HYBRIDSPEC: THROUGHPUT-AWARE SPECULATIVE DECODING FOR HYBRID GDN–MOE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

In speculative decoding, parallel drafters such as DFlash make drafting cheap, so target verification dominates each round. A central choice is the verification length, the number of tokens that the target processes in each round. Verifying more draft tokens can accept more of them, but work after the first rejection is wasted. For hybrid target models that combine Gated DeltaNet (GDN) and mixture-of-experts (MoE) layers, verification time grows much faster with this length than for dense models. In a single-request profile of Qwen3.6-35B-A3B with DFlash, it grew by 71% from length 4 to 16, almost entirely in GDN and MoE layers, compared with 4%–6% for a dense Qwen3-4B target. Each GDN layer must update a recurrent state at every token and recover it at the end of the accepted prefix. The MoE cost depends on the experts that the tokens activate, which are known only once verification runs. We present HybridSpec, which selects the verification length on the GPU from draft-side acceptance estimates and measured target latency. Verification then runs this length plus one extra token up to the first MoE router, before any expert runs. There, a lightweight routing correction uses the observed expert assignments to decide whether to keep that token. The rest of verification runs only on the selected tokens. Block-parallel GDN kernels avoid storing a state after every token and reconstruct the accepted state of every GDN layer in one launch. In isolated benchmarks at verification lengths 4 to 16 with every draft token accepted, these kernels, including accepted-state reconstruction, are $1.54\times$–$4.04\times$ faster than the default kernels. With three GDN–MoE target models in single-request greedy decoding, HybridSpec is $1.75\times$–$1.88\times$ faster than autoregressive decoding and $1.30\times$–$1.33\times$ faster than DFlash.

1 INTRODUCTION

Speculative decoding accelerates autoregressive generation by using a drafter to propose multiple tokens and a target model to verify them in a single forward pass, accepting draft tokens until the first rejection (Leviathan et al., 2023; Chen et al., 2023). Recent parallel and semi-autoregressive drafters, such as the diffusion-based DFlash, generate a block of draft tokens in one forward pass (Hui et al., 2026; Chen et al., 2026a; Cheng et al., 2026), making drafting inexpensive and shifting the main cost of each draft-and-verify round to verification, as Figure 1a shows. This makes the verification length, i.e., the number of draft tokens processed by the target in each round, a key design choice. A longer verification length may accept more tokens, but working on any tokens after the first rejection incurs wasted computation.

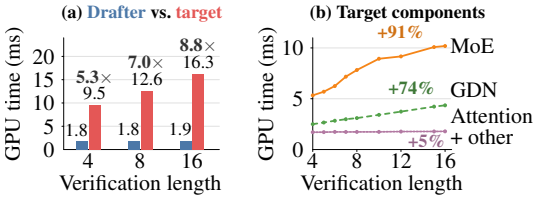


Figure 1: Profiled GPU time per verification call for Qwen3.6-35B-A3B with DFlash. (a) Times in ms, with target/drafter ratios in bold. (b) Bold percentages are increases from length 4 to 16.

Verification time grows much more rapidly with speculative length for the hybrid architectures used in recent open models such as Jamba, MiniMax-01, Kimi Linear, and Qwen3.5 (Lieber et al., 2024; MiniMax et al., 2025; Kimi Team et al., 2025; Qwen Team, 2026b) than for dense models. These architectures combine linear-attention or state-space layers, such as Gated DeltaNet (GDN) (Yang et al., 2025), with mixture-of-experts (MoE) layers. A GDN layer maintains a fixed-size recurrent

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.

state rather than a growing key–value cache, while an MoE layer routes each token to only a small subset of experts, 8 of 256 in the models we evaluate. In our profile of Qwen3.6-35B-A3B with a DFlash drafter at batch size one, increasing the verification length from 4 to 16 raises verification time by 71%, and Figure 1b shows that almost all of the increase comes from the GDN and MoE layers. In contrast, verification time for a dense Qwen3-4B target increases by only 4%–6%, since its batch-one cost is dominated by weight loading. For GDN, the default SGLang verifier (Zheng et al., 2024) updates the recurrent state at every token and stores a full copy after each update to enable rollback to the accepted prefix. For MoE, verification cost depends on expert reuse: additional tokens are inexpensive when they reuse already active experts but more costly when they activate new ones. Because experts are selected only when the target executes, a verification length chosen beforehand can only estimate this cost.

Existing length selectors, such as Sequoia (Chen et al., 2024), DSpark (Cheng et al., 2026), D-Cut (Liu et al., 2026), and EVICT (Pan et al., 2026), decide before verification from draft confidence or profiled costs. EcoSpec predicts draft-token experts with a separately trained model (Xie et al., 2026), and FASER prunes tokens at intermediate target layers (Chen et al., 2026b). None of them uses the target's own expert assignments before the experts run.

HybridSpec selects the verification length based on measured target-model latency and acceptance estimates derived from the drafter's own signals. Since both GDN and MoE costs increase with the number of verified tokens, reducing the verification length lowers both costs. HybridSpec then verifies this length plus one additional draft token up to the first MoE router, which we call the router boundary. At this point, a lightweight routing correction uses the observed expert assignments to retain the extra token only when its expected acceptance benefit outweighs the additional verification latency. To further accelerate GDN, HybridSpec employs kernels that process all verification tokens in parallel without storing per-token recurrent states. Our contributions are as follows:

- **Throughput-aware length selection.** A GPU-resident policy chooses the verification length to maximize decode throughput, and a lightweight routing correction can add one token at the router boundary. In exact arithmetic, every selected length provably preserves the target's greedy outputs and sampling distribution.
- **Two-stage verification at the router boundary.** Verification pauses after the first MoE router and resumes only on the selected tokens, with no CPU round trip or recomputation.
- **Parallel GDN verification and state reconstruction.** GPU kernels verify all tokens of a round in parallel within each GDN layer and rebuild the accepted states of all GDN layers in one launch. In isolated benchmarks with every draft token accepted, they are $1.54\times$–$4.04\times$ faster than the default SGLang kernels at lengths 4 to 16.
- **End-to-end speedups.** On three GDN–MoE target models under greedy decoding, HybridSpec is $1.75\times$–$1.88\times$ faster than autoregressive decoding and $1.30\times$–$1.33\times$ faster than vanilla DFlash, which verifies every draft token.

2 Background and Related Work

2.1 Speculative decoding and adaptive verification

Speculative decoding verifies draft tokens with the target model, accepting them until the first mismatch, after which the target generates its own token (Leviathan et al., 2023; Chen et al., 2023). This process exactly preserves the target model's greedy output and sampling distribution. Tree-based methods verify multiple candidate continuations in a single target pass (Miao et al., 2024). Parallel drafters generate multiple tokens simultaneously using extra decoding heads (Cai et al., 2024), a shared learnable hidden state (Hui et al., 2026), or a lightweight diffusion model (Chen et al., 2026a). EAGLE-3 instead drafts autoregressively using features from multiple target layers (Li et al., 2025), while DSpark combines parallel and sequential drafting (Cheng et al., 2026). In this work, we fix the parallel drafter and focus on reducing verification cost.

Adaptive methods dynamically determine how many draft tokens to verify or how speculation should be structured. SpecDec++ formulates draft-length selection as a stopping problem (Huang et al., 2025a), while online adaptation adjusts speculative configurations during inference (Liu et al., 2025). EAGLE-2 constructs its draft tree based on draft confidence (Li et al., 2024), and Sequoia optimizes

speculation trees using draft probabilities together with hardware costs (Chen et al., 2024). DSpark selects verification lengths from prefix-level acceptance estimates and an engine-specific throughput profile (Cheng et al., 2026), while D-Cut jointly prunes draft tokens across a batch using draft confidence and a runtime cost model (Liu et al., 2026). For MoE targets, Cascade adapts verification budgets (Saxena et al., 2025), EVICT truncates draft trees using drafter signals and profiled verification costs (Pan et al., 2026), and MoESD studies speculative decoding for sparse models (Huang et al., 2025b). EcoSpec prioritizes draft-tree nodes likely to reuse experts using a small model that predicts the target's expert assignments (Xie et al., 2026). FASER, designed for continuous-batching serving, prunes tokens predicted to be rejected at intermediate target layers (Chen et al., 2026b).

2.2 Gated DeltaNet and chunkwise verification

GDN is a linear-attention layer that keeps a fixed-size state matrix as a memory from keys to values (Katharopoulos et al., 2020; Yang et al., 2025). In the evaluated models, GDN layers are interleaved with softmax-attention layers, and every layer ends with an MoE block. For one head at token position i, let $q_i, k_i \in \mathbb{R}^{d_k}$ be the query and key, $v_i \in \mathbb{R}^{d_v}$ the value, and $S_i \in \mathbb{R}^{d_v \times d_k}$ the state after token i. Linear projections followed by a short causal convolution produce q_i, k_i, and v_i, and q_i and k_i are normalized to unit length. Each token decays the state by $\alpha_i \in (0, 1)$ and then corrects the value stored under its own key with a writing strength $\beta_i \in (0, 1)$. Both come from linear projections, and GDN applies

$$z_i = \beta_i\big(v_i - \alpha_i S_{i-1} k_i\big), \qquad S_i = \alpha_i S_{i-1} + z_i k_i^\top. \tag{1}$$

The update $z_i \in \mathbb{R}^{d_v}$ moves the value stored under k_i toward v_i by a fraction β_i, which is the delta rule (Schlag et al., 2021). The output $o_i = S_i q_i$ reads the updated state with the query, and we omit later output transformations such as normalization and the output gate.

Autoregressive decoding applies Equation (1) one token at a time. A verifier that does the same must store S_i, a $d_v \times d_k$ matrix per head, after every token, so that it can later restore the state at the end of whichever prefix is accepted. Verification can avoid this cost because all tokens of the target input, the tokens that the target processes in one round, are known in advance. Chunkwise formulations, developed for efficient training, process the tokens of a chunk in parallel (Dao & Gu, 2024; Yang et al., 2024). Number the tokens of the target input $1, \dots, w$, and let S_0 be the state before them. With the cumulative log-decay $g_i = \sum_{j=1}^{i} \log \alpha_j$, unrolling Equation (1) from S_0 gives:

$$S_i = e^{g_i} S_0 + \sum_{j=1}^{i} e^{g_i - g_j} z_j k_j^\top. \tag{2}$$

Substituting this expression for S_{i-1} into Equation (1) and stacking $k_i^\top$, $v_i^\top$, and $z_i^\top$ as the rows of $K \in \mathbb{R}^{w \times d_k}$ and $V, Z \in \mathbb{R}^{w \times d_v}$ gives one lower-triangular linear system for all w updates,

$$HZ = \operatorname{diag}(\beta)\big(V - \operatorname{diag}(e^g) K S_0^\top\big), \tag{3}$$

where $\beta = (\beta_1, \dots, \beta_w)$, $e^g = (e^{g_1}, \dots, e^{g_w})$, and H has ones on its diagonal and entries $H_{ij} = \beta_i e^{g_i - g_j} k_i^\top k_j$ for $j < i$. Solving this small $w \times w$ triangular system once therefore replaces the w sequential state updates. The solve and the dense matrix products that then give all w outputs use only S_0 and matrices with w rows, so no intermediate $d_v \times d_k$ state is formed. We call (z_i, k_i, g_i) the compact factors of token i, because Equation (2) reconstructs any state S_i from S_0 and the factors of tokens $1, \dots, i$. HybridSpec uses them to reconstruct the state after the accepted prefix.

