WHEN PARALLEL DRAFTER MEETS PARALLEL SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

DSpark-style parallel drafters have made speculative decoding highly effective, yet their draft phase remains serialized on the critical path of every round. Parallel speculative decoding (PSD) overlaps drafting with verification, yet existing methods must guess the accepted prefix and bonus token in advance: a wrong guess reverts the whole batch to serial drafting. We present DPara, a PSD framework that reuses effective parallel drafters yet guarantees backbone–verification overlap in every round, thereby eliminating this probabilistic fallback altogether. While the target verifies, DPara's diffusion backbone precomputes draft representations for *every* acceptance boundary with the bonus left unspecified; a lightweight autoregressive head then combines the revealed verification outcome with the matching precomputed representation to emit the next round's draft tokens almost instantly—fully parallelizing the dominant backbone forward with verification and leaving only the negligible head cost serial. Experiments on Qwen3-8B and Qwen3-14B across seven math, coding, and chat benchmarks show that DPara achieves average speedups of $3.21\times$ and $3.52\times$ over autoregressive decoding, surpassing the strongest serial and parallel speculative decoding baselines alike.

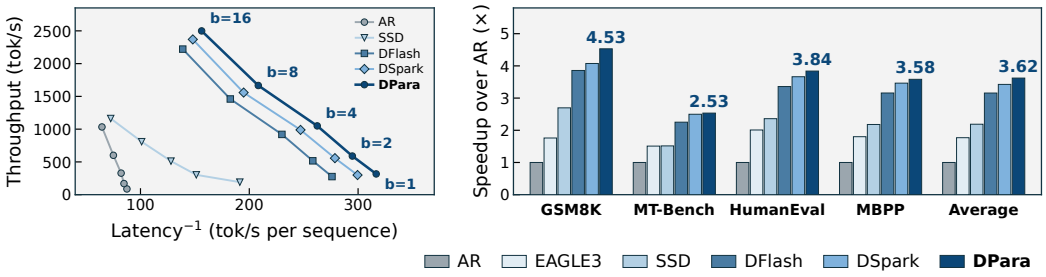

Figure 1: Performance of DPara and other baselines on Qwen3-14B. **Left**: throughput–latency frontier, averaging over the four benchmarks shown on the right; each curve traces batch sizes b=1, 2, 4, 8, 16, with batch labels on DPara. **Right**: end-to-end speedup over AR at b=1; DPara reaches an arithmetic mean of $3.62\times$ across these four benchmarks, exceeding DSpark on each.

1 INTRODUCTION

Speculative decoding (SD) accelerates large language model inference effectively while preserving the target distribution (Leviathan et al., 2023; Chen et al., 2023). Its end-to-end efficiency is governed by two quantities: *average acceptance length* and *drafting latency* on the critical path.

Diffusion-inspired parallel drafters, like DART (Liu et al., 2026) and DFlash (Chen et al., 2026), predict logits for multiple future positions in a single forward pass and permit greater model capacity under the same latency budget. Parallel logit prediction alone, however, does not capture dependencies among the sampled draft tokens. Subsequent designs such as DSpark (Cheng et al., 2026) and Domino (Huang et al., 2026) address this limitation by coupling the parallel diffusion backbone with a lightweight autoregressive head, which restores sequence consistency. Yet even under these advanced drafters, the conventional sequential draft–verify schedule keeps drafting on the critical path, where it accounts for 24%–32% of end-to-end latency (Appendix A).

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

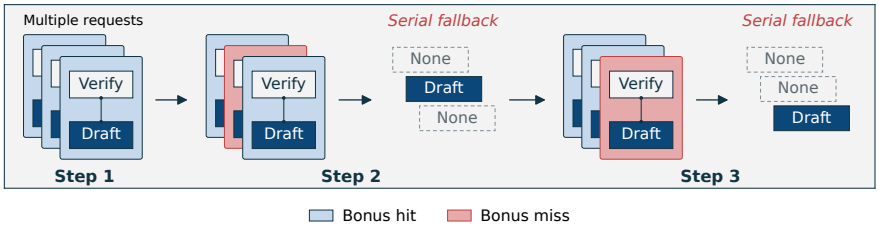

Figure 2: SSD drafts and verifies a batch of requests together. When any request's predicted bonus token mismatches (red), the whole batch loses its precomputed continuation and reverts to serial drafting. As the batch grows, the chance of at least one mismatch rises, so larger batches fall back more often and forfeit the parallel draft–verify gain.

