JETSPEC++: FORESIGHT PARALLEL DRAFTING WITH SELF REFINEMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding's effectiveness is governed by two quantities: the acceptance rate of drafted tokens and the cost of producing them. We show the two objectives can be co-optimized with algorithm-system co-design, and present JetSpec++, which retains a strong learned draft model while hiding its cost almost entirely. JetSpec++ combines three components: intra-block self refinement, which trains the draft model to iteratively correct its own parallel predictions within a block of tokens; inter-block foresight drafting, which begins the next block before the current one is verified; and pipelined drafting and verification, which overlaps the two stages so drafting's marginal wall-clock cost approaches zero. In comparison with state-of-the-art speculative decoding methods, JetSpec++ attains a wall-clock speedup of up to 6.74× over autoregressive (AR) decoding and 1.41× over DSpark, with up to 1250 TPS single-request throughput on GB200, and an average acceptance length of 7.60 across seven benchmarks, while preserving the target model's output distribution.

1 INTRODUCTION

Speculative decoding (SD) is a widely adopted technique for parallelizing large language model (LLM) generation (Leviathan et al., 2023; Chen et al., 2023). Its effectiveness is determined by two factors, the acceptance rate and the drafting cost. Earlier work showed that the target model's hidden states of the verified context can align the draft model with the target distribution and thereby raise the acceptance rate (Li et al., 2024a; 2025). DFlash employs parallel drafting and has further reduced per-token drafting cost (Chen et al., 2026), providing bigger headroom for higher throughput (Cheng et al., 2026) and lower latency (Hu et al., 2026a). A natural question to ask is whether we can further reduce drafting cost to zero on the critical path.

One solution is to further push down the drafting cost. Cheap drafting usually relies on model-free techniques such as prompt lookup decoding (Saxena, 2023; Somasundaram et al., 2025), retrieval from the text generated so far (Oliaro et al., 2025; Hu et al., 2025). However, model-free methods show high variance and low average acceptance rates, particularly when reusable patterns are sparse or fail to generalize. The question then becomes, can we retain the high acceptance rate of model-based drafting, while removing drafting cost from the draft-then-verify critical path?

The drafting cost can be removed from the critical path by overlapping drafting for the next round with verification in the current round (Figure 1). However, for a learned draft model conditioned on the target model's hidden states, the key challenge is that it must draft ahead before the next verified context becomes available (Zhang et al., 2026b). Accurately predicting this future context is therefore critical to maintaining draft quality.

We present JetSpec++, which maintains a target-conditioned parallel draft model and overlaps drafting for the next step with verification of the current step. To draft block $t{+}1$ while block t is still being verified, however, the draft model must operate without two pieces of information that become available only after verification: (1) the number of accepted draft tokens in block t and the corresponding bonus token, and (2) the target hidden states for the newly verified context in block t.

JetSpec++ addresses both missing components through *Foresight Drafting*. Rather than waiting for the verification outcome, the draft model anticipates multiple possible "world lines," each corresponding to a different acceptance-length outcome. For each request, a hidden-to-hidden adapter

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.

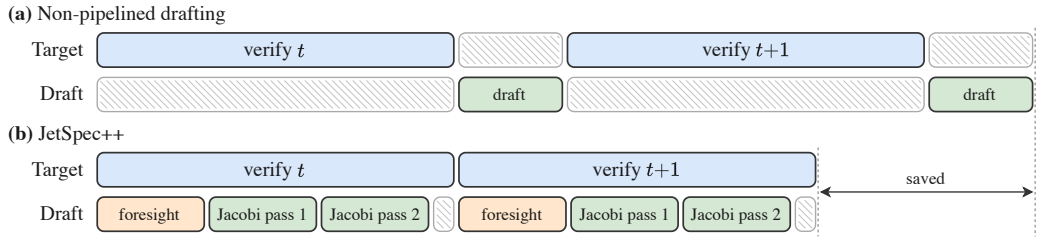


Figure 1: **Non-pipelined drafting versus JetSpec++.** (a) A draft model that reads the target's hidden states drafts only after verification. (b) JetSpec++ runs foresight and Jacobi passes while block t is verified. Segment lengths are schematic.

predicts the hidden states of the target verified context in block t from the already available verified context in block $t-1$. A lightweight decoder then maps these predicted states to possible anchor tokens. The predicted hidden states and anchor token are then used to draft block $t+1$.

Because drafting runs concurrently with target verification, additional draft-model computation incurs no extra latency as long as it remains within the verification window. JetSpec++ exploits this otherwise idle compute by augmenting JetSpec (Hu et al., 2026a)'s causal parallel draft model with Jacobi-style self-refinement, allowing the draft model to iteratively revise its parallel predictions over multiple numbers of function evaluations. Each additional refinement pass progressively moves the draft distribution toward a more autoregressive factorization with better target alignment.

In summary:

- We propose *Foresight Drafting*, which enables draft-verification pipelining for modern speculative decoding whose learned draft models depend on target-model hidden states. It anticipates possible verification outcomes and predicts the corresponding target hidden states and anchor tokens, allowing the next block to be drafted before the current block is verified.
- We introduce Jacobi-style self-refinement for JetSpec's causal parallel draft model, exploiting otherwise hidden drafting compute within the verification window to progressively strengthen causal token dependence and improve target alignment over multiple NFEs.
- We integrate JetSpec++ into a serving engine with CUDA graphs, where its end-to-end speedup is up to 6.74× and 1.41× over AR and DSpark, at up to 1,250 tokens per second for a single request, and cuts the per-round overhead beyond verification by up to 76% relative to DSpark (Table 2).

2 Related Work

Speculative Decoding. Speculative decoding drafts several tokens and verifies them in one target forward pass, and its rejection-sampling rule leaves the output distribution unchanged (Leviathan et al., 2023; Chen et al., 2023). Its speedup grows with the number of accepted tokens and shrinks with the cost of drafting them. To accept more tokens per pass, tree verification scores many candidate branches at once (Miao et al., 2024; Chen et al., 2024; Li et al., 2024b), knowledge distillation aligns the draft model with the target (Zhou et al., 2024; Liu et al., 2024; Goel et al., 2024). Medusa (Cai et al., 2024) predicts future positions from the last hidden state, EAGLE (Li et al., 2024a), EAGLE-3 (Li et al., 2025) and HASS (Zhang et al., 2025) draft from the target's features. To draft cheaply, parallel drafting reduces draft cost by predicting an entire block in one forward pass (An et al., 2026; Xiao et al., 2024). Recent methods further improve parallel drafting alignment by conditioning on target hidden states (Chen et al., 2026), but their parallel per-position predictions are not conditioned on the earlier realized draft tokens within the block (Wang et al., 2026c). Sequential heads restore this dependency at the cost of serial computation (Cheng et al., 2026; Huang et al., 2026; Zheng & Li, 2026). JetSpec (Hu et al., 2026a) introduces block-causal conditioning over hidden states, while xPress (Wang et al., 2026b) improves drafts through iterative refinements.

Parallel Decoding. Parallel decoding lets a model itself generate several of its own tokens per forward pass, without a separate draft model. Jacobi decoding updates every position of a block by fixed-point iteration and converges to the greedy output (Song et al., 2021; Santilli et al., 2023). Lookahead decoding (Fu et al., 2024) collects n-grams from the Jacobi trajectory and verifies them, while consistency distillation (Kou et al., 2024) and Jacobi forcing (Hu et al., 2026b) train an AR model to reach the fixed point in fewer iterations. Discrete diffusion models generate text by denoising all positions in parallel over several steps (Sahoo et al., 2024; Nie et al., 2025; Ye et al., 2025). Block-wise discrete diffusion denoises one block at a time and conditions each block on the earlier ones (Arriola et al., 2025; Wu et al., 2026a), while caching (Wu et al., 2026b), pseudo-trajectory distillation (Qian et al., 2026) and multi-block decoding (Wang et al., 2026a) further accelerate diffusion inference. JetSpec++ applies Jacobi iteration to the draft model rather than the target.

Draft-Verification Pipelining. Prior work hides drafting latency by running the draft model asynchronously with target verification. Because the verification outcome is not yet known, recent methods hedge against multiple possible outcomes: SpecBranch (Shen et al., 2026) explores alternative branches, while speculative speculative decoding (SSD) (Kumar et al., 2026) prepares drafts for likely verification outcomes. SwiftSpec (Zhang et al., 2026b) similarly uses asynchronous tree drafting to remove draft latency from the critical path. Target-conditioned draft models make pipelining harder because the representation of the context under verification is unavailable. Prior approaches obtain an early target-side signal from partial-depth computation (Bhendawade et al., 2025; Venkatesha et al., 2025; Yu et al., 2026). D^{2}SD (Zhang et al., 2026a) forms a distribution over the number of accepted tokens k from the draft model's own confidences to choose the prefix lengths that a second draft prepares. In contrast, JetSpec++ predicts the missing target hidden states directly, enabling a target-conditioned parallel draft model to speculate ahead without executing the target model.

3 PRELIMINARIES

An AR model generates one token per forward pass, and each pass is bound by memory bandwidth, so compute sits idle. Speculative decoding (SD) (Leviathan et al., 2023; Chen et al., 2023) uses this idle compute to verify several drafted tokens in one forward pass. A cheap draft model q proposes γ tokens $d_1, \ldots, d_\gamma$ after the verified context x, and the target model p scores all of them in a single forward pass. From left to right, draft token d_i is accepted with probability $\min\{1, p(d_i \mid x, d_{<i})/q(d_i \mid x, d_{<i})\}$. At the first rejection the rest of the draft is discarded and the target model samples a token of its own, the **bonus token**, from the residual distribution $\propto \max\{p - q, 0\}$. If every draft token is accepted, the bonus token is sampled from p at position $\gamma + 1$. This procedure is lossless: it preserves the target model's generation distribution exactly.

Each round commits the accepted prefix plus one bonus token, and its expected number of committed tokens is the **acceptance length** τ. A round costs one drafting step and one verification step, which take times T_{draft} and T_{verify}. The per-committed-token latency is therefore $L = (T_{\text{draft}} + T_{\text{verify}})/\tau$, so a more accurate draft model raises τ and a faster draft model lowers T_{draft}.

3.1 PARALLEL DRAFTING

An AR draft model needs γ sequential forward passes, so T_{draft} grows with γ. Conditioned on the target hidden states h_x of the verified context, modern parallel draft models such as DFlash take the previous bonus token as the **anchor** y_0, fill the other γ positions with mask tokens, and predict all γ positions in one forward pass of the draft model q_θ with parameters θ (Chen et al., 2026):

$$q_\theta(y_1, \ldots, y_\gamma \mid x, h_x, y_0) \;=\; \prod_{i=1}^{\gamma} q_\theta(y_i \mid x, h_x, y_0) \tag{1}$$

T_{draft} then barely depends on γ, but each position is predicted without the tokens chosen at the other positions of the block. JetSpec (Hu et al., 2026a) makes the draft model causal, so in one forward pass every position attends to the hidden states at its own position and at the positions before it. We build on this architecture to enable causal token factorization via multi-pass Jacobi iterations.