Speculative decoders for recurrent targets differ in how they recover the state after the accepted prefix. STree accumulates state transitions to verify token trees (Wu et al., 2025). SpecLA, Bole, and TreeWY retain compact factors from which the accepted state is reconstructed (Wang et al., 2026b;a; Ghantasala, 2026). Other hybrid and recurrent systems likewise avoid storing a state for every verified token (Wang et al., 2024; Wu et al., 2024). Concurrent work, ReplaySSM, verifies drafts with a chunkwise triangular solve and writes the state back only when its ring buffer of recent GDN updates is flushed (Liou & Dao, 2026). Its reference implementation in vLLM (Kwon et al., 2023) targets large serving batch sizes, whereas HybridSpec targets single-request decoding. Equation (3) is the standard chunkwise computation. HybridSpec contributes fused, block-parallel verification kernels and a commit kernel that reconstructs the accepted state of every GDN layer in one launch.

3 HYBRIDSPEC

HybridSpec selects the verification length entirely on the GPU and splits target verification at the router boundary: immediately after the MoE router in the first layer (Layer 0) assigns experts, but before any routed expert is executed. Figure 2 illustrates one decoding round, and Algorithm 1 in Appendix A.1 details the complete procedure.

3.1 SETTING AND THROUGHPUT OBJECTIVE

We first formulate HybridSpec for greedy decoding with sequence batch size $B = 1$. In each round, a fixed drafter produces a proposal block of D draft tokens. Together with the anchor, the newest emitted token that the target has not yet processed, this block forms a full target input of length $W = D + 1$. HybridSpec retains the first $M \leq D$ draft tokens, which we call the *proposal prefix*, and verifies the corresponding target input of length $w = M + 1$. Thus D and M count draft tokens, whereas W and w count target input tokens, including the anchor. In all evaluated models, Layer 0 applies a GDN layer followed by an MoE block.

At the start of a round, x_0 denotes the anchor and d_i the i-th draft token. To make the first M draft tokens eligible for acceptance, the target processes $(x_0, d_1, \ldots, d_M)$. If a nonterminal round accepts $A \leq M$ draft tokens, it emits $R = A + 1$ tokens and commits $m = A + 1$ input tokens. These counts are equal but refer to different sequences: the emitted tokens are $(d_1, \ldots, d_A, x_{\text{new}})$, whereas the committed input tokens are $(x_0, d_1, \ldots, d_A)$. Here x_{new} is the token that the target emits after the accepted prefix. The old anchor is therefore consumed even when $A = 0$, and x_{new} becomes the next anchor. Both sequences are truncated at an end-of-sequence (EOS) token or the output limit.

Verifying only a prefix must not change the output. Under greedy decoding, HybridSpec emits the same tokens as target-only decoding in exact arithmetic, as Proposition 1 in Appendix C.2 proves under the assumptions stated there. Under temperature sampling, the verifier accepts each draft token with the probability that the target assigns to it and, after a rejection, samples a replacement from the target's distribution with that token excluded. Any verification length fixed before these draws, including the length after the routing correction, therefore preserves the output distribution of target-only sampling in exact arithmetic, as Proposition 2 proves.

Let $\mathcal{I}$ denote the information available when selecting a verification length and A_D the number of draft tokens accepted from the full proposal block. A proposal prefix of length M has $\min(A_D, M)$ accepted draft tokens, and the round emits one more token. The prefix-acceptance probability $p_i(\mathcal{I}) = \mathbb{P}(A_D \geq i \mid \mathcal{I})$ is the probability that the first i draft tokens are accepted. For a nonterminal round, the expected number of emitted tokens is $G(M \mid \mathcal{I}) = 1 + \sum_{i=1}^{M} p_i(\mathcal{I})$. We compare all values of M at the same initial state and proposal block. Let $L(M \mid \mathcal{I})$ denote the expected latency of a complete round, including drafting, all target computation, length-selection and graph-launch overhead, acceptance, and state commit. The objective is pooled decode throughput, the total number of emitted tokens divided by the total decode time. For a fixed set of rounds with information $\mathcal{I}_1, \ldots, \mathcal{I}_N$, choosing M_r in round r gives the pooled throughput

$$\frac{\sum_{r=1}^{N} G(M_r \mid \mathcal{I}_r)}{\sum_{r=1}^{N} L(M_r \mid \mathcal{I}_r)}. \tag{4}$$

This ratio of sums is not the average of the per-round ratios G/L, so the best choice in one round depends on the throughput of all rounds. Let $\rho^\star$ be the maximum of Equation (4). By Dinkelbach's theorem (Dinkelbach, 1967), a set of choices attains $\rho^\star$ exactly when every round maximizes $G(M_r \mid \mathcal{I}_r) - \rho^\star L(M_r \mid \mathcal{I}_r)$. Appendix B analyzes this per-round score.

3.2 TWO-STAGE VERIFICATION AT THE ROUTER BOUNDARY

After drafting, the draft-side selector chooses the nominal verification length w_0 from the drafter's signals and writes it to GPU memory (step 1 in Figure 2). Let $w_{\text{rem}} \in \{2, \ldots, W\}$ be the largest verification length that the remaining output budget allows, but at least 2. Stage 0 covers the first $\min(w_0 + 1, w_{\text{rem}})$ tokens of the target input and stops at the router boundary, before any routed expert runs. HybridSpec captures in advance one CUDA Graph, a recorded sequence of GPU kernels, for each possible Stage-0 length. A conditional graph node (NVIDIA, 2026) reads the Stage-0

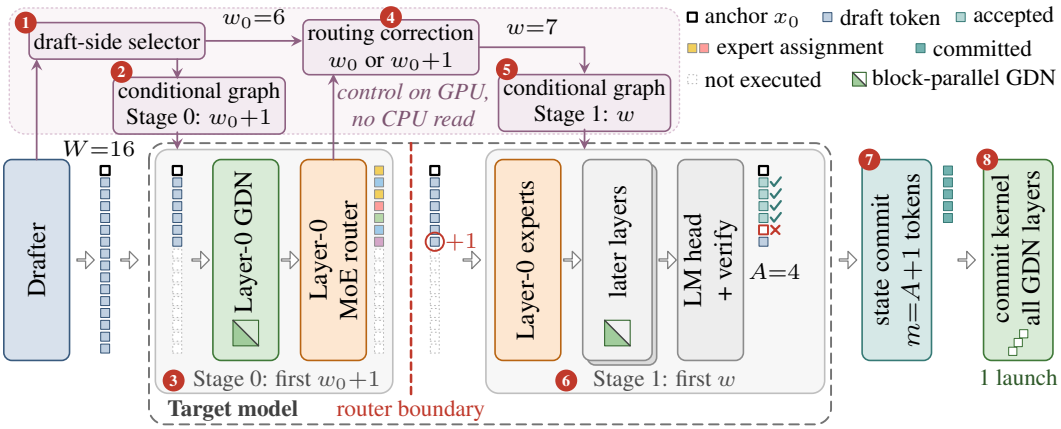


Figure 2: One round of HybridSpec in single-request greedy decoding, with illustrative values: nominal verification length $w_0 = 6$, executed verification length $w = 7$, and $A = 4$ accepted draft tokens. Circled numbers mark the steps described in Section 3.2.

length and launches the matching graph without a CPU read of the length (step 2). Stage 0 then runs these tokens through the Layer-0 GDN layer and MoE router (step 3). It retains the hidden states and residuals needed to resume the layer, plus each token's expert assignments and routing weights.

At the router boundary, the routing correction decides whether to keep d_{w_0}, the extra draft token that Stage 0 processed beyond the first w_0 tokens (step 4). The draft-side selector chose w_0 before the target ran, so it could charge d_{w_0} only the average expert cost. The actual cost is lower if d_{w_0} reuses experts already selected by the first w_0 tokens and higher if it activates new ones. Because Stage 0 has already routed d_{w_0}, the correction counts the new Layer-0 experts that it adds and uses this count to predict its expert cost in all MoE layers. It keeps d_{w_0}, setting the corrected verification length to $w_c = w_0 + 1$, only if its acceptance probability exceeds the predicted latency that it adds, converted into tokens at the pooled throughput rate, and otherwise sets $w_c = w_0$, as Equation (6) formalizes. The executed verification length is $w = \min(w_c, w_{\text{rem}})$. Stage 1 then processes the first w tokens from the Layer-0 experts through the final layer without repeating Stage 0. HybridSpec likewise captures one Stage-1 graph for each possible value of w, and a second conditional graph launches the graph for w (step 5). This graph reads the retained values of the first w tokens, builds the attention and GDN metadata for length w, and executes the final normalization and projection and the greedy verification of the $w - 1$ draft tokens (step 6).

Once verification accepts A draft tokens, the Stage-1 graph ends with the commit, which advances the attention and convolution caches, as well as the target hidden states that the drafter reads, through the $m = A + 1$ committed tokens (step 7). Layer-0 GDN retains compact factors for the Stage-0 tokens, and later GDN layers retain factors for the w selected tokens. A commit kernel then rebuilds the accepted state of every GDN layer from these factors in one launch (step 8).

3.3 Draft-side selection and routing correction

The length policy has two parts: the draft-side selector, which makes the main decision before target execution, and a lightweight routing correction, which runs after Stage 0 and can add one token. The fixed drafter supplies a top-1 confidence and an entropy for each of its $D = 15$ draft tokens. Before Layer 0, the selector calibrates the confidence and computes an entropy-change feature within the proposal block. It estimates conditional acceptance probabilities $\hat{c}_i \approx \mathbb{P}(A_D \geq i \mid A_D \geq i-1, \mathcal{I})$ and prefix probabilities $\hat{p}_i = \prod_{j=1}^{i} \hat{c}_j$, so $1 \geq \hat{p}_1 \geq \cdots \geq \hat{p}_D \geq 0$. In terms of verification length $w = M+1$, the expected number of emitted tokens G is $1+\sum_{i=1}^{w-1} p_i$. Over the nominal verification lengths $\mathcal{W}_0 = \{2, \ldots, W\}$, the selector chooses

$$w_0 = \arg\max_{w \in \mathcal{W}_0} \left[1 + \sum_{i=1}^{w-1} \hat{p}_i - \rho C_{\text{tgt}}(w) \right], \tag{5}$$

where the target-cost curve $C_{\mathrm{tgt}}(w)$ is the measured target latency at length w and $\rho > 0$ is a calibrated estimate of $\rho^\star$ that converts latency into tokens. The score is the per-round Dinkelbach objective with estimated acceptance probabilities. It uses C_{tgt} in place of L, because length-independent costs do not change its maximizer. Ties favor the shorter target input.

