# H-SPEC: PARALLEL SPECULATIVE DECODING WITHOUT A DRAFTER-SIDE KV CACHE

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

## ABSTRACT

Block diffusion drafters such as DFlash and DSpark reduce drafting latency in speculative decoding by predicting multiple future tokens in parallel. These drafters condition their prediction on the target model by projecting target hidden states at every input position into a separate drafter-side KV cache. This drafter-side cache consumes extra GPU memory per request, reducing maximum concurrency. Maintaining separate caches can also lead to fragmentation or padding in paged serving systems, increasing memory waste. Direct target KV reuse alone eliminates this extra cache but reduces draft quality at later block positions.

We propose a hybrid target-context injection method that complements direct target KV reuse with target hidden states only at the last input position, requiring no separate drafter-side KV cache. Based on this design, we propose H-Spec, a hybrid Mamba-attention parallel drafter. Mamba modules are initialized with projected last-token target hidden states, while attention modules reuse target KVs in place. Despite its recurrent formulation, Mamba's parallel scan allows H-Spec to preserve block-parallel drafting. Across three targets and three workloads, H-Spec achieves 5.1–17.3% higher peak throughput across the evaluated concurrency levels and 4.3–24.6% lower average KV utilization than the best baseline for each metric. Despite having no drafter-side KV cache, H-Spec preserves draft quality, improving mean accepted length by 5.0–13.3% over the best baseline.

## 1 INTRODUCTION

Large language models (LLMs) generate text *autoregressively*: each output token is conditioned on all preceding tokens. Speculative decoding (Leviathan et al., 2023) has emerged as a primary solution to this sequential bottleneck: a lightweight draft model predicts the next $k$ tokens; the larger target model then verifies the $k$ tokens in a single forward pass. Traditional drafters predict future tokens autoregressively (Cheng et al., 2024; Li et al., 2024a;b; 2025); recent works instead leverage *block diffusion models* as drafters to predict $k$ tokens in parallel, further improving speedup (Chen et al., 2026; Huang et al., 2026; Cheng et al., 2026; Karimi et al., 2026; Inco AI, 2026).

Block diffusion drafters incorporate target-model context through *KV injection*, keeping their predictions closely conditioned on the target model but requiring a separate drafter-side KV cache. For every input token, target-model hidden states are projected into Key and Value representations, which the drafter consumes as prefix context when predicting the next block. This process introduces several challenges for concurrent serving. First, drafter-side KV memory grows with concurrency and context length, reducing serving headroom and maximum concurrency under fixed GPU memory capacity. Second, different target and drafter KV entry sizes can cause padding or oversized pages in serving systems, increasing memory waste (Zhang et al., 2025). Third, different KV access patterns complicate cache management, which can introduce further memory waste (vLLM, 2026). Therefore, we ask: ***can we preserve strong target conditioning in the drafter without using a separate drafter-side KV cache?***

A straightforward solution to the above question is to let the block drafter reuse the target model's existing KV cache directly. To evaluate this approach, we modify the KV injection mechanism in a representative block drafter, DFlash (Chen et al., 2026), to reuse target KVs in place. Under a matched training recipe, direct target-KV reuse largely preserves draft acceptance at early positions in the block compared to vanilla DFlash, but consistently degrades at later positions across multiple



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.

evaluated tasks (Figure 3), suggesting weaker target-context conditioning than hidden-state-based injection. Eliminating the drafter-side KV cache, therefore, requires more than direct target-KV reuse to sustain draft quality throughout the block.

Motivated by the study above, we propose a hybrid context-injection method that achieves strong target conditioning without the drafter-side KV cache. Specifically, we combine two complementary sources of target-model context: in-place target KV reuse and target-model hidden states only at the last input position. Under causal attention, hidden states at the last input position are conditioned on the entire prefix, providing a full input summary with $\mathcal{O}(1)$ storage; the target model's existing KVs provide fine-grained position-wise information at no additional memory cost.

We propose **H-Spec**, a hybrid Mamba-attention parallel drafter that consumes the two target-context sources through different modules. The Mamba modules (Gu & Dao, 2024; Dao & Gu, 2024) are initialized with projected target-model hidden states at the last input position, as the prefix summary provided in these hidden states aligns naturally with the role of Mamba's recurrent state. Attention modules consume the target-model KV cache in place, leveraging its position-wise context. Despite its recurrent formulation, Mamba evaluates all draft positions through a parallel scan, allowing H-Spec to preserve block-parallel drafting.

We compare H-Spec with reproduced baselines under a unified training recipe. Across three target models and math/coding/chat workloads, H-Spec improves generation throughput while reducing KV cache utilization across concurrency levels. Figure 1 shows one example setting, where the upper-left Pareto frontier indicates faster generation with lower KV memory use. This result demonstrates H-Spec's better scalability than drafters that maintain a separate KV cache. At batch size 1, H-Spec improves mean accepted length and inter-token latency (ITL) speedup over the best baseline, showing that hybrid context injection provides effective target conditioning.

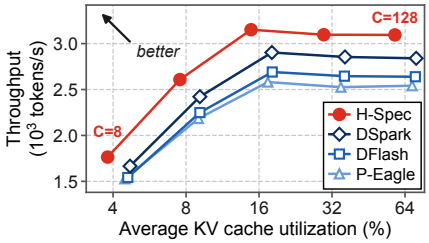



Figure 1: **H-Spec achieves the highest throughput and lowest KV utilization** across 8–128 concurrent requests on MATH workload/Qwen3-8B target.

To summarize, we identify the drafter-side KV cache as a systems bottleneck: it increases memory consumption which reduces maximum concurrency, and complicates cache management in serving systems. We eliminate this extra cache with a hybrid context-injection mechanism that preserves effective target conditioning and improves serving efficiency. More broadly, our results establish concurrent-serving efficiency as an important consideration in speculative drafter design.

## 2 Background and Motivation

**Speculative decoding speedup.** In each speculative decoding step, a lightweight draft model $\mathcal{M}_D$ proposes $k$ future tokens, which the target model $\mathcal{M}_T$ verifies in one forward pass (Leviathan et al., 2023). The achieved speedup depends on both draft acceptance and drafter runtime (details in Appendix §A). Following Sadhukhan et al. (2025), the average per-token latency is $L_{sd} = (T_D(k) + T_T)/\tau$, where $T_D(k)$ is the time to draft $k$ tokens, $T_T$ is the target verification time, and $1 \leq \tau \leq k+1$ is the mean accepted length (MAL), including the target-generated "bonus" token. The overall speedup is $L_T/L_{sd}$, where $L_T$ is per-token latency without speculative decoding.

**Block diffusion for parallel drafting.** Traditional drafters output the next $k$ predictions autoregressively (Cheng et al., 2024; Li et al., 2024a;b; 2025). Recent block diffusion drafters improve draft speed by generating $k$ tokens in one forward pass. A representative solution, DFlash (Chen et al., 2026), uses a stack of Transformer layers with attention and MLP modules. Domino (Huang et al., 2026) and DSpark (Cheng et al., 2026) further append a lightweight sequential head that refines each draft position based on preceding information, combining parallel drafting with causal correction.

**KV injection in block drafting.** LLM hidden representations capture long-range dependencies and encode information useful for predicting future tokens (Samragh et al., 2025). Existing block drafters therefore employ a KV injection mechanism that leverages target hidden states (Figure 2). At every input position, hidden states from selected target-model layers are fused and projected into KV representations. The drafter attends to these KVs as prefix context when predicting the next block, keeping draft predictions closely conditioned on the target model and improving acceptance.

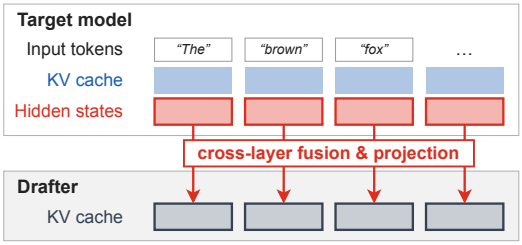



Figure 2: **KV injection in existing block diffusion drafters.** Target hidden states at each input position are projected to a drafter KV entry, separate from the target model's.

Table 1: **Drafter-side KV adds memory overhead.** Per-token KV entry size across target models and their corresponding official DFlash drafters. KVs are in bf16.

| Target model | Target KV /token | Draft KV /token | KV/token increase |
|---|---|---|---|
| Qwen3-4B | 144 KiB | 20 KiB | 1.139× |
| Qwen3-8B | 144 KiB | 20 KiB | 1.139× |
| Llama3.1-8B-Instruct | 128 KiB | 20 KiB | 1.156× |
| Qwen3.5-27B | 64 KiB | 24 KiB | 1.375× |
| Qwen3.5-397B-A17B | 30 KiB | 24 KiB | 1.800× |

**Challenges.** KV injection provides strong target-context conditioning; however, it requires a separate drafter-side KV store in addition to the target's (Table 1). This creates several challenges:

- **Reduced serving headroom.** The drafter-side KV cache scales in memory footprint with batch size and context length, reducing headroom for serving more requests concurrently under fixed GPU memory capacity. Recent drafters adapt sliding-window attention (SWA), but SWA only reduces memory for requests exceeding the window size. The window size should also be large enough to support effective prefix context, e.g., 2,048 tokens (Inco AI, 2026), leaving non-negligible per-request KV memory even when capped by SWA. Additionally, LLMs increasingly adopt memory-efficient architectures to support higher concurrency (Lenz et al., 2025; NVIDIA et al., 2025; Qiu et al., 2026); maintaining a separate drafter-side KV cache offsets these savings.
- **Different KV entry sizes.** In paged serving systems, GPU memory for the KV cache is divided into blocks for efficient allocation (Kwon et al., 2023). Target and draft models often differ in per-token KV size (Table 1), and the block layout must reconcile these differences. For example, vLLM requires all blocks to have the same page size, i.e., physical size in bytes (vLLM, 2026). A page size that is a common multiple of different KV entry sizes avoids padding but can be excessively large, leaving blocks partially filled (Zhang et al., 2025). Alternatively, padding can also reconcile heterogeneous KV sizes but still leaves some allocated memory unused.
- **Different access patterns.** Target and draft models can differ in KV access patterns (e.g., full vs. sliding-window attention), complicating memory management and potentially wasting memory. For example, vLLM groups layers by KV access pattern and requires a uniform page size across groups (vLLM, 2026), which can lead to unused padded layer slots when groups have unbalanced numbers of layers. Appendix §B elaborates on this challenge and presents a case study.

These challenges motivate us to design a new drafter that needs no separate KV cache while retaining key properties of existing drafters, including draft quality and parallel drafting capability.

## 3 Method: H-Spec

### 3.1 A straightforward solution: DFlash with direct target-KV reuse

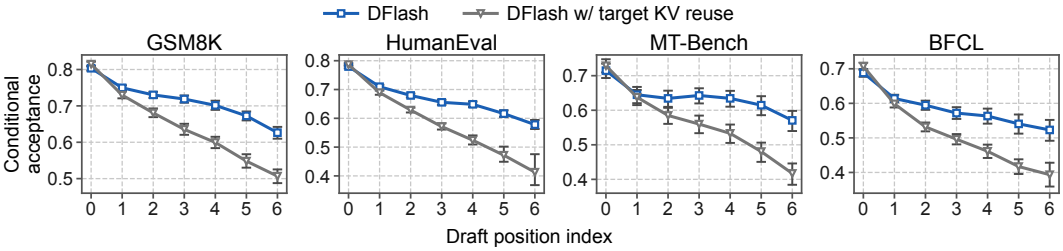



Figure 3: **Reusing target KV alone reduces draft quality at later positions.** Per-position conditional acceptance for DFlash and the target-KV-reuse variant on Qwen3-8B. Error bars are 95% CIs over 10,000 request-level bootstrap resamples; other CIs in the paper are computed similarly.