3.2 JACOBI DECODING FOR PARALLEL DRAFTING

Applying parallel decoding to an AR LLM computes tokens from a γ-token block, $y_i = \arg\max_v p(v \mid x, y_{<i})$ for $i = 1, \ldots, \gamma$, in parallel with one forward pass. Jacobi decoding (Song et al., 2021; Santilli et al., 2023) formulates this process as solving these γ equations by fixed-point iteration, updating every position from the previous iteration r:

$$y_i^{(r+1)} \;=\; \arg\max_{v} \; p\big(v \mid x,\, y_{<i}^{(r)}\big), \qquad i = 1, \ldots, \gamma \tag{2}$$

Under a causal mask, each Jacobi iteration requires a single forward pass. The resulting iterates form a **Jacobi trajectory** that converges to the AR fixed point under greedy decoding (Santilli et al., 2023). JetSpec++ trains the draft model on both fully noised inputs and partially converged generations, teaching it to progressively map intermediate states toward the target-model fixed point (Section 4.3).

4 METHODS

In this section, we describe how JetSpec++ works. Draft-verification pipelining (Section 4.1) overlaps drafting with verification, reducing the critical-path latency for decoding and yielding the throughput model in Eq. 4. The challenge is that, in a pipelined round, the draft model must begin before the verification outcome is available. JetSpec++ addresses this with *Foresight Drafting* (Section 4.2), which learns predictive features that recover most of the acceptance-length loss (Appendix B). Finally, intra-block causal self-refinement (Section 4.3) makes use of the bubble exposed by pipelining to further increase the acceptance length.

4.1 DRAFT-VERIFICATION PIPELINING

Draft-verification pipelining drafts block $t+1$ while block t is being verified. In SSD's terms (Kumar et al., 2026), the draft process anticipates the **verification outcome** (k, a), the number k of accepted draft tokens together with the bonus anchor token a, and fills a **speculation cache** with one pre-drafted block per anticipated outcome. When verification returns, the realized (k, a) is matched. A **cache hit** makes the matching pre-drafted block the next block, and a **cache miss** falls back to an ordinary speculative step. Every committed token is verified by the rejection sampling acceptance rule (Leviathan et al., 2023), so the output distribution is unchanged.

Foresight Drafting (Section 4.2) predicts the missing states and estimates the distribution of the outcome from the prediction (Figure 2). The draft process of Figure 2 (a) is hidden when

$$T_{\text{draft}} = (T_{\text{adapter}} + T_{\text{dec}}) + (J+1)\, T_{\text{pass}}(|S|) + T_{\text{host}} \;\le\; T_{\text{verify}} \tag{3}$$

where T_{adapter} and T_{dec} account for adapter and decoder overheads in foresight drafting, with T_{adapter} counting every adapter call of the round, $T_{\text{pass}}(|S|)$ is the time of one batched draft-model forward over the set S of selected outcomes, J is the number of Jacobi refinement passes, and T_{host} is the time of the CPU work between GPU phases. A choice of more selected outcomes raises $T_{\text{pass}}(|S|)$, so under Eq. 3 it leaves fewer refinement passes J.

Throughput Model. Through draft-verification pipelining, on a hit the critical path overhead reduces to $\max(T_{\text{verify}}, T_{\text{draft}}) + T_{\text{comm}}$, where T_{comm} is the communication time between the two processes. On a miss, the draft model drafts a fallback block from the verified context in one pass (J_0), which adds T_{fallback} to the critical path, and the same round verifies it. JetSpec++ in Tables 1 and 2 serves this rule. With p_{hit} being cache hit probability, and τ_{hit} and τ_{miss} being pre-drafted block and fallback block acceptance lengths, we can write

$$\text{throughput} \;=\; \frac{p_{\text{hit}}\, \tau_{\text{hit}} + (1 - p_{\text{hit}})\, \tau_{\text{miss}}}{\max\big(T_{\text{verify}},\, T_{\text{draft}}\big) + T_{\text{comm}} + (1 - p_{\text{hit}})\, T_{\text{fallback}}} \tag{4}$$

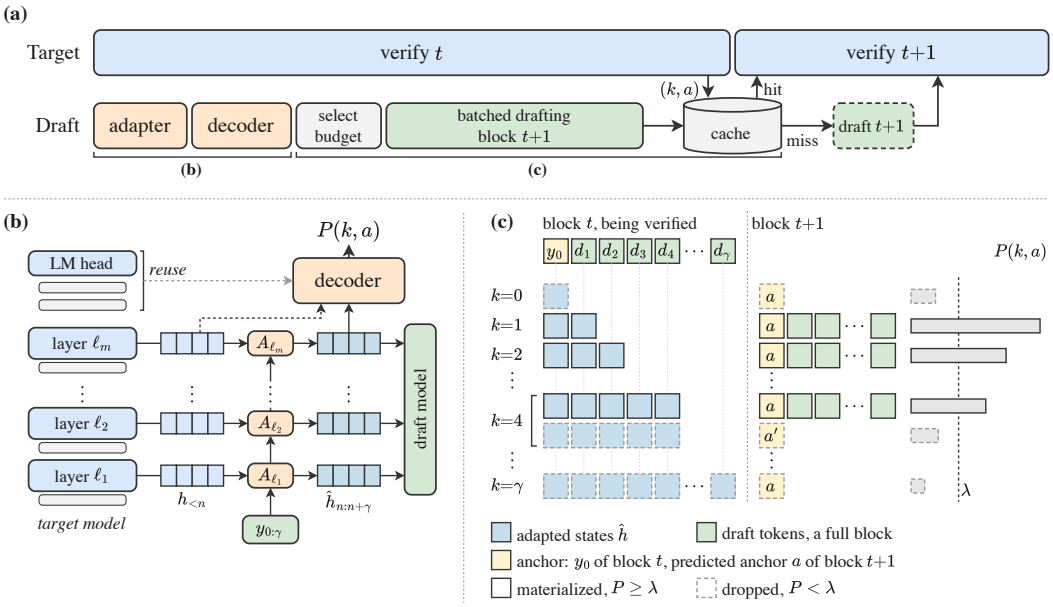


Figure 2: **Illustration of Foresight Drafting.** (a) *Foresight Drafting* predicts a budget S of outcomes (Eq. 7) in one batch with $J{+}1$ passes. The returned outcome (k, a) is looked up in the speculation cache. A hit immediately triggers verification, and a miss falls back to an ordinary speculative step (Section 4.1). (b) At each target layer ℓ_i, the adapter attends to the verified prefix states $h_{<n}$ and predicts the current-block states $\hat{h}_{n:n+\gamma}$. A decoder converts the top-layer hidden state predictions into the outcome distribution $P(k, a)$ of Eq. 6. (c) The speculation cache for block $t{+}1$ contains outcome k, the draft tokens $\{d_k\}$ that outcome k accepts, the corresponding adapted states $\hat{h}$ the draft model will read, and the decoded bonus token a as the anchor of block $t{+}1$. An outcome whose calibrated score (Eq. 11) reaches the threshold λ will be pre-drafted as a γ-token block.

In the numerator, since τ_{miss} is smaller than τ_{hit}, throughput grows with p_{hit}. Foresight drafting (Section 4.2) raises p_{hit} while keeping τ_{hit} close to its value under true hidden states (Section 5), and Jacobi iterations (Section 4.3) raise τ_{hit}.

Placement. JetSpec++ targets per-request latency. Because the two processes are separate, they can run on separate devices, as in SSD (Kumar et al., 2026). The two processes can also share one GPU with concurrent execution. We report both placements in Section 5.

4.2 Inter-Block Foresight Drafting

Foresight drafting consists of an adapter, a decoder, and a draft-model backbone, where data flow is illustrated in Figure 2. Under each possible outcome k, positions $1, \ldots, k$ of block t hold the known draft tokens $d_1, \ldots, d_k$, so the missing target-side hidden states belong to known tokens and are predictable by a learned map from the tokens up to position j. This allows training a specialized adapter to predict hidden states up to a position j within block t.

Hidden-to-Hidden Adapter. The adapter predicts, in one parallel pass, the target hidden states of all $\gamma + 1$ positions of the block under verification. At each target layer ℓ the adapter attends to the cached prefix at that layer and predicts the hidden states of the new future positions:

$$\hat{h}^{\ell}_{n+j} \;=\; A_{\ell}\big(\, h^{\ell}_{<n},\; \hat{h}^{\ell^-}_{n:n+j} \,\big), \qquad j = 0, \ldots, \gamma \tag{5}$$

Here n is the position of the anchor y_0, the bonus token of the latest verification. A_ℓ is the adapter module for layer ℓ. $h^{\ell}_{<n}$ is the target's cached context states at layer ℓ, the same as h^{ℓ}_{x} in Eq. 1, and $\hat{h}^{\ell^-}$ is the adapter's prediction at the next lower target layer that the draft model reads. The adapter is trained separately from the draft model. During training, each layer's adapter reads the

prediction of the module below it $\hat{h}^{\ell-}_{n:n+j}$, as at inference, and is trained using a Smooth L1 loss plus a cosine distance between the predicted and the true states against the target's true hidden states $h^{\ell}_{n:n+j}$ (Appendix E).

Decoder. The decoder turns the adapter's predictions into the outcome distributions. The decoder reuses frozen weights from the target's last few layers – above the deepest layer the adapter predicts, followed by the target's final norm and LM head. It attends to the verified context through keys and values computed from the target's true states (Appendix E). The decoder then runs under a causal mask over the adapter's top-layer predictions $\hat{h}_n, \ldots, \hat{h}_{n+\gamma}$. Position j yields $\hat{p}(\cdot \mid j)$, an estimate of the target's next-token distribution after the prefix up to j. The estimate at position $j-1$ assigns draft token d_j the probability $\hat{p}_j = \hat{p}(d_j \mid j-1)$. For each $k < \gamma$,

$$P(k) = \hat{p}_1 \cdots \hat{p}_k \,(1 - \hat{p}_{k+1}), \qquad P(k, a) \;=\; \hat{p}_1 \cdots \hat{p}_k \,\hat{p}(a \mid k), \quad a \neq d_{k+1} \tag{6}$$

If all γ draft tokens are accepted, then $P(\gamma) = \prod_{j=1}^{\gamma} \hat{p}_j$ and $P(\gamma, a) = P(\gamma)\,\hat{p}(a \mid \gamma)$. One decoder pass therefore scores every outcome at once, since row j serves both the acceptance of d_{j+1} and the bonus distribution at $k = j$ (Figure 3 in Appendix D.1). With a perfect decoder, Eq. 6 is the exact distribution of the outcome under speculative sampling at any temperature, using argmax draft in Eq. 8 (see Appendix D.1). In Appendix E, we show that the current design closely approximates the perfect decoder, matching the target's greedy token on 93.5% of held-out positions, while fine-tuning on adapter predictions does not raise top-1 agreement beyond the confidence interval (Appendix C).