The routing correction uses the expert assignments observed in Stage 0 only to revise the latency estimate, and it leaves the acceptance probabilities $\hat{p}_i$ unchanged. Let $U_0(w)$ be the number of distinct Layer-0 experts selected by the first w tokens of the target input. Extending the target input from w_0 to $w_0 + 1$ adds draft token d_{w_0} and raises the expected number of emitted tokens by $\hat{p}_{w_0}$. Because drafting and Stage 0 are complete when the correction runs, the extension changes only the latency of Stage 1. The curve increment $C_{\mathrm{tgt}}(w_0 + 1) - C_{\mathrm{tgt}}(w_0)$ in Equation (5) differs from this change in two ways that the correction can account for. First, it includes the cost s of running one more token through the Layer-0 GDN layer and router, which Stage 0 has already paid for d_{w_0}. Second, it charges expert time at its calibration average. At batch size one, each expert processes only a few tokens, so MoE latency depends mainly on the number of distinct experts. Each distinct (layer, expert) pair therefore adds about η to the latency. Let n_{all} be the number of new (layer, expert) pairs that d_{w_0} adds across all MoE layers. The curve thus charges about $\eta\,\bar{n}_{\mathrm{all}}(w_0)$ for them, where $\bar{n}_{\mathrm{all}}(w_0)$ is the average of n_{all} at this position. Stage 0 observes only the Layer-0 part of n_{all}, $n_{\mathrm{new}} = U_0(w_0 + 1) - U_0(w_0)$. Because later-layer routing correlates with Layer-0 routing, as Appendix D.1 shows, the correction replaces $\bar{n}_{\mathrm{all}}(w_0)$ with $\bar{n}_{\mathrm{all}}(w_0 \mid n_{\mathrm{new}})$, the average of n_{all} given n_{new}. Subtracting s from the curve increment and making this replacement gives ΔL_{c}, the estimated Stage-1 latency increase. The correction applies the additive score of Equation (5), with the same rate ρ, to this one-step choice:

$$\Delta L_{\mathrm{c}} = C_{\mathrm{tgt}}(w_0 + 1) - C_{\mathrm{tgt}}(w_0) - s + \eta\big(\bar{n}_{\mathrm{all}}(w_0 \mid n_{\mathrm{new}}) - \bar{n}_{\mathrm{all}}(w_0)\big),$$
$$w_{\mathrm{c}} = \begin{cases} w_0 + 1, & \text{if } \hat{p}_{w_0} > \rho\,\Delta L_{\mathrm{c}}, \\ w_0, & \text{otherwise.} \end{cases} \tag{6}$$

Because w_0 maximizes the score in Equation (5) under the same curve, $\hat{p}_{w_0} \le \rho\,[C_{\mathrm{tgt}}(w_0 + 1) - C_{\mathrm{tgt}}(w_0)]$ whenever $w_0 < W$. The correction can therefore expand only if d_{w_0} costs less than the curve predicts, that is, only if $s > \eta\,\big(\bar{n}_{\mathrm{all}}(w_0 \mid n_{\mathrm{new}}) - \bar{n}_{\mathrm{all}}(w_0)\big)$.

The correction adds at most one token because Stage 0 must process every token that the correction might add, at cost s per token in every round. If $w_0 = W$ or $w_{\mathrm{rem}} \le w_0$, Stage 0 does not cover position $w_0 + 1$, and the correction does not expand. Acceptance parameters are fitted on the calibration prompts, latency is measured on the evaluated runtime, and all parameters are frozen before test evaluation. Appendix A.2 describes the calibration.

3.4 Parallel GDN verification and accepted-state reconstruction

When a GDN layer processes u tokens, the default SGLang verifier updates its state one token at a time and writes a full state after every token, requiring $O(u\,d_v d_k)$ elements per head. HybridSpec instead treats the target input as a single chunk and processes all u tokens in parallel, as Figure 3 shows. Its block-parallel triangular-solve kernel solves Equation (3), computes the outputs from the result, and retains only the compact factors (z_i, k_i, g_i), which require $O(u(d_v + d_k))$ elements. Both counts exclude S_0 and workspace. The Layer-0 GDN layer precedes the router, so it runs on the Stage-0 tokens with $u = \min(w_0 + 1, w_{\mathrm{rem}})$, and later GDN layers run with $u = w$ in Stage 1. For $2 \le w \le 16$, the kernels are compiled for each length, so they evaluate only the selected tokens, and exploit the small size of H with closed-form 4×4 block inverses and an exact truncated series. A joint producer computes all input projections and the convolution in one launch.

After verification, each GDN layer must advance its state through the $m = A + 1$ committed tokens, namely the anchor and the accepted draft tokens, as the right side of Figure 3 shows. Let $Z_{1:m}$ and $K_{1:m}$ denote the first m rows of Z and K in Equation (3). With S_0 the layer's state at the start of the round, the accepted state is Equation (2) at $i = m$. In matrix form, it reads

$$S_m = e^{g_m} S_0 + Z_{1:m}^{\top}\,\mathrm{diag}\,\big(e^{g_m - g_1}, \ldots, e^{g_m - g_m}\big) K_{1:m}. \tag{7}$$

Because H is lower triangular, the first m rows of Z depend only on the first m tokens. Discarding the rejected suffix therefore leaves $Z_{1:m}$ unchanged in exact arithmetic, so the commit kernel reuses the factors computed during verification. The HybridSpec kernels store k_i and g_i rather than decay-scaled keys, so the commit kernel can form the decay weights $e^{g_m - g_i}$ for any m. Accepted-state

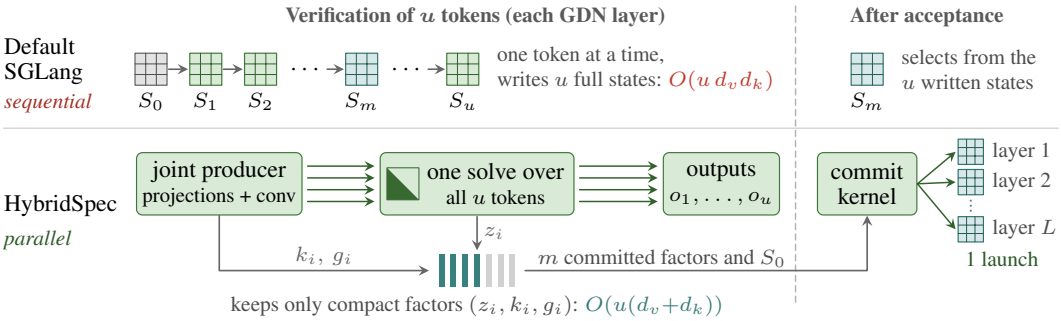


Figure 3: GDN verification and accepted-state reconstruction in the default SGLang verifier and in HybridSpec. The default verifier steps through the u tokens one at a time, whereas HybridSpec processes all u tokens in parallel and rebuilds the accepted states of all GDN layers in parallel, in one launch. Teal marks the accepted state and the factors of the m committed tokens.

reconstruction is independent across GDN layers, so one commit kernel launch reconstructs the state of every GDN layer. The kernel accumulates in 32-bit floating point (FP32). Only this reconstruction is parallel across layers, and the target forward pass remains sequential.

4 Evaluation

We first compare HybridSpec end to end against six baselines across three target–drafter pairs. We then use matched ablations and component-level measurements to isolate the contributions of adaptive length selection, routing correction, and the GDN kernels.

4.1 Experimental setup

Models and workloads. We evaluate Qwen3.5-35B-A3B, Qwen3.6-35B-A3B, and Qwen3.5-122B-A10B-GPTQ-INT4 (Qwen Team, 2026b;c;a), with their official DFlash drafters, in SGLang 0.5.16 with one request at a time on NVIDIA A100 80 GB GPUs. Prompts come from seven categories of the SPEED-Bench Qualitative split (Abramovich et al., 2026): coding, math, reasoning, retrieval-augmented generation (RAG), summarization, writing, and roleplay. A seeded split gives 280 calibration and 280 test prompts, 40 per category in each. Table 1 reports greedy decoding, the primary setting, and Appendices E.6 and E.7 report temperature-one sampling and batched decoding.

Baselines. We compare HybridSpec with autoregressive (AR) decoding, vanilla DFlash, D-Cut (Liu et al., 2026), EVICT (Pan et al., 2026), ReplaySSM (Liou & Dao, 2026), and EVICT + ReplaySSM. All speculative baselines use the same DFlash drafter as HybridSpec. We adapt D-Cut and EVICT to the DFlash proposal block and use the SGLang implementation of ReplaySSM. Vanilla DFlash verifies the full target input of length $W = D + 1$ in every round. D-Cut and EVICT shorten the proposal block before verification. ReplaySSM replaces the default GDN verification procedure, and EVICT + ReplaySSM combines both optimizations.

Metrics. The primary metric is the speedup ratio (SR), defined as the pooled post-prefill decode throughput normalized by that of the corresponding AR baseline. Acceptance length (AL) is the number of tokens emitted per verification round, including the accepted draft tokens and the additional token generated by the target. Appendix E details the full protocol.

4.2 End-to-end performance

In Table 1, the mean SR of HybridSpec is $1.88\times$ on Qwen3.5-35B-A3B, $1.83\times$ on Qwen3.6-35B-A3B, and $1.75\times$ on Qwen3.5-122B-A10B. Vanilla DFlash reaches $1.45\times$, $1.37\times$, and $1.32\times$, and EVICT, the strongest baseline on all three pairs, reaches $1.59\times$, $1.51\times$, and $1.65\times$. HybridSpec has the highest SR in every category, and its per-category SR ranges from $1.38\times$ for roleplay on Qwen3.5-122B-A10B to $2.39\times$ for coding on Qwen3.5-35B-A3B. And we further evaluate correctness and task quality in Appendix D.3.

Table 1: End-to-end comparison under greedy decoding. AL is the number of tokens emitted per verification round, and SR is the decode-throughput speedup over paired autoregressive (AR) decoding. Summ. denotes summarization, and Avg. is the mean over the seven categories.

