget tasks. LC0.90 has scores for the five-task subset and the long-budget suite, but not the full seven-task suite.

The 16k suite contains MATH-500 (500), AIME (30), and LiveCodeBench (1,055). LC0.90 is evaluated across two targets and three rounds using the same per-question inputs as the other methods. Its DSPEC/LC1 speedups are $5.968 \pm 0.030\times$ on LLaDA2 and $3.025 \pm 0.005\times$ on SDAR, using three-task geometric means. LLaDA2 MATH-500 changes from 75.53 to 92.53 for DSPEC when increasing the cap, while LC1 changes from 76.80 to 93.60.

All per-round task scores and original exact-format GPQA scores are retained with the source data.

Table 10: Per-task quality for the matched single-GPU SimSD comparisons (%). Scores use the five labeled speed workloads at a 2k output cap; eager and CUDA Graph rows use different DSPEC checkpoints.

| Execution | Method | GSM8K | MATH | AIME | HEval | LCB |
|---|---|---|---|---|---|---|
| HF eager | SimSD | 89.40 | 71.00 | 10.00 | 60.37 | 19.00 |
| | DSPEC | 89.60 | 79.00 | 13.33 | 65.24 | 23.20 |
| HF CUDA Graph | SimSD | 88.40 | 74.00 | 10.00 | 63.41 | 19.00 |
| | DSPEC | 90.20 | 78.20 | 10.00 | 68.90 | 23.60 |

Table 11: Confidence-threshold comparison on GSM8K, MATH-500, AIME 2025, HumanEval, and LiveCodeBench (1,694 questions) at batch one (%, three-round means). Scores use the speed-run inputs.

| Target | Method | GSM8K | MATH | AIME | HEval | LCB | Avg. |
|---|---|---|---|---|---|---|---|
| LLaDA2.0-mini | LC1 | 88.33 | 76.80 | 13.33 | 87.80 | 40.00 | 61.25 |
| | LC0.95 | 87.93 | 76.40 | 13.33 | 88.41 | 38.53 | 60.92 |
| | LC0.90 | 87.60 | 76.40 | 13.33 | 86.59 | 38.00 | 60.38 |
| | LC0.5 | 70.80 | 59.60 | 3.33 | 38.41 | 5.80 | 35.59 |
| | DSPEC | 89.00 | 75.53 | 20.00 | 87.20 | 39.80 | 62.31 |
| SDAR-8B | LC1 | 91.00 | 76.60 | 16.67 | 68.29 | 24.60 | 55.43 |
| | LC0.95 | 91.00 | 76.20 | 16.67 | 68.29 | 24.60 | 55.35 |
| | LC0.90 | 91.00 | 76.40 | 13.33 | 68.29 | 24.00 | 54.61 |
| | LC0.5 | 80.20 | 66.60 | 0.00 | 46.95 | 13.80 | 41.51 |
| | DSPEC | 90.80 | 78.80 | 10.00 | 69.51 | 23.00 | 54.42 |

Table 12: Quality with a 16k output cap (%, batch one, three-round means): full MATH-500, AIME 2025, and LiveCodeBench, 1,585 questions. LC1, LC0.95, and LC0.90 denote Fast-dLLM's confidence-threshold rule in SGLang at thresholds 1.0, 0.95, and 0.90. Bold marks the highest displayed score per target, including ties.

| | LLaDA2.0-mini | | | | SDAR-8B | | | |
|---|---|---|---|---|---|---|---|---|
| Task | LC1 | LC0.95 | LC0.90 | DSPEC | LC1 | LC0.95 | LC0.90 | DSPEC |
| MATH-500 | **93.60** | 93.53 | 91.80 | 92.53 | 76.80 | 76.40 | 76.40 | **79.00** |
| AIME 2025 | 26.67 | 28.89 | **33.33** | **33.33** | **16.67** | **16.67** | 13.33 | 10.00 |
| LiveCodeBench | **40.92** | 40.32 | 38.48 | 39.94 | **24.36** | **24.36** | 24.08 | 22.84 |

Table 13: Long-budget quality at batch one (%, three-round means): 16k cap on all 1,585 questions. $\Delta$ is DSPEC minus LC1 in percentage points, computed before display rounding. Cap-hit rates cover LC1 / LC0.95 / DSPEC only.

| Target | Task | LC1 | LC0.95 | LC0.90 | DSPEC | $\Delta$pp | Cap hits (%) |
|---|---|---|---|---|---|---|---|
| LLaDA2.0-mini | MATH | 93.60 | 93.53 | 91.80 | 92.53 | −1.07 | 0.73 / 0.67 / 0.07 |
| | AIME | 26.67 | 28.89 | 33.33 | 33.33 | +6.67 | 0.00 / 8.89 / 0.00 |
| | LCB | 40.92 | 40.32 | 38.48 | 39.94 | −0.98 | 10.36 / 10.71 / 8.88 |
| SDAR-8B | MATH | 76.80 | 76.40 | 76.40 | 79.00 | +2.20 | 0.80 / 1.00 / 0.40 |
| | AIME | 16.67 | 16.67 | 13.33 | 10.00 | −6.67 | 3.33 / 3.33 / 0.00 |
| | LCB | 24.36 | 24.36 | 24.08 | 22.84 | −1.52 | 2.94 / 3.03 / 6.26 |

Table 14: GPQA scoring sensitivity (%, three-round means, 198 questions, 16k cap). Post-hoc final-line format normalization is applied uniformly to all methods and supplies the headline quality scores.