Outcome Selection. Given the outcome distributions, the draft process still needs to select the outcomes to pre-draft. Due to limited draft-side compute, the draft process selects, computes and caches a finite set of B most promising outcomes. Outcomes are disjoint events, so the hit probability of a cache is the sum of its entries' probabilities:

$$S^{\star} \;=\; \arg\max_{|S| \leq B} \sum_{(k,a) \in S} P(k, a) \tag{7}$$

The optimum is the B most probable outcomes. In practice, decoder scores are imperfect, especially at longer lengths where errors compound through Eq. 6, so we calibrate the outcome scores on held-out prompts and retain candidates above a split-conformal threshold (Angelopoulos & Bates, 2023), up to cache size B. The threshold is fit offline to include the realized outcome at a target rate on held-out rounds; at inference, this fixed threshold yields a variable number of cached outcomes across rounds. Calibration affects only which outcomes are pre-drafted, never the verified output (Appendix C). Unlike SSD (Kumar et al., 2026), which allocates fixed anchors across acceptance lengths, Eq. 7 selects the cache dynamically from each round's outcome scores. We show SSD's geometric fan-out arises as a special case under its assumptions (Proposition 2, Appendix D.2).

Batched Drafting. Having selected the outcomes to predict, each outcome (k, a) is then drafted in the same way as an ordinary block. Its anchor is a, its context ends with $d_1, \ldots, d_k$, and the draft model reads the adapter's predictions at the positions up to k in place of the target's hidden states. The predicted states are the only approximation, and verification keeps the output lossless whatever they are. All selected outcomes are drafted as one batch, with the refinement passes of Section 4.3.

4.3 Intra-Block Self Refinements

A parallel draft model marginalizes each position over its possible predecessors, so its marginals are what PRESTO (Wang et al., 2026c) calls *prefix-blind*. DSpark (Cheng et al., 2026) gives an example: when the context admits *"of course"* and *"no problem"*, the block may read *"of problem"*. Sequential heads and separate refiners repair prefix-blindness at the cost of a sequential or recurrent module. We instead let the draft model refine its own draft in parallel, with no added module, in the time that pipelining frees under verification.

Inference. Eq. 1 factorizes the block into per-position marginals, whereas the target generates it token by token, $p(y_{1:\gamma} \mid x, y_0) = \prod_{i=1}^{\gamma} p(y_i \mid x, y_{<i})$. JetSpec++ makes the draft model follow this causal, token-level factorization through Jacobi iterations. Our draft model takes *natural-language tokens*, not mask embeddings, as the inputs, and attends causally within the block (Section 3.1),

so each position can condition on the tokens drafted before it. Refinement is a Jacobi iteration (Section 3.2) of the draft model on its own draft:

$$y_i^{(r+1)} \;=\; \arg\max_{v}\; q_\theta\big(v \,\big|\, x,\; h_x,\; y_{\le i}^{(r)}\big), \qquad i = 1, \ldots, \gamma \tag{8}$$

Here $y^{(r)}$, which we call J_r, is the block after r iteration passes. J_0 is the initial draft, whose inputs are tokens drawn from the verified context, and y_0 is the anchor. Causal attention lets position i read $y_{\le i}$ for each iteration. The block therefore moves from the factorization of Eq. 1 toward an explicit causal token factorization, in which each position conditions on the drafted tokens $y_{<i}$ before it. At a fixed point, reached within $r \le \gamma$ passes, every token is consistent with the drafted prefix from previous iterations.

Training. A draft model trained only on masked blocks is never trained to condition on tokens at its inputs. Following Jacobi forcing (Hu et al., 2026b), we train on blocks that pair a clean prefix with a noisy suffix, the shape of a partially converged Jacobi iterate. The first m positions hold the true tokens and the rest hold noise tokens $\tilde{y}$, which we draw from the sequence's own context rather than from Jacobi trajectories. We train the draft model to match the target model at every position:

$$\mathcal{L} \;=\; \mathbb{E}_m \sum_{i=1}^{\gamma} \mathcal{D}\Big(p(\cdot \mid x, y_{<i}) \,\Big\|\, q_\theta\big(\cdot \mid x, h_x, [\, y_{0:m};\, \tilde{y}_{m+1:\gamma}\,]_{\le i}\big)\Big) \tag{9}$$

where $\mathcal{D}(\cdot \,\|\, \cdot)$ denotes a divergence. We use forward KL, which outperforms reverse KL and cross-entropy SFT in our ablations (Section 5.4). The clean prefix length m is sampled per block from a fixed schedule (Appendix F), so the model is trained to repair any clean-prefix block in Jacobi decoding. Appendix F provides our clean-prefix selection rule and details on our training setup.

Jacobi Iteration Budget and Other Bubble-Fill Optimizations. The draft process refines each selected outcome with up to J Jacobi passes after the initial draft J_0, subject to the latency budget in Eq. 3. After J_0, an optional adapter refresh predicts the target hidden states corresponding to the block's current draft tokens; its cost is included in T_{adapter}. The refresh stops at the first position where the cumulative product of the draft model's top-1 probabilities falls below a threshold, and subsequent refinement passes use the refreshed states as context.

5 Experiments

5.1 Setup

Models and baselines. The target model is Qwen3-8B with thinking mode disabled. We compare JetSpec++ with DFlash (Chen et al., 2026), DSpark (Cheng et al., 2026), Domino (Huang et al., 2026), JetSpec (Hu et al., 2026a) and SSD (Kumar et al., 2026), the last with its Qwen3-0.6B draft model. All draft models except SSD's are trained on the same 800k sequences regenerated by the target (Appendix F). The draft models are trained with a block size of 16, and during inference we test the checkpoints with both budget 16 and 32. DSpark runs without its confidence-based scheduling, so every method verifies a block of fixed size.

Benchmarks. We evaluate on GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021), AIME25, HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), LiveCodeBench (LCB) (Jain et al., 2025) and MT-Bench (Zheng et al., 2023), with the first 128 prompts of a fixed shuffle, all 30 AIME25 problems and both turns of the 80 MT-Bench questions (Appendix F).

Evaluation protocol. Every method runs on the same serving engine at batch size 1 and generates at most 2048 tokens, stopping at EOS or at the onset of a repetition loop (Appendix F). Table 1 runs in eager mode on NVIDIA A100 GPUs, and Table 2 on NVIDIA GB200 GPUs with CUDA graph optimization for better tokens per second (TPS) throughput. On a miss, JetSpec++ drafts a fallback block from the verified context in one pass, as DFlash and DSpark consistently draft every block (Section 4.1). We report both greedy decoding and temperature 1 with rejection sampling (Appendix D.1). Speedup is measured in TPS, prefill included, relative to AR on the same GPU, and τ is the mean number of tokens committed per verification round.

Table 1: Speedup over autoregressive decoding and acceptance length τ at verification budgets 16 and 32, in the setup of Section 5.1. JetSpec++ runs J_3 with two adapter refreshes. GPUs gives the placement of Section 4.1: with one, the draft model of JetSpec++ shares the target's GPU, and with two it runs on its own GPU (Appendix E).

			GSM8K		MATH-500		AIME25		HumanEval		MBPP		LCB		MT-Bench	
Method	GPUs	Budget	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Temperature = 0																
DFlash	1	16	4.93	5.91	6.17	7.59	5.82	6.89	4.16	5.16	4.10	4.91	4.47	5.45	2.71	3.24
		32	4.16	5.18	5.18	6.67	4.89	6.07	3.50	4.53	3.50	4.31	3.62	4.70	2.43	2.95
DSpark	1	16	5.95	7.31	7.05	9.01	6.85	8.35	5.11	6.57	5.07	6.17	5.40	6.92	3.15	**3.89**
		32	5.27	7.06	6.33	8.66	5.80	7.73	4.51	6.25	4.40	5.85	4.70	6.38	2.93	3.76
Domino	1	16	5.27	6.91	6.53	8.74	6.26	8.09	4.46	6.06	4.54	5.90	4.64	6.27	2.89	3.74
		32	4.24	6.25	5.33	7.96	5.13	7.37	3.66	5.61	3.68	5.29	3.98	5.92	2.39	3.49
JetSpec	1	16	4.69	5.90	5.88	7.43	5.51	6.78	4.11	5.28	3.96	4.92	4.17	5.45	2.52	3.16
		32	4.66	6.02	5.94	7.81	5.52	7.02	4.02	5.34	3.95	4.96	4.15	5.53	2.54	3.19
JetSpec++	1	16	6.56	7.49	7.47	9.10	6.98	8.05	5.40	6.61	5.37	6.15	5.82	6.93	3.22	3.81
		32	7.19	8.28	9.06	10.64	8.24	9.42	5.61	7.01	5.53	**6.45**	6.22	**7.49**	3.25	3.88
SSD	2	5	0.97	4.06	0.92	4.19	0.95	4.05	0.89	3.82	0.82	3.59	0.88	3.81	0.64	3.01
JetSpec++	2	16	6.83	7.48	8.04	9.09	7.17	8.00	5.82	6.62	5.50	6.15	5.99	6.98	3.45	3.81
		32	**7.49**	**8.34**	**9.74**	**10.79**	**8.90**	**9.55**	**6.16**	**7.04**	**5.89**	6.41	**6.44**	7.48	**3.55**	3.88
Temperature = 1																
DFlash	1	16	4.22	5.19	4.40	5.37	3.49	4.25	3.41	4.26	3.45	4.24	3.85	4.78	2.28	2.80
		32	3.75	4.70	3.81	4.85	3.27	4.16	3.10	3.97	3.01	3.81	3.27	4.20	2.14	2.64
DSpark	1	16	4.85	6.10	4.88	6.13	3.68	4.69	4.11	5.32	4.11	5.18	4.69	6.05	2.60	**3.26**
		32	4.46	5.99	4.48	6.03	3.49	4.67	3.74	5.08	3.67	4.91	4.04	5.54	2.38	3.17
Domino	1	16	4.38	5.86	4.37	5.94	3.50	4.79	3.67	4.97	3.69	4.96	4.02	5.50	2.39	3.15
		32	3.64	5.47	3.86	5.87	3.12	4.62	3.08	4.68	2.97	4.47	3.37	5.27	1.97	2.95
JetSpec	1	16	4.07	5.24	4.27	5.45	3.43	4.30	3.39	4.39	3.48	4.41	3.71	4.95	2.19	2.80
		32	4.11	5.34	4.34	5.62	3.32	4.30	3.43	4.47	3.28	4.29	3.55	4.66	2.17	2.76
JetSpec++	1	16	4.86	6.00	4.63	6.28	3.59	4.81	4.14	5.44	3.97	5.04	4.64	5.84	2.40	3.17
		32	5.54	**6.85**	4.90	6.73	3.72	**4.98**	4.10	5.46	4.16	**5.34**	4.80	6.19	2.37	3.17
SSD	2	5	0.89	3.82	0.78	3.72	0.71	3.31	0.79	3.50	0.73	3.32	0.83	3.69	0.53	2.69
JetSpec++	2	16	5.25	6.21	5.19	6.22	3.95	4.79	4.34	5.11	4.21	5.08	4.90	5.94	**2.74**	3.19
		32	**5.70**	6.68	**5.77**	**6.82**	**4.11**	4.94	**4.57**	**5.55**	**4.47**	5.24	**5.18**	**6.30**	2.70	3.19

5.2 Results

At the same number of passes, the draft model keeps 95 to 98% of the τ it reaches from the target's true states (Appendix B), and each adapter refresh narrows the offline gap to DSpark, from 0.52 tokens at J_2 without refresh to a lead of 0.10 tokens at J_3 with two refreshes (Appendix F). In Table 1, at temperature 0 and budget 16, JetSpec++ outperforms DSpark by 2–10% with similar τ, despite DSpark drafting from true verified-context states. At budget 32, the lead grows to 11–43%: DFlash, DSpark, and Domino lose τ beyond their training block length, while the causal draft models JetSpec and JetSpec++ continue to improve, consistent with JetSpec's reported scaling. Using each method's faster budget, JetSpec++ leads by up to 29% on one GPU and 38% on two GPUs, where it reaches 9.74× over AR on MATH-500; at temperature 1, the gains narrow to 14% and 18%, respectively. The verification outcome is then sampled, so it is pre-drafted less often (Table 13), and each miss drafts a block on the critical path.