Parallel speculative decoding (PSD) attacks this serialization directly by overlapping draft generation with target verification (Liu et al., 2025; McDanel, 2024; Shen et al., 2025; Kumar et al., 2026). On behalf of these methods, SSD (Kumar et al., 2026) extends speculation to verification itself: while the target verifies the current step, the drafter predicts the accepted prefix and *bonus token* and generates subsequent draft tokens in advance. A matching outcome makes the next step's draft tokens immediately available, eliminating drafting latency from the critical path. A mismatch, however, reverts SSD to serial speculative decoding (Figure 2), and because the probability of at least one mismatch grows with batch size, the entire batch is forced back to serial execution ever more frequently—at batch size 16, up to 96% of steps fall back (Appendix B). Outcome-dependent overlap therefore exchanges a best-case gain for a regression that intensifies with serving load. This tension raises a natural question:

*Can we **completely eliminate probable serial fallback** from parallel speculative decoding?*

We answer this question affirmatively by severing the probabilistic dependence on bonus-token prediction: the key is to separate the computation that must await the verification outcome from the computation that can proceed independently of it. The DSpark-style architecture affords precisely this separation—the expensive diffusion backbone computes draft representations while verification is still in flight, and the lightweight autoregressive head incorporates the actual bonus token afterward. Deferring bonus-token conditioning to the head yields **backbone–verification overlap in every round, without any prediction-induced fallback**.

We present DPara, a PSD framework that makes this overlap independent of the verification outcome. DPara prepares diffusion backbone prediction outputs for *every* possible accepted prefix boundary and leaves the bonus token unspecified until verification completes. Because this guarantee holds for each request individually, enlarging the batch cannot reintroduce the prediction-induced serial fallback. Covering all acceptance boundaries in a single pass, however, requires a backbone that predicts from multiple feature-less anchors—a regime the standard DFlash model does not satisfy. We close this gap by finetuning: M-DFlash initializes from the published DSpark checkpoint and adapts *only the diffusion backbone* to the multiple-anchor, feature-less regime, introducing no new trainable parameters and keeping the AR head frozen as published. Provisioned to complete within the verification window, the diffusion backbone—the dominant component of drafting latency—is fully hidden, leaving only the lightweight autoregressive head on the critical path. DPara thereby preserves the effectiveness of parallel drafters while excising backbone execution entirely from the serial draft–verify schedule.

Figure 1 previews the empirical gains. Across math, coding, and chat benchmarks on Qwen3-8B and Qwen3-14B, DPara exceeds other baselines on every model–dataset pair, averaging $3.21\times$ and $3.52\times$ speedup over autoregressive decoding (§4.2). Against DSpark, the strongest serial drafter, this corresponds to average improvements of 11.1% and 5.1%; against the parallel baselines PEARL and SSD on Qwen3-14B, the average speedup rises by 31% and 64%. The advantage does not erode under batching: DPara stays ahead of DSpark at every batch size from 1 to 16 and retains over $2\times$ AR throughput at batch 16, while outcome-dependent methods such as SSD suffer a markedly steeper throughput drop-off as batches grow (Figure 1). The gains likewise survive a leaner drafting budget: DPara-A10 retains 98% of DPara's speedup while cutting the drafting-device cost by $\sim 90\%$, whereas DPara-single retains 90%–95% at small batches (§4.3). Ablations further confirm that finetuning only the diffusion backbone raises the average acceptance length by 30.7%.

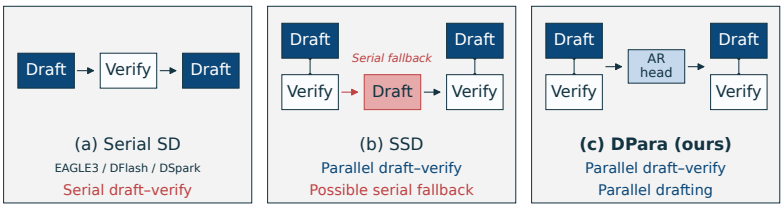

Figure 3: Scheduling of drafting and verification across speculative decoding paradigms. **(a) Serial SD**: autoregressive (EAGLE3) or parallel drafters (DFlash/DSpark) run draft–verify in sequence, leaving drafting on the critical path. **(b) SSD**: overlaps drafting with verification, but a mispredicted bonus token forces serial fallback. **(c) DPara (ours)**: the backbone overlaps verification for every acceptance outcome while drafting stays parallel.