| Target | Method | Exact format | Format normalized |
|---|---|---|---|
| LLaDA2.0-mini | LC1 | 39.39 | 39.39 |
| | LC0.95 | 41.75 | 41.75 |
| | LC0.5 | 28.79 | 29.29 |
| | DSPEC | 40.07 | 40.07 |
| SDAR-8B | LC1 | 0.51 | 26.26 |
| | LC0.95 | 0.51 | 26.26 |
| | LC0.5 | 0.51 | 27.27 |
| | DSPEC | 0.00 | 23.74 |

Table 15: LC0.5 comparison on GSM8K, MATH-500, HumanEval, and LiveCodeBench (1,664 questions, TP1, eager, 2k cap). Absolute throughput pools these tasks and three rounds. Speedup is the three-round mean ± sample SD of task geometric means; quality averages tasks and rounds equally. Learned Draft uses the main checkpoint.

| Target | Batch | LC0.5 tok/s | Learned / LC0.5 | LC0.5 score | Learned score | Δpp |
|---|---|---|---|---|---|---|
| LLaDA2.0-mini | 1 | 177.7 | $1.062 \pm 0.009$ | 43.65 | 72.88 | +29.23 |
| | 8 | 418.7 | $1.226 \pm 0.005$ | 44.07 | 72.65 | +28.58 |
| | 32 | 928.1 | $1.015 \pm 0.009$ | 43.15 | 72.05 | +28.90 |
| | 64 | 1236.4 | $1.027 \pm 0.012$ | 43.91 | 72.40 | +28.50 |
| | 128 | 1489.4 | $1.028 \pm 0.017$ | 42.39 | 72.25 | +29.86 |
| SDAR-8B | 1 | 66.7 | $1.632 \pm 0.003$ | 51.89 | 65.53 | +13.64 |
| | 8 | 198.6 | $1.585 \pm 0.011$ | 52.19 | 63.65 | +11.46 |
| | 32 | 389.6 | $1.380 \pm 0.005$ | 51.58 | 64.69 | +13.11 |
| | 64 | 495.0 | $1.353 \pm 0.017$ | 52.85 | 64.57 | +11.72 |
| | 128 | 603.3 | $1.191 \pm 0.012$ | 51.44 | 64.63 | +13.19 |

**Aggressive confidence-threshold decoding.** LC0.5 uses the same frozen speed inputs and scoring rules, with three rounds on each target and batch. Its full-quality evaluation covers 3,807 questions per target and round, using eager execution and 2k caps except GPQA at 16k. Table 2 compares these scores with the main Learned Draft checkpoint.

Table 16: SDAR-8B, HF, one H20, batch one. Speedup is Learned Draft / baseline (eight-workload mean ± SD, three rounds); scores are five-task means on 1,694 speed inputs. S2D2 uses one quality round; other quality scores average three rounds. Drafter checkpoints differ across execution modes.

| Execution | Baseline | dSpec head training | Speedup | Baseline score | Learned score |
|---|---|---|---|---|---|
| Eager | SimSD | 10k | $3.713 \pm 0.030$ | 49.95 | **54.08** |
| Eager | S2D2 | 10k | $2.281 \pm 0.011$ | 50.67 | **54.08** |
| CG | SimSD | 1 epoch | $3.451 \pm 0.009$ | 50.96 | **54.18** |

**Single-GPU speculative comparisons.** S2D2 uses Config-A with temperature zero while retaining its original randomized acceptance and resampling. The supplement uses Transformers 4.57.1 and PyTorch 2.11.0, matches the input-token hashes of the comparison methods, and preserves the

Table 17: Single-GPU HF eager comparison on the five-task speed subset (%). S2D2 scores its first speed round; the SimSD and 10k-step dSpec scores average three rounds.

| Task | $n$ | SimSD | S2D2 | DSPEC |
|---|---|---|---|---|
| GSM8K | 500 | 89.40 | 89.60 | 89.60 |
| MATH-500 | 500 | 71.00 | 75.00 | 79.00 |
| AIME 2025 | 30 | 10.00 | 13.33 | 13.33 |
| HumanEval | 164 | 60.37 | 52.44 | 65.24 |
| LiveCodeBench | 500 | 19.00 | 23.00 | 23.20 |

Table 18: Single-GPU HF eager absolute throughput (tok/s). Each round pools eight workloads; entries are the mean $\pm$ sample SD over three rounds.

| Method | Tok/s |
|---|---|
| SimSD | $26.587 \pm 0.172$ |
| DSPEC (10k) | $101.634 \pm 0.313$ |
| S2D2 | $43.324 \pm 0.023$ |

original scoring rules. S2D2 quality is scored on the first of three speed rounds. On HumanEval, 42 responses lack the required entry function and remain failures under the common scorer. Its dSpec reference uses the 10k checkpoint; the two-GPU study uses the one-epoch checkpoint.

# E SUPPLEMENTARY DRAFTER ANALYSIS

Table 19: Paired LLaDA2.0-mini drafting ablations. Accepted length is control / candidate; speedup is candidate / control. AR / Parallel use the earlier schedule, and CE / Prefix-TV the main schedule, each at 10k steps. Hidden / Delta uses the main schedule in HF, one round of 2,774 inputs, without a matched timing run.

| Comparison | Batch | Accepted length | Speedup |
|---|---|---|---|
| AR → Parallel | 1 | 7.534 / 6.066 | $1.863 \pm 0.005$ |
| | 8 | 7.522 / 6.121 | $1.812 \pm 0.009$ |
| | 32 | 7.587 / 6.181 | $1.507 \pm 0.007$ |
| CE → Prefix-TV | 1 | 5.777 / 5.997 | $1.033 \pm 0.002$ |
| | 8 | 5.787 / 6.059 | $1.018 \pm 0.001$ |
| Hidden → Delta | 1 | 5.154 / 6.040 | – |