5.3 System Performance

Table 2 measures how much drafting latency JetSpec++ removes from each draft-verification round and how that translates into end-to-end speedup. We evaluate all methods under the same CUDA-graph serving setup (Appendix G), closer to production inference. At J_2, JetSpec++ is 1.41× faster than DSpark on GSM8K and 1.37× faster on MATH-500, despite slightly lower τ (6.68 vs. 7.26 and 8.44 vs. 9.05), showing that the gain comes from hiding drafting latency. For an 8B target, graphed verification takes about 5.5 ms, enough to hide two refinement passes but not an adapter refresh. DSpark adds 3.70 and 3.74 ms of drafting per round, whereas JetSpec++ exposes only 0.89 and 1.15 ms, with the realized outcome already pre-drafted in 94.2% and 92.4% of rounds. SSD, designed

for larger targets, reaches only 1.48× and 1.56×: its autoregressive draft model outlasts verification by 8.32 and 8.21 ms per round, and each round commits at most five tokens (τ of 4.07 and 4.20). As target size grows, verification becomes more expensive relative to drafting, creating a larger latency window for additional refinement passes and adapter refreshes. We therefore expect JetSpec++ to benefit even more in larger-model settings such as targets of 32B and 70B parameters.

Table 2: Throughput and round time on GSM8K and MATH-500 at temperature 0 and budget 16, on NVIDIA GB200 GPUs with CUDA graphs (Section 5.1). JetSpec++ runs without adapter refresh. Speedup is over AR, round and exposed (its part beyond verification) are in ms, and hit is the share of rounds whose outcome was pre-drafted (Appendix G).

		GSM8K				MATH-500			
Method	GPUs	tok/s (speedup)	τ	round (exposed)	cache hit	tok/s (speedup)	τ	round (exposed)	cache hit
AR	1	190.9 (1.00×)	1.00	5.20	N/A	186.1 (1.00×)	1.00	5.32	N/A
DSpark	1	746.9 (3.91×)	7.26	9.68 (3.70)	N/A	920.4 (4.95×)	9.05	9.76 (3.74)	N/A
DFlash	1	633.3 (3.32×)	5.87	9.23 (3.35)	N/A	790.1 (4.25×)	7.61	9.57 (3.44)	N/A
SSD	2	281.5 (1.48×)	4.07	14.09 (8.32)	96.0%	289.2 (1.56×)	4.20	13.05 (8.21)	85.6%
JetSpec++, J_1	2	988.6 (5.22×)	6.42	6.36 (0.79)	90.4%	1232.6 (6.67×)	8.12	6.38 (0.86)	88.6%
JetSpec++, J_2	2	1056.8 (5.54×)	6.68	6.24 (0.89)	94.2%	1249.7 (6.74×)	8.44	6.49 (1.15)	92.4%

5.4 Ablation Studies

Table 3 (Appendix A) isolates the draft model's side of JetSpec++ against controls, under the condition in its caption. Appendix B and Appendix C evaluate the adapter and the decoder.

Refinement. Table 3 tests whether the draft model can refine its own draft. A draft model trained on token inputs with every block fully noised drafts as well at J_0 but does not refine, so refinement is learned from the blocks with a clean prefix, which the prefix-length schedule of Appendix F supplies. With this schedule but mask inputs, a draft model keeps only 42% and 55% of the gain from J_0 to J_3. At J_0 the other three draft models differ from ours by at most 0.11 tokens, so the inputs and the schedule decide how much a draft model refines, not how well it drafts in one pass. The final draft models show the same split (Table 6). The released JetSpec draft model, which is trained on mask inputs, falls from 6.03 at J_0 to 2.40 at J_1 on GSM8K and from 7.69 to 2.59 on MATH-500, while the JetSpec++ draft model rises from 5.85 at J_0 to 7.30 at J_∞, run to its fixed point, and from 7.72 to 9.24, with the first pass taking 67% and 69% of this gain.

Training loss and attention. In Tables 4 and 5 (Appendix A), forward KL leads SFT (cross-entropy to the next token of the training sequence) and reverse KL in every cell, by 0.06 to 0.29 tokens over SFT and 1.44 to 2.77 tokens over reverse KL, each by at least 2.5 standard errors of the paired per-prompt difference. A draft model with bidirectional attention within the block matches the causal one at J_0 and trails by 0.06 and 0.17 tokens at J_3, beyond two standard errors only on MATH-500, so causal attention gains only with refinement passes.

Adapter and decoder. At J_0, the adapter's predicted states recover 96% and 98% of the gap between copying the last verified state and the true states (Appendix B), and the training-free decoder stays within 0.15 tokens of a fine-tuned one (Appendix C).

5.5 Limitations

Our serving integration can be further improved from systems optimization. In particular, draft-side CUDA graphs, fused kernels, and optimized communication could reduce drafting overhead and enable additional refinement passes and adapter refreshes to further improving throughput.

6 Conclusion

We presented JetSpec++, which drafts the next block while the current one is verified. Foresight Drafting predicts the target states and the verification outcome before verification returns. We train the causal parallel draft model on blocks with a clean prefix and a noisy suffix, so that self-refinement can spend the freed draft-side time on Jacobi passes. On GB200 GPUs, JetSpec++ decodes up to 6.74× faster than AR decoding and 1.41× faster than DSpark.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on technical formulations and qualitative data analysis, we also used AI to help with method implementations. We have not used generative AI tools to generate synthetic datasets, develop theoretical models, assist in writing proofs, propose hypotheses, translate content, clean or reformat datasets, or interpret experimental results. Additionally, we used generative AI tools to refine manuscript text for clarity and readability, suggest improvements to presentation and structure, and assist with literature search and verification. We reviewed all AI-assisted work and manually verified technical claims, citations, and suggested revisions against the original sources and our own experiments. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the model architecture, training objectives, inference procedure, and system configuration throughout the main text and appendices. All datasets and benchmarks used in our experiments are publicly available. We document the training hyperparameters, evaluation, and hardware settings needed to reproduce our results. We additionally report ablations for the draft model, adapter, decoder, outcome selection, and refinement procedure to isolate the contribution of each component. In addition, we provide the experimental code in the supplementary material.

ETHICS STATEMENT

All authors have read and adhere to the ICLR Code of Ethics. This work does not involve human subjects, sensitive personal data, or experiments with the potential to cause harm. No confidential or proprietary data were used. The methods and experiments were conducted in accordance with principles of research integrity, fairness, and transparency. All conclusions are the sole responsibility of the authors.

REFERENCES

Zihao An, Taichi Liu, Ziqiong Liu, Dong Li, Ruofeng Liu, and Emad Barsoum. PARD-2: Target-aligned parallel draft model for dual-mode speculative decoding. arXiv preprint arXiv:2605.08632, 2026. URL http://arxiv.org/abs/2605.08632v1.

Anastasios N. Angelopoulos and Stephen Bates. Conformal prediction: A gentle introduction. *Foundations and Trends® in Machine Learning*, 16(4):494–591, 2023. doi: 10.1561/2200000101.

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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. arXiv preprint arXiv:2108.07732, 2021. URL http://arxiv.org/abs/2108.07732v1.

Nikhil Bhendawade, Kumari Nishu, Arnav Kundu, Chris Bartels, Minsik Cho, and Irina Belousova. Mirror speculative decoding: Breaking the serial barrier in LLM inference. arXiv preprint arXiv:2510.13161, 2025. URL http://arxiv.org/abs/2510.13161v2.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

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 http://arxiv.org/abs/2302.01318v1.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Oz335dV48X.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021. URL http://arxiv.org/abs/2107.03374v2.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*, volume 37, pp. 129531–129563, 2024. doi: 10.52202/079017-4116. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ea1f5f0878d43ff4fb8bf64ef4a2326c-Paper-Conference.pdf.

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 http://arxiv.org/abs/2607.05147v1.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021. URL http://arxiv.org/abs/2110.14168v2.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079, 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

Raghavv Goel, Mukul Gagrani, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. Direct alignment of draft model for speculative decoding with chat-fine-tuned LLMs. arXiv preprint arXiv:2403.00858, 2024. URL http://arxiv.org/abs/2403.00858v4.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Lanxiang Hu, Zhaoxiang Feng, Yulun Wu, Haoran Yuan, Yujie Zhao, Yu-Yang Qian, Bojun Wang, Peng Zhao, Daxin Jiang, Yibo Zhu, Tajana Rosing, and Hao Zhang. JetSpec: Breaking the scaling ceiling of speculative decoding with parallel tree drafting. arXiv preprint arXiv:2606.18394, 2026a. URL http://arxiv.org/abs/2606.18394v3.

Lanxiang Hu, Siqi Kou, Yichao Fu, Samyam Rajbhandari, Tajana Rosing, Yuxiong He, Zhijie Deng, and Hao Zhang. Fast and accurate causal parallel decoding using Jacobi forcing. In *Forty-third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=ORy2hyPd4G.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. SAM decoding: Speculative decoding via suffix automaton. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12187–12204, 2025. doi: 10.18653/v1/2025.acl-long.595. URL https://aclanthology.org/2025.acl-long.595/.

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 http://arxiv.org/abs/2605.29707v1.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. CLLMs: Consistency large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 25426–25440, 2024. URL https://openreview.net/forum?id=8uzBOVmh8H.

Tanishq Kumar, Tri Dao, and Avner May. Speculative speculative decoding. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/1b96f01343ff10150e6719eb163e1536-Paper-Conference.pdf.