The most straightforward way to eliminate the drafter-side KV cache is to let the drafter attend directly to the target model's existing KV cache. We evaluate this approach on DFlash, which extracts hidden states from five selected target-model layers, then fuses and projects them into drafter-side

KVs for its five-layer block diffusion drafter. Our variant retains five layers, but each layer directly attends to the KV cache of one selected target layer. We train both under a matched recipe.

As shown in Figure 3, direct target-KV reuse largely retains DFlash's draft accuracy at early positions (i.e., indices 0 and 1), with differences of $-2.07$ to $+1.91$ pp. in conditional acceptance rates on four domains. However, the gap widens at later positions, reaching up to 6.3/12.5/16.5 pp. at indices 2/4/6 across these domains. Overall, direct target-KV reuse does not sustain the draft quality achieved by conventional hidden-state-based KV injection throughout the block.

### 3.2 Hybrid target-context injection

An important difference between hidden states and KV representations is the contextual information they provide for drafting. At each input position, hidden states integrate information from preceding tokens and encode useful information for predicting future tokens (Samragh et al., 2025). In contrast, target KVs provide position-wise information for retrieval by future queries, but lack an explicit contextual summary over the entire input.

Based on this observation, we propose a novel hybrid target-context injection method that complements direct KV reuse with target-model hidden states only at the last input position. Under causal attention, hidden states at the last position are conditioned on all preceding input tokens, providing the drafter with a compact summary of the full input sequence. Meanwhile, target KVs provide fine-grained position-wise context that complements this compact summary. Importantly, processing and storing hidden states at only the last input position incurs $\mathcal{O}(1)$ overhead relative to input length, eliminating the $\mathcal{O}(N)$ drafter-side KV cache. The remaining challenge is to design a parallel drafter architecture that can make effective use of both context sources.

### 3.3 H-Spec architecture

We propose H-Spec (Figure 4), a parallel drafter based on our hybrid target-context injection. Each context source is consumed by a module aligned with the information it provides: Mamba is initialized from projected last-token hidden states, while attention consumes target KVs in place.

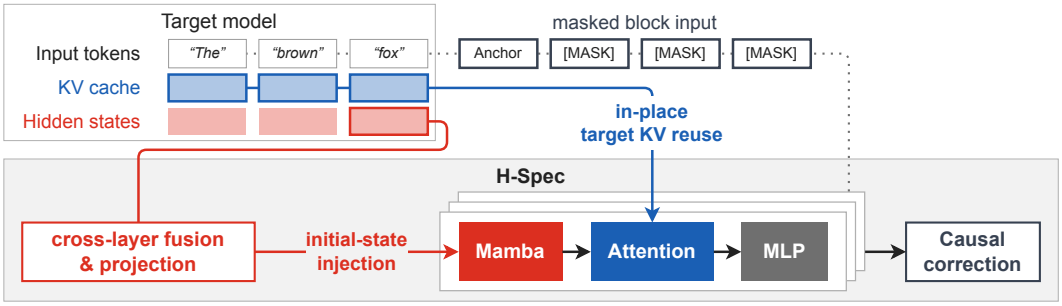



Figure 4: **H-Spec overview.** H-Spec initializes Mamba modules from projected target hidden states only at the last input position and reuses target KVs in place through attention. A lightweight causal correction follows the hybrid backbone to improve inter-token dependencies.

#### 3.3.1 Mamba with state initialization from last-token hidden states

Mamba is a recurrent selective state-space architecture that compresses preceding context into a fixed-size state, which is updated at each position and carried forward to condition subsequent tokens (Gu & Dao, 2024). Given input $x_i$ and preceding state $S_{i-1}$, the state is updated as $S_i = \bar{A}_i S_{i-1} + \bar{B}_i x_i$, where $\bar{A}_i$ and $\bar{B}_i$ are input-dependent. Mamba-2 keeps the underlying selective recurrent formulation but uses a more structured per-head state transition under its State Space Duality (SSD) (Dao & Gu, 2024). H-Spec thus uses Mamba-2 for its improved runtime efficiency.

**Initial state injection from last-token hidden states.** As discussed, target hidden states at the last input position are conditioned on the full preceding context, and are naturally aligned with the role of Mamba's recurrent state. We therefore use these hidden states to initialize Mamba: the last-token hidden states are fused across selected target-model layers and projected to serve as Mamba's initial recurrent state, conditioning subsequent draft generation on the target model's full input context.

**Cross-layer hidden state fusion.** Let $h_n^{(m)}$ and $\mathcal{M}$ denote the target-model hidden state at the last input position $n$ from layer $m$, and the set of selected target layers for context injection, respectively. H-Spec obtains the initial Mamba-2 state $S_{\text{init}}$ via:

$$F_n(\mathcal{M}) = \text{Proj}_r\Big(\text{Concat}\Big(\{h_n^{(m)}\}_{m\in\mathcal{M}}\Big)\Big), \qquad S_{\text{init}} = \text{Proj}_{d_S}(F_n(\mathcal{M}))$$

where $r$ is the intermediate rank used for cross-layer fusion and $d_S$ denotes the flattened dimensionality of Mamba-2's recurrent state. H-Spec shares the same $\text{Proj}_r$ and $\text{Proj}_{d_S}$ weights across all drafter layers, so the initial state is only computed once per drafter forward pass.

**Parallel-drafting compatibility.** Despite its recurrent nature, Mamba can evaluate sequence positions in parallel due to its *linear* state recurrence. In H-Spec, the Mamba kernel starts from the injected initial state and computes the state evolution across all draft positions in one pass, avoiding sequential execution. This allows the hybrid Mamba-attention backbone to retain parallel drafting.

**Input-dependent selectivity.** Following existing block diffusion drafters, we do not add positional embeddings to the masked block input. Although the first Mamba module receives the same mask-token embedding at each draft position, its recurrent scan produces position-dependent hidden states. Attention modules additionally incorporate positional information through RoPE (§3.3.2). As a result, Mamba modules after the first receive hidden states that vary across draft positions, preserving input-dependent selectivity across the block.

### 3.3.2 Attention modules with in-place target-KV reuse

In each H-Spec layer, the attention module follows the Mamba module, providing fine-grained, position-wise context complementary to Mamba's compact recurrent formulation.

**KV dimension.** H-Spec's attention modules match the target model's KV dimensions, ensuring that target KVs can be reused as is. We also match H-Spec's number of attention and KV heads to the target, although in practice reusing only a subset of target KV heads may suffice and could further reduce attention overhead; we leave this design space for future work.

**Layer-wise KV mapping.** Each H-Spec layer's attention module consumes the KV from one target model layer. This allows H-Spec to incorporate target-model information from multiple depths while preserving in-place KV reuse. Cross-layer KV fusion would instead require storing an additional fused KV in memory, reintroducing the drafter KV cache.

**Attention mask.** We use causal attention over the masked block, with reused target KVs serving as prefix context to keep the draft closely conditioned on the target model. We use sliding-window attention (SWA) with a 2,048-token window for long-context stability (Eldenk et al., 2026).

**Positional alignment for masked block inputs.** Because target KVs are stored post-RoPE in memory, masked-block Q/K vectors must use the target model's RoPE $\theta$ with position indices that continue from the target prefix, ensuring consistent positional encoding across the prefix and draft block.

### 3.3.3 Causal correction

Block diffusion models process all drafting positions in parallel but can weaken causal dependencies between adjacent positions, reducing coherence in the drafted sequence. Recent works therefore augment the parallel backbone with auxiliary components that strengthen inter-position dependencies. H-Spec specifically adopts DSpark's Markov head (Cheng et al., 2026) after the hybrid backbone, introducing a first-order Markov dependency by adjusting the predicted logits at each draft position based on the prediction at the immediately preceding position.

Causal correction is part of H-Spec's architecture but not a contribution of this work, as the hybrid backbone is independent of, and can be paired with, any specific mechanism. We leave evaluating other causal-correction mechanisms with the hybrid backbone as important future work.

**Architecture configuration.** To ensure a fair comparison, we match H-Spec's trainable and total parameter counts to the recommended 5-layer DFlash/DSpark settings (Chen et al., 2026; Cheng et al., 2026). Because each H-Spec layer additionally contains a Mamba module, we use a shallower four-layer drafter: three full Mamba-attention-MLP layers and one partial Mamba-MLP layer. Retaining Mamba rather than attention in the partial layer keeps H-Spec closest to the baseline parameter budget. We detail the remaining H-Spec configurations and dimension settings in Appendix §C.1.

# 4 EVALUATION

**Target models.** We conduct experiments on Llama3.1-8B-IT (Grattafiori et al., 2024) and Qwen3 4B and 8B (Yang et al., 2025), covering both general-purpose and reasoning LLM families.

**Baselines and training configurations.** We compare H-Spec with parallel drafters: P-Eagle (Hui et al., 2026), DFlash (Chen et al., 2026), and DSpark (Cheng et al., 2026). Since the official baseline checkpoints use heterogeneous training recipes, we reproduce all baselines and H-Spec using Speculators (Red Hat, 2025), the speculative-decoding training framework for the vLLM ecosystem, under a unified training recipe for controlled comparison. We train all drafters for 5 epochs on 100K samples drawn from Magpie (Xu et al., 2025) and Ultrachat (Ding et al., 2023), with matched optimizer steps. For Qwen3 targets, we use thinking-enabled responses generated by the target models for better drafter alignment. Appendix §C details remaining baseline and training settings.

**Metrics.** We report closed-loop throughput and average KV cache utilization (%) during request serving, across concurrency levels. Together, these metrics show how end-to-end generation speed and KV cache pressure scale with concurrency.

**Additional settings.** Unless otherwise specified, we use the default sampling parameters per target model (temp=0.6/topP=0.9 for Llama3.1; temp=0.6/topP=0.95/topK=20 for Qwen3), and 7 drafted tokens per step. We report sensitivity on these settings in §4.4. We measure generation speed and KV cache utilization using vLLM 0.25.1 (Kwon et al., 2023) on one NVIDIA A100 80G GPU.

## 4.1 SERVING PERFORMANCE USING VLLM

We evaluate workloads from three domains following DSpark's setting: MATH (Hendrycks et al., 2021), LiveCodeBench (Jain et al., 2025), and Alpaca (Taori et al., 2023), using 1,000 requests each.