	Coding		Math		Reasoning		RAG		Summ.		Writing		Roleplay		Avg.	
Method	AL	SR	AL	SR	AL	SR	AL	SR	AL	SR	AL	SR	AL	SR	AL	SR
Qwen3.5-35B-A3B																
AR target	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×
Vanilla DFlash	6.43	1.89×	5.40	1.54×	4.91	1.43×	6.07	1.72×	4.96	1.38×	3.92	1.13×	3.53	1.05×	5.03	1.45×
D-Cut	6.12	1.98×	4.97	1.64×	4.61	1.54×	5.76	1.80×	4.65	1.48×	3.70	1.26×	3.37	1.19×	4.74	1.55×
EVICT	6.18	1.99×	5.15	1.69×	4.68	1.58×	5.88	1.85×	4.78	1.54×	3.77	1.29×	3.36	1.20×	4.83	1.59×
ReplaySSM	6.41	1.82×	5.53	1.54×	4.82	1.36×	6.04	1.66×	4.93	1.34×	3.92	1.10×	3.51	1.03×	5.02	1.41×
EVICT + ReplaySSM	5.94	1.91×	5.11	1.65×	4.60	1.52×	5.69	1.76×	4.72	1.50×	3.73	1.26×	3.33	1.17×	4.73	1.54×
HybridSpec	6.20	2.39×	5.10	1.97×	4.62	1.87×	5.87	2.18×	4.80	1.84×	3.69	1.51×	3.27	1.42×	4.79	1.88×
Qwen3.6-35B-A3B																
AR target	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×
Vanilla DFlash	5.58	1.56×	5.41	1.58×	4.79	1.42×	5.87	1.63×	4.85	1.37×	3.66	1.06×	3.33	0.99×	4.79	1.37×
D-Cut	5.36	1.66×	5.11	1.68×	4.58	1.54×	5.62	1.71×	4.67	1.50×	3.53	1.19×	3.21	1.12×	4.58	1.48×
EVICT	5.37	1.68×	5.17	1.70×	4.63	1.58×	5.63	1.73×	4.72	1.54×	3.52	1.21×	3.17	1.13×	4.60	1.51×
ReplaySSM	5.65	1.54×	5.49	1.55×	4.83	1.39×	5.69	1.53×	4.87	1.34×	3.66	1.04×	3.36	0.98×	4.79	1.34×
EVICT + ReplaySSM	5.37	1.63×	5.18	1.65×	4.60	1.53×	5.59	1.66×	4.69	1.48×	3.54	1.18×	3.19	1.10×	4.59	1.46×
HybridSpec	5.28	2.05×	5.14	2.09×	4.62	1.88×	5.55	2.07×	4.74	1.85×	3.50	1.46×	3.16	1.39×	4.57	1.83×
Qwen3.5-122B-A10B																
AR target	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×
Vanilla DFlash	6.83	1.67×	5.60	1.41×	5.13	1.29×	6.31	1.56×	5.33	1.31×	4.13	1.05×	3.54	0.95×	5.27	1.32×
D-Cut	6.29	1.89×	5.23	1.70×	4.85	1.59×	5.95	1.81×	5.01	1.60×	3.85	1.32×	3.44	1.24×	4.95	1.59×
EVICT	6.05	1.93×	5.09	1.76×	4.79	1.66×	5.94	1.90×	4.82	1.63×	3.72	1.37×	3.33	1.29×	4.82	1.65×
ReplaySSM	6.59	1.56×	5.78	1.42×	5.08	1.25×	6.37	1.53×	5.26	1.28×	4.06	1.02×	3.60	0.95×	5.25	1.29×
EVICT + ReplaySSM	6.31	1.89×	5.10	1.68×	4.70	1.56×	5.99	1.82×	4.97	1.60×	3.78	1.32×	3.38	1.25×	4.89	1.59×
HybridSpec	6.02	2.06×	5.10	1.84×	4.71	1.77×	5.85	2.01×	4.91	1.72×	3.75	1.46×	3.34	1.38×	4.81	1.75×

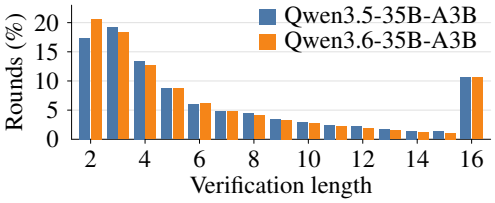


Figure 4: Executed verification lengths of HybridSpec on the 280 test prompts. Vanilla DFlash verifies 16 tokens per round.

HybridSpec has a lower AL than vanilla DFlash but a higher SR mainly because it verifies fewer tokens in each round. On the two 35B pairs, the mean verification length of HybridSpec is 6.4 and 6.2, compared with 16 for vanilla DFlash, and Figure 4 shows that about half of its rounds verify at most 4 tokens. Relative to vanilla DFlash, HybridSpec lowers mean AL by 0.24 on Qwen3.5-35B-A3B and 0.22 on Qwen3.6-35B-A3B, or 5% on both. It leaves 9.6 and 9.8 draft tokens per round unverified, and the AL difference corresponds to fewer than 3% of them. Most skipped tokens would therefore have been rejected. Meanwhile, the decode time per round, the pooled decode time divided by the number of rounds, falls from 22.9 to 16.5 ms on Qwen3.5-35B-A3B and from 23.0 to 16.3 ms on Qwen3.6-35B-A3B. In every category on both 35B pairs, AL falls by at most 7%, and time per round falls by 24%–32%. Time per round falls most for writing and roleplay, the two categories with the lowest AL. In the isolated benchmarks of Table 5, with every draft token accepted, the GDN verification and reconstruction kernels save 0.12 ms per round at length 2 and 1.44 ms at length 16.

D-Cut and EVICT reach an AL close to that of HybridSpec (4.58–4.95 versus 4.57–4.81) but a lower SR (1.48×–1.65× versus 1.75×–1.88×), so HybridSpec spends less time per round. Four design differences can contribute. First, both baselines choose the length that maximizes a per-round ratio of expected tokens to profiled step latency (Liu et al., 2026; Pan et al., 2026), whereas HybridSpec maximizes pooled throughput through the per-round score of Equation (5). Second, both form acceptance estimates directly from drafter confidences, whereas HybridSpec calibrates its estimates on the calibration prompts. Third, both read the selected length on the CPU to launch the matching verification graph, whereas HybridSpec launches it with a conditional graph node and never copies the length to the CPU. Fourth, HybridSpec uses its own GDN kernels. EVICT + ReplaySSM, which adds prior GDN verification to EVICT, still trails HybridSpec by 0.16–0.37 in mean SR. Without recalibration, HybridSpec also speeds up the four other SPEED-Bench Qualitative categories. In Table 2, where DFlash denotes vanilla DFlash, HybridSpec reaches a mean SR of 1.69× on their 320 prompts with Qwen3.6-35B-A3B, versus 1.31× for vanilla DFlash, and leads in every category.

Table 2: Unseen SPEED-Bench categories on Qwen3.6-35B-A3B, evaluated without recalibration.

	DFlash		HybridSpec	
Category	AL	SR	AL	SR
Humanities	3.89	1.18×	3.65	1.55×
QA	3.95	1.19×	3.64	1.56×
STEM	4.27	1.28×	3.95	1.66×
Multilingual	5.27	1.57×	4.98	2.01×
Avg.	4.35	1.31×	4.06	1.69×

Table 3: Component gains on Qwen3.6-35B-A3B. Gain is the SR of HybridSpec divided by that of the variant, and w is the mean verification length.

Component	Replaced by	w	AL	SR	Gain
HybridSpec	–	6.2	4.57	1.83×	–
Length selection	Static length 7	7	3.99	1.54×	1.19×
Length selection	Length 16	16	4.77	1.46×	1.25×
Routing correction	Nominal-only	6.1	4.48	1.73×	1.06×
GDN kernels	SGLang kernels	6.2	4.51	1.74×	1.05×
All components	Vanilla DFlash	16	4.79	1.37×	1.33×

4.3 Ablations

Table 3 removes one component of HybridSpec at a time on Qwen3.6-35B-A3B while keeping the others unchanged. We report the gain as the SR of HybridSpec divided by that of each variant, representing the speedup contributed by the removed component relative to its stated replacement. Vanilla DFlash removes all components. Removing length selection also eliminates two-stage verification and the routing correction, since both depend on the selected length. We therefore evaluate two fixed-length variants that use the HybridSpec GDN kernels within a single CUDA Graph. The calibrated static length is 7, the fixed length with the highest pooled throughput on the calibration prompts among screened candidates, while length 16 matches vanilla DFlash. The nominal-only control removes only the routing correction: it keeps $w = w_0$ under the same two-stage verification, with Stage 0 processing only the first w_0 tokens. Thus, its measured gain captures the routing correction net of the cost of processing the extra Stage-0 token. Finally, the GDN variant retains the full length-selection policy but replaces the HybridSpec GDN kernels with the default SGLang kernels.

Length selection provides the largest gain: $1.19\times$ over the calibrated static length and $1.25\times$ over length 16. Although HybridSpec verifies fewer tokens per round than the static-length variant, 6.2 versus 7, it achieves a higher AL, 4.57 versus 3.99, by adapting the verification length to each round. The routing correction provides a further $1.06\times$ gain over the nominal-only control. The HybridSpec GDN kernels contribute $1.05\times$ at the lengths selected by HybridSpec. At length 16, the SR ratio between the fixed-length variant and vanilla DFlash indicates a larger kernel gain of $1.07\times$, consistent with the greater savings at longer verification lengths in Table 5.

4.4 GDN kernel microbenchmarks

We measure the latency of GDN verification plus accepted-state reconstruction for the 30 GDN layers of Qwen3.6-35B-A3B on one A100, and Table 5 in Appendix D.2 gives all measured lengths. With every draft token accepted, the HybridSpec kernels are $1.33\times$–$1.35\times$ faster than the default SGLang kernels at lengths 2 and 3 and $1.54\times$–$4.04\times$ faster at lengths 4 to 16. The speedup grows with length, consistent with the default kernels writing a full state after every token. With its default launch parameters, as in Table 1, ReplaySSM is slower than the default SGLang kernels at every length, consistent with its lower mean SR than vanilla DFlash. The HybridSpec kernels are $1.95\times$–$6.23\times$ faster than it. These isolated benchmarks start from precomputed projections and exclude the joint producer and two-stage verification. The kernels also save memory, as Appendix D.4 shows. At batch size one, the default verifier reserves 1.88 GB, 40% of the size of the attention-cache pool, to store the state after every token, whereas the HybridSpec kernels need no such buffer.

5 Conclusion

Choosing the verification length for hybrid GDN–MoE targets trades expected output progress against recurrent and routed-expert work. HybridSpec makes this choice on the GPU from acceptance estimates and measured target latency and refines it with a lightweight routing correction at the first MoE router. The rest of verification runs only on the selected tokens, with GDN kernels that verify all tokens in parallel and reconstruct every accepted state in one launch. On three GDN–MoE target models in single-request greedy decoding, HybridSpec is $1.75\times$–$1.88\times$ faster than autoregressive decoding and $1.30\times$–$1.33\times$ faster than vanilla DFlash.

AI USE STATEMENT

The authors used generative AI tools for manuscript writing and language editing, coding assistance and code review, and checking reported results against experimental records. AI-generated output was not treated as evidence. The authors reviewed all AI-assisted text, code, and checks and take full responsibility for the content of this paper.

ETHICS STATEMENT

This work studies language-model inference efficiency and does not involve human-subject research or the collection of personal data. Cheaper inference lowers the cost of both beneficial and harmful uses of the underlying models. Because HybridSpec preserves the target's outputs in exact arithmetic, safety, privacy, and fairness remain properties of the target model and its deployment.

REPRODUCIBILITY STATEMENT

Sections 3–4 specify the problem formulation, method, correctness conditions, and reported measurements. Appendices A–E provide the round procedure, calibration details, derivations and proofs, the profiling and kernel-benchmark setups, and the end-to-end evaluation details. We will release the code upon acceptance.

REFERENCES

Talor Abramovich, Maor Ashkenazi, Izzy Putterman, Benjamin Chislett, Tiyasa Mitra, Bita Darvish Rouhani, Ran Zilberstein, and Yonatan Geifman. SPEED-Bench: A unified and diverse benchmark for speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://arxiv.org/abs/2604.09557.

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*. PMLR, 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 https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026a. URL https://arxiv.org/abs/2602.06036.

Wenyan Chen, Chengzhi Lu, Yanying Lin, and Dmitrii Ustiugov. FASER: Fine-grained phase management for speculative decoding in dynamic LLM serving. arXiv preprint arXiv:2604.20503, 2026b. URL https://arxiv.org/abs/2604.20503.

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.

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.