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, 2024a. URL https://proceedings.mlr.press/v235/li24bt.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, 2024b. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

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 38*, pp. 151568–151587, 2025. doi: 10.52202/085713-4562.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 31131–31146, 2024. URL https://proceedings.mlr.press/v235/liu24y.html.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024. doi: 10.1145/3620666.3651335.

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. In *Advances in Neural Information Processing Systems*, volume 38, pp. 50608–50646, 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/48b383b24230e0e6e649d9c98dae4d8c-Paper-Conference.pdf.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. SuffixDecoding: Extreme speculative decoding for emerging AI applications. In *Advances in Neural Information Processing Systems*, volume 38, pp. 126326–126354, 2025. doi: 10.52202/085713-4211. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/b7aea253ab34a773967f1e4cdea9e4fb-Paper-Conference.pdf.

Yu-Yang Qian, Junda Su, Lanxiang Hu, Peiyuan Zhang, Zhijie Deng, Peng Zhao, and Hao Zhang. d3LLM: Ultra-fast diffusion LLM using pseudo-trajectory distillation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=rzBAQT2Fkg.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184, 2024. doi: 10.52202/079017-4135. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/eb0b13cc515724ab8015bc978fdde0ad-Paper-Conference.pdf.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. Accelerating transformer inference for translation via parallel decoding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12336–12355, 2023. doi: 10.18653/v1/2023.acl-long.689. URL https://aclanthology.org/2023.acl-long.689/.

Apoorv Saxena. Prompt lookup decoding. GitHub repository, November 2023. URL https://github.com/apoorvumang/prompt-lookup-decoding. Accessed 2026-09-22.

Yuhao Shen, Junyi Shen, Quan Kong, Tianyu Liu, Yao Lu, and Cong Wang. SpecBranch: Speculative decoding via hybrid drafting and rollback-aware branch parallelism. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/67089958e98b243d5cc1881ad60418b8-Paper-Conference.pdf.

Shwetha Somasundaram, Anirudh Phukan, and Apoorv Saxena. PLD+: Accelerating LLM inference by leveraging language model artifacts. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 6090–6104, 2025. doi: 10.18653/v1/2025.findings-naacl.338. URL https://aclanthology.org/2025.findings-naacl.338/.

Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. Accelerating feedforward computation via parallel nonlinear equation solving. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9791–9800, 2021. URL https://proceedings.mlr.press/v139/song21a.html.

Yeshwanth Venkatesha, Souvik Kundu, and Priyadarshini Panda. Fast and cost-effective speculative edge-cloud decoding with early exits. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=PTIUjARnbc.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, Kai Yu, and Zhijie Deng. Diffusion LLMs can do Faster-Than-AR inference via discrete diffusion forcing. In *International Conference on Learning Representations*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/04bb76a98d9f32226b3beba7bd26a51f-Paper-Conference.pdf.

Zheng Wang, Davis Wertheimer, Yu Chin Fabian Lim, Mudhakar Srivatsa, Raghu K. Ganti, Minjia Zhang, and Naigang Wang. xPress: Parallel refinement for diffusion drafters in speculative decoding. arXiv preprint arXiv:2608.02438, 2026b. URL http://arxiv.org/abs/2608.02438v1.

Zheng Wang, Zhifan Ye, Qi Cheng, Yonggan Fu, Ziyan Wang, Feng Zhu, Haozhe Zhao, Jan Kautz, Pavlo Molchanov, Humphrey Shi, and Minjia Zhang. PRESTO: Prefix-aligned tree drafting for diffusion speculative decoding. arXiv preprint arXiv:2607.22634, 2026c. URL http://arxiv.org/abs/2607.22634v1.

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://proceedings.iclr.cc/paper_files/paper/2026/file/d0865cbe51d35ec322f9af9db7806fc7-Paper-Conference.pdf.

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://proceedings.iclr.cc/paper_files/paper/2026/file/5d8d4e6061c3ba96c240b7fa1ae3471d-Paper-Conference.pdf.

Zilin Xiao, Hongming Zhang, Tao Ge, Siru Ouyang, Vicente Ordonez, and Dong Yu. ParallelSpec: Parallel drafter for efficient speculative decoding. arXiv preprint arXiv:2410.05589, 2024. URL http://arxiv.org/abs/2410.05589v1.

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. doi: 10.48550/arXiv.2508.15487. URL https://arxiv.org/abs/2508.15487.

Yijiong Yu, Huazheng Wang, Shuai Yuan, Ruilong Ren, and Ji Pei. Speculative pipeline decoding: Higher-accuracy drafting with hidden latency via pipeline parallelism. arXiv preprint arXiv:2605.30852, 2026. URL http://arxiv.org/abs/2605.30852v3.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/575286a73f238b6516ce0467d67eadb2-Paper-Conference.pdf.

Liyuan Zhang, Jiarui Zhang, Jinwei Yao, Ran Yan, Yuchen Yang, Jiahao Zhang, Tongkai Yang, Yi Wu, and Binhang Yuan. D^2SD: Accelerating speculative decoding with dual diffusion draft models. arXiv preprint arXiv:2606.04446, 2026a. URL http://arxiv.org/abs/2606.04446v1.

Ziyi Zhang, Ziheng Jiang, Chengquan Jiang, Menghan Yu, Size Zheng, Haibin Lin, Xin Liu, and Henry Hoffmann. SwiftSpec: Disaggregated speculative decoding and fused kernels for low-latency LLM inference. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 2197–2211, 2026b. doi: 10.1145/3779212.3790246.

Hong-Kai Zheng and Piji Li. DeLS-Spec: Decoupled long-short contexts for parallel speculative drafting. arXiv preprint arXiv:2607.07409, 2026. URL http://arxiv.org/abs/2607.07409v1.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

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://proceedings.iclr.cc/paper_files/paper/2024/file/8766fbc68e1ed1cdef712ce273e0a363-Paper-Conference.pdf.

A SELF REFINEMENT

Table 3 isolates refinement for the ablation draft model (Section 5.4), and Table 6 repeats it for the final draft models, at temperature 0 and 1.

Table 3: Can the draft model refine its own draft? Acceptance length τ over refinement passes at temperature 0 for the two-epoch ablation draft model, trained with token or mask inputs, and on blocks with a clean prefix drawn by the prefix-length schedule of Appendix F or on fully noised blocks only. The target's true states, 164 prompts per benchmark, 256 tokens. Paired per prompt, every other row trails ours at J_1 and J_3 by at least 9 standard errors.

		GSM8K			MATH-500		
Inputs	Training blocks	J_0	J_1	J_3	J_0	J_1	J_3
token (ours)	clean prefix (ours)	4.42	4.97	5.14	5.97	6.82	7.17
	fully noised	4.44	4.44	4.44	6.08	6.07	6.07
mask	clean prefix	4.45	4.66	4.76	5.97	6.35	6.63
	fully noised	4.41	1.23	1.24	6.04	1.28	1.29

Loss ablation. The three arms of Table 4 differ only in the objective of Eq. 9 (Appendix F). The ordering is consistent with what each objective asks of the draft model. Forward KL matches the target's full next-token distribution at every position, including the positions that follow a noisy suffix, so the draft model learns how the target's prediction shifts once the tokens before a position change, which is the signal a refinement pass uses. SFT supervises only the single next token of the training sequence, which suffices for a one-pass draft but carries less of this information, so its gap is larger after refinement passes. Reverse KL is mode-seeking. Where the draft model cannot represent the target's distribution, it commits to one mode, which need not contain the target's greedy token, and the objective places no penalty on the target mass that the draft model drops, so later passes have little signal to correct toward.

Table 4: Training loss. τ in the setting of Table 3. Paired per prompt, forward KL leads SFT by at least 2.5 standard errors and reverse KL by at least 24 in every cell.

	GSM8K			MATH-500		
Loss	J_0	J_1	J_3	J_0	J_1	J_3
forward KL (ours)	4.42	4.97	5.14	5.97	6.82	7.17
SFT	4.35	4.82	4.96	5.86	6.65	6.88
reverse KL	2.98	3.10	3.13	4.06	4.33	4.39

Attention ablation. Table 5 changes only the attention within the draft block. At J_0 every block position holds a token drawn from the verified context, so attending to later positions adds little information, and the two masks draft equally well. In a refinement pass, a causal position reads only the refined tokens before it, which matches the target's left-to-right factorization, whereas a bidirectional position also reads the unrefined tokens after it, which are the least reliable part of the block. The gain is larger on MATH-500, where the gap grows from 0.11 tokens at J_1 to 0.17 at J_3, while it stays near 0.06 on GSM8K. MATH-500 also has longer accepted prefixes (τ of 7.17 against 5.14 at J_3), so more of the accepted positions depend on a refined prefix.

Table 5: Attention within the draft block. τ in the setting of Table 3. Paired per prompt, the causal draft model leads by more than two standard errors only on MATH-500 at J_1 and J_3.

	GSM8K			MATH-500		
Attention	J_0	J_1	J_3	J_0	J_1	J_3
causal (ours)	4.42	4.97	5.14	5.97	6.82	7.17
bidirectional	4.43	4.90	5.08	5.94	6.71	7.00

Table 6: Acceptance length τ over refinement passes for the final draft models at temperature 0 and 1: the target's true states, 2048 tokens, 32 prompts per benchmark. The released JetSpec draft model is trained on mask inputs. J_∞ runs at most 16 passes and stops early once the draft stops changing. It is not run for the released JetSpec draft model (—).

	GSM8K				MATH-500			
Draft model	J_0	J_1	J_3	J_∞	J_0	J_1	J_3	J_∞
Temperature 0								
JetSpec (released)	6.03	2.40	2.44	—	7.69	2.59	2.64	—
JetSpec++	5.85	6.82	7.24	7.30	7.72	8.76	9.17	9.24
Temperature 1								
JetSpec (released)	5.21	2.31	2.31	—	5.90	2.35	2.39	—
JetSpec++	5.19	5.90	6.02	6.13	5.79	6.35	6.36	6.52

B Adapter Ablations

Table 7 measures the adapter with the final draft model. At J_2 without refresh, drafting from the adapter's predicted states keeps 95 to 98% of the acceptance length that the same draft model reaches from the target's true states, the ceiling of this recovery.

Table 7: Offline τ of the final draft model from the target's true states and from the adapter's predicted states, at budget 16, temperature 0 and no refresh, on the prompts of Table 12. The last row divides the τ from predicted states by the τ from true states at the same J. The predicted row is the refresh-0, budget-16 row of Table 12.

states	GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench
true, J_0	5.70	7.52	7.14	5.09	4.81	6.11	3.09
true, J_2	6.88	8.86	8.47	6.12	5.71	7.12	3.48
predicted, J_2	6.67	8.52	8.07	5.90	5.59	6.84	3.34
predicted ÷ true, J_2	0.970	0.961	0.953	0.965	0.980	0.961	0.960

Table 8 holds the draft model fixed and varies the source of the target states at positions the target has not yet verified. All reads use the two-epoch ablation draft model under the condition of Table 3 (Appendix A), whose row for our draft model is the ceiling here. The adapter has the twelve blocks and the loss of Appendix E and is trained for two epochs on the same 239k mathematics sequences. No refresh runs in any row.