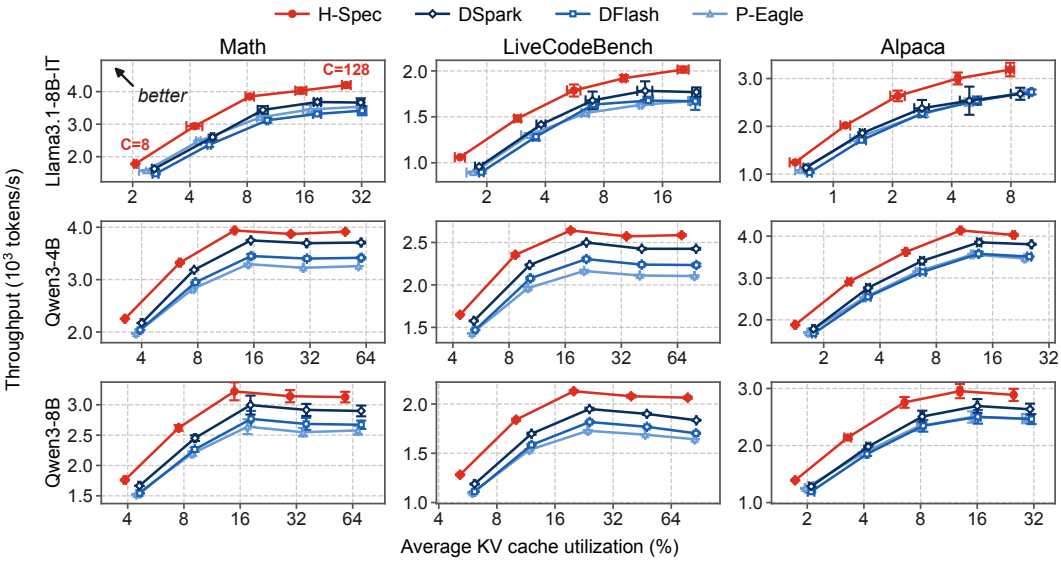



Figure 5: **Serving throughput and KV cache utilization under different concurrency.** Within a curve, five marks from left to right correspond to $C = 8/16/32/64/128$ concurrent requests. Results are averaged from 3 repeats, and error bars show standard deviation (st.d.).

H-Spec achieves the highest throughput and lowest KV cache utilization at every evaluated concurrency level (Figure 5). Relative to the best baseline, peak throughput improves by 13.0–17.3%, 5.1–7.4%, and 7.6–9.7% across the three tasks for the three targets, respectively. By eliminating the drafter-side KV cache, H-Spec reduces average KV cache utilization by 4.3–24.6% relative to the lowest-utilization baseline in each setting, leaving more headroom for higher concurrency. H-Spec also avoids potential memory waste from padding and fragmentation due to heterogeneous target and drafter KV caches; Appendix §E.1 further analyzes the padding behavior in vLLM.

**Preliminary comparison with DFlash-2.** Appendix §E.2 compares H-Spec with DFlash-2 (Inco AI, 2026) on Qwen3-8B. H-Spec achieves higher throughput and lower KV-cache utilization across

all tasks and concurrency levels. These gains are consistent with DFlash-2 retaining DFlash's KV injection mechanism, including the drafter-side cache and its associated serving overheads.

### 4.2 Single-request accepted length and speedup

We evaluate the mean-accepted-length (MAL, denoted $\tau$) and inter-token-latency (ITL) speedup relative to no-speculative-decoding under batch size 1, measuring draft quality and speedup separate from concurrent contention. We use a broader 8-task set covering math, QA, chat, RAG, summarization, translation, code, and tool calling. The first 6 tasks are adopted from Spec-Bench (Xia et al., 2024), while we add HumanEval (Chen et al., 2021) and BFCL (Patil et al., 2025) to broaden coding and tool-calling coverage.

Table 2: **H-Spec improves accepted draft length and generation speed under batch size 1.** MAL ($\tau$) and speedup (*Spd.*) at batch size 1 over no-speculative-decoding. Max 4,096 tokens per response. H-Spec results are tinted; best values are bolded. Statistical uncertainty reported in §D.1.

| | **Math** | | **QA** | | **Chat** | | **RAG** | | **Summ.** | | **Transl.** | | **Code** | | **Tool** | | **Avg.** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| L-8B-IT | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 3.08 | *2.82*× | **2.73** | ***2.34***× | 2.59 | *2.14*× | 2.62 | *2.02*× | 2.44 | *1.89*× | 1.66 | *1.35*× | 3.62 | *2.59*× | 2.78 | *2.12*× | 2.69 | *2.16*× |
| DFlash | 2.99 | *2.57*× | 2.19 | *1.65*× | 2.69 | *2.30*× | 2.29 | *1.83*× | 2.11 | *1.67*× | 1.44 | *1.20*× | 3.46 | *2.88*× | 2.63 | *2.09*× | 2.47 | *2.03*× |
| DSpark | 3.27 | *2.77*× | 2.63 | *2.32*× | **3.03** | *2.49*× | 2.48 | *1.95*× | 2.27 | *1.79*× | 1.46 | *1.20*× | 3.82 | *3.09*× | 2.77 | *2.23*× | 2.72 | *2.23*× |
| H-Spec | **4.24** | ***3.23***× | 2.40 | *2.21*× | 2.99 | ***2.64***× | **2.96** | ***2.31***× | **2.77** | ***2.21***× | **1.80** | ***1.43***× | **4.29** | ***3.53***× | **3.19** | ***2.54***× | **3.08** | ***2.51***× |
| Q-4B | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 3.76 | *2.77*× | 2.84 | *2.15*× | 3.04 | *2.18*× | 2.92 | *2.08*× | 2.39 | *1.74*× | 2.69 | *2.10*× | 3.41 | *2.37*× | 2.87 | *2.08*× | 2.99 | *2.18*× |
| DFlash | 3.74 | *2.93*× | 2.67 | *2.22*× | 2.91 | *2.31*× | 2.68 | *2.12*× | 2.28 | *1.80*× | 2.43 | *2.03*× | 3.39 | *2.52*× | 2.80 | *2.20*× | 2.86 | *2.26*× |
| DSpark | 4.02 | *3.04*× | 2.90 | *2.31*× | 3.16 | *2.43*× | 2.92 | *2.22*× | 2.45 | *1.87*× | 2.58 | *2.11*× | 3.69 | *2.70*× | 2.99 | *2.30*× | 3.09 | *2.37*× |
| H-Spec | **4.17** | ***3.19***× | **3.04** | ***2.43***× | **3.32** | ***2.58***× | **3.12** | ***2.32***× | **2.62** | ***2.01***× | **2.74** | ***2.23***× | **3.83** | ***2.82***× | **3.09** | ***2.42***× | **3.24** | ***2.50***× |
| Q-8B | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 3.74 | *2.91*× | 2.65 | *2.19*× | 2.98 | *2.33*× | 2.83 | *2.16*× | 2.36 | *1.88*× | 2.63 | *2.14*× | 3.36 | *2.56*× | 2.83 | *2.22*× | 2.92 | *2.30*× |
| DFlash | 3.68 | *2.97*× | 2.58 | *2.17*× | 2.85 | *2.35*× | 2.58 | *2.11*× | 2.22 | *1.82*× | 2.31 | *1.99*× | 3.29 | *2.60*× | 2.68 | *2.21*× | 2.77 | *2.28*× |
| DSpark | 3.91 | *3.17*× | 2.69 | *2.27*× | 3.11 | *2.52*× | 2.74 | *2.21*× | 2.34 | *1.89*× | 2.42 | *2.03*× | 3.57 | *2.81*× | 2.84 | *2.32*× | 2.95 | *2.40*× |
| H-Spec | **4.13** | ***3.36***× | **2.97** | ***2.44***× | **3.32** | ***2.69***× | **3.03** | ***2.46***× | **2.63** | ***2.12***× | **2.74** | ***2.31***× | **3.82** | ***3.00***× | **3.06** | ***2.50***× | **3.21** | ***2.61***× |

As shown in Table 2, H-Spec improves the average unweighted $\tau$ across eight tasks by 13.3%, 5.0%, and 8.7%, and Spd. by 12.6%, 5.3%, and 8.7% over the best baseline for Llama, Qwen3 4B, and 8B targets, respectively. Paired bootstrap 95% CIs for all six comparisons sit above zero, indicating that H-Spec's gains are statistically significant (Appendix §D.1). Unlike the straightforward solution presented in §3.1, these results show that H-Spec with the hybrid context injection can eliminate the extra drafter-side KV cache without sacrificing draft quality.

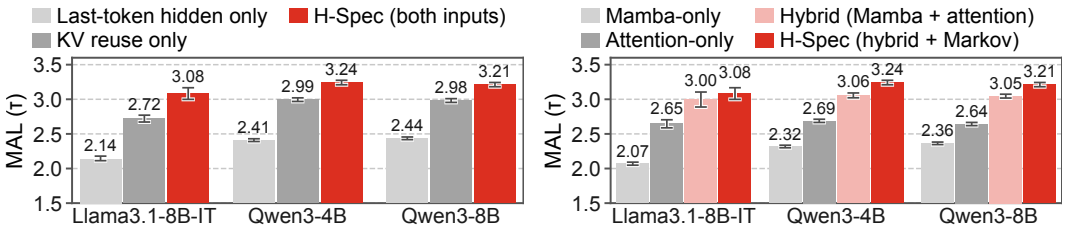
**Extended evaluation tasks.** Appendix §E.3 evaluates the 9-task suite released by the DSpark authors. Across all tasks and three target models, H-Spec improves both $\tau$ and Spd. over the best baseline, with statistically significant gains in every comparison.

### 4.3 Ablation studies

All ablation variants are trained from scratch using the same training recipe as the full H-Spec. For the remaining analyses in the paper, we report unweighted average $\tau$ and Spd. from the eight tasks used in Table 2, with per-task results omitted for brevity.

**Target-context source ablation.** We validate our central hypothesis: the last-token hidden states and the KVs of the target model provide complementary context for drafting. We therefore keep the H-Spec architecture unchanged and ablate one context source at a time. In the *Last-token-hidden-only* variant, attention is restricted to masked block positions, removing target KVs from the prefix context; in the *KV-reuse-only* variant, the Mamba states are initialized with zeros, and weights for cross-layer fusion and projection of the last-token hidden states are no longer needed.

Figure 6a shows that using either target-context source alone reduces draft quality compared to full hybrid context injection. With the drafter architecture fixed, using only last-token hidden states or target KVs reduces average $\tau$ by 24.1–30.5% and 7.1–11.7%, respectively, across the three target models. These results confirm that the two context sources have complementary benefits.



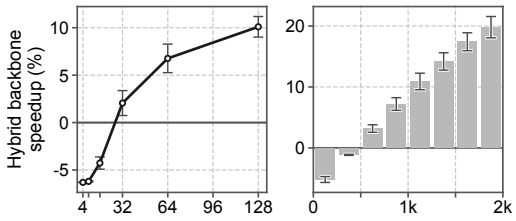

(a) **Target-context source ablation.** (b) **Drafter architecture ablation.**

Figure 6: **Ablation studies.** 6a: Hybrid context injection improves $\tau$ over either context source alone under a fixed drafter architecture. 6b: The Mamba-attention architecture improves $\tau$ over either component alone under roughly matched parameter counts. Error bars are 95% bootstrap CIs.

**Drafter architecture ablation.** The study above ablates input context only. Here, we also change the drafter architecture: *Mamba-only* and *Attention-only* with only Mamba-MLP and attention-MLP layers, respectively; *Hybrid* combines Mamba and attention. The first two have five layers to roughly match H-Spec in total parameter count, with at most 5.2% deviation across three target models; detailed configurations are in Appendix §C.4. Figure 6b shows that the hybrid backbone improves $\tau$ over Mamba-only by 28.9–45.1% and over attention-only by 13.3–15.3% under roughly matched parameter counts, confirming that the combined architecture improves over either module alone.