This paper makes the following contributions:

- The **first** parallel speculative decoding framework powered by DSpark-style parallel drafters, establishing **state-of-the-art** inference performance across both serial and parallel baselines.
- DPara, a PSD framework that covers all acceptance boundaries **without any serial backbone fallback** and hides the backbone latency entirely within the verification window, enabled by M-DFlash, a multiple-anchor backbone finetuned from a published DSpark checkpoint.
- An extensive evaluation on Qwen3-8B and Qwen3-14B: DPara exceeds the best serial and parallel baselines on every model–dataset pair (up to $4.53\times$, averaging $3.21\times$/$3.52\times$ over AR) and sustains its advantage from batch 1 to 16; further explorations of heterogeneous drafting and single-GPU colocation retain most of the speedup under a reduced drafting budget.

2 Background and Related Work

2.1 Speculative Decoding

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) alleviates memory-bound decoding by amortizing the weight reads of a large target model $\mathcal{M}_t$ across multiple tokens. Given a prefix C, a lightweight drafter $\mathcal{M}_d$ proposes a block $\tilde{y}_{1:K} \sim q(\cdot \mid C)$, which $\mathcal{M}_t$ scores in one forward pass. Distribution-preserving rejection sampling retains an accepted prefix and appends a *bonus token* b: a correction sampled at the first rejection, or an additional target sample after full acceptance.

2.2 Parallel Drafter

To overcome the sequential drafting overhead of autoregressive drafters such as EAGLE3 (Li et al., 2025), DART (Liu et al., 2026) and DFlash (Chen et al., 2026) draw inspiration from diffusion language models (dLLMs) and predict a block of future positions in one forward pass, substantially reducing draft-generation cost. Independent prediction at different positions, however, can produce incoherent draft sequences and declining acceptance length (τ). DART applies *N-gram* corrections to its parallel logits, whereas Domino (Huang et al., 2026) and DSpark (Cheng et al., 2026) learn a lightweight autoregressive head that conditions each position on the preceding sampled tokens, which preserves the efficient parallel backbone while restoring local causal dependence, leading to longer τ. Nevertheless, the drafter and verifier remain sequentially coupled across decoding steps: **the drafter must finish before verification can begin, leaving drafting on the critical path and making it a nontrivial component of end-to-end latency**.

2.3 Parallel speculative decoding

Figure 3 contrasts the execution schedules of serial SD, SSD, and DPara. PSD overlaps next-step drafting with current-step verification to hide the drafting interval that otherwise delays the next verification. The drafter must therefore generate tentative continuations before the accepted prefix and *bonus token* are known. PEARL (Liu et al., 2025) and AMUSD (McDanel, 2024) generate continuations during verification under an all-accepted assumption, which provides only limited opportunities for parallelism. SpecBranch (Shen et al., 2025) and SSD (Kumar et al., 2026) predict likely rejection positions and bonus tokens to reduce rollback. In all these methods, an unanticipated verification outcome can invalidate the precomputed continuation and fall back to *serial speculation*, **so none of them guarantees draft–verify parallelism without fallback in every round**.