Table 8: Acceptance length τ for each source of the missing target states: the floor copies the last verified state into every new position, the adapter predicts the states, and the ceiling reads the target's true states, which exist only after verification. Paired per prompt and before rounding, the ceiling leads the adapter by 0.05 to 0.19 tokens, with a 95% bootstrap interval that excludes zero in every cell.

	GSM8K			MATH-500		
States for new positions	J_0	J_1	J_3	J_0	J_1	J_3
floor	2.53	2.84	2.98	3.18	3.65	3.87
adapter	4.35	4.85	5.04	5.91	6.65	6.97
ceiling	4.42	4.97	5.14	5.97	6.82	7.17

The adapter closes 96% and 98% of the gap in τ between the floor and the ceiling at J_0 on GSM8K and MATH-500, and 94 to 98% over all six cells.

C Decoder Ablations

We compare the decoder of Appendix E, the target's upper layers reused without training (the *training-free decoder*), with two fine-tuned copies. The *CE-trained decoder* is fine-tuned with cross-entropy to the target's greedy token on the predictions of the adapter evaluated here, and the *KL-trained decoder* with KL divergence to the target's full next-token distribution on the predictions of a different adapter. All measurements use the two-epoch draft model and adapter of Appendix B, so we evaluate on GSM8K and MATH-500.

C.1 Calibration Procedure

The served rule has three fitted parts: the six coefficients of Eq. 11, the anchor temperature β, and the thresholds. There is one threshold for each of the first two draft passes and one shared by the third and later passes. Split conformal prediction then sets each threshold so that the realized outcome is cached on 93% of the calibration rounds that the threshold serves. A threshold with no calibration rounds takes the value pooled over all passes.

We capture the calibration rounds by serving a fixed pooled set of 56 prompts. GSM8K, MATH-500, HumanEval, MBPP and LiveCodeBench each contribute 8 prompts, drawn after the prompts that the benchmark uses for evaluation. UltraChat and the GSM8K training split contribute 8 prompts each in place of MT-Bench and AIME25, whose prompts are all used for evaluation. One fit per block size serves every benchmark.

Every cached block is verified by the target like any other draft, so calibration changes the hit rate and never the verified output.

C.2 Reused Versus Fine-Tuned Decoder

Table 9 reports top-1 agreement, the fraction of block positions where the decoder's most probable token equals the target's greedy token.

Table 9: Offline replay. Agreement of the decoder with the target's greedy token on 200 held-out math sequences (379,305 positions), using the two-epoch ablation adapter under its serving attention mask. Intervals are 95% sequence-clustered intervals. KL is the divergence from the target's next-token distribution to the decoder's.

decoder	top-1 agreement	KL to target
training-free (reused)	.9666 [.9645, .9686]	.043
CE-trained, 6,000 updates	.9678 [.9656, .9699]	.247
KL-trained	.9582 [.9555, .9607]	.046

Fine-tuning does not raise top-1 agreement beyond the 95% intervals of Table 9, and the reused layers need no retraining when the adapter changes.

Table 10 serves the three decoders with everything else fixed, each with the calibration and thresholds of the calibration procedure above refit on its own calibration rounds.

The training-free decoder matches the CE-trained decoder in joint hit and τ in every setting except MATH-500 with CUDA graphs, the only setting whose intervals exclude zero. There the cap of 4 outcomes truncates its larger cache, and τ drops by 0.15 tokens per round (2.2%).

Table 10: Served comparison of the decoders with the two-epoch draft model and adapter of Appendix B, 32 evaluation prompts per benchmark, 256 generated tokens each, temperature 0. Joint hit and τ are measured on the rounds with a cache, excluding the first round of each request and each round that follows a miss. Joint hit is the fraction of these rounds whose realized outcome is cached. Cached is the mean number of cached outcomes per round, at most 16 in eager mode and at most 4 with CUDA graphs. The difference rows give CE-trained minus training-free with prompt-clustered 95% bootstrap intervals.

	GSM8K			MATH-500		
Decoder	joint hit	cached	τ	joint hit	cached	τ
Eager, cache of at most 16 outcomes						
training-free	.941	3.89	4.820	.939	6.16	6.735
CE-trained	.934	2.79	4.828	.934	2.81	6.776
KL-trained	.942	4.27	4.816	.945	5.41	6.754
difference	−.008 [−.017, +.001]		+.007 [−.031, +.047]	−.005 [−.017, +.009]		+.041 [−.020, +.111]
CUDA graphs, cache of at most 4 outcomes						
training-free	.905	2.92	4.791	.833	3.46	6.653
CE-trained	.916	2.51	4.826	.921	2.57	6.805
KL-trained	.888	3.06	4.817	.867	3.37	6.693
difference	+.011 [−.003, +.027]		+.035 [−.005, +.076]	+.088 [+.061, +.117]		+.152 [+.052, +.253]

C.3 Selection Rules and the Cache Size

The served rule scores each outcome (k, a) by the calibrated score of Eq. 11. Table 11 compares it with rules that each remove one source of information. The uncalibrated rule uses the decoder's $P(k, a)$ of Eq. 6. The draft-model rule scores k by the product of the draft model's confidences c_i (Appendix E). The no-covariate rule uses the frequency of each k on the calibration rounds, the same in every round. The static grid caches a fixed number of the most probable values of k and, for each, a fixed number of the most probable anchors, chosen on the calibration rounds within the cache size B of Eq. 7. These rules rank anchors by the decoder. The geometric fan-out is the cache shape of SSD in Proposition 2(a), with α and ρ of Assumption 2 fitted on the calibration rounds and the F_k rounded to integers that sum to at most B. We read it with the draft model's anchors, as in SSD, and with the decoder's anchors.

Table 11: Offline replay. Joint hit at a cache size of B outcomes per round, on the eager evaluation rounds of Table 10. Each cell gives training-free / CE-trained decoder.

	GSM8K			MATH-500		
rule	B = 4	B = 16	B = 64	B = 4	B = 16	B = 64
calibrated score (served, no threshold)	.910 / .934	.984 / .983	.992 / .991	.835 / .938	.967 / .992	.995 / .997
decoder, uncalibrated, Eq. 6	.878 / .878	.970 / .951	.987 / .979	.719 / .906	.888 / .975	.962 / .992
draft model k	.671 / .710	.947 / .942	.989 / .982	.695 / .757	.934 / .959	.992 / .993
no covariate	.574 / .627	.887 / .901	.984 / .975	.511 / .589	.844 / .911	.987 / .992
static grid	.834 / .889	.957 / .969	.985 / .984	.759 / .894	.929 / .972	.991 / .994
geometric fan-out, draft model anchors	.259 / .259	.529 / .531	.781 / .782	.279 / .273	.682 / .680	.912 / .912
geometric fan-out, decoder anchors	.513 / .531	.850 / .872	.979 / .982	.373 / .414	.798 / .877	.973 / .987

The calibrated score leads every other rule in every cell, by the widest margin at $B = 4$, and the geometric fan-out, whose shape is fixed before the round, trails it at every cache size, further with the draft model's own anchors.

D Theory of Foresight Drafting

D.1 Exactness of the Outcome Distribution under Speculative Sampling

Figure 3 shows how one decoder pass supplies every factor of Eq. 6.

Assumption 1 (Exact decoder)**.** *The draft tokens $d_1, \ldots, d_\gamma$ of the block are a deterministic function of the verified context x, as for the argmax draft of Eq. 8 and for drafts from predicted states. Verification uses the acceptance rule of Section 3, and $\hat{p}(\cdot \mid j) = p(\cdot \mid x, d_{\le j})$, the target's next-token distribution after prefix j at the sampling temperature, which is the point mass on the argmax at temperature* 0*.*

Proposition 1 (Exact outcome distribution)**.** *Under Assumption 1, at any temperature, let $o^\star = (k^\star, a^\star)$ be the realized outcome of the block, where $k^\star$ is the number of accepted draft tokens and $a^\star$ is the committed bonus token. Then $\Pr[o^\star = (k, a) \mid x] = P(k, a)$ of Eq. 6 for every outcome (k, a), with $a \neq d_{k+1}$ when $k < \gamma$.*

Proof. Write $p_i(\cdot) := p(\cdot \mid x, d_{<i})$ for the target distribution at draft position i, and $\hat{p}_i := \hat{p}(d_i \mid i-1)$. Then

$$\Pr[\text{accept } d_i] = \min\{1, p_i(d_i)/1\} = p_i(d_i).$$

Given the accepted prefix, each acceptance test uses a fresh draw, so the acceptance probabilities of successive positions multiply. For $k < \gamma$ with positive rejection probability, rejection at $k+1$ has probability $1 - p_{k+1}(d_{k+1})$, and the residual is $\max\{p_{k+1}(\cdot) - \delta_{d_{k+1}}(\cdot), 0\}/[1 - p_{k+1}(d_{k+1})]$, where $\delta_{d_{k+1}}$ is the point mass at d_{k+1}. Thus, for $a \neq d_{k+1}$,

$$\Pr[o^\star = (k, a) \mid x] = \Big(\prod_{i=1}^{k} p_i(d_i)\Big)[1 - p_{k+1}(d_{k+1})]\frac{p_{k+1}(a)}{1 - p_{k+1}(d_{k+1})} = \Big(\prod_{i=1}^{k} \hat{p}_i\Big)\hat{p}(a \mid k).$$