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 https://arxiv.org/abs/2110.14168.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. PMLR, 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Werner Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967. URL https://doi.org/10.1287/mnsc.13.7.492.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, Chenhan Yu, Ye Yu, Weijia Chen, Sajad Norouzi, Jingyu Liu, Shiyi Lan, Ligeng Zhu, Jin Wang, Jindong Jiang, Morteza Mardani, Mehran Maghoumi, Song Han, Ante Jukić, Nima Tajbakhsh, Jan Kautz, and Pavlo Molchanov. Nemotron-Labs-Diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. arXiv preprint arXiv:2607.05722, 2026. URL https://arxiv.org/abs/2607.05722.

Sneha Murthy Ghantasala. TreeWY: Speculative verification for Gated DeltaNet hybrids. arXiv preprint arXiv:2608.20961, 2026. URL https://arxiv.org/abs/2608.20961.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. SpecDec++: Boosting speculative decoding via adaptive candidate lengths. In *Conference on Language Modeling (COLM)*, 2025a. URL https://openreview.net/forum?id=Y131N9fUbU.

Zongle Huang, Lei Zhu, Zongyuan Zhan, Ting Hu, Weikai Mao, Xianzhi Yu, Yongpan Liu, and Tianyu Zhang. MoESD: Unveil speculative decoding's potential for accelerating sparse MoE. In *Advances in Neural Information Processing Systems*, volume 38, pp. 138874–138909, 2025b. doi: 10.52202/085713-4176.

Mude Hui, Xin Huang, Jaime Campos Salas, Yue Sun, Nathan Pemberton, Xiang Song, Ashish Khetan, and George Karypis. P-EAGLE: Parallel-drafting EAGLE with scalable training. arXiv preprint arXiv:2602.01469, 2026. URL https://arxiv.org/abs/2602.01469.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun Du. Kimi Linear: An expressive, efficient attention architecture. arXiv preprint arXiv:2510.26692, 2025. URL https://arxiv.org/abs/2510.26692.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165.

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. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.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. Association for Computational Linguistics, 2024. 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*, volume 38, pp. 151568–151587, 2025. doi: 10.52202/085713-4562.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid Transformer-Mamba language model. arXiv preprint arXiv:2403.19887, 2024. URL https://arxiv.org/abs/2403.19887.

Ze-Wei Liou and Tri Dao. ReplaySSM: Cache SSM inputs, not state. Dao AI Lab blog, June 15, 2026. URL https://dao-lab.ai/blog/2026/replayssm/.

Jiesong Liu, Brian Park, and Xipeng Shen. A drop-in solution for on-the-fly adaptation of speculative decoding in large language models. In *Proceedings of the 63rd Annual Meeting of*

the Association for Computational Linguistics (Volume 1: Long Papers), pp. 9778–9794. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.482. URL https://aclanthology.org/2025.acl-long.482/.

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, 2026. URL https://arxiv.org/abs/2607.14647.

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. ACM, 2024. doi: 10.1145/3620666.3651335.

MiniMax, Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, Enwei Jiao, Gengxin Li, Guojun Zhang, Haohai Sun, Houze Dong, Jiadai Zhu, Jiaqi Zhuang, Jiayuan Song, Jin Zhu, Jingtao Han, Jingyang Li, Junbin Xie, Junhao Xu, Junjie Yan, Kaishun Zhang, Kecheng Xiao, Kexi Kang, Le Han, Leyang Wang, Lianfei Yu, Liheng Feng, Lin Zheng, Linbo Chai, Long Xing, Meizhi Ju, Mingyuan Chi, Mozhi Zhang, Peikai Huang, Pengcheng Niu, Pengfei Li, Pengyu Zhao, Qi Yang, Qidi Xu, Qiexiang Wang, Qin Wang, Qiuhui Li, Ruitao Leng, Shengmin Shi, Shuqi Yu, Sichen Li, Songquan Zhu, Tao Huang, Tianrun Liang, Weigao Sun, Weixuan Sun, Weiyu Cheng, Wenkai Li, Xiangjun Song, Xiao Su, Xiaodong Han, Xinjie Zhang, Xinzhu Hou, Xu Min, Xun Zou, Xuyang Shen, Yan Gong, Yingjie Zhu, Yipeng Zhou, Yiran Zhong, Yongyi Hu, Yuanxiang Fan, Yue Yu, Yufeng Yang, Yuhao Li, Yunan Huang, Yunji Li, Yunpeng Huang, Yunzhi Xu, Yuxin Mao, Zehan Li, Zekang Li, Zewei Tao, Zewen Ying, Zhaoyang Cong, Zhen Qin, Zhenhua Fan, Zhihang Yu, Zhuo Jiang, and Zijia Wu. MiniMax-01: Scaling foundation models with lightning attention. arXiv preprint arXiv:2501.08313, 2025. URL https://arxiv.org/abs/2501.08313.

NVIDIA. CUDA programming guide, section 4.2: CUDA Graphs. Online documentation, last updated September 10, 2026. URL https://docs.nvidia.com/cuda/cuda-programming-guide/04-special-topics/cuda-graphs.html.

Lehan Pan, Ziyang Tao, Xiao Wang, Zhimeng Zhao, Meng Wang, and Yanyong Zhang. Making every verified token count: Adaptive verification for MoE speculative decoding. arXiv preprint arXiv:2605.00342, 2026. URL https://arxiv.org/abs/2605.00342.

Qwen Team. Qwen3.5-122B-A10B-GPTQ-Int4. Hugging Face model card, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-122B-A10B-GPTQ-Int4.

Qwen Team. Qwen3.5-35B-A3B. Hugging Face model card, 2026b. URL https://huggingface.co/Qwen/Qwen3.5-35B-A3B.

Qwen Team. Qwen3.6-35B-A3B. Hugging Face model card, 2026c. URL https://huggingface.co/Qwen/Qwen3.6-35B-A3B.

Anish Saxena, Po-An Tsai, Hritvik Taneja, Aamer Jaleel, and Moinuddin Qureshi. Utility-driven speculative decoding for mixture-of-experts. arXiv preprint arXiv:2506.20675, 2025. URL https://arxiv.org/abs/2506.20675.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366. PMLR, 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19. ACM, 2019. doi: 10.1145/3315508.3329973.

Junxiong Wang, Daniele Paliotta, Avner May, Alexander M. Rush, and Tri Dao. The Mamba in the Llama: Distilling and accelerating hybrid models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62432–62457, 2024. doi: 10.52202/079017-1996.

Li Wang, Yi Su, Xiabao Wu, Chiran You, Yongchao Liu, Zhan Qiu, Juelu Zhang, Jiajun Zheng, Fangxin Liu, Jie Zhang, Chen Tian, and Chengying Huan. Bole: Efficient tree speculation for hybrid-attention language models. arXiv preprint arXiv:2608.01651, 2026a. URL https://arxiv.org/abs/2608.01651.

Zhibin Wang, Xuying Han, Zhaohua Yang, Fuliang Liu, Xue Li, Rong Gu, Sheng Zhong, and Chen Tian. SpecLA: Efficient speculative decoding for linear-attention models. arXiv preprint arXiv:2607.16673, 2026b. URL https://arxiv.org/abs/2607.16673.

Yangchao Wu, Yonatan Dukler, Matthew Trager, Alessandro Achille, Wei Xia, and Stefano Soatto. Snakes and ladders: Accelerating SSM inference with speculative decoding. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop*, volume 262 of *Proceedings of Machine Learning Research*, pp. 292–304. PMLR, 2024. URL https://proceedings.mlr.press/v262/wu24a.html.

Yangchao Wu, Zongyue Qin, Alex Wong, and Stefano Soatto. STree: Speculative tree decoding for hybrid state space models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 128121–128141, 2025. doi: 10.52202/085713-3855.

Jincheng Xie, Runheng Liu, Heyan Huang, Yawen Ling, Hanbin Dai, Yu Zheng, and Wen Hu. Less experts, faster decoding: Cost-aware speculative decoding for mixture-of-experts. arXiv preprint arXiv:2607.12696, 2026. URL https://arxiv.org/abs/2607.12696.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, volume 37, pp. 115491–115522, 2024. doi: 10.52202/079017-3668.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

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, pp. 62557–62583, 2024. doi: 10.52202/079017-2000.

A HybridSpec Inference Details

A.1 Round procedure

Algorithm 1 lists one round of Section 3.2, and its comments give the corresponding steps of Figure 2. The lengths w_0, u, and w stay in GPU memory, and lines 5 and 12 launch the CUDA Graphs for lengths u and w through conditional graph nodes. Under temperature sampling, the point-mass rule of Appendix C.3 replaces lines 13 and 14, and all other lines are unchanged. At an EOS token or the output limit, the emitted and committed tokens are truncated as in Section 3.1.

Algorithm 1 One round of HybridSpec

Input: anchor x_0, target caches and GDN states, drafter context, and length limit w_{rem} from the remaining output budget
Parameters: calibrated acceptance model, C_{tgt}, ρ, s, η, and the tables $\bar{n}_{\mathrm{all}}(w_0)$ and $\bar{n}_{\mathrm{all}}(w_0 \mid n)$
Output: emitted tokens, next anchor, and updated caches, GDN states, and drafter context

1: Draft the proposal block $(d_1, \ldots, d_D)$ with the top-1 confidence and entropy of each token
2: Estimate $\hat{c}_1, \ldots, \hat{c}_D$ and set $\hat{p}_i \leftarrow \prod_{j=1}^{i} \hat{c}_j$ for $i = 1, \ldots, D$ ▷ App. A.2
3: $w_0 \leftarrow \arg\max_{w \in \mathcal{W}_0} \left[1 + \sum_{i=1}^{w-1} \hat{p}_i - \rho\, C_{\mathrm{tgt}}(w)\right]$ ▷ Eq. (5), step 1
4: $u \leftarrow \min(w_0 + 1, w_{\mathrm{rem}})$ ▷ Stage-0 length
5: Stage 0: run Layer 0 through its MoE router on the first u tokens of $(x_0, d_1, \ldots, d_D)$, retaining compact factors, hidden states, residuals, expert assignments, and routing weights ▷ steps 2–3
6: $w_{\mathrm{c}} \leftarrow w_0$ ▷ step 4
7: **if** $u = w_0 + 1$ **then** ▷ Stage 0 covers position $w_0 + 1$
8: $\quad n_{\mathrm{new}} \leftarrow U_0(w_0 + 1) - U_0(w_0)$
9: $\quad \Delta L_{\mathrm{c}} \leftarrow C_{\mathrm{tgt}}(w_0 + 1) - C_{\mathrm{tgt}}(w_0) - s + \eta \left(\bar{n}_{\mathrm{all}}(w_0 \mid n_{\mathrm{new}}) - \bar{n}_{\mathrm{all}}(w_0)\right)$ ▷ Eq. (6)
10: $\quad w_{\mathrm{c}} \leftarrow w_0 + \mathbf{1}\left[\hat{p}_{w_0} > \rho\, \Delta L_{\mathrm{c}}\right]$
11: $w \leftarrow \min(w_{\mathrm{c}}, w_{\mathrm{rem}})$ ▷ executed verification length
12: Stage 1: resume Layer 0 at its experts for the first w tokens, run the remaining layers, and retain their GDN compact factors ▷ steps 5–6
13: $y_i \leftarrow$ target argmax after $(x_0, d_1, \ldots, d_{i-1})$ for $i = 1, \ldots, w$
14: $A \leftarrow \max\{a \in \{0, \ldots, w-1\} : d_i = y_i \text{ for all } i \le a\}$ and $x_{\mathrm{new}} \leftarrow y_{A+1}$
15: Advance the attention and convolution caches and the drafter context through the $m = A + 1$ committed tokens ▷ step 7
16: Reconstruct the accepted state of every GDN layer in one kernel launch ▷ Eq. (7), step 8
17: **return** $(d_1, \ldots, d_A, x_{\mathrm{new}})$, with x_{new} as the next anchor

A.2 GPU-resident selection of verification length

Draft-side selector. The selector reads the top-1 confidence and the entropy h_i of each draft token from GPU memory. It calibrates the confidence per position with isotonic regression. Let $\bar{h}_i = (h_i + \bar{h}_{i-1})/2$, with $\bar{h}_0 = h_1$, be an exponential moving average of the entropy within the proposal block. A logistic model then combines the calibrated confidence, the position, their interaction, $\log(1+h_i)$, and the entropy change $\log(h_i/\bar{h}_{i-1})$. Its outputs are the conditional acceptance probabilities $\hat{c}_i$. One kernel forms the prefix probabilities and evaluates Equation (5) for all nominal verification lengths. All selector parameters are fitted on the calibration prompts (Appendix E.3).