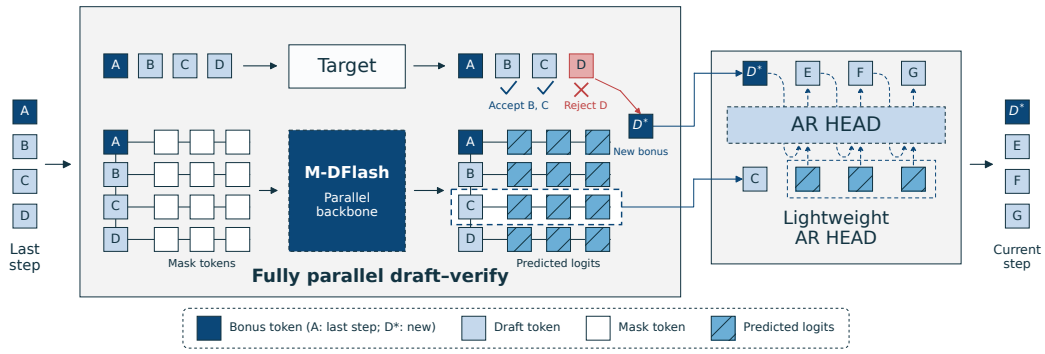

Figure 4: One decoding step of DPara. With $\boxed{A}$ as the bonus token from last step, the target verifies $\boxed{ABCD}$ while the parallel backbone concurrently precomputes d future logits for every acceptance boundary. Accepting $\boxed{BC}$ and rejecting $\boxed{D}$ yields bonus token $\boxed{D^*}$. The lightweight AR head combines $\boxed{D^*}$ with the logits following $\boxed{C}$ to generate next step's draft tokens $\boxed{EFG}$ quickly.

3 DPara Design

In this section, we will go through the design of DPara to guarantee complete backbone–verification overlap in every PSD step.

3.1 Backbone requirement of DPara

DPara places a requirement on the *diffusion backbone* that the standard DFlash model does not satisfy. Under the conventional serial schedule, drafting runs *after* verification: the target has already finished verification, so a fresh target feature exists at every position before bonus token—*the bonus itself carries no feature of its own*. DFlash (Chen et al., 2026) anchors on this *single* token, appended to the fully featured prefix, to predict the logits of the next d positions in one backbone forward. DPara instead launches the backbone *while* the target is still verifying the current block (Figure 4). Write the draft tokens from last step as A, B, C, D, where A is the previous bonus and B, C, D are the drafts being scored, and let $r \in \{0, \ldots, d\}$ be the number of accepted drafts and b the new bonus. At launch time the backbone knows neither r nor b; all it holds are the tokens carried over from the previous round, of which the accepted prefix $A, t_1 \ldots, t_r$ forms $k = r + 1$ (≥ 1) anchor tokens that have *not* yet passed through the target and therefore carry *no* target features. This requires:

> The DPara's backbone must predict the next d positions from **multiple, feature-less anchors**, and must do so for every possible r at once, since the true acceptance length is not revealed until verification finishes.

This requirement, however, does not change *what* the backbone computes; it *generalizes* it. DFlash already predicts future logits from *a single anchor* given the fully featured prefix; DPara only asks it to extend this *from a single anchor to multiple anchors*. Because multiple-anchor prediction is a strict extension of DFlash's single-anchor semantics rather than a different function, we can obtain the backbone DPara needs by finetuning DFlash (Figure 5). We call this multiple-anchor diffusion backbone **M-DFlash** (Multiple-anchor-token DFlash).

3.2 Finetuning DFlash to M-DFlash

M-DFlash introduces *no new trainable network parameters*. We initialize it from the official DSpark checkpoint (Cheng et al., 2026)—a DFlash-family drafter that couples the block-diffusion backbone with a lightweight Markov autoregressive (AR) head—and adapt *only the backbone part* to the multiple-anchor, feature-less regime above. The AR head is precisely the component that lets DPara defer bonus conditioning to the post-verification stage, so we keep it frozen as published.

Algorithm 1 One steady-state round of DPara (draft length d).

Require: feature cache C; feature-less draft block $t_{0:d}$ (t_0 = previous bonus)
Ensure: accepted tokens, bonus b; updated C and next block

// launched together, run concurrently

Precompute (drafter): M-DFlash on spine $t_{0:d}$, cache C
  $\rightarrow$ base logits $\ell_{1:d}^{(r)}$ for all $r \in \{0{:}d\}$
Verify (target): $\mathcal{M}_t$ forwards $t_{0:d}$
  $\rightarrow$ accepted length r, bonus b, features of $t_{0:r}$

// barrier: (r, b) now known

select precomputed branch r ▷ enumerated $\Rightarrow$ no fallback
$y_0 \leftarrow b$
for $i = 1, \ldots, d$ **do** ▷ AR head: lightweight cost
  $z_i \leftarrow \ell_i^{(r)} + WE(y_{i-1}); \quad y_i \sim \mathrm{softmax}(z_i)$
end for
$C \leftarrow [\, C, C_{0:r}^{new} \,]$ ▷ drop rejected $t_{r+1:d}$
$t_{0:d} \leftarrow (b,\, y_{1:d})$ ▷ b = next feature-less anchor
return accepted $t_{1:r}$, bonus b ▷ stop at EOS / length budget

Trainable and frozen parameters. We train the backbone blocks (attention, MLP, and their normalizations), the projection that fuses the concatenated multi-layer target features, and the input and output normalizations. We freeze the entire target $\mathcal{M}_t$, which only supplies online features and supervision, together with the drafter's token embedding, its full-vocabulary LM projection, and both the AR head and the confidence head. Only the backbone faces a changed input regime—*multiple feature-less anchors instead of a single one*—so it is the only part that must adapt.