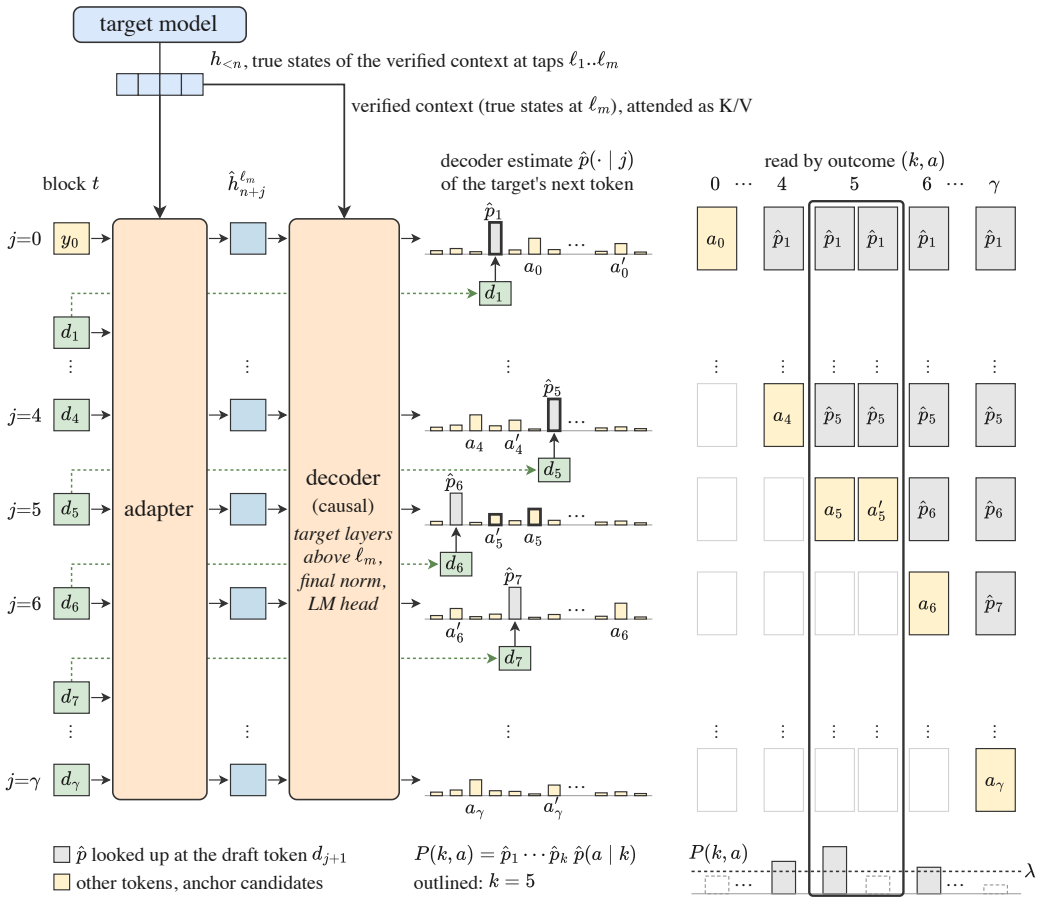


Figure 3: **One decoder pass scores every outcome.** The adapter predicts the block's states from the verified context, and the decoder reads them with the verified context as keys and values. Row j is the decoder's $\hat{p}(\cdot \mid j)$. For the outcomes with $k = 5$ (outlined), rows 0 to 4 supply the acceptance probabilities $\hat{p}_1, \ldots, \hat{p}_5$ of the draft tokens, row 5 supplies the bonus distribution with d_6 excluded, and later rows are not read. The bars give the score $P(k, a)$ of Eq. 6 against the threshold λ: solid bars reach λ and are cached, and dashed bars are dropped. Heights are schematic. In serving, the score uses the calibrated factors of Eq. 11 and is capped at B outcomes (Appendix E).

If $p_{k+1}(d_{k+1}) = 1$, then $p_{k+1}(a) = 0$ for every $a \neq d_{k+1}$, and both sides vanish. For $k = \gamma$, the bonus token is drawn directly from $p_{\gamma+1}(\cdot)$, hence

$$\Pr[o^\star = (\gamma, a) \mid x] = \Big(\prod_{i=1}^{\gamma} \hat{p}_i\Big)\hat{p}(a \mid \gamma).$$

Both expressions are Eq. 6. □

With a deterministic draft, the acceptance rule of Section 3 thus accepts d_i with probability $p_i(d_i)$ and otherwise commits a draw from p_i with d_i excluded. This equals drawing $y \sim p_i$, accepting d_i if and only if $y = d_i$, and committing y otherwise, so every committed token is the target's own draw.

D.2 Relation to the Geometric Fan-Out of SSD

SSD (Kumar et al., 2026) caches F_k anchors for accepted length k, under schedules fixed across rounds, whereas Eq. 7 selects a cache anew each round.

Assumption 2 (Rank-space law)**.** *The draft model has constant acceptance rate α, with*

$$P(k) = \begin{cases} \alpha^k(1-\alpha), & k < \gamma, \\ \alpha^\gamma, & k = \gamma. \end{cases}$$

For the continuous optimization, the hit mass of the top F anchors conditional on k, ranked in decreasing order of $P(k,\cdot)$, is modeled by the surrogate curve $1 - F^{-\rho}$ for real $F \geq 1$, for some $\rho > 0$. Both parts follow SSD: $P(k)$ is the acceptance model of its Theorem 12 and the curve the per-length form of its power-law hit rate (its Definition 11), zero at $F = 1$ as there, except that SSD ranks anchors by the draft model's logits.

For round t, let I_t be the features the selection observes, such as the outputs of the decoder and the draft model. Let $\pi_t(o) := \Pr[o^\star = o \mid I_t]$, with $\pi_t(S) := \sum_{o \in S} \pi_t(o)$, and let $S_t^\star$ solve Eq. 7 with π_t in place of P.

Proposition 2 (Adaptive selection contains the geometric fan-out)**.**

(a) Under Assumption 2, let the relaxed problem maximize $\sum_k P(k)\,(1 - F_k^{-\rho})$ over $F_k \in \{0\} \cup [1,\infty)$ with $\sum_k F_k \leq B$ for the cache size B of Eq. 7, where the hit mass at length k is 0 when $F_k = 0$. At the maximizer, each active length k, one with $F_k \geq 1$, receives

$$F_k = F_0 \alpha^{k/(1+\rho)} \quad (k < \gamma), \qquad F_\gamma = F_0 \alpha^{\gamma/(1+\rho)} (1-\alpha)^{-1/(1+\rho)},$$

the geometric fan-out of SSD, where F_0 is the scale that makes the active F_k sum to B, equal to the allocation of length 0 when that length is active.

(b) For every round and every I_t-chosen cache S_t with $|S_t| \leq B$, including fixed $\{F_k\}$ with $\sum_k F_k \leq B$,

$$\pi_t(S_t^\star) \geq \pi_t(S_t).$$

If $\pi_t(k, a_{k,s})$, the probability of the s-th ranked anchor $a_{k,s}$ at length k, is the same in every round, then $S_t^\star$ is one fixed allocation and the best fixed allocation ties.

Proof of (a). The marginal of $P(k)(1 - F_k^{-\rho})$ is $P(k)\rho F_k^{-\rho-1}$. Equalizing active marginals at the multiplier μ of the constraint $\sum_k F_k \leq B$ gives

$$P(k)\rho F_k^{-\rho-1} = \mu \quad \Longrightarrow \quad F_k \propto P(k)^{1/(1+\rho)},$$

and hence the stated series. On $F_k \geq 1$ the objective is concave, so for a fixed set of active lengths this condition is sufficient. Lengths with $P(k)\rho < \mu$ receive no cache entry, $F_k = 0$, because $F_k = 1$ adds no hit mass and the marginal $P(k)\,\rho$ at $F_k = 1$ is below μ. SSD derives this condition in its Theorem 18. The restriction $F_k \in \{0\} \cup [1,\infty)$, the concavity argument and the inactive lengths are not part of SSD's statement. □

Proof of (b). Each $S_t = S(I_t)$ is feasible in Eq. 7, so $\pi_t(S_t^\star) \geq \pi_t(S_t)$ pointwise, and the inequality survives expectation over rounds. If $\pi_t(k, a_{k,s})$ does not depend on t, the B largest values select the same ranks (k, s) in every round, and the decreasing order makes them the top F_k ranks at each k, a fixed allocation. □

In serving, the selection scores outcomes by an estimate $\hat{\pi}_t$ of π_t. For any cache $\hat{S}_t$ of at most B outcomes, let $\eta_t := \max_{|S| \leq B} |\pi_t(S) - \hat{\pi}_t(S)|$ and $\epsilon_t := \max_{|S| \leq B} \hat{\pi}_t(S) - \hat{\pi}_t(\hat{S}_t)$. Then $\pi_t(S_t^\star) - \pi_t(\hat{S}_t) \leq 2\eta_t + \epsilon_t$ in every round.

E Implementation Details

Adapter architecture. The adapter predicts the target's states at the five layers that the draft model of Appendix F reads, 1, 9, 17, 25 and 33 of the 36 target layers, indexed from 0. It has twelve transformer blocks, one stream per layer, the module A_ℓ of Eq. 5. The streams for layers 1, 9 and 17 have one block each, initialized from that layer. The stream for layer 25 has four blocks, initialized from layers 19, 21, 23 and 25, and the stream for layer 33 has five, initialized from layers 27, 29, 30, 32 and 33. Each block is a copy of a whole target layer, and the adapter adds no projection or head of its own.

Decoder. The decoder reuses the two target layers above layer 33, layers 34 and 35, together with the target's final norm and LM head, unchanged. It computes the keys and values of the verified context on the draft process, from the layer-33 states that the target sends after each verification. With the final adapter, the decoder agrees with the target's greedy token on 93.5% of held-out positions (95% interval 92.5% to 94.3%), measured on 200 held-out sequences from the training mixture, 100 math and 100 non-math.

Adapter training. The frozen target computes the adapter's regression targets online for each training batch. Let $\mathbf{h}_{n+j}$ be the concatenation of the target's true states h^{ℓ}_{n+j} at the five tap layers, $\hat{\mathbf{h}}_{n+j}$ the adapter's prediction of it, and ω_j the probability that a served round reads block position j. For a block whose anchor is at position n, the loss is

$$\mathcal{L}_{\text{adapter}} \;=\; \sum_{j=0}^{\gamma} \frac{\sqrt{\omega_j}}{\sum_{j'=0}^{\gamma} \sqrt{\omega_{j'}}} \Big[\operatorname{SmoothL1}\big(\hat{\mathbf{h}}_{n+j}, \mathbf{h}_{n+j}\big) + 1 - \cos\big(\hat{\mathbf{h}}_{n+j}, \mathbf{h}_{n+j}\big) \Big], \tag{10}$$

averaged over the blocks of the batch. The Smooth L1 term is averaged over the coordinates of the concatenated vector with threshold 1, and the cosine is taken over the whole concatenated vector. The probability ω_j is 1 at positions 0 and 1 and falls to 0.027 at position 15. Training tiles every training sequence into consecutive blocks of 16 positions, prompt tokens included. The adapter trains on the 800k sequences of the draft model, less 24k held-out evaluation sequences, for six epochs with AdamW, a peak learning rate of 3×10^{-4} with linear warmup over the first 4% of steps and cosine decay, a global batch of 8 sequences of at most 3,072 tokens, and seed 42.

Adapter refresh. The refresh of Section 4.3 stops at the first position i_{stop} where the product of the draft model's top-1 probabilities falls below 0.1, and the predicted states of positions 0 to i_{stop} replace the previous ones in the draft model's context. The refresh emits no tokens.

Calibrated score. Each factor $\hat{p}_i$ of Eq. 6 is replaced by $\tilde{p}_i$, a logistic link on a vector $\boldsymbol{\phi}_i$ of six features,

$$\tilde{p}_i \;=\; \sigma\big(\mathbf{w}^{\top} \boldsymbol{\phi}_i\big), \qquad G(k) = \prod_{i \le k} \tilde{p}_i, \qquad \tilde{P}(k) = G(k) - G(k+1), \quad G(\gamma+1) = 0. \tag{11}$$

The six features are the logit of the draft model's probability c_i of d_i in its final pass, the logit of the decoder's agreement $\hat{p}_i$, an indicator of whether d_i is the decoder's top token, the logit of the decoder's strongest alternative at position $i-1$, the relative position $i/(\gamma-1)$ and a bias.