Cost calibration. Runs on the evaluated runtime fix the verification length at each supported value and measure the mean decode time per round on the calibration prompts. A quadratic in w is fitted to these times, and $C_{\mathrm{tgt}}(w)$ is the fit minus a fixed per-round cost that does not depend on w. Calibration also measures s, the latency that one more token adds to the Layer-0 GDN layer and router, as the slope of Stage-0 latency in the number of tokens. The constant η is the latency that one more distinct expert adds to one MoE layer at batch size one. Every MoE layer of the evaluated models routes each token to 8 of 256 experts. Two measurements support the per-expert model. For Qwen3.6-35B-A3B at verification length 16, the target latency of one verification call in each of 80 requests has a correlation of 0.999 with that call's number of distinct (layer, expert) pairs. For the Qwen3.5-122B-A10B MoE kernel with 16 input rows, and hence 128 token–expert

assignments, median latency grows almost linearly with the number of distinct experts, from 0.11 ms at 8 experts to 1.18 ms at 128. Because the number of assignments is fixed, token work adds at most the fitted intercept of 0.06 ms. The fitted slopes give η, $4.18\,\mu$s for the two 35B models, which share one architecture, and $8.8\,\mu$s for Qwen3.5-122B-A10B. The table value $\bar{n}_{\mathrm{all}}(w_0)$ is the sum of two averages for the token at position $w_0 + 1$ on the calibration prompts: the number of new Layer-0 experts and, as Appendix D.1 describes, the number of new (layer, expert) pairs in the later layers. The conditional value $\bar{n}_{\mathrm{all}}(w_0 \mid n)$ replaces the first average with n and the second with its average over the tokens at that position that add n new Layer-0 experts. Cells with fewer than 30 tokens use a linear fit in n for each position. The rate ρ is the fixed point of Dinkelbach's iteration (Dinkelbach, 1967). It equals the estimated pooled throughput of the draft-side selector on the rounds of the capture run that Appendix E.3 describes. Its numerator sums $1 + \sum_{i=1}^{w_0-1} \hat{p}_i$ over the rounds, and its denominator sums $C_{\mathrm{tgt}}(w_0)$ plus the fixed per-round cost removed from C_{tgt}.

B Analysis of the Throughput Objective

Because A_D is an integer, $\min(A_D, M) = \sum_{i=1}^{M} \mathbf{1}[A_D \geq i]$. Taking a conditional expectation and adding the token emitted after the accepted prefix gives G for nonterminal rounds (Section 3.1).

For $M' > M$ at a common state, suppress $\mathcal{I}$ and let $\Delta G = G(M') - G(M) = \sum_{i=M+1}^{M'} p_i$ and $\Delta L = L(M') - L(M)$. With positive latencies, M' has a higher G/L exactly when

$$\Delta G\, L(M) > G(M)\, \Delta L. \tag{8}$$

Replacing the current rate $G(M)/L(M)$ with the pooled rate ρ turns Equation (8) into the additive test $\Delta G > \rho\, \Delta L$. This is the form of both the draft-side score in Equation (5) and the routing correction in Equation (6). Under the additive test, even a draft token that is certain to be accepted helps only if its added latency is below $1/\rho$, the pooled latency per emitted token. Because the measured latency curve can be nonconvex or even nonmonotone, the selector scores every nominal verification length rather than stopping at the first unfavorable comparison.

For serial rounds at $B = 1$, the throughput of a stationary policy π converges, under standard ergodicity conditions, to $\mathbb{E}_\pi[R]/\mathbb{E}_\pi[T_{\mathrm{round}}]$, where T_{round} is the round latency. This limit is a ratio of expectations rather than an average of per-round ratios, so Section 3.1 defines the objective as pooled throughput. During decoding, later rounds depend on earlier choices, so ρ only approximates the optimal pooled throughput of the decoding process.

C Correctness Analysis

C.1 Triangular GDN factorization and state reconstruction

By induction on Equation (1), Equation (2) holds for every i. Substituting it for S_{i-1} into z_i and using $\alpha_i e^{g_{i-1}} = e^{g_i}$ gives

$$z_i + \sum_{j<i} \beta_i e^{g_i - g_j} (k_i^\top k_j) z_j = \beta_i v_i - \beta_i e^{g_i} S_0 k_i. \tag{9}$$

With the matrices of Section 2.2, stacking Equation (9) over $i = 1, \ldots, w$ gives the triangular system in Equation (3). In matrix form, $H = I + \mathrm{tril}_{-1}\left(\mathrm{diag}(\beta)(\Gamma \odot KK^\top)\right)$. Here Γ is the decay-aware causal mask of Gated DeltaNet (Yang et al., 2025), with $\Gamma_{ij} = e^{g_i - g_j}$ for $j \leq i$ and 0 otherwise, $\odot$ is the elementwise product, and tril_{-1} keeps the entries below the diagonal.

The first m equations depend only on the leading $m \times m$ block of H and the first m rows of the right-hand side. Thus appending or removing later tokens cannot change the compact factors of the prefix in exact arithmetic. Equation (2) at $i = m$ is Equation (7) written as a sum, so reusing the leading factors reconstructs the correct accepted state.

Substituting Equation (2) into $o_i = S_i q_i$ gives $o_i = e^{g_i} S_0 q_i + \sum_{j=1}^{i} e^{g_i - g_j} z_j (k_j^\top q_i)$. Stacking $q_i^\top$ and $o_i^\top$ as the rows of $Q \in \mathbb{R}^{w \times d_k}$ and $O \in \mathbb{R}^{w \times d_v}$ gives

$$O = \mathrm{diag}(e^g) Q S_0^\top + (\Gamma \odot QK^\top) Z. \tag{10}$$

Once Equation (3) yields Z, Equation (10) evaluates all GDN outputs $o_1, \ldots, o_w$ with dense matrix products. These outputs match the recurrent computation. A numerical implementation still requires stable accumulation and appropriate handling of extreme decay.

C.2 Correctness of prefix truncation under greedy decoding

The analysis uses five assumptions. Arithmetic is exact. Causal prefix invariance holds, meaning that removing suffix tokens does not change any target operator's computation for earlier tokens. Prefix states are recovered exactly: after each round, the commit sets every GDN recurrent state, convolution cache, and attention cache to the target's state after the committed tokens. Processing several tokens in one forward pass gives the same per-token results as processing them one at a time. Stage 1 resumes Layer 0 from the retained Stage-0 values of the first w tokens and reads no retained value of a later token. The fourth assumption holds for attention with a key–value cache, for the causal convolution of GDN, for position-wise layers, and for top-k MoE routing and expert computation without capacity limits. It also holds for the GDN recurrence, because Appendix C.1 shows that computing the updates of a block of tokens at once gives the same outputs and states as the token-by-token recurrence. These assumptions exclude MoE capacity rules that change an earlier token's expert computation when later tokens are removed.

Proposition 1. *Under these assumptions, greedy decoding with HybridSpec emits the same tokens as target-only greedy decoding for any verification length that each round selects, provided that both stop at the first EOS token or at the output limit.*

Proof. The proof maintains an invariant. After prefill and at the start of each round, the emitted tokens form a prefix of the target-only greedy continuation computed without stopping, and every persistent cache holds the state that the target alone computes for the tokens before the anchor. By causal prefix invariance and because Stage 1 reads only the retained values of the first w tokens, the extra Stage-0 token does not change the target logits for the anchor and the retained draft tokens. Because $m \leq w$, its retained factors never enter the commit. Because every cache holds this state and a multi-token pass matches one-token passes, the logits at the anchor equal those of target-only decoding. If the first draft token matches their argmax, accepting it extends the prefix, and the same argument applies to each later accepted draft token. At the first mismatch, or after the last retained draft token, the target output at that position is the next token of target-only decoding. Because prefix states are recovered exactly, committing exactly the anchor and the accepted draft tokens restores the invariant for the next round. Each round emits at least one token, so both processes truncate the same continuation at the first EOS token or at the output limit. □

In floating point, the multi-token kernels compute the same quantities with a different operation order and precision. This can flip the argmax when logits are nearly tied, after which later tokens can also differ. Appendix C.3 reuses the cache part of the invariant for temperature sampling.

C.3 Exactness under temperature sampling

Fix the prompt and consider a round with anchor x_0 and proposal prefix $d_1, \ldots, d_M$. For $i = 1, \ldots, M+1$, let p_i^{t} be the sampling distribution that the verifier computes at position i, that is, for the next token after the prompt, the tokens emitted before the round, and $d_1, \ldots, d_{i-1}$. It includes temperature scaling and any top-k or top-p filtering.

The verifier applies speculative sampling (Leviathan et al., 2023; Chen et al., 2023) with each draft token treated as a proposal of probability one. We call this the point-mass rule. For $i = 1, \ldots, M$, the verifier draws a uniform random number $\xi_i \in [0, 1]$ and accepts d_i if $\xi_i \leq p_i^{\mathrm{t}}(d_i)$. At the first rejection, it emits a token drawn from p_i^{t} with d_i removed and the remaining probabilities renormalized. The round then ends. If all M draft tokens are accepted, it emits a token drawn from p_{M+1}^{t}. We assume that each draw is independent of everything determined before it. SGLang implements this rule with its default acceptance thresholds, which our configuration uses.

Let y_i be the token that the round emits at position i. Let $\mathcal{F}_i$ be everything determined before the draws at position i, including the proposal block, M, and all earlier tokens and draws. If the round reaches position i, that is, if $i \leq M+1$ and $d_1, \ldots, d_{i-1}$ are accepted, then

$$\mathbb{P}(y_i = v \mid \mathcal{F}_i) = p_i^{\mathrm{t}}(v) \quad \text{for every token } v. \tag{11}$$

The draws at position i are independent of $\mathcal{F}_i$. If $i = M + 1$, the verifier draws y_i from p^{t}_{M+1}. If $i \le M$, the token d_i is emitted only through acceptance, which has probability $p^{\mathrm{t}}_i(d_i)$. Any other token v requires a rejection, with probability $1 - p^{\mathrm{t}}_i(d_i)$, and then a draw of v, with probability $p^{\mathrm{t}}_i(v)/\big(1 - p^{\mathrm{t}}_i(d_i)\big)$. The product is $p^{\mathrm{t}}_i(v)$. If $p^{\mathrm{t}}_i(d_i) = 1$, a rejection never occurs and $p^{\mathrm{t}}_i(v) = 0$ for every $v \neq d_i$, so the equation still holds. Because Equation (11) holds for every proposal block and every M, the verification length changes only how a token is produced, not its distribution.