Shared causal spine with isolated bidirectional branches. M-DFlash covers all $d + 1$ acceptance boundaries in one backbone forward using the attention pattern in Figure 6. In the illustrated example ($d = 3$), the previous bonus A and draft tokens B, C, D form a shared, feature-less causal spine. Each anchor attends to the cached target features, itself, and earlier anchors, but never to later anchors or mask tokens. The four groups of three masks correspond to the possible accepted prefixes $\boxed{A}$, $\boxed{AB}$, $\boxed{ABC}$, and $\boxed{ABCD}$. Within each group, the masks attend bidirectionally to one another, read the cached target features, and access only the corresponding anchor prefix; all other branches remain invisible. For example, the branch following C reads A, B, C but not D, so its predictions remain valid if verification accepts B, C and rejects D. The triangular light blue region and diagonal medium blue blocks therefore encode two complementary constraints: causal sharing along the spine and bidirectional attention within isolated branches. Unlike an ordinary triangular causal mask, this tree-structured mask computes the shared anchors once while producing d future logits for every possible acceptance boundary.

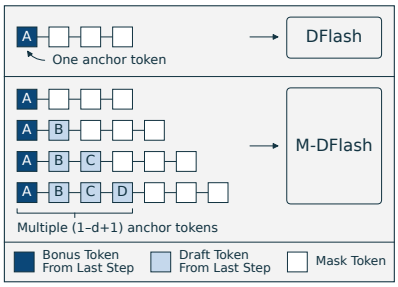

Figure 5: Input regimes of DFlash and M-DFlash. **Top**: DFlash anchors on a single forwarded token appended to a fully featured prefix. **Bottom**: M-DFlash predicts the next d logits from multiple ($1{\sim}d{+}1$) feature-less anchor tokens for every acceptance boundary in one forward pass—the regime required to launch the backbone during verification.

Loss. For a selected boundary, let ℓ_i be the base logits at mask position i and let the frozen AR head add a correction from the previously sampled token,

$$z_i = \ell_i + WE(y_{i-1}), \qquad y_0 = b, \quad i = 1, \ldots, d,$$

where E is the head's independent low-rank token embedding and W projects it to the vocabulary; the head reads only the preceding token. Writing $q_i = \mathrm{softmax}(z_i)$ for the student and p_i for the target distribution at the same position, we minimize a position-weighted mixture of cross-entropy

and full-vocabulary probability distance (Cheng et al., 2026),

$$\mathcal{L} = \frac{\sum_i m_i\, w_i \left[\, 0.1 \left(-\log q_i(y_i) \right) + 0.9 \sum_{v \in \mathcal{V}} |q_i(v) - p_i(v)| \right]}{\sum_i m_i\, w_i}, \qquad w_i = e^{-(i-1)/\gamma},$$

with decay $\gamma = 4$ and m_i a validity mask that keeps only positions whose supervision falls inside the assistant span. The distribution-matching term aligns the drafter's proposal with the target's acceptance behavior over the whole vocabulary; the confidence loss is disabled, so the fixed budget d is never dynamically truncated.

3.3 DPara Algorithm

Algorithm 1 details one steady-state round. DPara maintains a target-feature cache C over the accepted history and a feature-less draft block $t_{0:d}$ carried over from the previous round, where t_0 is the previous bonus and $t_{1:d}$ are the drafts now being scored. The M-DFlash backbone runs on the shared spine $t_{0:d}$ concurrently with the target verifying that same token block, and emits base logits for *all* $d+1$ acceptance boundaries. This is exactly where DPara departs from SSD (Kumar et al., 2026), which predicts the accepted prefix and bonus token to pregenerate the next draft and reverts to serial speculation whenever that prediction misses. DPara instead predicts future logits, and it never has to *guess the verification outcome*: it commits to no particular accepted prefix or particular bonus token, so whatever (r, b) the verifier returns, a finished branch is already waiting.