Table 20: Formal Hidden / Delta evaluation on the five labeled speed workloads (%); 2,774 total speed inputs, HF eager, batch one, 10k steps, one round. Cap hits cover all eight speed workloads.

| Head | GSM8K | MATH | AIME | HEval | LCB | Cap hits |
|---|---|---|---|---|---|---|
| Hidden | 89.60 | 76.60 | 20.00 | 88.41 | 38.60 | 14.38 |
| Delta | 87.80 | 77.20 | 13.33 | 89.02 | 39.40 | 14.06 |

Table 21: Paired drafting-ablation task quality (%) and generation lengths. Scores cover the five labeled speed workloads; output length and cap-hit rates cover all eight. Three repeated rounds use the same questions.

| Batch | Method | GSM8K | MATH | AIME | HEval | LCB | Output tokens | Cap hits |
|---|---|---|---|---|---|---|---|---|
| 1 | AR | 87.40 | 73.00 | 13.33 | 84.15 | 38.60 | 769.4 | 17.30 |
| | Parallel | 89.00 | 73.80 | 16.67 | 84.76 | 38.60 | 707.6 | 14.28 |
| 8 | AR | 88.00 | 73.60 | 16.67 | 81.71 | 34.60 | 775.3 | 17.09 |
| | Parallel | 89.40 | 74.60 | 16.67 | 87.20 | 38.40 | 714.2 | 14.10 |
| 32 | AR | 88.00 | 73.20 | 16.67 | 84.15 | 38.40 | 761.5 | 16.62 |
| | Parallel | 89.60 | 75.00 | 13.33 | 85.98 | 39.93 | 715.3 | 14.67 |
| 1 | CE | 90.20 | 73.40 | 10.00 | 87.80 | 38.60 | 736.7 | 15.36 |
| | Prefix-TV | 88.00 | 76.00 | 16.67 | 85.98 | 39.60 | 710.3 | 14.06 |
| 8 | CE | 87.00 | 75.20 | 13.33 | 87.20 | 39.40 | 737.8 | 15.25 |
| | Prefix-TV | 88.80 | 76.40 | 16.67 | 84.76 | 38.20 | 704.2 | 13.84 |

**Ablation names.** *Delta* denotes the output-anchor parameterization in equation (4); *Hidden* retains the same target-feature input but omits the additive anchor logits. *Self-Draft* uses $Z^0$ without a learned correction. AR and Parallel instead distinguish sequential versus simultaneous proposal generation.

The formal Hidden/Delta comparison uses main-schedule 10k LLaDA2 heads, HF eager, batch one, and identical approximate verification. Delta raises acceptance by 17.2%; table 20 reports the per-task scores. This comparison measures the effect of the additive target output anchor with matched conditioning and architecture. The AR/Parallel pair uses matched earlier-schedule checkpoints and evaluates the trade-off between accepted length and execution cost (tables 19 and 21).

Table 22: Matched dSpec variants: Self-Draft versus Learned Draft. Accepted length and cap-hit rates are Self-Draft / Learned Draft; speedup is Learned Draft / Self-Draft (mean ± SD). Both use the same verification and recovery path; learned heads complete one epoch.

| Target | Batch | Accepted length | Speedup | Cap hits (%) |
|---|---|---|---|---|
| LLaDA2.0-mini | 1 | 5.575 / 6.161 | $1.050 \pm 0.005$ | 15.43 / 14.20 |
| | 8 | 5.643 / 6.135 | $1.039 \pm 0.001$ | 15.47 / 14.31 |
| | 32 | 5.752 / 6.215 | $1.034 \pm 0.002$ | 15.10 / 14.89 |
| SDAR-8B | 1 | 2.481 / 2.500 | $0.954 \pm 0.004$ | 3.53 / 2.99 |
| | 8 | 2.484 / 2.515 | $0.956 \pm 0.004$ | 3.42 / 3.64 |
| | 32 | 2.479 / 2.498 | $0.938 \pm 0.006$ | 3.39 / 2.96 |

Table 23: Large-batch extension of the matched earlier-schedule LLaDA2 10k AR/Parallel experiment: three rounds, 2,774 inputs per round, 2k cap. Tok/s averages pooled round throughput; the last column averages eight-workload geometric means. Accepted length excludes the target-supplied first token.

| Batch | Drafter | Tok/s | Accepted | Cap hits (%) | Parallel / AR |
|---|---|---|---|---|---|
| 64 | AR | $857.24 \pm 1.08$ | 7.732 | 16.98 | $1.390 \pm 0.002$ |
| 64 | Parallel | $1139.38 \pm 1.95$ | 6.266 | 14.06 | $1.390 \pm 0.002$ |
| 128 | AR | $1168.39 \pm 1.30$ | 7.770 | 17.56 | $1.187 \pm 0.000$ |
| 128 | Parallel | $1341.24 \pm 1.79$ | 6.323 | 14.17 | $1.187 \pm 0.000$ |

**Self-Draft and Learned Draft.** Self-Draft projects the masked-block target hidden states through the frozen vocabulary head and takes argmax, with no learned drafter forward pass. Both variants share verification and recovery; vocabulary projection remains inside timing. This matched run uses both targets at batches 1/8/32 with three rounds; only Learned Draft uses a one-epoch head. Self-Draft provides the direct proposal: learned refinement increases accepted draft length on both targets, while the net throughput benefit depends on the target and drafting cost. LLaDA2 gains 3.4–5.0% throughput; SDAR's throughput decreases by 4.4–6.2% (table 22). This SGLang run measured speed, lengths, and truncation without task scoring. The SDAR HF two-GPU quality control appears separately in table 32. Hidden and Self-Draft belong to different backend/checkpoint groups and are not ranked directly.