**Drafter runtime scaling.** Throughput depends on both MAL and drafter runtimes. To isolate the effects, we compare the runtime of the hybrid Mamba-attention and block diffusion backbones from the Qwen3-8B drafters under a controlled setting: synthetic requests with 16 input and 2,048 output tokens, and all drafted tokens are accepted per step. Figure 7 reports the hybrid backbone's relative speedup in forward-pass time (positive = hybrid faster), averaged over decoding steps across concurrency levels $C$ (7a), and across input lengths (binned by 256 tokens) at fixed $C = 128$ (7b).

At low concurrency, the hybrid backbone drafts more slowly, suggesting a deeper computational graph at matched parameter counts, but scales more favorably as $C$ increases and is faster at $C = 32$. This is consistent with the hybrid backbone using fewer attention modules, thereby reducing KV-cache read traffic per added request. At short input lengths, the hybrid backbone drafts more slowly, but becomes faster beyond roughly 512 tokens. For a fixed draft length, Mamba reads a fixed-size injected state per drafting step and has $\mathcal{O}(1)$ runtime in input length; attention runtime instead grows linearly with the attended context length. Fewer attention modules thus improve the hybrid's runtime scaling for longer inputs.




(a) Concurrency (b) Input tokens

Figure 7: **Hybrid vs. block diffusion runtime scaling.** Forward-pass speedup of the hybrid over the block-diffusion backbone across (a) concurrency and (b) input length at fixed $C = 128$. Results are averaged from 5 waves per concurrency level and error bars show st.d. across waves.

### 4.4 Sensitivity Analyses

**Sensitivity to sampling parameters.** We validate that H-Spec's lead in $\tau$ persists under extended sampling parameter configurations beyond the defaults used in Table 2. As shown in Table 3, under greedy sampling (temp=0, topP/topK disabled), H-Spec improves $\tau$ over the best baseline by 4.8–12.0% across three targets. Under untruncated sampling (temp=1, topP=1, topK disabled), where all drafters decline, H-Spec still leads in $\tau$ by 4.7–8.4%.

Table 3: **Sensitivity to sampling parameters.** MAL under greedy/untruncated sampling. Statistical uncertainty reported in §D.2.

| | Llama3.1-8B-IT | | Qwen3-4B | | Qwen3-8B | |
|---|---|---|---|---|---|---|
| **Method** | Greedy | Untrunc. | Greedy | Untrunc. | Greedy | Untrunc. |
| P-Eagle | 2.93 | 1.99 | 3.22 | 2.75 | 3.09 | 2.72 |
| DFlash | 2.57 | 2.02 | 3.09 | 2.66 | 2.91 | 2.57 |
| DSpark | 2.86 | 2.04 | 3.33 | 2.83 | 3.15 | 2.73 |
| H-Spec | **3.28** | **2.20** | **3.49** | **2.97** | **3.42** | **2.96** |

Due to space constraints, we report results only for the Qwen3-8B target model in the remaining analyses and defer results for the other target models to Appendix §E.4.

Table 4: **Sensitivity to draft length.** MAL ($\tau$) and speedup for draft length $k \in \{3, 5, 7\}$. Statistical uncertainty reported in §D.3.

| | $k=3$ | | $k=5$ | | $k=7$ | |
|---|---|---|---|---|---|---|
| **Method** | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 2.57 | *2.05*× | 2.84 | *2.26*× | 2.92 | *2.30*× |
| DFlash | 2.46 | *1.99*× | 2.68 | *2.17*× | 2.77 | *2.28*× |
| DSpark | 2.57 | *2.07*× | 2.90 | *2.32*× | 2.95 | *2.40*× |
| H-Spec | **2.68** | ***2.17***× | **3.07** | ***2.46***× | **3.21** | ***2.61***× |

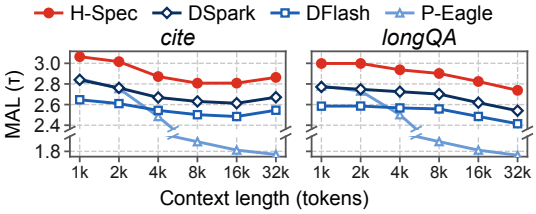



Figure 8: **Sensitivity to context length.** MAL ($\tau$) across context lengths on *cite* and *longQA* tasks. Statistical uncertainty reported in §D.4.

**Sensitivity to draft length.** We vary the inference-time draft length over $k \in \{3, 5, 7\}$ while using the same drafters trained with $k = 7$. As shown in Table 4, H-Spec remains the best-performing drafter across all tested $k$ values. Across $k = 3/5/7$, H-Spec's $\tau$ margin over the best baseline is 4.3/5.8/8.7%, and the speedup margin is 5.3/6.0/8.7%, respectively. Speedup follows the same overall trend, showing that H-Spec is robust to inference-time draft-length changes and generally gains a larger advantage at longer draft lengths.

**Sensitivity to context length.** Previous work shows draft quality can degrade on longer contexts (Eldenk et al., 2026). We therefore validate H-Spec's robustness on two tasks from *HELMET* (Yen et al., 2025): *cite* and *longQA*, which support evaluation across multiple context-length bins. As Figure 8 shows, H-Spec's $\tau$ margin over the best baseline remains 6.8–9.2% across both tasks and all context lengths. From 1K to 32K, the margin changes only from +7.5 to +7.2% on *cite* and remains +7.8% on *longQA*, showing H-Spec's advantage is sustained as context length increases.

**Transferability to finetuned targets.** We serve drafters unchanged on targets finetuned from their training target, reported in Appendix §E.5. Overall, H-Spec retains the best $\tau$ and Spd. under target-model weight and/or chat template shifts, demonstrating zero-shot transferability without retraining.

## 5 Related Work, Limitations, Conclusion and Future Work

**Related work.** Earlier drafters such as the Eagle series (Li et al., 2024a;b; 2025) and ReDrafter (Cheng et al., 2024) predict tokens autoregressively. P-Eagle (Hui et al., 2026) extends Eagle to parallel drafting by predicting multiple future positions from a shared learnable hidden state. DFlash (Chen et al., 2026) instead uses a block-diffusion drafter, conditioning on target-model hidden states by projecting them into a separate drafter-side KV cache. Subsequent drafters such as Domino (Huang et al., 2026), DSpark (Cheng et al., 2026), DBlast (Karimi et al., 2026), and DFlash-2 (Inco AI, 2026) retain the block-diffusion structure but improve inter-token dependencies through different auxiliary components. H-Spec instead deviates from the conventional block-diffusion architecture by introducing hybrid target-context injection through a Mamba-attention architecture, eliminating the separate drafter-side KV cache.

An orthogonal direction augments the target model to predict multiple future tokens without a separate drafter, such as Medusa (Cai et al., 2024), MTP (Gloeckle et al., 2024), and Orthrus (Nguyen et al., 2026). H-Spec instead follows the standard speculative-decoding setup that leaves the target model entirely unchanged.

**Limitations.** H-Spec requires its drafter-side K and V dimensions to match the target model's to reuse target KVs as-is, introducing an additional architectural constraint compared to hidden-state-based KV injection. The hybrid backbone drafts more slowly at lower concurrency or shorter input lengths, reflecting a deeper computational graph at matched parameter counts; at higher concurrency or longer inputs, this trend reverses as the hybrid architecture scales more favorably (Figure 7).

**Conclusion and future work.** In this work, we identify the drafter-side KV cache as an avoidable bottleneck in existing block diffusion drafters. We eliminate it with a novel hybrid target-context injection mechanism that combines last-token target hidden states with in-place target-KV reuse. Building on this mechanism, we propose H-Spec, a parallel drafter that improves serving throughput while reducing KV cache utilization. More broadly, our results highlight system efficiency under concurrent serving as an important consideration in drafter architecture design. Future work will explore alternative target-layer selection strategies, partial target-KV head reuse to reduce drafter attention overhead, and additional causal correction schemes for the hybrid backbone.

### AI USE STATEMENT

In this work, we used generative AI to provide feedback on experiments. Additionally, we used generative AI tools to modify scientific figures, edit software code, brainstorm research ideas during the initial stages of the project, polish writing and presentations (section structure, paper title ideas, grammar), and find related work. We have reviewed all AI-assisted work, including code and model checkpoints, to the best of our abilities. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

This work does not raise any ethical considerations.

### REPRODUCIBILITY STATEMENT

We provide code implementations to train and serve H-Spec checkpoints in the supplementary materials. We will publish all H-Spec and reproduced baseline checkpoints used in the experiments to facilitate future research. We provide a detailed description of all H-Spec, baseline, training, and experimental configurations in §4 and Appendix §C to ensure full reproducibility.

## REFERENCES

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*, ICML'24. JMLR.org, 2024. 9

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. 1, 2, 5, 6, 9, 19

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374. 7

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, 2026. URL https://arxiv.org/abs/2607.05147. 1, 2, 5, 6, 9, 19, 20

Yunfei Cheng, Aonan Zhang, Xuanyu Zhang, Chong Wang, and Yi Wang. Recurrent drafter for fast speculative decoding in large language models, 2024. URL https://arxiv.org/abs/2403.09919. 1, 2, 9

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*, ICML'24. JMLR.org, 2024. 2, 4

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.183. URL https://aclanthology.org/2023.emnlp-main.183/. 6, 19

Doğaç Eldenk, Payal Mohapatra, Yigitcan Comlek, Kaan Oktay, Hongyang Zhang, and Stephen Xia. Attention drift: What autoregressive speculative decoding models learn, 2026. URL https://arxiv.org/abs/2605.09992. 5, 9, 19, 21

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024. 9

Raghavv Goel, Sudhanshu Agrawal, Mukul Gagrani, Junyoung Park, Yifan Zao, He Zhang, Tian Liu, Yiping Yang, Xin Yuan, Jiuyan Lu, Chris Lott, and Mingu Lee. Vocabtrim: Vocabulary pruning for efficient speculative decoding in llms, 2025. URL https://arxiv.org/abs/2506.22694. 19

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh,

Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783. 6

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2. 2, 4

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021. 6

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding, 2026. URL https://arxiv.org/abs/2605.29707. 1, 2, 9

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, 2026. URL https://arxiv.org/abs/2602.01469. 6, 9, 20

Inco AI. DFlash 2: Keep Drafting Parallel, August 2026. URL https://inco.ai/blog/dflash2/. 1, 3, 6, 9, 19, 22

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL. 6

Amirmohammad Karimi, Chao Gao, and Negar Hassanpour. Dblast: Dependent block drafting for stochastic speculative decoding, 2026. URL https://arxiv.org/abs/2608.05448. 1, 9

Sungkyun Kim, Jaemin Kim, Dogyeong Yun, Jiho Shin, Junyeol Lee, and Jiwon Seo. Speculative verification: Exploiting information gain for speculative decoding. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 36290–36307, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1809. URL https://aclanthology.org/2026.findings-acl.1809/. 19