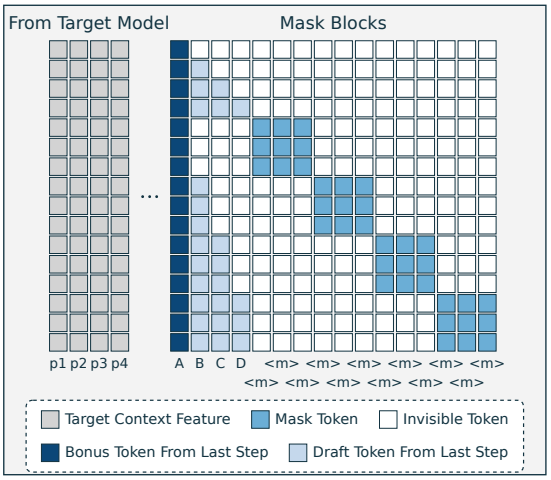

Figure 6: M-DFlash attention mask for $d = 3$. Colored cells indicate visible connections and white indicate masked connections. All queries access cached target features (gray). The previous bonus A (dark blue) and drafts B, C, D (light blue) form a shared causal spine. Four bidirectional three-mask branches (medium blue) read the anchor prefixes $\boxed{A}$, $\boxed{AB}$, $\boxed{ABC}$, and $\boxed{ABCD}$, respectively, without attending to one another.

Only the head stays on the critical path. When verification returns (r, b), DPara selects branch r and runs *only* the lightweight AR head to turn its corresponding predicted logits into the next draft $y_{1:d}$; since the backbone already overlapped with verification, this low-rank correction is the sole additional cost exposed between two rounds. The verified features of $t_{0:r}$ are appended to C and the rejected suffix dropped, while the bonus b—sampled at the last accepted position and therefore carrying no target feature *yet*—becomes the feature-less t_0 of the next round, to be forwarded until next verification. Approximating a round as

$$T_{\text{step}} \approx \max\big(T_{\text{verify}}, T_{\text{precompute}}\big) + T_{\text{Head}},$$

the diffusion backbone term is hidden whenever it fits inside the verification window.

4 Experiments

4.1 Experiments Setup

Target models. We train M-DFlash drafters for two models of the Qwen3 family (Qwen3-8B and Qwen3-14B (Yang et al., 2025)); the detailed training recipe is provided in Appendix E.

Hardware. All training is conducted on a server equipped with NVIDIA GB200 GPUs (4 GPUs, 268 GB). The main results are obtained on an NVIDIA H800 node (8 GPUs, 80 GB each). We also conduct additional inference experiments on NVIDIA GB200 GPUs. For PSD methods, including PEARL, SSD, and DPara, we use one additional GPU dedicated to drafting. We further explore an NVIDIA A10 (1×24GB) as a resource-constrained drafting device (§4.3).

Table 1: DPara against parallel (PSD) and serial (SD) speculative decoding baselines on Qwen3-8B and Qwen3-14B. Within each model block, the upper group of rows lists PSD baselines, the middle group serial SD baselines, and the last row DPara. We report decoding speedup over the AR baseline and the average acceptance length τ, with batch size 1, greedy decoding, up to 512 new tokens, draft length 7. SSD-E3 takes corresponding EAGLE3 weight of target as draft model. SSD and PEARL in Qwen3-14B take Qwen3-0.6B as draft model.