**When learned refinement pays off.** Equation (9) restates $C_l/T_l > C_s/T_s$, where $C$ is committed tokens and $T$ total step time, both per proposal and pooled over workloads; the values in section 4.3 divide output tokens and wall time by the number of proposals in the matched run. Only when non-drafting costs are unchanged, so $T_l = T_s + T_d$, does the right-hand side reduce to the added drafting time $T_d/T_s$. Our end-to-end comparisons use measured total step times to account for changes in recovery frequency and state-construction cost.

Actual commitments differ from accepted draft length. For the fully masked path before stopping-rule truncation, let $A$ be accepted draft tokens and $F$ indicate full acceptance. The step commits $A + 2 - F$ tokens: rejection includes the first token and correction, whereas full acceptance includes the first token but no bonus correction. Thus its mean is $C = \mathbb{E}[A] + 2 - \Pr(F)$. Initialization, partial-block fallback, and final output trimming lie outside this step model; the measured complete-generation throughput remains the criterion for choosing a configuration.

## F Block Transfer and Inference Width

At a fixed inference width, native short-block training improves accepted length by 1.0–4.0% and throughput by 1.2–4.9% over the long-trained head. The measured efficiency differences are modest,

Table 24: Training-block and inference-block choices on LLaDA2 (matched 10k heads, one round). All throughput values use the $B = 32$, $K = 32$ row as 1.000 within each batch; bold marks the fastest configuration. Compare rows at the same $K$ to isolate training-block choice.

| | | Accepted draft tokens | | | Relative throughput | | |
|---|---|---|---|---|---|---|---|
| Train block $B$ | Infer block $K$ | Batch 1 | Batch 8 | Batch 32 | Batch 1 | Batch 8 | Batch 32 |
| 8 | 8 | 4.17 | 4.20 | 4.17 | 0.794 | 0.835 | 0.951 |
| 32 | 8 | 4.05 | 4.04 | 4.05 | 0.774 | 0.800 | 0.906 |
| 16 | 16 | 5.53 | 5.54 | 5.53 | 0.970 | **1.025** | **1.129** |
| 32 | 16 | 5.41 | 5.45 | 5.47 | 0.947 | 1.002 | 1.116 |
| 32 | 32 | 6.00 | 6.07 | 6.11 | **1.000** | 1.000 | 1.000 |

Table 25: Quality of native short-block and train-long/infer-short heads on the five labeled speed workloads (%). Each paired comparison fixes inference width $K$ and requested batch. Cap-hit rates cover the eight-workload suite; one round.

| Batch | $K$ | Head | GSM8K | MATH | AIME | HEval | LCB | Avg. | Cap hits |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 8 | Native | 87.80 | 73.40 | 16.67 | 93.29 | 40.00 | 62.23 | 16.55 |
| | | $B = 32$ | 87.80 | 73.80 | 10.00 | 89.02 | 40.40 | 60.20 | 15.36 |
| | 16 | Native | 89.40 | 75.60 | 20.00 | 86.59 | 39.40 | 62.20 | 14.35 |
| | | $B = 32$ | 90.00 | 75.00 | 10.00 | 89.02 | 39.60 | 60.72 | 13.84 |
| 8 | 8 | Native | 87.40 | 74.00 | 16.67 | 89.02 | 40.40 | 61.50 | 16.37 |
| | | $B = 32$ | 86.80 | 75.40 | 10.00 | 89.02 | 39.60 | 60.16 | 15.39 |
| | 16 | Native | 88.80 | 75.80 | 10.00 | 87.20 | 39.20 | 60.20 | 14.71 |
| | | $B = 32$ | 88.00 | 74.60 | 23.33 | 88.41 | 40.00 | 62.87 | 14.46 |
| 32 | 8 | Native | 88.00 | 72.60 | 16.67 | 92.07 | 39.40 | 61.75 | 16.29 |
| | | $B = 32$ | 86.60 | 74.80 | 6.67 | 90.24 | 41.00 | 59.86 | 15.21 |
| | 16 | Native | 87.00 | 74.80 | 10.00 | 87.80 | 39.60 | 59.84 | 14.96 |
| | | $B = 32$ | 88.40 | 75.40 | 20.00 | 89.02 | 38.20 | 62.20 | 13.91 |

while quality and truncation effects are task-dependent. Across inference widths, the best observed throughput configuration shifts: $K = 32$ is fastest at batch one, whereas a native $K = 16$ head is fastest at batch 32. This second observation changes target block scheduling as well as proposal width and is separate from the fixed-$K$ transfer comparison. Only one round was collected, so no cross-round error bars are reported.