Each outcome is scored by $\tilde{P}(k)\, P_\beta(a \mid k)$, where $P_\beta(a \mid k) \propto \hat{p}(a \mid k)^{1/\beta}$ is the tempered bonus distribution of outcome k, renormalized over the decoder's top 32 candidates, with d_{k+1} removed when $k < \gamma$. Appendix C fits $\mathbf{w}$ and β on calibration prompts disjoint from the evaluation prompts. In the round that follows a cache miss, only outcomes with $k = 0$ are scored.

Serving configurations. On one GPU, JetSpec++ runs its draft process as a second process on the target's GPU, sharing it through NVIDIA Multi-Process Service (MPS). On two, the draft process has its own GPU, as SSD's does. SSD drafts 4 tokens per round, a block of 5 with the anchor, and pre-drafts up to three verification outcomes (one on MATH-500, its best setting). JetSpec++ caches at most $B = 16$ outcomes per round in Table 1 and $B = 4$ in Table 2, and the calibrated rule decides how many of them a round fills. Each block size, 16 and 32, has its own calibration fit (Appendix C), used by every row at that block size. The two refreshes of Table 1 follow the first and the second pass.

F TRAINING AND EVALUATION DETAILS

Data and schedule. The draft model is trained on 800k sequences whose responses are generated by the target model in non-thinking mode, for six epochs, with learning rate 3×10^{-4}, a cosine schedule with linear warmup over the first 4% of steps, a global batch of 16 sequences and seed 42. Ablations use a separate set of 239k mathematics sequences, also with responses generated by the target, for two epochs with the same learning rate, schedule, global batch and seed.

Blocks. Each training sequence has at most 3,072 tokens. We sample up to 512 anchor positions per sequence. An anchor must be a supervised response token and must leave room for a full block of 16 positions, the anchor and $\gamma = 15$ draft positions. The prefix length m of Eq. 9 is 0, a fully noised block, with probability 0.25, and each of $1, \ldots, 15$ with probability 0.05.

Architecture. The draft model has five transformer layers. It reads the target's hidden states at the five tap layers of Appendix E through a linear fusion of the concatenated states followed by a normalization, and it ends with a final normalization. It reuses the target's embedding and LM head, which stay frozen, and trains 1.05B parameters.

Loss. Eq. 9 is the forward KL divergence at temperature 1, applied to the supervised draft positions and averaged over all of them in a batch, with no decay along the block. The loss ablations of Table 4 (Appendix A) change only this objective: SFT is the cross-entropy to the next token of the training sequence, which the target generated, and reverse KL is $D_{\mathrm{KL}}(q \,\|\, p)$ at temperature 1, where q is the draft model's distribution and p the target's.

Baselines. The DFlash, DSpark, JetSpec and Domino draft models are trained for six epochs on the same 800k sequences as our draft model. JetSpec is the checkpoint released with Hu et al. (2026a), and its settings below are those of that release. The DFlash, DSpark and JetSpec draft models also share our maximum length, anchor count, warmup and seed. DFlash and JetSpec have our five layers and tap layers and fill every draft position of a training block with a mask token. DFlash follows its published recipe without the positional loss decay, attends bidirectionally within the block and is trained with the SFT loss at our learning rate and global batch. JetSpec attends causally within the block and is trained with our loss and learning rate at a global batch of 8. Our DSpark reproduces DSpark with 15 draft tokens after the anchor, its Markov head and its confidence head, and keeps its loss, a weighted sum of 0.1 cross-entropy and 0.9 L1 loss with a positional decay rate of 7, at learning rate 6×10^{-4}. It differs from the published recipe in two settings: six epochs instead of ten, and a global batch of 8 instead of 512. JetSpec reports its main results with at most 3,072 generated tokens, so its rows here are a re-run at our 2048-token cap and differ from the numbers in its paper.

Evaluation. MT-Bench contributes the first turn of its 80 questions and the second turn of 79, since one first-turn answer fills the 2048-token cap and leaves no room for its second turn in a 4096-token context. The second turn of every method follows the same first-turn answer, generated by the target with AR. For each benchmark, τ and throughput are pooled over its generations, as committed tokens over verification rounds and tokens over time. τ counts the bonus token of each round, and the token the target produces at prefill belongs to no round. In the last round of a generation, τ counts only the tokens committed before the end-of-sequence token or the generation cap. Refinement stops early at a fixed point, so J_r runs at most $r + 1$ forward passes.

Serving. In bf16, floating-point non-associativity across verification shapes changes a small fraction of greedy trajectories for every speculative method, so bf16 outputs are not compared across methods. JetSpec++ publishes the last refinement pass that has finished when verification is ready, so at temperature 1 its drafts, and with them the target's random draws, vary between repeated reads with the same seed, and Table 1 reports one timed read.

Repetition cut. At temperature 0, the greedy output of some prompts falls into a verbatim repetition loop that runs to the 2048-token cap, and repeated text drafts almost perfectly. Tables 1 and 2 therefore treat an output that reaches the cap as a repetition loop if its final 64 tokens recur verbatim at least three times in it, and stop it at the first occurrence of that 64-token span. The stop is booked like an EOS stop: tokens are credited up to the onset, the round that crosses it is credited only up to the onset, and time is counted through that round. Every method, AR included, is cut at its own onsets, and the prompt counts are unchanged. The rule applies to each MT-Bench turn separately. Outputs that reach the cap without a repetition loop are kept.

Offline acceptance length. At budget 16 and temperature 0 (Table 12), the pooled paired gap of JetSpec++ to DSpark goes from 0.52 tokens at J_2 without refresh to 0.08 tokens at J_3 with one refresh and to a lead of 0.10 tokens with two.

Table 12: Offline acceptance length τ of chain drafting at verification budgets 16 and 32, at temperature 0 and 1. JetSpec++ drafts from the adapter's predicted states. Refresh counts the adapter's re-predictions between draft-model passes (Section 4.3), and J counts the refinement passes. 128 prompts per benchmark (AIME25 30, MT-Bench 80 first turns), at most 2048 generated tokens. JetSpec++ is the final draft model. A dash marks a setting that does not apply to the method, and the baselines draft in one pass. Served speedup and τ are in Table 1.

Method	Budget	Refresh	J	GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench
Temperature = 0										
DFlash	16	—	—	5.85	7.64	7.27	5.06	4.87	5.90	3.07
	32	—	—	5.19	6.74	6.28	4.58	4.32	5.62	2.79
DSpark	16	—	—	7.30	9.14	8.59	6.37	6.08	7.31	3.70
	32	—	—	7.04	8.80	7.99	6.27	5.85	7.76	3.57
Domino	16	—	—	6.85	8.81	8.39	5.92	5.78	6.75	3.61
JetSpec	16	—	—	5.86	7.51	7.08	5.18	4.89	5.93	3.02
	32	—	—	6.05	7.92	7.34	5.41	4.91	6.79	3.01
JetSpec++	16	0	2	6.67	8.52	8.07	5.90	5.59	6.84	3.34
	32			7.30	9.99	9.17	6.40	5.80	8.42	3.36
	16	1	1	6.79	8.50	8.09	5.98	5.58	7.05	3.41
	32			7.30	9.79	8.92	6.43	5.78	8.54	3.42
	16	1	3	7.28	9.02	8.61	6.37	5.98	7.44	3.55
	32			8.07	10.91	9.90	7.03	6.34	9.31	3.59
	16	2	3	7.54	9.22	8.81	6.58	6.17	7.62	3.63
	32			8.46	11.37	10.22	7.32	6.55	9.59	3.67
Temperature = 1										
DFlash	16	—	—	5.32	5.52	4.39	4.36	4.22	4.91	2.80
	32	—	—	4.54	4.91	4.01	4.07	3.80	4.16	2.52
DSpark	16	—	—	6.28	6.18	5.11	5.26	5.15	6.07	3.25
	32	—	—	6.11	6.09	4.47	5.10	5.06	5.62	3.13
Domino	16	—	—	5.91	6.27	4.47	4.94	4.99	5.57	3.07
JetSpec	16	—	—	5.20	5.43	4.49	4.44	4.23	4.85	2.71
	32	—	—	5.26	5.50	4.40	4.42	4.34	4.77	2.72
JetSpec++	16	0	2	5.84	6.06	4.58	4.97	4.83	5.25	2.94
	32			6.19	6.58	4.98	5.15	5.00	5.31	2.96
	16	1	1	5.93	5.93	4.89	5.00	4.84	5.63	3.00
	32			6.26	6.18	4.97	5.28	4.97	5.86	3.01
	16	1	3	6.10	6.05	4.91	5.35	5.10	6.07	3.15
	32			6.77	6.93	5.17	5.58	5.40	6.26	3.18
	16	2	3	6.42	6.33	4.88	5.48	5.20	5.86	3.19
	32			7.18	7.02	5.25	5.68	5.47	7.10	3.20

Hit rate. Table 13 gives the share of rounds whose verified outcome was pre-drafted, for JetSpec++ at budget 16 on one GPU, from the timed reads of Table 1.

Table 13: Hit rate of JetSpec++ in Table 1 at budget 16 on one GPU: the share of rounds whose verified outcome was pre-drafted.

Temperature	GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench
0	.964	.962	.934	.941	.933	.887	.891
1	.910	.866	.808	.881	.880	.827	.812

G SERVING MEASUREMENTS

Protocol. Table 2 comes from one dedicated GB200 node, on which GSM8K and MATH-500 run in parallel on the two sockets, two GPUs each, with no other process on a socket during a session. Each benchmark is measured in sessions that run methods back to back: one for AR, DSpark and DFlash, one each for JetSpec++ at J_1 and J_2, and one for SSD. Every session also reads AR twice and starts and ends with a DSpark read, and each row's speedup and ratio to DSpark divide by the means of these reads in its own session. The AR, DSpark and DFlash rows show the first session, and each JetSpec++ row is the mean of two reads within its session. A warm-up run compiles every recipe before the first session.

Correctness checks. Each draft round that JetSpec++ replays from a CUDA graph is recomputed eagerly in the same round and compared with the graph's outputs, and in fp32 the output of JetSpec++ is compared with that of AR token by token on 8 prompts. No round differed, and all 8 fp32 outputs matched AR.

Definitions and adapter refresh. Round is the median time per round, exposed is the mean of the round time minus the verification time, so round minus exposed is not the median verification time, and hit rate is the share of rounds whose outcome was pre-drafted. Table 2 runs without adapter refresh, because on GB200 one refresh after the first pass lengthens the draft side to about 10 ms, against a verification of about 5.5 ms. In a separate read on 8 GSM8K prompts, the refresh raises τ at J_2 to 0.96 of DSpark's, while the throughput falls to 0.91 of DSpark's.