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023. 3, 6, 22

Barak Lenz, Opher Lieber, Alan Arazi, Amir Bergman, Avshalom Manevich, Barak Peleg, Ben Aviram, Chen Almagor, Clara Fridman, Dan Padnos, Daniel Gissin, Daniel Jannai, Dor Muhlgay, Dor Zimberg, Edden M. Gerber, Elad Dolev, Eran Krakovsky, Erez Safahi, Erez Schwartz, Gal Cohen, Gal Shachaf, Haim Rozenblum, Hofit Bata, Ido Blass, Inbal Magar, Itay Dalmedigos, Jhonathan Osin, Julie Fadlon, Maria Rozman, Matan Danos, Michael Gokhman, Mor Zusman, Naama Gidron, Nir Ratner, Noam Gat, Noam Rozen, Oded Fried, Ohad Leshno, Omer Antverg, Omri Abend, Or Dagan, Orit Cohavi, Raz Alon, Ro'i Belson, Roi Cohen, Rom Gilad, Roman Glozman, Shahar Lev, Shai Shalev-Shwartz, Shaked Haim Meirom, Tal Delbari, Tal Ness, Tomer Asida, Tom Ben Gal, Tom Braude, Uriya Pumerantz, Josh Cohen, Yonatan Belinkov, Yuval Globerson, Yuval Peleg Levy, and Yoav Shoham. Jamba: Hybrid transformer-mamba language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=JFPaD7lpBD. 3

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023. 1, 2, 17

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024a. 1, 2, 9

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen

(eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, Miami, Florida, USA, November 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/. 1, 2, 9

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=4exx1hUffq. 1, 2, 9

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, 2026. URL https://arxiv.org/abs/2607.14647. 19

Chien Van Nguyen, Chaitra Hegde, Van Cuong Pham, Ryan A. Rossi, Franck Dernoncourt, and Thien Huu Nguyen. Orthrus: Memory-efficient parallel token generation via dual-view diffusion, 2026. URL https://arxiv.org/abs/2605.12825. 9

NVIDIA, :, Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, Aleksandr Shaposhnikov, Alex Kondratenko, Alexander Bukharin, Alexandre Milesi, Ali Taghibakhshi, Alisa Liu, Amelia Barton, Ameya Sunil Mahabaleshwarkar, Amir Klein, Amit Zuker, Amnon Geifman, Amy Shen, Anahita Bhiwandiwalla, Andrew Tao, Anjulie Agrusa, Ankur Verma, Ann Guan, Anubhav Mandarwal, Arham Mehta, Ashwath Aithal, Ashwin Poojary, Asif Ahamed, Asit Mishra, Asma Kuriparambil Thekkumpate, Ayush Dattagupta, Banghua Zhu, Bardiya Sadeghi, Barnaby Simkin, Ben Lanir, Benedikt Schifferer, Besmira Nushi, Bilal Kartal, Bita Darvish Rouhani, Boris Ginsburg, Brandon Norick, Brandon Soubasis, Branislav Kisacanin, Brian Yu, Bryan Catanzaro, Carlo del Mundo, Chantal Hwang, Charles Wang, Cheng-Ping Hsieh, Chenghao Zhang, Chenhan Yu, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Christopher Parisien, Collin Neale, Cyril Meurillon, Damon Mosk-Aoyama, Dan Su, Dane Corneil, Daniel Afrimi, Daniel Lo, Daniel Rohrer, Daniel Serebrenik, Daria Gitman, Daria Levy, Darko Stosic, David Mosallanezhad, Deepak Narayanan, Dhruv Nathawani, Dima Rekesh, Dina Yared, Divyanshu Kakwani, Dong Ahn, Duncan Riach, Dusan Stosic, Edgar Minasyan, Edward Lin, Eileen Long, Eileen Peters Long, Elad Segal, Elena Lantz, Ellie Evans, Elliott Ning, Eric Chung, Eric Harper, Eric Tramel, Erick Galinkin, Erik Pounds, Evan Briones, Evelina Bakhturina, Evgeny Tsykunov, Faisal Ladhak, Fay Wang, Fei Jia, Felipe Soares, Feng Chen, Ferenc Galko, Frank Sun, Frankie Siino, Gal Hubara Agam, Ganesh Ajjanagadde, Gantavya Bhatt, Gargi Prasad, George Armstrong, Gerald Shen, Gorkem Batmaz, Grigor Nalbandyan, Haifeng Qian, Harsh Sharma, Hayley Ross, Helen Ngo, Herbert Hum, Herman Sahota, Hexin Wang, Himanshu Soni, Hiren Upadhyay, Huizi Mao, Huy C Nguyen, Huy Q Nguyen, Iain Cunningham, Ido Galil, Ido Shahaf, Igor Gitman, Ilya Loshchilov, Itamar Schen, Itay Levy, Ivan Moshkov, Izik Golan, Izzy Putterman, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jatin Mitra, Jeffrey Glick, Jenny Chen, Jesse Oliver, Jian Zhang, Jiaqi Zeng, Jie Lou, Jimmy Zhang, Jinhang Choi, Jining Huang, Joey Conway, Joey Guman, John Kamalu, Johnny Greco, Jonathan Cohen, Joseph Jennings, Joyjit Daw, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kai Xu, Kan Zhu, Kari Briski, Katherine Cheung, Katherine Luna, Keith Wyss, Keshav Santhanam, Kevin Shih, Kezhi Kong, Khushi Bhardwaj, Kirthi Shankar, Krishna C. Puvvada, Krzysztof Pawelec, Kumar Anik, Lawrence McAfee, Laya Sleiman, Leon Derczynski, Li Ding, Lizzie Wei, Lucas Liebenwein, Luis Vega, Maanu Grover, Maarten Van Segbroeck, Maer Rodrigues de Melo, Mahdi Nazemi, Makesh Narsimhan Sreedhar, Manoj Kilaru, Maor Ashkenazi, Marc Romeijn, Marcin Chochowski, Mark Cai, Markus Kliegl, Maryam Moosaei, Matt Kulka, Matvei Novikov, Mehrzad Samadi, Melissa Corpuz, Mengru Wang, Meredith Price, Michael Andersch, Michael Boone, Michael Evans, Miguel Martinez, Mikail Khona, Mike Chrzanowski, Minseok Lee, Mohammad Dabbah, Mohammad Shoeybi, Mostofa Patwary, Nabin Mulepati, Najeeb Nabwani, Natalie Hereth, Nave Assaf, Negar Habibi, Neta Zmora, Netanel Haber, Nicola Sessions, Nidhi Bhatia, Nikhil Jukar, Nikki Pope, Nikolai Ludwig, Nima Tajbakhsh, Nir Ailon, Nirmal Juluru, Nishant Sharma, Oleksii Hrinchuk, Oleksii Kuchaiev, Olivier Delalleau, Oluwatobi Olabiyi, Omer Ullman Argov, Omri Puny, Oren Tropp, Ouye Xie, Parth Chadha, Pasha Shamis, Paul Gibbons, Pavlo Molchanov, Pawel Morkisz, Peter Dykas, Peter Jin, Pinky Xu, Piotr Januszewski, Pranav Prashant Thombre, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Qing Miao, Qiyu Wan, Rabeeh Karimi

Mahabadi, Rachit Garg, Ran El-Yaniv, Ran Zilberstein, Rasoul Shafipour, Rich Harang, Rick Izzo, Rima Shahbazyan, Rishabh Garg, Ritika Borkar, Ritu Gala, Riyad Islam, Robert Hesse, Roger Waleffe, Rohit Watve, Roi Koren, Ruoxi Zhang, Russell Hewett, Russell J. Hewett, Ryan Prenger, Ryan Timbrook, Sadegh Mahdavi, Sahil Modi, Samuel Kriman, Sangkug Lim, Sanjay Kariyappa, Sanjeev Satheesh, Saori Kaji, Satish Pasumarthi, Saurav Muralidharan, Sean Narentharen, Sean Narenthiran, Seonmyeong Bak, Sergey Kashirsky, Seth Poulos, Shahar Mor, Shanmugam Ramasamy, Shantanu Acharya, Shaona Ghosh, Sharath Turuvekere Sreenivas, Shelby Thomas, Shiqing Fan, Shreya Gopal, Shrimai Prabhumoye, Shubham Pachori, Shubham Toshniwal, Shuoyang Ding, Siddharth Singh, Simeng Sun, Smita Ithape, Somshubra Majumdar, Soumye Singhal, Stas Sergienko, Stefania Alborghetti, Stephen Ge, Sugam Dipak Devare, Sumeet Kumar Barua, Suseella Panguluri, Suyog Gupta, Sweta Priyadarshi, Syeda Nahida Akter, Tan Bui, Teodor-Dumitru Ene, Terry Kong, Thanh Do, Tijmen Blankevoort, Tim Moon, Tom Balough, Tomer Asida, Tomer Bar Natan, Tomer Ronen, Tugrul Konuk, Twinkle Vashishth, Udi Karpas, Ushnish De, Vahid Noorozi, Vahid Noroozi, Venkat Srinivasan, Venmugil Elango, Victor Cui, Vijay Korthikanti, Vinay Rao, Vitaly Kurin, Vitaly Lavrukhin, Vladimir Anisimov, Wanli Jiang, Wasi Uddin Ahmad, Wei Du, Wei Ping, Wenfei Zhou, Will Jennings, William Zhang, Wojciech Prazuch, Xiaowei Ren, Yashaswi Karnati, Yejin Choi, Yev Meyer, Yi-Fu Wu, Yian Zhang, Yigong Qin, Ying Lin, Yonatan Geifman, Yonggan Fu, Yoshi Subara, Yoshi Suhara, Yubo Gao, Zach Moshe, Zhen Dong, Zhongbo Zhu, Zihan Liu, Zijia Chen, and Zijie Yan. Nvidia nemotron 3: Efficient and open intelligence, 2025. URL https://arxiv.org/abs/2512.20856. 3

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk. 7

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao, Bo Zheng, and Dayiheng Liu. On the design of qwen3.8-next architecture: Evaluation, efficiency, and training stability, 2026. URL https://arxiv.org/abs/2608.30320. 3

Red Hat. Speculators: A unified library for speculative decoding algorithms in llm serving. https://github.com/vllm-project/speculators, 2025. 6, 22

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 6835–6850, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/13f972adf12bdf886583d48cd528002f-Paper-Conference.pdf. 2

Alexander Samarin, Sergei Krutikov, Anton Shevtsov, Sergei Skvortsov, Filipp Fisin, and Alexander Golubev. LK losses: Direct acceptance rate optimization for speculative decoding. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ZqnVidXtxV. 20

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential, 2025. URL https://arxiv.org/abs/2507.11851. 2, 4

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023. 6

vLLM. Hybrid kv cache manager, 2026. URL https://docs.vllm.ai/en/stable/design/hybrid_kv_cache_manager/. 1, 3, 17

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey

of speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.456. URL https://aclanthology.org/2024.findings-acl.456/. 7

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned LLMs with nothing. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Pnk7vMbznK. 6, 19

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388. 6

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. Helmet: How to evaluate long-context language models effectively and thoroughly. In *International Conference on Learning Representations (ICLR)*, 2025. 9

Chen Zhang, Kuntai Du, Shu Liu, Woosuk Kwon, Xiangxi Mo, Yufeng Wang, Xiaoxuan Liu, Kaichao You, Zhuohan Li, Mingsheng Long, Jidong Zhai, Joseph Gonzalez, and Ion Stoica. Jenga: Effective memory management for serving llm with heterogeneity. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25, pp. 446–461, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400718700. doi: 10.1145/3731569.3764823. URL https://doi.org/10.1145/3731569.3764823. 1, 3