Table 26: Inference-width and training-head sweep. One evaluation round; throughput and speedup follow appendix C.

| Batch | $K$ | Head | tok/s | Accepted | vs. LC1 | vs. $B32/K32$ |
|---|---|---|---|---|---|---|
| 1 | 8 | $B = 8$ | 127.0 | 4.174 | 3.725 | 0.794 |
| | 8 | $B = 32$ | 123.4 | 4.052 | 3.631 | 0.774 |
| | 16 | $B = 16$ | 153.2 | 5.525 | 4.214 | 0.970 |
| | 16 | $B = 32$ | 149.9 | 5.413 | 4.115 | 0.947 |
| | 32 | $B = 32$ | 158.4 | 6.003 | 4.257 | 1.000 |
| 8 | 8 | $B = 8$ | 384.9 | 4.198 | 2.995 | 0.835 |
| | 8 | $B = 32$ | 365.9 | 4.038 | 2.870 | 0.800 |
| | 16 | $B = 16$ | 470.6 | 5.536 | 3.462 | 1.025 |
| | 16 | $B = 32$ | 464.3 | 5.446 | 3.382 | 1.002 |
| | 32 | $B = 32$ | 462.4 | 6.065 | 3.321 | 1.000 |
| 32 | 8 | $B = 8$ | 831.7 | 4.173 | 2.345 | 0.951 |
| | 8 | $B = 32$ | 800.5 | 4.047 | 2.236 | 0.906 |
| | 16 | $B = 16$ | 1021.9 | 5.526 | 2.523 | 1.129 |
| | 16 | $B = 32$ | 999.8 | 5.470 | 2.494 | 1.116 |
| | 32 | $B = 32$ | 897.5 | 6.108 | 2.352 | 1.000 |

## G Execution Ablation Context

Table 27: Incremental execution speedup with matched one-epoch drafters. Each row adds to the preceding configuration; separate execution is 1.000 within each target and batch. Three-round means; bold marks the fastest displayed value. SDs are in table 28.

| | LLaDA2.0-mini | | | SDAR-8B | | |
|---|---|---|---|---|---|---|
| Configuration | Batch 1 | Batch 8 | Batch 32 | Batch 1 | Batch 8 | Batch 32 |
| Separate execution | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| + Folded verification | 1.010 | 1.009 | 0.944 | **1.306** | 1.195 | 1.067 |
| + Batched recovery | **1.012** | **1.649** | **1.988** | 1.305 | **1.305** | **1.197** |

Table 28: System-ablation speedups (three-round mean ± sample SD). The middle column isolates adding batched recovery after folded verification; the last column compares the complete configuration with separate execution.

| Target | Batch | Folded / Separate | Complete / Folded | Complete / Separate |
|---|---|---|---|---|
| LLaDA2.0-mini | 1 | $1.010 \pm 0.003$ | $1.003 \pm 0.002$ | $1.012 \pm 0.003$ |
| | 8 | $1.009 \pm 0.002$ | $1.634 \pm 0.001$ | $1.649 \pm 0.003$ |
| | 32 | $0.944 \pm 0.003$ | $2.107 \pm 0.010$ | $1.988 \pm 0.013$ |
| SDAR-8B | 1 | $1.306 \pm 0.001$ | $0.999 \pm 0.002$ | $1.305 \pm 0.003$ |
| | 8 | $1.195 \pm 0.006$ | $1.092 \pm 0.003$ | $1.305 \pm 0.002$ |
| | 32 | $1.067 \pm 0.003$ | $1.122 \pm 0.004$ | $1.197 \pm 0.003$ |

Separate execution runs approximate verification first, then jointly constructs committed KV and next-block conditioning, grouping requests by committed length. Folded verification prepares candidate states during verification and reuses them after full acceptance; rejected proposals require recon-

Table 29: Closed-loop system-ablation context. Accepted length, mean output length, and cap-hit rate are aggregated across three rounds. No task-quality scoring was collected for this run.

| Target | Batch | Configuration | Accepted | Output tokens | Cap hits (%) |
|---|---|---|---|---|---|
| LLaDA2.0-mini | 1 | Separate execution | 6.151 | 719.0 | 14.46 |
| | | + Folded verification | 6.174 | 713.2 | 14.10 |
| | | + Batched recovery | 6.174 | 713.2 | 14.10 |
| | 8 | Separate execution | 6.173 | 706.6 | 14.13 |
| | | + Folded verification | 6.190 | 717.9 | 14.60 |
| | | + Batched recovery | 6.134 | 713.1 | 14.28 |
| | 32 | Separate execution | 6.235 | 715.9 | 14.28 |
| | | + Folded verification | 6.212 | 720.7 | 14.78 |
| | | + Batched recovery | 6.222 | 721.3 | 14.89 |
| SDAR-8B | 1 | Separate execution | 2.509 | 336.3 | 3.64 |
| | | + Folded verification | 2.500 | 327.5 | 2.99 |
| | | + Batched recovery | 2.500 | 327.5 | 2.99 |
| | 8 | Separate execution | 2.497 | 322.8 | 2.78 |
| | | + Folded verification | 2.508 | 331.5 | 3.28 |
| | | + Batched recovery | 2.515 | 338.0 | 3.64 |
| | 32 | Separate execution | 2.508 | 332.4 | 3.28 |
| | | + Folded verification | 2.503 | 326.1 | 2.92 |
| | | + Batched recovery | 2.495 | 324.2 | 2.88 |

struction. Batched recovery combines different committed lengths using padding and per-request visibility and position IDs (appendix H). Fused vocabulary projection remains on, and CUDA Graphs remain off.

Full-block acceptance is approximately 2.5% on LLaDA2 and 76% on SDAR. These rates are consistent with the observed benefits of state reuse on SDAR and batched recovery on LLaDA2. Implementation checks used three batches of eight requests per target, with rejections at different positions. On these inputs, the checks confirmed matching effective tokens, positions, and masks, unchanged inspected prefix KV, and matching checked KV copies; matching compute shapes removed observed no-fold verification-logit differences. No task-quality scores were collected for the system run. The later fixed-length profile in table 35 is a separate diagnostic.

## H Verification Layout and Training Details

**Four-region layout.** Let $P = |c|$ be the committed prefix length. Region indices below are one-based, while target position IDs are zero-based. All queries read the existing prefix KV. The within-call visibility in table 31 is applied at every target layer; a dash denotes no visible keys in that region. These definitions specify the non-strict L2R path used by DSpec.