Model	Method	MATH				CODING						CHAT				AVG.	
		GSM8K		MATH-500		HumanEval		MBPP		CodeAlpaca		Alpaca		MT-Bench			
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Q3-8B	SSD-E3	1.51×	2.39	1.54×	2.42	1.72×	2.75	1.58×	2.49	1.67×	2.62	1.29×	2.17	1.34×	2.26	1.52×	2.44
	EAGLE3	1.44×	2.83	1.46×	2.84	1.67×	3.26	1.51×	2.98	1.58×	3.18	1.18×	2.45	1.27×	2.57	1.44×	2.87
	DFlash	3.38×	6.08	3.28×	5.88	2.93×	5.25	2.80×	4.95	2.77×	4.95	1.81×	3.28	1.94×	3.56	2.70×	4.85
	DSpark	3.48×	6.51	3.46×	6.39	3.15×	5.82	3.00×	5.51	3.01×	5.55	2.00×	3.72	2.13×	4.01	2.89×	5.36
	DPara	**4.17**×	5.82	**3.95**×	5.41	**3.50**×	4.78	**3.27**×	4.48	**3.25**×	4.47	**2.10**×	2.92	**2.25**×	3.12	**3.21**×	4.43
Q3-14B	SSD	2.70×	5.47	2.80×	5.63	2.36×	4.87	2.18×	4.47	2.05×	4.39	1.39×	3.17	1.51×	3.47	2.14×	4.50
	SSD-E3	1.76×	2.48	1.80×	2.48	2.02×	2.80	1.82×	2.54	1.91×	2.68	1.48×	2.17	1.57×	2.28	1.77×	2.49
	PEARL	3.35×	3.70	3.58×	3.85	2.95×	3.18	2.47×	2.81	2.58×	2.81	1.84×	1.98	2.03×	2.20	2.68×	2.93
	SD	2.21×	5.47	2.28×	5.65	1.99×	4.86	1.85×	4.46	1.76×	4.39	1.31×	3.17	1.42×	3.47	1.83×	4.50
	EAGLE3	1.76×	2.94	1.74×	2.93	2.01×	3.35	1.80×	3.04	1.86×	3.24	1.42×	2.42	1.51×	2.57	1.73×	2.93
	DFlash	3.86×	6.03	3.76×	5.92	3.36×	5.20	3.16×	4.89	3.13×	4.97	2.07×	3.24	2.25×	3.55	3.08×	4.83
	DSpark	4.07×	6.53	4.01×	6.39	3.66×	5.79	3.46×	5.48	3.45×	5.60	**2.32**×	3.71	2.50×	4.02	3.35×	5.36
	DPara	**4.53**×	5.73	**4.32**×	5.37	**3.84**×	4.76	**3.58**×	4.45	**3.49**×	4.46	**2.32**×	2.91	**2.53**×	3.16	**3.52**×	4.41

Benchmarks. We evaluate DPara on seven datasets across three workload categories: **Math** for mathematical reasoning, including GSM8K (Cobbe et al., 2021) and MATH-500 (Lightman et al., 2023); **Chat** for conversation and instruction following, including MT-Bench (Zheng et al., 2023) and Alpaca (Taori et al., 2023); and **Coding** for code generation, including HumanEval (Chen et al., 2024), MBPP (Austin et al., 2021), and CodeAlpaca (Lhoest et al., 2021).

4.2 MAIN RESULT

Gains over both serial and parallel speculation. DPara achieves average speedups over autoregressive decoding of $3.21\times$ on Qwen3-8B and $3.52\times$ on Qwen3-14B across seven benchmarks at batch size 1 (Table 1). It attains the highest reported speedup on every model–dataset pair, with a tie with DSpark on Qwen3-14B Alpaca. The advantage spans math, coding, and chat, and extends beyond methods that serialize drafting and verification: on Qwen3-14B, DPara improves the average speedup over PEARL and SSD by 31% and 64%, respectively. These results show that effective backbone–verification overlap translates into lower end-to-end decoding latency. Across the evaluated models and workloads, DPara achieves state-of-the-art speedups across both serial and parallel speculative decoding methods.

Balancing draft acceptance and critical-path cost. DPara is designed around a joint optimization of acceptance length and drafting cost for end-to-end latency: it preserves comparatively high acceptance through a high-capacity diffusion backbone while hiding the backbone forward pass entirely within verification, leaving only the lightweight AR head on the drafting critical path. This balance yields the best overall end-to-end decoding latency among the evaluated methods. Specifically, DPara retains average acceptance lengths of 4.43 and 4.41 on Qwen3-8B and Qwen3-14B, respectively, compared with 5.36 for DSpark, the strongest serial SD baseline (Table 1). This acceptance gap reflects the information constraints imposed by PSD: when the backbone starts, neither the accepted length nor the new bonus is known, and fresh target features for the draft tokens carried over from the previous round are unavailable. Unlike serial drafting, which conditions on a resolved acceptance boundary and a fully featured prefix, DPara must precompute all boundaries from feature-less anchors and defer bonus conditioning to the AR head. The resulting information deficit constrains the acceptance achievable by the parallel backbone relative to serial drafting. Nevertheless, the end-to-end results show that removing backbone execution from the critical path more than compensates for this acceptance gap: DPara improves average speedup over DSpark by 11.1% on Qwen3-8B and 5.1% on Qwen3-14B. The net benefit is lower latency per generated token, achieved by retaining strong draft acceptance without paying for a serial backbone forward pass.