Weilin Zhao, Tengyu Pan, Xu Han, Yudi Zhang, Ao Sun, Yuxiang Huang, Kaihuo Zhang, Weilun Zhao, Yuxuan Li, Jie Zhou, Hao Zhou, Jianyong Wang, Zhiyuan Liu, and Maosong Sun. FR-spec: Accelerating large-vocabulary language models via frequency-ranked speculative sampling. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3909–3921, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.198. URL https://aclanthology.org/2025.acl-long.198/. 19

## Appendix Table of Contents


**A Extended background on speculative decoding** **17**

**B Case study: KV access patterns and memory waste** **17**

**C Experimental settings** **18**

C.1 H-Spec architecture configuration . . . . . 18

C.2 Baseline configuration . . . . . 19

C.3 Training recipe . . . . . 19

C.4 Settings for drafter architecture ablation . . . . . 20

**D Statistical uncertainty analysis in evaluation results** **20**

D.1 Single-request MAL and speedup . . . . . 20

D.2 Sensitivity to sampling parameters . . . . . 21

D.3 Sensitivity to inference-time draft length . . . . . 21

D.4 Sensitivity to context length . . . . . 21

**E Extended evaluation** **21**

E.1 KV cache allocation analysis . . . . . 21

E.2 Preliminary comparison with DFlash-2 . . . . . 22

E.3 Single-request evaluation on DeepSpec tasks . . . . . 23

E.4 Extended sensitivity analysis . . . . . 24

E.5 Zero-shot transfer to finetuned target models . . . . . 24


## A Extended background on speculative decoding

**Definition.** Speculative decoding (Leviathan et al., 2023) losslessly accelerates the inference of a target LLM $\mathcal{M}_T$ with a smaller draft model $\mathcal{M}_D$. In each decoding iteration, given an input sequence $L$, $\mathcal{M}_D$ generates $k$ draft tokens $t_1 \dots t_k$ with probabilities $P_D(t_i|L, t_1 \dots t_{i-1})$. $\mathcal{M}_T$ then performs verification with a single forward pass over the concatenated sequence $L, t_1 \dots t_k$, obtaining $P_T(t_i|L, t_1 \dots t_{i-1})$ for $1 \le i \le k+1$.

A draft token $t_i$ is accepted if all preceding drafts are accepted and

$$\min(1, \frac{P_T(t_i|L, t_1 \dots t_{i-1})}{P_D(t_i|L, t_1 \dots t_{i-1})}) \ge \alpha$$

where $\alpha \sim U(0, 1)$ and $1 \le i \le k$.

**Target-generated "bonus" token.** Each verification produces one target-generated "bonus" token in addition to accepted drafts. If $t_i$ is the first rejected draft, all subsequent drafts are rejected, and $t_i$ is replaced by sampling from the corrected target distribution conditioned on $L, t_1 \dots t_{i-1}$. If all $k$ drafts are accepted, a bonus token $t_{k+1}$ is sampled with probability $P_T(t_{k+1}|L, t_1 \dots t_k)$.

## B Case study: KV access patterns and memory waste

Target and draft models can differ in KV access patterns, complicating memory management in serving systems. We use vLLM's Hybrid KV Cache Manager (vLLM, 2026) as an example. In vLLM, layers are organized into KV cache groups based on their access patterns, with layers in the

same group sharing the same block table. To manage these groups within a shared memory pool, vLLM requires uniform page sizes and pads imbalanced groups to a common group size, which can leave part of the allocated KV memory unused.

We demonstrate this memory inefficiency using two target models: Qwen3 4B and 8B, both of which contain 36 full-attention layers. We pair each target with a DFlash checkpoint, which contains five sliding-window layers. Each DFlash layer has matched KV dimensions to the target model.

We analyze vLLM's memory allocation decisions through its startup logs. The five sliding-window drafter layers form a single KV cache group due to a shared access pattern. Because the drafter and target models have matched per-layer KV sizes, the 36 full-attention layers are also organized into five-layer KV groups to ensure a globally matched page size. This results in eight full-attention KV groups, with one group having four unused padded layer slots.

Overall, managing caches with different access patterns can introduce padded layer slots in vLLM, especially when layer counts across access patterns do not align with a common KV cache group size. This further motivates us to eliminate the drafter-side KV cache and reduce cache management complexity in serving runtimes.

# C Experimental settings

## C.1 H-Spec architecture configuration

This section details H-Spec's architecture configuration. To ensure controlled comparisons, we match the attention, MLP, and, where applicable, Markov-head dimensions between H-Spec and DFlash/DSpark, while closely matching their total trainable parameter counts. This helps ensure performance differences reflect the drafter architecture rather than mismatched drafter capacity or configurations.

**Number of drafter layers.** Available official DFlash and DSpark checkpoints for our evaluated targets use five drafter layers. We keep the attention and MLP dimensions matched to these baselines, but each H-Spec layer additionally contains a Mamba module. To keep the total parameter count closely matched, H-Spec therefore uses four layers: three full Mamba-attention-MLP layers and one partial Mamba-MLP layer.

**Attention.** H-Spec matches the target model's attention configuration, including the Q, K, and V dimensions and the numbers of attention and KV heads, so that target-model KVs can be reused directly. All available official DFlash and DSpark checkpoints for our evaluated targets also match these attention configurations to their respective target models. Thus, although the baselines are not constrained by target-KV reuse, H-Spec's target-matching requirement does not introduce an additional attention-configuration difference.

**MLP.** We set the MLP hidden dimension to match the target model, consistent with conventions in the official DFlash and DSpark checkpoints.

**Mamba-2.** We use a Mamba head dimension of 64, with the state dimension, number of groups, and convolution kernel size set to 4, and the expansion factor set to 1. We vary the number of Mamba heads across target models to minimize the remaining parameter-count gap with the baselines: 44 and 48 heads for Qwen3 4B and 8B, respectively, and 56 heads for Llama3.1-8B-IT. Here, the product of the number of Mamba heads and head dimension specifies the SSM dimension, which can be smaller than the expanded hidden dimension. This keeps the difference in trainable parameters within 2.9% compared to DSpark across all evaluated targets.

We set $r = 4{,}096$ for Llama3.1/Qwen3-8B, and $2{,}560$ for Qwen3-4B as the intermediate rank for cross-layer fusion (§3.3.1), corresponding to the target model's hidden dimension. $d_S$, the flattened dimensionality of Mamba-2's recurrent state, equals the product of the number of Mamba heads, the per-head dimension, and the state dimension; the projected state is reshaped to [n_heads, head_dim, d_state] before Mamba initialization.

**Markov rank.** We use a Markov rank of 256 for causal correction, matching DSpark.

**Target-layer mapping.** Existing block-diffusion drafters fuse and project hidden states from five evenly spaced target-model layers: layers 1, 8, 15, 22, and 30 for Llama3.1-8B-IT, and layers 1, 9,

17, 25, and 34 for Qwen3 4B and 8B. For a controlled comparison, H-Spec draws target context from the same layers. The last-token hidden states from all five layers are fused and projected into the Mamba initial states, while the three attention modules reuse the KV caches from the last three selected layers: 15, 22, and 30 for Llama3.1-8B-IT, and 17, 25, and 34 for Qwen3. This ensures that H-Spec's advantage does not result from using different target-model layer representations.

### C.2 Baseline configuration

We report and justify any non-trivial configuration settings during baseline reproduction. Overall, we standardize each configuration choice across methods based on released checkpoints or prior literature, aiming to control the comparison rather than optimize for any individual method.

**Unified attention implementation.** Official baseline checkpoints use different attention implementations, such as Llama-style or Qwen3 attention. To ensure that comparisons reflect drafter architecture rather than attention implementation, we use Qwen3Attention for H-Spec and all reproduced baselines. This follows the official DFlash implementation, which uses Qwen3 attention in its drafters for both Qwen3 and non-Qwen3 target models.

**Sliding-window attention (SWA).** We use SWA with a window size of 2,048 for H-Spec and all reproduced baselines. SWA is used in the official DSpark and DFlash-2 releases (Inco AI, 2026), and has also been shown to improve long-context stability for EAGLE-style architectures (Eldenk et al., 2026). We therefore standardize on SWA for controlled attention-window configuration.

**Pruned drafter vocabulary.** All methods use the target model's LM-head weights. Following prior work on vocabulary pruning (Zhao et al., 2025; Goel et al., 2025), we restrict drafter prediction to the 32,000 most frequent target-vocabulary tokens observed in the training corpus. This pruning uniformly reduces drafting latency for all methods while maintaining lossless generation.

**Fixed-length verification.** We use fixed-length verification, where the target model verifies all drafted tokens in every iteration. Adaptive verification strategies, such as DSpark's confidence-scheduled verification (Cheng et al., 2026), Speculative-Verification (Kim et al., 2026), or D-cut (Liu et al., 2026), operate at the verification stage and are orthogonal to the drafter architecture studied in this work. We therefore keep the verification procedure fixed across methods so that performance differences reflect the drafter itself rather than differences in verification policy.

### C.3 Training recipe

We use a unified training recipe for H-Spec and all reproduced baseline methods, while retaining method-specific training mechanisms where applicable:

**Training corpus.** We use 100K samples randomly drawn from Magpie (Xu et al., 2025) and Ultrachat (Ding et al., 2023) in a 60:40 ratio, proportional to the sizes of the original datasets. The scale of this corpus matches the 100K-sample setting used for DFlash's ablation experiments (Chen et al., 2026). For Qwen3 4B and 8B, we use target-generated responses with thinking enabled rather than the original responses, allowing all drafters to better align with the thinking behavior.

**Preprocessing.** Every sample is tokenized into a training sequence and truncated to at most 8,192 tokens. We use Multipack sampling to pack these into 8,192-token sequences, with block-diagonal attention masks preventing attention across examples. Within each packed sequence, we randomly select up to 3,072 anchor positions, each corresponding to a block start for H-Spec and the block diffusion baselines, i.e., DFlash and DSpark, or a chain start for P-Eagle.

**Optimizer settings and steps.** We use matched optimizer settings, learning-rate schedules, and optimizer steps for all drafters to ensure a controlled comparison. Specifically, we use AdamW with a peak learning rate of $6 \times 10^{-4}$, weight decay of 0.01, and gradient clipping at a global norm of 1.0, with 1% linear warmup followed by half-cosine decay to zero. All methods train for 5 epochs on the same 100K samples across two GPUs with identical packing. With one packed sequence per GPU per optimization step, this gives a global batch size of two packed sequences (16,384 tokens), containing up to 6,144 anchors. This yields matched optimizer steps: 27,086 for Llama3.1-8B-IT and 104,032 for Qwen3 targets.

**Loss function and block-position weighting.** Following DSpark (Cheng et al., 2026), we use a compound loss combining cross-entropy and total variation (Samarin et al., 2026) for all drafters, weighted by 0.1 and 0.9, respectively. The Markov heads in DSpark and H-Spec are trained with the same loss. For H-Spec, DFlash, and DSpark, we apply block-position loss weighting as in DFlash and DSpark: $w_k = \exp(-(k-1)/\gamma)$, where $w_k$ is the loss weight at masked block position $1 \le k \le 7$ and $\gamma = 4$. P-Eagle instead uses its own position-dependent sampling scheme: COD sampling retains a fraction $\max(0.6^d, 0.2)$ of anchors at depth $0 \le d \le 6$ (Hui et al., 2026).