Table 30: Inputs and logical positions in the folded forward. $D$ and $V$ form the approximate verifier; $C$ and $N$ prepare candidate generation states.

| Region | Input | Index range | Position ID | Role |
|---|---|---|---|---|
| $D_i$ | $\hat{x}_i$ | $1 \le i \le B-1$ | $P+i-1$ | Draft context |
| $V_i$ | `[MASK]` | $1 \le i \le B-1$ | $P+i$ | Predict $\tilde{y}_{i+1}$ |
| $C_i$ | $\hat{x}_i$ | $1 \le i \le B$ | $P+i-1$ | Candidate KV |
| $N_i$ | `[MASK]` | $1 \le i \le B$ | $P+B+i-1$ | Next-block anchor |

Table 31: Within-call attention visibility. Each row lists the keys visible to the indicated query, in addition to the committed prefix.

| Query | $D_j$ | $V_j$ | $C_j$ | $N_j$ |
|---|---|---|---|---|
| $D_i$ | $j \le i$ | $j \ge i$ | – | – |
| $V_i$ | $j \le i$ | $j \ge i$ | – | – |
| $C_i$ | – | – | All | – |
| $N_i$ | – | – | All | All |

**Algorithm 1** One fully masked DSPEC block step (before stopping-rule truncation)

**Require:** Prefix $c$, draft variant, valid cached state if available
Obtain the anchor $(H^0, Z^0)$ from cache or a target pass
**if** Self-Draft **then**
    Form $\hat{x}_{1:B}$ directly using equation (2)
**else**
    Compute $\Delta H$, $Z^{\mathrm{d}}$ and proposals using equations (4) and (5)
**end if**
$(\tilde{y}, K_{\mathrm{cand}}, S_{\mathrm{next}}) \leftarrow \mathrm{Fold}(c, \hat{x}_{1:B})$
Compute first mismatch $r$ using equation (6)
**if** $r = B + 1$ **then**
    Commit $\hat{x}_{1:B}$
    Adopt candidate KV state $K_{\mathrm{cand}}$
    Cache $S_{\mathrm{next}}$ for the next block
**else**
    Commit $\hat{x}_{1:r-1}$ and correction $\tilde{y}_r$
    Invalidate speculative suffix and next-block states
    Recover from the actual corrected prefix
**end if**
Enforce stopping rules; release state for completed requests

**Why verification is approximate.** The shared $D/V$ computation permits indirect dependencies absent from independently reconstructed partial blocks. For example, when $B \ge 3$, $V_1$ predicts position 2 and can read $V_2$, which can itself read $D_2$ containing $\hat{x}_2$. Across layers, the prediction for position 2 can therefore depend on the proposal being checked. L2R prefix acceptance does not remove this dependence. Conversely, $C/N$ do not read $D/V$, and $C$ does not read $N$. Given identical prefix KV, inputs, positions, and visibility, the $C/N$ computation matches separate state construction for the same candidate block in real arithmetic. It can therefore be combined with verification without introducing the cross-region dependencies of $D/V$.

**Separate execution and recovery.** Without folding, verification uses only $D/V$, with $2(B-1)$ queries. After acceptance and correction, a joint state-construction pass consumes the actual committed tokens followed by $B$ masks. It constructs the committed KV and the next-block anchor together, including after full acceptance. With folding, full acceptance reuses $C/N$, while a mismatch requires the same reconstruction from the corrected prefix. The grouped implementation batches requests with equal committed lengths.

For batched recovery, let request $j$ commit $r_j$ tokens and let $R = \max_j r_j$. Each request uses $R + B$ physical query slots: $r_j$ committed tokens, $R - r_j$ padding slots, and $B$ next-block masks. Committed queries read the prefix and all real committed tokens; next-block queries additionally read all next-block masks. Neither reads padding. Padding queries have self-visibility and prefix visibility but cannot influence valid queries. Crucially, next-block position IDs are $P_j + r_j, \ldots, P_j + r_j + B - 1$, independent of the padding width. One batched forward produces the states; only the real committed KV entries are copied into the request cache.

**Emitted and accepted tokens.** Before EOS or output-budget truncation, full acceptance emits $B$ tokens, of which $B-1$ are accepted drafter positions; a mismatch emits $r$ tokens, of which $r-2$ are

accepted drafter positions. A mismatch at $r = B$ still triggers recovery despite emitting $B$ tokens. Full acceptance produces no additional bonus token beyond the block.

**Initialization and partial blocks.** Algorithm 1 describes the fully masked speculative-block path. Initialization and cache misses compute target anchors, and cached next-block states are checked against the request's current prefix length. A batch containing a partially filled block takes a target-only fallback instead of this speculative path: SDAR selects the highest-confidence masked position, whereas LLaDA2 uses L2R selection. EOS and output-budget truncation are applied to emitted tokens.

**Implementation notes.** SGLang maintains committed KV and reusable anchors per request, adopting candidate states only after full acceptance. For Learned Draft, the single drafter layer uses QK-normalized grouped-query attention, a dense SwiGLU feed-forward network, output normalization, and local positions $0, \ldots, B-1$. All target parameters remain frozen. The drafter normalizes the embedding and anchor streams separately before $W_f$ projects their concatenation to the drafter hidden size. Teacher generation runs in SGLang, with hidden states transferred through Mooncake and distributions reconstructed through the frozen head; softmax and overlap are computed over the full vocabulary in FP32. Vocabulary projection is fused as $H^0 W^\top + \Delta H W^\top = (H^0 + \Delta H) W^\top$. Grouped versus batched recovery and fused versus unfused projections are checked separately as implementation variants. Initialization, cache misses, and rejected proposals can require additional target forwards, and the $4B - 2$ queries of a folded call make its cost differ from an ordinary block forward.