Proposition 2. *Adopt the assumptions of Proposition 1. Assume also that verification uses the point-mass rule and that each round fixes its proposal block and M before its verification draws. Both may depend on the prompt and on everything determined in earlier rounds. Assume further that generation stops at the first EOS token or at the output limit, as in target-only sampling. Then the output has the same distribution as target-only sampling.*

Proof. The cache part of the invariant in Appendix C.2 does not use the acceptance rule. At the start of each round, every cache therefore holds the state that the target alone computes for the tokens before the anchor. With the other assumptions, and because the verifier applies the same temperature and filtering as target-only sampling, p^{t}_i is the target-only sampling distribution given the prompt and all tokens emitted before position i. By Equation (11), each emitted token follows this distribution given $\mathcal{F}_i$. This distribution depends only on the earlier emitted tokens, which $\mathcal{F}_i$ contains, so the same holds given only these tokens. The round and position that produce each emitted token are determined before its draws, so this holds for every emitted token in turn. By the chain rule, every sequence of emitted tokens has the same probability as under target-only sampling. Both processes apply the same stopping rule, so their outputs have the same distribution. □

Like Proposition 1, this result assumes exact arithmetic. In floating point, p^{t}_i differs slightly from the target-only sampling distribution, so the two output distributions agree only approximately.

In HybridSpec, the draft tokens are the drafter's top-1 tokens, and the verification length depends on the drafter's confidence and entropy values, the Layer-0 expert assignments of the Stage-0 tokens, and the remaining output budget, but on no verification draw. Proposition 2 therefore covers the full length policy, including the routing correction.

The argument depends on the point-mass rule. The general rule (Leviathan et al., 2023; Chen et al., 2023) samples d_i from a draft distribution q^{d}_i, accepts it with probability $\min\{1, p^{\mathrm{t}}_i(d_i)/q^{\mathrm{d}}_i(d_i)\}$, and otherwise draws from the normalized positive part of $p^{\mathrm{t}}_i - q^{\mathrm{d}}_i$. Under that rule, the emitted token follows p^{t}_i only on average over $d_i \sim q^{\mathrm{d}}_i$, not for each value of d_i. A length rule that reads d_i can then bias the output, because the draft tokens that reach verification may no longer follow q^{d}_i. The routing correction is such a rule, because it reads the expert assignment of d_{w_0} before deciding whether to add that token. The point-mass rule removes this dependence, since Equation (11) holds for every draft token.

D Experimental Details

D.1 Profile across verification lengths

The profile used Qwen3.6-35B-A3B with its DFlash drafter, SGLang 0.5.16, and one NVIDIA A100 GPU, at batch size one with a context limit of 8,192 tokens. SGLang used its default Triton (Tillet et al., 2019) MoE configuration, because no tuned configuration was available for this model. The workload was ten GSM8K test prompts (Cobbe et al., 2021), each decoded greedily for 32 new tokens. Each run set the DFlash block size to the verification length w, so $D = M = w - 1$.

At $w = 4$, 8, and 16, mean profiled GPU time per measured verification call was 1.795, 1.817, and 1.856 ms for the drafter and 9.527, 12.648, and 16.318 ms for target verification. The target-to-drafter time ratios were therefore $5.3\times$, $7.0\times$, and $8.8\times$. From $w = 4$ to 16, drafter time increased by 3.4%, whereas target time increased by 71.3%. Drafter and target intervals may overlap under the scheduler, so these values do not form an end-to-end latency decomposition.

Across the nine measured verification lengths, an unweighted ordinary least-squares fit to GDN time gives $1.89 + 0.154w$ ms, with coefficient of determination $R^2 = 0.999$. A log–log fit to MoE time gives $2.66w^{0.50}$ ms (log-space $R^2 = 0.976$). From $w = 4$ to 16, profiled GPU time for GDN and

Table 4: Qwen3.6-35B-A3B target-side GPU time across verification lengths, in milliseconds per measured verification call. The total is the union of slightly overlapping component intervals, so the components do not sum exactly to it.

w	Measured calls	Target total	GDN	MoE	Softmax attention	Other
4	101	9.527	2.498	5.331	0.686	1.019
5	85	10.055	2.659	5.691	0.696	1.017
6	80	10.811	2.835	6.248	0.710	1.026
7	74	11.876	2.984	7.171	0.705	1.026
8	72	12.648	3.096	7.826	0.698	1.037
10	71	14.075	3.413	8.934	0.705	1.033
12	71	14.659	3.734	9.165	0.719	1.050
15	71	16.078	4.224	10.085	0.722	1.057
16	70	16.318	4.350	10.187	0.725	1.065

MoE increased by 74.1% and 91.1%, whereas softmax-attention and other target-side time increased by 5.7% and 4.5%. Each verification length was measured in a separate run, so the fitted trends do not isolate the effect of length at a fixed decoding state.

Dense-target comparison. A separate comparison on one NVIDIA A100 GPU paired a dense Qwen3-4B target with DFlash at context lengths of 2,048, 8,192, and 32,768 tokens. Over 15 samples at each setting, the median latency of its captured target-verification CUDA Graph was 3.98%, 5.44%, and 6.08% higher at $w = 16$ than at $w = 4$ for the three context lengths.

Layer-0 routing as a proxy. To estimate the later-layer counts, a separate run on the calibration prompts verifies at length 16 with greedy decoding and at most 1,024 new tokens and records the top-8 experts of every token in every layer. For each draft position, let n_0 be the number of new Layer-0 experts that the token adds and $n_{>0}$ the number of new (layer, expert) pairs that it adds in layers 1–39. After the mean of each position is subtracted, n_0 and $n_{>0}$ have a correlation of 0.49 on Qwen3.6-35B-A3B and 0.52 on Qwen3.5-35B-A3B, and a least-squares fit gives 9.0 and 10.0 later-layer pairs per new Layer-0 expert.

D.2 Kernel microbenchmark details

The benchmark runs all 30 GDN layers of Qwen3.6-35B-A3B at batch size one, starting from random bfloat16 (BF16) projections and FP32 states. It captures the verification and the state commit of each implementation as CUDA Graphs. Instead of modeling acceptance, it fixes the number of committed tokens at $m = w$, where every draft token is accepted, or at $m = \lceil w/2 \rceil$. Each latency is the median of 30 samples of 10 graph replays. ReplaySSM uses its SGLang kernels with their default launch parameters, as in the end-to-end evaluation, and a ring buffer of at least twice the verification length. Because ReplaySSM folds this buffer into the state only when the buffer is flushed, its verification latency is the mean over 300 alternating verification and commit rounds. At the lengths in Table 5, a second sweep reproduced every total latency within 0.5%, except the default kernels at length 2, which differed by 13%. ReplaySSM was slower than the default SGLang kernels at every length. In a separate run with launch parameters tuned for batch size one, ReplaySSM was $1.05\times$–$1.71\times$ faster than the default kernels over lengths 2 to 16, and HybridSpec remained $1.16\times$–$2.42\times$ faster than it. At $m = \lceil w/2 \rceil$, the latency of the default kernels changes by less than 0.2%, because they write a state after every token during verification. The latencies of HybridSpec and ReplaySSM fall by 2%–6% and 14%–31%, respectively, because both fold only the committed tokens into the state. The all-accepted speedups over the default kernels in Section 4.4 are therefore conservative, whereas the all-accepted setting favors HybridSpec against ReplaySSM.

Triangular solve for short inputs. The matrix H in Equation (3) is unit lower triangular with at most 16 rows, so $X = I - H$ is strictly lower triangular and $X^{16} = 0$. Lengths 4, 6, and 8 use closed-form inverses of 4×4 blocks, and lengths 6 and 8 join two such blocks with one correction term. Lengths 9 to 16 use the exact identity $H^{-1} = (I + X + X^2 + X^3)(I + X^4 + X^8 + X^{12})$, which needs only matrix products. The other lengths, 2, 3, 5, and 7, form the inverse row by row in at most six steps on a matrix with at most seven rows, and each step computes a whole row at once.

Table 5: Latency (ms) of GDN verification plus accepted-state reconstruction for the 30 GDN layers of Qwen3.6-35B-A3B on one A100 when all $m = w$ input tokens are committed, that is, when every draft token is accepted. Speedups of HybridSpec are given for $m = w$ and for $m = \lceil w/2 \rceil$, when about half of the input tokens are committed. ReplaySSM uses its default launch parameters, as in Table 1.

	Latency (ms)			Speedup (×), $m = w$ / $m = \lceil w/2 \rceil$	
w	SGLang	ReplaySSM	HybridSpec	vs. SGLang	vs. ReplaySSM
2	0.4726	0.6926	0.3552	1.33 / 1.35	1.95 / 1.66
3	0.5012	0.8058	0.3721	1.35 / 1.38	2.17 / 1.90
4	0.5920	0.9192	0.3847	1.54 / 1.62	2.39 / 1.90
8	0.9733	1.5050	0.4366	2.23 / 2.37	3.45 / 2.55
12	1.4263	2.4859	0.4491	3.18 / 3.38	5.54 / 4.59
16	1.9111	2.9451	0.4728	4.04 / 4.29	6.23 / 4.56

For every length, all heads run in parallel, and the products with keys, values, and S_0 are matrix products over all tokens of the block, so verification remains block-parallel.

Kernel correctness. Before timing, the benchmark compares the HybridSpec kernels with the default SGLang kernels on one GDN layer at lengths 2, 4, 8, and 16. Each comparison runs two consecutive rounds that commit w and $w/2$ tokens, in both orders. The largest relative error in Frobenius norm is 4.3×10^{-3} for outputs and 1.7×10^{-3} for the reconstructed state. With FP32 instead of BF16 arithmetic in the HybridSpec kernels, these errors fall below 10^{-4} and 10^{-6}.

D.3 Correctness and task quality

We check correctness in two steps, both on Qwen3.6-35B-A3B under greedy decoding. The first step tests whether splitting verification at the router boundary changes outputs. It compares HybridSpec with and without the split, where the unsplit form runs all of Layer 0 before choosing the verification length. On 16 prompts that each generate 1,024 tokens, HybridSpec runs 3,114 target forward passes with verification lengths 3 to 16 and produces the same output tokens, acceptance histograms, and numbers of rounds in both forms. It never reads the selected length on the CPU. The second step tests task quality. On all 1,319 GSM8K test problems (Cobbe et al., 2021), HybridSpec solves 96.06% and vanilla DFlash 96.21%. The one-sided 95% lower confidence bound of the paired difference, -0.71 points, is above the noninferiority margin of -1 point fixed before evaluation. The two systems do not emit identical tokens, which reordered floating-point kernels can cause by flipping near-tied logits, so this step compares task quality rather than exact token agreement.