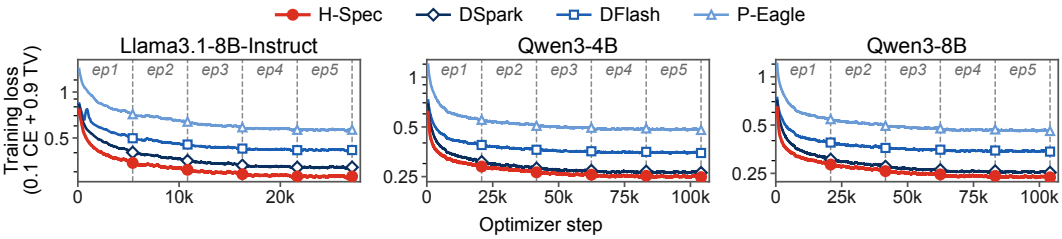



Figure 9: **Training loss for H-Spec and all reproduced baselines.** For each target model, all methods are trained for an identical number of optimizer steps, with vertical dotted lines marking epoch boundaries. Loss values are shown as moving averages over 1% of the run for readability.

Figure 9 shows the training loss for all methods over three target models. All methods converge stably within the matched number of optimizer steps.

### C.4 Settings for drafter architecture ablation

This section provides detailed configurations for the three drafter variants studied in the architecture ablation in §4.3. All three variants omit the Markov correction head: *Mamba-only* uses Mamba-MLP layers with last-token hidden-state injection, *Attention-only* uses attention-MLP layers with target-KV reuse, and *Hybrid* combines Mamba and attention. The full H-Spec adds the Markov head on top of the hybrid backbone.

For the *Mamba-only* and *Attention-only* variants, we keep the dimensions of the Mamba/attention and MLP modules unchanged from the full H-Spec, and increase both to five layers for parameter matching, with at most a 5.2% difference in total parameter count. In the full H-Spec, Mamba initial-state injection uses last-token hidden states from five selected target layers, while the three attention modules reuse KVs from three of the five selected layers. In *Attention-only*, the two additional attention-MLP layers reuse KVs from the remaining two target layers, keeping the overall set of selected target layers unchanged.

## D Statistical uncertainty analysis in evaluation results

We report 95% bootstrap confidence intervals (CIs) to quantify the statistical uncertainty in MAL ($\tau$) and ITL speedup, and use paired bootstrap analyses to assess whether H-Spec's improvements over the baselines are statistically significant.

For single-task $\tau$ and Spd., we estimate 95% CIs using 10,000 bootstrap resamples of the task's requests: each resample is drawn with replacement from the original requests, and we recompute $\tau$ and Spd. on the resampled set. For results averaged over the eight tasks, we independently resample the requests within each task, recompute the task-level $\tau$ and Spd., and then average each metric across the eight tasks. We repeat this procedure 10,000 times to obtain the bootstrap distribution of the task-average $\tau$ and Spd., from which we estimate their 95% CIs.

### D.1 Single-request MAL and speedup

Complementing Table 2, we assess whether H-Spec's improvement over baselines is statistically significant, both when averaged across eight tasks and separately per task.

**Average across all tasks.** We conduct paired bootstrap analyses between H-Spec and DSpark, which is the best baseline in task-average $\tau$ and Spd. for all three target models. The 95% CIs of H-Spec's relative improvement over DSpark in $\tau$ are [+9.0%, +17.5%], [+4.3%, +5.6%], [+8.0%,

+9.4%], and in Spd. are [+10.2%, +15.1%], [+4.7%, +5.9%], [+7.9%, +9.4%], across Llama3.1, Qwen3 4B and 8B target models, respectively. All CIs lie entirely above 0, demonstrating the statistical significance of H-Spec's gains.

**Per task/model setting.** Across eight tasks and three target models, H-Spec achieves the best $\tau$ in 22/24 settings and the best Spd. in 23/24 settings. We conduct paired bootstrap analyses for each of the 45 (task, target model, metric) settings where H-Spec leads, comparing H-Spec with the corresponding best baseline in that setting. In 42/45 cases, the 95% CI lies above zero, showing that H-Spec's lead is statistically significant. In the remaining three cases (Spd. in Math and Chat on Llama3.1, and $\tau$ in Transl. on Qwen3-4B), the lead over the best baseline is not statistically significant as the CI includes zero, but the improvement remains significant compared with the second-best baseline.

### D.2 Sensitivity to sampling parameters

Complementing Table 3, we conduct paired bootstrap analyses between H-Spec and the best baseline (P-Eagle for Llama3.1/Greedy, DSpark in all remaining settings) under all sampling configurations and target model settings. Under greedy sampling, the 95% CI of H-Spec's $\tau$ improvement over the best baseline is [+8.1%, +15.3%], [+3.7%, +5.8%] and [+7.6%, +9.4%], for Llama3.1, Qwen3 4B and 8B target models, respectively. For untruncated sampling, the 95% CIs are [+0.6%, +18.6%], [+4.1%, +5.2%] and [+7.5%, +9.5%]. All CIs lie entirely above zero, demonstrating that H-Spec's gain in $\tau$ is statistically significant across multiple sampling parameter settings.

### D.3 Sensitivity to inference-time draft length

To complement Table 4, we run paired bootstrap analyses between H-Spec and DSpark, the best baseline across all inference-time draft-length settings. For $k = 3/5/7$, the 95% bootstrap CI for H-Spec's $\tau$ margin over DSpark is [+3.8%, +4.8%], [+5.1%, +6.4%], and [+8.0%, +9.4%]; for speedup, the CIs are [+4.8%, +5.8%], [+5.5%, +6.5%], and [+7.9%, +9.4%]. All paired CIs lie above zero, showing that H-Spec's lead is statistically significant across various inference-time draft length settings.

### D.4 Sensitivity to context length

To complement Figure 8, we conduct paired bootstrap analyses between H-Spec and the best baseline at every (task, context length bin) setting; the best baseline is P-Eagle for context length 1K on both tasks, and DSpark in remaining settings. The 95% CI is above zero for every setting, with the tightest margin being [+5.8%, +8.6%] on *cite* at 32K context. This result shows that H-Spec's hybrid context injection remains effective across context lengths, suggesting robustness to the attention drift (Eldenk et al., 2026).

## E Extended evaluation

### E.1 KV cache allocation analysis

This section studies whether eliminating the drafter-side KV cache also avoids additional KV memory allocation and padding in paged memory management (discussed in §2). We compare the allocated KV bytes per token for the block diffusion drafters (DFlash, DSpark) and H-Spec against no-speculative-decoding. Unlike the raw KV entry sizes in Table 1, this measurement reflects vLLM's actual cache allocation after accounting for sliding-window attention and padding, and directly determines how many tokens can fit within a fixed KV memory budget. We also report the padded layer slots per drafter to directly assess memory waste (§B). All numbers are directly reported or derived from vLLM initialization logs.

As shown in Table 5, both DFlash and DSpark increase per-token KV footprint by 9.9–12.5% relative to no-speculative-decoding, after accounting for padding and sliding-window. H-Spec instead directly reads existing target-model KV cache blocks, incurring *zero* additional KV memory usage.

Table 5: **KV cache allocation in vLLM.** Per-token KV cache footprint and padded layers compared with serving the target model without speculative decoding. Parentheses show the relative increase in KV footprint over no speculative decoding. Padded layer slots are as explained in §B, with more padding indicating higher GPU memory waste.

| Target model | Method | KV footprint (KiB/token) | Padded layers |
|---|---|---|---|
| Llama3.1-8B-IT (32 layers) | No spec. | 128.0 | 0 |
| | DFlash | 140.6 (+9.9%) | 3 |
| | DSpark | 140.6 (+9.9%) | 3 |
| | H-Spec | **128.0 (+0.0%)** | **0** |
| Qwen3-4B (36 layers) | No spec. | 144.0 | 0 |
| | DFlash | 162.0 (+12.5%) | 4 |
| | DSpark | 162.0 (+12.5%) | 4 |
| | H-Spec | **144.0 (+0.0%)** | **0** |
| Qwen3-8B (36 layers) | No spec. | 144.0 | 0 |
| | DFlash | 162.0 (+12.5%) | 4 |
| | DSpark | 162.0 (+12.5%) | 4 |
| | H-Spec | **144.0 (+0.0%)** | **0** |

Without speculative decoding, all three target models have no padded layer slots, since all layers use full attention and form a single KV cache group. DFlash and DSpark introduce a second access pattern through their five SWA drafter layers, which form a separate five-layer group. The target-model layers are therefore also partitioned into five-layer groups; because 32 and 36 are not divisible by five, this requires three padded layer slots for Llama3.1 and four for each Qwen3 target. Including the drafter cache group, padding occupies 3 of 40 allocated layer slots (7.5%) for Llama3.1 and 4 of 45 (8.9%) for Qwen3. H-Spec also uses SWA in its attention modules, but reuses existing target KV blocks in place and registers no drafter-side KV cache, so it adds no additional KV cache groups or padded layer slots.

### E.2 Preliminary comparison with DFlash-2

We provide a preliminary comparison with DFlash-2 (Inco AI, 2026), the most recent follow-up work from the DFlash team. DFlash-2 keeps the block diffusion drafter backbone and the KV injection mechanism (Figure 2) as in the original DFlash, but with two additions: (1) a two-tap convolution kernel before and after every attention and MLP module that mixes each position's hidden representation with that of its preceding position, and (2) a path selector that scores adjacent pairs among the top-16 candidates at each draft position and selects a coherent drafted sequence. Together, these additions better capture inter-token dependencies and provide a competitive alternative to existing causal correction mechanisms.

We reproduce DFlash-2 under the matched training recipes used in §4 on the Qwen3-8B target model. Following §4.1, we report decoding throughput (output tokens per second) and average KV cache utilization (%) during request serving under concurrency levels $C \in \{8, 16, 32, 64, 128\}$, on the MATH, LiveCodeBench, and Alpaca workloads.

**Disclaimer:** We implemented DFlash-2's training and inference pipelines using the Speculators (Red Hat, 2025) and vLLM (Kwon et al., 2023) frameworks to the best of our ability. Overall, our DFlash-2 implementation outperforms both DFlash and DSpark under a unified training recipe, consistent with the trend reported by the DFlash-2 authors.

As shown in Figure 10, H-Spec improves maximum throughput over DFlash-2 by 4.2% on MATH, 2.4% on LiveCodeBench, and 8.4% on Alpaca, while reducing average KV cache utilization by 9.7% to 20.2% across all concurrency and task settings. DFlash-2 trails H-Spec in maximum throughput by the smallest margin among all baselines, reflecting its improved drafting quality over DSpark. However, DFlash-2 does not narrow the KV cache utilization gap, since its block diffusion backbone and KV injection mechanism remain unchanged.