**Learned Draft: teacher order and loss reduction.** The parallel trainer requires the recorded teacher order to equal $0, \ldots, B-1$ and computes prefix products directly along that position dimension. For a validity mask $m_i \in \{0, 1\}$, define $n = \sum_i m_i$ and $\bar{\alpha}_i = m_i \alpha_i + (1 - m_i)$. The implemented per-block loss is

$$\ell = 1 - \frac{\sum_{k=1}^{B} m_k \prod_{i=1}^{k} \bar{\alpha}_i}{\max(n, 1)}, \qquad \mathcal{L} = \mathrm{mean}_{\mathrm{blocks}}\, \ell. \tag{11}$$

Invalid positions contribute a factor of one to the product and are excluded from the length sum. A fully invalid block would contribute a constant one with zero gradient. The first position remains supervised and enters every prefix product, although deployment overrides its proposal with the anchor prediction. The CE control instead uses the teacher's argmax as a hard label and averages cross-entropy over valid tokens. Thus, for incomplete blocks, the two objectives also differ in their reduction weights, and the CE ablation compares complete objectives rather than isolating prefix coupling from positionwise TV.

# I HF EVALUATION WITH TWO GPUS

**Scope and execution settings.** We evaluate nine configurations on SDAR-8B-Chat with two H20 GPUs per request, BF16, and batch one. Native, FreeDave, S2D2, DSPEC (Self-Draft), and the one-epoch DSPEC (Learned Draft) checkpoint use target tensor parallelism of degree two. Native uses fixed $K = 4$, four-step confidence-order decoding. Native and both DSPEC variants are tested with eager execution and CUDA Graphs (CG). SimSD[1] uses its draft/target pipeline on separate GPUs, with CG, fused denoising, and speculative target extension; fused linear layers and folded draft extension are disabled. Thus, all configurations use two GPUs, but their device layouts differ. The main text summarizes these nine configurations; the separate single-GPU comparisons remain in tables 16 to 18.

Each configuration is evaluated for three rounds on 3,777 questions across six tasks. AIME is not included. All tasks use a 2k output cap except GPQA-Diamond, which uses 16k. Regular-task shards evaluate methods sequentially on the same GPU pair, rotating method order across rounds. GPQA uses mixed device placement, so its throughput is reported descriptively.

**Main-table aggregation.** Table 5 aggregates GSM8K, MATH-500, HumanEval, LiveCodeBench, and IFEval for both its speed and quality summaries. For each task, speed uses output tokens pooled over the three rounds divided by their total runtime; each configuration's ratio to Native in the same execution mode is then geometrically averaged across the five tasks. Eager and CG use separate Native denominators. Quality first averages each task over rounds and then weights these five tasks equally. GPQA's normalized score is shown separately, and its mixed-placement throughput does not enter the speedup.

Table 32: SDAR HF quality with two H20s per request (batch one, three rounds, %). Target TP2 is used except for SimSD, which places draft and target on separate GPUs. GPQA uses 16k and final-line normalization; other tasks use 2k. The six-task suite has 3,777 questions.

| Method | Mode | GSM8K | MATH | HEval | LCB | IFEval | GPQA |
|---|---|---|---|---|---|---|---|
| Native | Eager | 90.45 | 76.80 | 67.07 | 24.27 | 61.00 | 23.74 |
| FreeDave | Eager | 90.75 | 75.60 | 67.07 | 24.74 | 62.05 | 22.73 |
| S2D2 | Eager | 89.84 | 75.53 | 49.59 | 22.37 | 59.33 | 19.36 |
| DSPEC (Self-Draft) | Eager | 90.75 | 78.40 | 68.29 | 23.22 | 63.34 | 25.76 |
| DSPEC (Learned Draft) | Eager | 90.98 | 77.20 | 67.68 | 22.94 | 62.42 | 24.24 |
| Native | CG | 90.45 | 77.00 | 67.07 | 23.79 | 61.49 | 26.77 |
| DSPEC (Self-Draft) | CG | 90.75 | 78.40 | 68.29 | 23.22 | 63.34 | 25.76 |
| DSPEC (Learned Draft) | CG | 90.98 | 77.20 | 67.68 | 22.94 | 62.42 | 24.24 |
| SimSD | CG | 88.40 | 74.00 | 63.41 | 18.48 | 56.25 | 20.20 |

**Quality and efficiency.** Table 32 reports every headline task, using the same final-line GPQA normalization as the main study. Quality differences are task-dependent, and Self-Draft remains competitive on this SDAR configuration. On the five regular headline tasks, it is slightly faster than Learned Draft in both execution modes and scores higher on four tasks; Learned Draft scores higher on GSM8K. Table 33 reports per-domain pooled throughput. Eager and CG runs also differ in their cache-path implementations.