D.4 Memory for verification states

Because the default SGLang verifier stores the GDN state after every token, it preallocates, for each request slot, one state per GDN layer for every position of the full target input. For Qwen3.6-35B-A3B, whose 30 GDN layers each hold a $32 \times 128 \times 128$ state in FP32, this buffer takes $16 \times 30 \times 32 \times 128 \times 128 \times 4$ bytes, or 0.94 GB, per slot. At batch size one, SGLang allocates two slots, and the server log of our runs on one A100 reports 1.88 GB for the buffer. The same log reports a pool of 112,687 tokens for the attention caches of the target and the drafter, which take about 44 KB per token. The buffer therefore equals the caches of about 45,000 tokens, 40% of the pool. The HybridSpec kernels retain only the compact factors, which take about 0.01 GB for 16 tokens across the 30 layers even in FP32, so they do not need this buffer. Releasing it raises the maximum context length that fits in memory by about 40%.

The saving grows with the number of concurrent requests, because SGLang allocates one slot per request plus one. With four concurrent requests, vanilla DFlash allocates 4.69 GB for the buffer and has room for 12,472 cached tokens, whereas the batched version of HybridSpec in Appendix E.7 allocates no buffer and caches 122,512 tokens.

E End-to-End Evaluation Details

E.1 Models, baselines, and metrics

The targets are the official Qwen checkpoints of Qwen3.5-35B-A3B and Qwen3.6-35B-A3B in BF16 and of Qwen3.5-122B-A10B in GPTQ-Int4. The drafters are the official DFlash checkpoints that z-lab released for these targets, and all checkpoints come from Hugging Face. Each pair is calibrated and evaluated separately. Qwen3.5-122B-A10B uses two-way tensor parallelism across two GPUs without NVLink. The prompts come from seven of the eleven SPEED-Bench Qualitative categories (Section 4.1), and a seeded random split divides their 560 prompts into the calibration and test halves. Each request generates at most 1,024 new tokens, as in the SPEED-Bench evaluation of Fu et al. (2026), and thinking mode is enabled.

Baselines. D-Cut and EVICT follow their papers. At batch size one, the budget allocation of D-Cut reduces to a per-request pruning depth. As in its paper, D-Cut restricts this budget to four fixed fractions of the full target input, so it chooses among verification lengths 4, 8, 12, and 16. The draft tree of EVICT reduces to the single DFlash chain. EVICT + ReplaySSM applies the truncation of EVICT and verifies the retained prefix with ReplaySSM. The drafter generates all D draft tokens before any length decision, so all speculative systems incur the same drafting cost.

Metrics. Pooled decode throughput (TPS) is the number of tokens emitted after the prefill-produced anchor divided by the wall-clock decode time, pooled over the requests of a category. The decode time includes CPU-side control, graph dispatch, synchronization, and padding. SR is the ratio of pooled TPS to that of AR on the same prompts. AL is the number of emitted tokens divided by the number of verification rounds, which corresponds to G in Section 3.1. Category averages are arithmetic means over the seven categories.

E.2 Comparisons and ablations

End-to-end comparisons. All systems share prompts, stopping rules, precision, and output limits. Systems that select a verification length use the calibration split and a cost function calibrated for their own runtime. Each system runs in three sessions, each with a fresh server process, and the sessions are pooled.

Ablations. Each ablation keeps the drafter, prompts, precision, and calibration split of HybridSpec. To select the calibrated static length, a sweep on a subset of the calibration prompts keeps up to three fixed lengths within 5% of the fastest. These candidates and length 16 then run on all calibration prompts, and the one with the highest pooled throughput is used. The GDN variant replaces the HybridSpec verification and commit kernels with the default SGLang kernels.

E.3 Calibration data

Calibration runs. Calibration runs separately for each target–drafter pair and uses only the calibration prompts, which are disjoint from the test prompts. A capture run verifies all D draft tokens in every round with greedy decoding. It records the drafter's confidences and entropies, the number of new Layer-0 experts that each draft token adds, and the number of accepted draft tokens. These rounds train the acceptance model, give the Layer-0 averages in $\bar{n}_{\mathrm{all}}$, and are the rounds over which ρ is computed. Runs with a fixed verification length measure C_{tgt}, and a separate run at length 16 gives the later-layer averages, as Appendices A.2 and D.1 describe.

Acceptance-model training. Because the capture run verifies all D draft tokens, a shorter verification never censors the labels of later draft tokens. Training $\hat{c}_i$ uses the label $\mathbf{1}[A_D \geq i]$ among rounds with $A_D \geq i - 1$. Rounds cut short by EOS or the output limit are excluded.

E.4 Correctness checks

The GSM8K check compares HybridSpec with vanilla DFlash. HybridSpec solves 1,267 problems and vanilla DFlash solves 1,269, a paired difference of -0.15 percentage points. The lower confi-

dence bound is a normal approximation from the 20 problems on which the two systems disagree, 11 solved only by vanilla DFlash and 9 only by HybridSpec. A second paired run gives 96.21% without the routing correction and 96.13% with it.

The router-boundary check uses 16 prompts spanning math, coding, writing, and chat, and each request ignores EOS and generates exactly 1,024 tokens. The split and unsplit forms also process the same numbers of tokens before and after the length decision. After Stage 0 finishes, overwriting its retained tensors for tokens beyond the selected length leaves every output unchanged. The run records no graph recapture and no rerun of the Layer-0 router.

E.5 Unseen categories

The four SPEED-Bench Qualitative categories outside the study, humanities, QA, STEM, and multilingual, contribute all 80 of their prompts each. The prompts are rendered as in the main evaluation, and none matches a study prompt. HybridSpec uses the parameters of Table 1 without recalibration. HybridSpec is faster than vanilla DFlash in every category, by $1.28\times$ for multilingual to $1.31\times$ for QA. Its mean verification length is 5.7, compared with 6.2 on the test prompts.

E.6 Temperature-one sampling

Table 6 repeats the comparison of Table 1 for the two 35B pairs under temperature-one sampling without top-p or top-k truncation. HybridSpec keeps the parameters calibrated under greedy decoding. Its mean SR is $1.57\times$ on Qwen3.5-35B-A3B and $1.60\times$ on Qwen3.6-35B-A3B, compared with $1.20\times$ and $1.21\times$ for vanilla DFlash and $1.34\times$ and $1.38\times$ for EVICT, the strongest baseline. HybridSpec has the highest SR in every category. Sampling lowers the AL of every speculative method, so every SR is lower than under greedy decoding. The lead of HybridSpec over vanilla DFlash is $1.31\times$ and $1.32\times$, compared with $1.30\times$ and $1.33\times$ under greedy decoding.

Table 6: End-to-end comparison under temperature-one sampling, laid out as in Table 1.

	Coding		Math		Reasoning		RAG		Summ.		Writing		Roleplay		Avg.	
Method	AL	SR	AL	SR	AL	SR	AL	SR	AL	SR	AL	SR	AL	SR	AL	SR
Qwen3.5-35B-A3B																
AR target	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×
Vanilla DFlash	5.29	1.56×	4.18	1.23×	3.88	1.15×	5.20	1.50×	4.11	1.19×	3.23	0.96×	2.63	0.81×	4.07	1.20×
D-Cut	4.70	1.58×	3.84	1.33×	3.78	1.32×	4.97	1.62×	3.88	1.31×	3.13	1.11×	2.54	0.94×	3.84	1.31×
EVICT	4.81	1.61×	3.97	1.37×	3.86	1.35×	5.03	1.63×	3.96	1.33×	3.17	1.13×	2.56	0.96×	3.91	1.34×
ReplaySSM	5.06	1.45×	3.96	1.14×	4.01	1.16×	5.28	1.48×	4.12	1.16×	3.26	0.95×	2.67	0.81×	4.05	1.16×
EVICT + ReplaySSM	4.75	1.57×	3.86	1.34×	3.81	1.32×	4.89	1.58×	3.96	1.33×	3.12	1.10×	2.54	0.94×	3.85	1.31×
HybridSpec	4.88	1.91×	3.86	1.59×	3.82	1.57×	4.79	1.84×	3.98	1.57×	3.16	1.33×	2.59	1.16×	3.87	1.57×
Qwen3.6-35B-A3B																
AR target	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×	*n/a*	1.00×
Vanilla DFlash	4.77	1.42×	4.40	1.32×	4.22	1.26×	4.99	1.44×	4.25	1.24×	3.17	0.96×	2.70	0.83×	4.07	1.21×
D-Cut	4.63	1.58×	4.31	1.50×	3.96	1.38×	4.90	1.60×	4.13	1.40×	3.10	1.11×	2.61	0.97×	3.95	1.36×
EVICT	4.73	1.61×	4.25	1.49×	4.06	1.43×	4.86	1.60×	4.12	1.40×	3.09	1.12×	2.62	0.99×	3.96	1.38×
ReplaySSM	4.70	1.36×	4.43	1.29×	4.08	1.19×	4.99	1.40×	4.36	1.23×	3.15	0.92×	2.69	0.81×	4.06	1.17×
EVICT + ReplaySSM	4.65	1.53×	4.26	1.44×	3.99	1.36×	4.87	1.55×	4.14	1.36×	3.07	1.07×	2.62	0.95×	3.94	1.32×
HybridSpec	4.56	1.83×	4.19	1.73×	4.20	1.72×	4.78	1.83×	4.07	1.61×	3.15	1.35×	2.49	1.12×	3.92	1.60×

E.7 Batched decoding

The batched version of HybridSpec extends the length policy to a batch of B requests. All requests in a round share one verification length, so that the batch runs as one CUDA Graph of fixed shape. The policy chooses this length from $\{2, 4, 8, 12, 16\}$ by maximizing $\sum_{b=1}^{B} \hat{G}_b(w) - \rho_B\, C_B(w)$, where $\hat{G}_b(w)$ is the expected number of tokens that request b emits at length w, $C_B(w)$ is the measured latency of a round with B requests, and ρ_B is the Dinkelbach rate for batches of B requests. At $B = 1$, this rule reduces to Equation (5). A request that would prefer a shorter length still verifies all w tokens, because the batch already pays for w positions and extra verified tokens can only add accepted tokens. The length is fixed before verification, so the batch preserves the target's outputs in exact arithmetic. Stage 0 and the routing correction carry over: the correction counts the Layer-0 experts that the next candidate length adds to the union over the batch and expands to that length

if the added tokens are worth their latency. The latencies $C_B(w)$, the rates ρ_B, and the correction parameters are calibrated on the calibration prompts for batches of two and four requests, and a batch of three runs as a padded batch of four.

On the 280 test prompts of Qwen3.6-35B-A3B under greedy decoding, four concurrent clients each send a new request as soon as the previous one finishes. The batched version then emits 497 tokens per second in total, $1.10\times$ the 451 of vanilla DFlash and $1.39\times$ the 358 of autoregressive decoding, while vanilla DFlash with the HybridSpec GDN kernels emits 472. As at batch size one, it accepts fewer tokens per round than vanilla DFlash, 4.25 versus 4.57, yet emits more tokens per second. With the HybridSpec kernels, the batched runtime also has room for 122,512 cached tokens, 9.8 times the 12,472 tokens of vanilla DFlash, as Appendix D.4 details. The gain is smaller than at batch size one mainly because the length policy helps less: one length from five candidates must serve all requests, so it follows each request less closely.