We conclude that H-Spec and DFlash-2 make largely orthogonal contributions: H-Spec proposes novel target-context injection via a hybrid Mamba-attention backbone, while DFlash-2 introduces convolution and a path selector as additional components to a parallel drafter backbone to capture

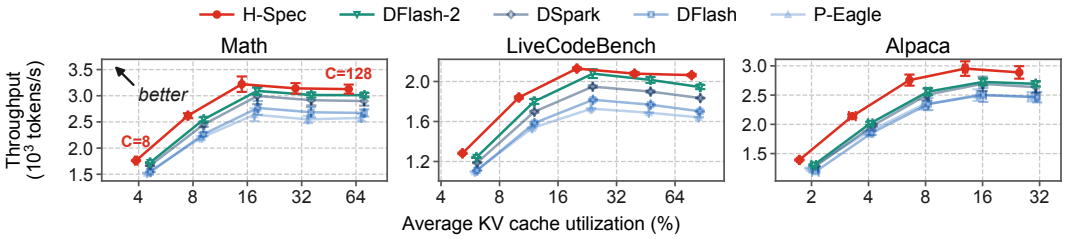



Figure 10: **H-Spec achieves higher throughput while maintaining lower KV cache utilization than DFlash-2 across concurrency levels.** Throughput and the average KV cache utilization during serving, under C=8–128 concurrent requests. Markers represent rising concurrency levels from left to right per curve. Results are averaged from three repeats, and error bars show st.d. across repeated runs.

inter-token dependencies. These components could be incorporated into our design: the convolution can be added to the hybrid backbone, while the path selector can replace the Markov head as the causal correction mechanism. We leave the evaluation and integration of these components as an important direction for future work.

### E.3 Single-request evaluation on DeepSpec tasks

Complementing §4.2, we provide additional mean accepted length ($\tau$) and ITL speedup evaluations on the nine-task benchmark suite released by the DSpark authors, which we refer to as the DeepSpec suite. These tasks are used in the original DFlash and DSpark papers and improve the coverage of our evaluation.

Table 6: **H-Spec leads in draft acceptance and speedup on the DeepSpec evaluation suite.** Mean accepted length ($\tau$) and batch-size-1 ITL speedup (Spd.) relative to no-speculative-decoding over nine tasks. Paired bootstrap 95% CIs for H-Spec's relative gains over the best baseline lie above zero for every task-target-metric setting, indicating statistical significance.

| | **Math** | | | | | | **Code** | | | | | | **Chat** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | gsm8k | | math500 | | aime25 | | livecodebench | | mbpp | | humaneval | | alpaca | | arena-hard-v2 | | mt-bench | |
| L-8B-IT | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 3.21 | *2.66*× | 3.96 | *3.19*× | 4.05 | *3.16*× | 2.75 | *2.25*× | 3.53 | *2.90*× | 3.50 | *2.87*× | 2.57 | *2.09*× | 2.13 | *1.68*× | 2.61 | *2.17*× |
| DFlash | 3.11 | *2.65*× | 3.78 | *3.14*× | 3.86 | *3.03*× | 2.67 | *2.23*× | 3.46 | *2.91*× | 3.26 | *2.72*× | 2.40 | *2.02*× | 2.08 | *1.63*× | 2.45 | *2.11*× |
| DSpark | 3.42 | *2.83*× | 4.17 | *3.37*× | 4.18 | *3.36*× | 2.90 | *2.37*× | 3.86 | *3.03*× | 3.67 | *3.05*× | 2.70 | *2.22*× | 2.08 | *1.65*× | 2.92 | *2.28*× |
| H-Spec | **4.02** | ***3.35***× | **4.64** | ***3.78***× | **4.78** | ***3.74***× | **3.33** | ***2.66***× | **4.21** | ***3.51***× | **4.20** | ***3.39***× | **3.01** | ***2.48***× | **2.31** | ***1.90***× | **3.14** | ***2.54***× |
| Q-4B | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 3.76 | *2.80*× | 3.83 | *2.81*× | 3.52 | *2.54*× | 3.07 | *2.17*× | 3.35 | *2.51*× | 3.37 | *2.49*× | 2.67 | *2.03*× | 2.27 | *1.65*× | 2.95 | *2.19*× |
| DFlash | 3.70 | *2.96*× | 3.79 | *3.00*× | 3.50 | *2.74*× | 2.98 | *2.27*× | 3.29 | *2.65*× | 3.29 | *2.59*× | 2.56 | *2.11*× | 2.19 | *1.71*× | 2.85 | *2.27*× |
| DSpark | 4.01 | *3.13*× | 4.12 | *3.18*× | 3.78 | *2.85*× | 3.26 | *2.41*× | 3.60 | *2.84*× | 3.62 | *2.77*× | 2.79 | *2.22*× | 2.31 | *1.74*× | 3.09 | *2.42*× |
| H-Spec | **4.14** | ***3.23***× | **4.27** | ***3.29***× | **3.91** | ***2.97***× | **3.40** | ***2.53***× | **3.76** | ***2.96***× | **3.76** | ***2.89***× | **2.92** | ***2.35***× | **2.38** | ***1.82***× | **3.21** | ***2.53***× |
| Q-8B | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 3.71 | *2.98*× | 3.83 | *3.04*× | 3.57 | *2.80*× | 3.02 | *2.34*× | 3.27 | *2.62*× | 3.33 | *2.63*× | 2.68 | *2.16*× | 2.25 | *1.77*× | 2.94 | *2.32*× |
| DFlash | 3.58 | *2.98*× | 3.75 | *3.07*× | 3.47 | *2.79*× | 2.92 | *2.34*× | 3.18 | *2.64*× | 3.18 | *2.63*× | 2.54 | *2.13*× | 2.14 | *1.76*× | 2.78 | *2.31*× |
| DSpark | 3.90 | *3.19*× | 4.09 | *3.31*× | 3.78 | *3.02*× | 3.15 | *2.49*× | 3.49 | *2.84*× | 3.50 | *2.83*× | 2.73 | *2.27*× | 2.25 | *1.81*× | 2.99 | *2.44*× |
| H-Spec | **4.14** | ***3.40***× | **4.32** | ***3.52***× | **4.00** | ***3.28***× | **3.37** | ***2.69***× | **3.69** | ***3.03***× | **3.74** | ***3.05***× | **2.93** | ***2.45***× | **2.39** | ***1.92***× | **3.22** | ***2.68***× |

As shown in Table 6, H-Spec continues to lead in draft acceptance and batch-size-1 ITL speedup. Averaged over 9 tasks, H-Spec improves in $\tau$ over the best baseline by 12.5%, 3.8%, and 6.4% for the Llama, Qwen3 4B, and 8B target models, respectively. For speedup, the margins over the best baseline are 13.1%, 4.3%, and 7.5% across three targets. Importantly, H-Spec leads in every task for every evaluated target model, with the pairwise 95% bootstrap CIs sitting above zero in all cases. These results further demonstrate H-Spec's consistent gains across diverse domains.

### E.4 Extended sensitivity analysis

Complementing §4.4, we report analysis experiments on additional target models: Llama3.1-8B-IT and Qwen3-4B. Overall, our analysis findings hold across target models.

Table 7: **Sensitivity to inference draft length.** MAL ($\tau$) and batch-size-1 speedup on Llama3.1-8B-IT and Qwen3-4B for draft lengths $k \in \{3, 5, 7\}$.

| | $k=3$ | | $k=5$ | | $k=7$ | |
|---|---|---|---|---|---|---|
| L-8B-IT | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 2.40 | *1.99*× | 2.68 | *2.18*× | 2.69 | *2.16*× |
| DFlash | 2.23 | *1.85*× | 2.38 | *1.93*× | 2.47 | *2.03*× |
| DSpark | 2.32 | *1.91*× | 2.64 | *2.14*× | 2.72 | *2.23*× |
| H-Spec | **2.55** | ***2.12***× | **2.98** | ***2.43***× | **3.08** | ***2.51***× |
| Q-4B | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 2.59 | *1.92*× | 2.87 | *2.12*× | 2.99 | *2.18*× |
| DFlash | 2.51 | *1.92*× | 2.79 | *2.11*× | 2.86 | *2.26*× |
| DSpark | 2.60 | *1.97*× | 2.93 | *2.20*× | 3.09 | *2.37*× |
| H-Spec | **2.69** | ***2.05***× | **3.09** | ***2.32***× | **3.24** | ***2.50***× |

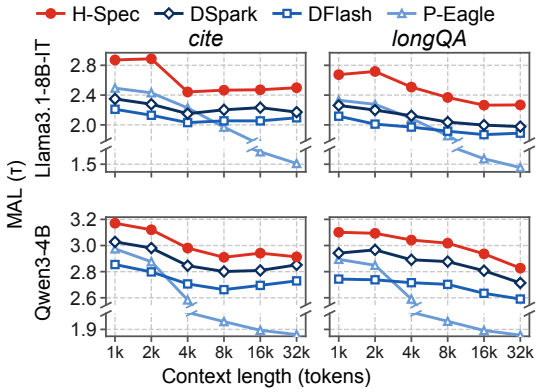



Figure 11: **Sensitivity to context length.** MAL ($\tau$) as a function of context length on *cite* and *longQA* for Llama3.1-8B-IT and Qwen3-4B.

**Sensitivity to draft length.** Table 7 shows that H-Spec continues to lead in $\tau$ and speedup across inference-time draft length for both target models. The margin in $\tau$ and speedup over the best baseline roughly increases with draft length, consistent with our observation on the Qwen3-8B target.

**Long-context generalization.** As shown in Figure 11, H-Spec achieves the best $\tau$ across context lengths on both *HELMET* tasks for the additional targets. This shows that hybrid context injection provides strong, sustained target conditioning as context length increases.

### E.5 Zero-shot transfer to finetuned target models

To evaluate the robustness of H-Spec's gains under target-model shifts, we serve drafters under zero-shot transfer on additional target models finetuned from their original training target. We report $\tau$ and Spd. averaged over eight augmented Spec-Bench tasks.

Table 8: **H-Spec remains the best drafter under zero-shot transfer.** Drafters trained for the base Llama3.1-8B-IT, Qwen3-4B, and Qwen3-8B target models are evaluated on two fine-tuned targets each. † indicates that the chat template changes in addition to the model weights.

| | **Llama3.1-8B-IT** | | | | **Qwen3-4B** | | | | **Qwen3-8B** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Selene-1 | | R1-Distill† | | SFT-Sci | | Jan-Nano† | | II-Medical | | R1-0528† | |
| **Method** | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* | $\tau$ | *Spd.* |
| P-Eagle | 2.71 | *2.08*× | 1.88 | *1.49*× | 2.55 | *1.89*× | 2.63 | *2.16*× | 2.54 | *2.08*× | 1.98 | *1.63*× |
| DFlash | 2.44 | *2.00*× | 1.72 | *1.39*× | 2.44 | *2.01*× | 2.53 | *2.12*× | 2.38 | *1.97*× | 1.90 | *1.58*× |
| DSpark | 2.71 | *2.14*× | 1.92 | *1.53*× | 2.62 | *2.02*× | 2.71 | *2.23*× | 2.49 | *2.04*× | 1.96 | *1.60*× |
| H-Spec | **2.94** | ***2.50***× | **2.26** | ***1.82***× | **2.71** | ***2.15***× | **2.89** | ***2.38***× | **2.69** | ***2.21***× | **2.11** | ***1.74***× |

As shown in Table 8, H-Spec remains the best drafter under both weight-only and weight-and-chat-template transfers, demonstrating the robustness of its gains to target-model shifts. Paired bootstrap 95% CIs show that H-Spec's gain over the best baseline is statistically significant in 11/12 (target transfer, metric) settings. The only exception is $\tau$ on SFT-Sci, where the CI for the gain over the best baseline, DSpark, includes zero; however, H-Spec's lead over the second-best baseline, P-Eagle, is statistically significant on that setting.