We run FreeDave through the released implementation's SDAR support[2], using its `tree_attention` draft mode, eight draft steps, Eager Mode (cross-block pre-drafting and pre-verification), and whole-EOS-block stopping rule. Reported output tokens stop at the first

[1] https://github.com/airevo2/SimSD-release

[2] https://github.com/cychomatica/FreeDave

Table 33: HF two-GPU throughput accompanying the quality run (pooled output tok/s per domain across three rounds). Regular tasks use the same GPU pair within each shard. GPQA* uses mixed device placement and is descriptive only. SimSD uses a draft/target pipeline; the other methods use target TP2.

| Method | Mode | GSM8K | MATH | HEval | LCB | IFEval | GPQA* |
|---|---|---|---|---|---|---|---|
| Native | Eager | 25.55 | 25.78 | 25.61 | 25.45 | 25.71 | 24.98 |
| FreeDave | Eager | 42.79 | 47.04 | 39.41 | 42.14 | 36.32 | 41.53 |
| S2D2 | Eager | 35.14 | 36.23 | 33.96 | 35.28 | 31.09 | 33.66 |
| DSPEC (Self-Draft) | Eager | 89.24 | 97.37 | 79.24 | 94.54 | 61.79 | 83.20 |
| DSPEC (Learned Draft) | Eager | 86.61 | 93.18 | 75.79 | 90.79 | 59.19 | 83.18 |
| Native | CG | 54.48 | 56.53 | 56.12 | 55.31 | 56.55 | 23.71 |
| DSPEC (Self-Draft) | CG | 183.34 | 204.35 | 168.00 | 195.09 | 134.52 | 89.54 |
| DSPEC (Learned Draft) | CG | 182.69 | 200.79 | 165.10 | 193.79 | 132.37 | 92.32 |
| SimSD | CG | 73.25 | 80.36 | 73.96 | 71.68 | 53.52 | 30.17 |

configured EOS, while the time spent computing beyond that EOS remains in the measured runtime. S2D2[3] retains its randomized acceptance and resampling.

[3] https://github.com/phymhan/S2D2

# J Cost Measurements and Implementation Diagnostics

The following fixed-state microbenchmarks, short-output profiles, and numerical checks characterize implementation behavior separately from the full-workload evaluations.

## J.1 Draft and Target Costs

Table 34: LLaDA2 fixed-state cost microbenchmark: block 32, synthetic prefix 1,024, folded target query 126; medians after 100 warmups and 500 timed calls. AR31 omits first-position priming; deployed AR32 includes it.

| Batch | Parallel ms | AR31 ms | AR32 ms | Target ms | Parallel/target | AR32/target |
|---|---|---|---|---|---|---|
| 1 | 1.385 | 58.199 | 59.551 | 19.265 | 0.072 | 3.091 |
| 8 | 2.898 | 60.053 | 61.589 | 27.084 | 0.107 | 2.274 |
| 32 | 8.169 | 60.286 | 62.089 | 85.538 | 0.095 | 0.726 |
| 64 | 15.875 | 60.661 | 62.412 | 155.326 | 0.102 | 0.402 |
| 128 | 31.568 | 60.970 | 62.748 | 298.122 | 0.106 | 0.210 |

The cost microbenchmark fixes the LLaDA2 block at 32, a synthetic target prefix of 1,024 tokens, and a 126-query folded target call. The drafter uses only block-local KV. Parallel drafting takes 7.2–10.7% of the measured target-call time across the five batches; deployed AR32 takes 21.0–309.1%. These call-cost ratios help explain the acceptance–cost trade-off. A short instrumented decode at batch eight observes median draft times of 3.33 ms for Parallel and 64.62 ms for AR over five and eight complete batch-eight calls, respectively. The corresponding fixed-state measurements are 2.90 and 61.59 ms. The instrumented decode and fixed-state measurements show similar draft-time scales.

Table 35: SGLang stage profile (one H20, TP1, one-epoch drafter, fixed 128 output tokens). Clean throughput is uninstrumented. Draft, target and remaining times are synchronized wall-clock intervals in ms per batch; instrumentation perturbs scheduling. The last column is synchronized / clean latency.

| Target | Batch | Mode | Clean tok/s | Draft ms | Target ms | Other ms | Perturbation |
|---|---|---|---|---|---|---|---|
| SDAR-8B | 1 | eager | 104.94 | 67.31 | 1021.79 | 174.67 | 1.036 |
| SDAR-8B | 1 | cg | 226.87 | 23.45 | 445.66 | 157.33 | 1.110 |
| SDAR-8B | 8 | eager | 473.46 | 153.75 | 1642.66 | 397.36 | 1.014 |
| SDAR-8B | 8 | cg | 717.83 | 103.86 | 1066.58 | 361.29 | 1.074 |
| LLaDA2.0-mini | 1 | eager | 130.10 | 20.76 | 899.84 | 59.35 | 0.996 |
| LLaDA2.0-mini | 1 | cg | 402.72 | 4.79 | 276.75 | 52.65 | 1.051 |
| LLaDA2.0-mini | 8 | eager | 630.97 | 48.69 | 1485.28 | 116.68 | 1.017 |
| LLaDA2.0-mini | 8 | cg | 984.14 | 24.64 | 940.89 | 117.16 | 1.041 |

The separate SGLang stage study fixes output length at 128 tokens and tests both targets at batches one and eight with eager and CG execution. It records uninstrumented throughput, asynchronous event timing, and a serial synchronized profile. Table 35 reports the clean throughput and synchronized intervals with the measured perturbation factor. Target computation dominates these profiles, while draft costs depend on target and execution mode. The synchronized intervals include the effects of inserted waits.

## J.2 CUDA Graph Numerical Checks

**Padding and output agreement.** On eight short diagnostic requests per target, the original padded CG path matches eager outputs for 7/8 SDAR requests and 2/8 LLaDA2 requests. Using CG only

for exact captured batch sizes, with eager fallback otherwise, yields 8/8 agreement for both targets. Across 169 same-state comparisons (69 SDAR and 100 LLaDA2), repeated replay and eager execution of the captured forward at the same shape produce elementwise-identical logits and hidden states. The observed differences arise when padding changes computation shapes; checked prefix KV entries remain unchanged.