The advantage persists under batching. DPara remains faster than DSpark at every measured batch size from 1 to 16 on every evaluated benchmark, for both target models (Table 2 shows GSM8K and HumanEval). The single-request advantage therefore survives batched execution, rather than disappearing as more requests share each forward pass. At batch size 16, DPara retains $2.55\times$ and $2.17\times$ speedups over AR on Qwen3-8B, and $2.95\times$ and $2.52\times$ on Qwen3-14B, for GSM8K and HumanEval, respectively. Its throughput exceeds DSpark's by approximately 6%–8% across these four settings. Nevertheless, batching reduces the relative benefit of speculation: absolute throughput increases, but speedup over the correspondingly batched AR baseline declines for both methods. Outcome-independent overlap thus does not imply batch-invariant speedup; the empirical result is a sustained advantage over DSpark and more than $2\times$ AR throughput throughout the measured range.

Table 2: Batched inference on Qwen3-8B/14B (Q3-8B/14B). Each method reports throughput (tok/s) and speedup over AR.

Model	Task	Method	Batch size				
			1	2	4	8	16
Q3-8B	GSM8K	AR	141	277	529	937	1520
		DSpark	491 3.48×	901 3.25×	1507 2.85×	2426 2.59×	3627 2.39×
		DPara	588 4.17×	1052 3.80×	1772 3.35×	2736 2.92×	3869 2.55×
	Human-Eval	AR	141	269	532	957	1684
		DSpark	445 3.15×	815 3.02×	1464 2.75×	2283 2.39×	3382 2.01×
		DPara	495 3.50×	914 3.39×	1565 2.94×	2567 2.68×	3654 2.17×
Q3-14B	GSM8K	AR	87	171	330	597	1020
		DSpark	355 4.07×	647 3.79×	1142 3.47×	1813 3.03×	2847 2.79×
		DPara	395 4.53×	721 4.22×	1269 3.85×	1999 3.35×	3010 2.95×
	Human-Eval	AR	87	170	333	622	1080
		DSpark	319 3.66×	598 3.52×	1082 3.25×	1688 2.71×	2523 2.34×
		DPara	335 3.84×	634 3.74×	1140 3.42×	1824 2.93×	2724 2.52×

4.3 Further explorations of resource-constrained drafting

Dedicating a GPU to drafting lets PSD reduce latency by overlapping backbone execution with verification, but incurs an additional hardware cost. We therefore explore two complementary ways to cut the drafting budget while retaining DPara's latency advantage: drafting on a much weaker GPU paired with the H800 target (*DPara-A10*, §4.3.1), and colocating drafting and verification on one GPU (*DPara-single*, §4.3.2). Both are compared with the default two-GPU DPara; implementation details are deferred to Appendix D.

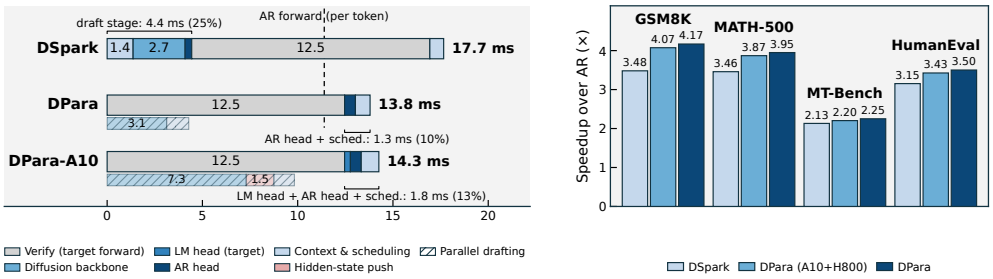

Figure 7: **Left**: anatomy of one speculation round on Qwen3-14B (GSM8K, batch 1); bars are measured critical paths and hatched sub-bars show drafting work overlapped with verification. DSpark serializes its draft stage (4.4 ms, 25% of the step); DPara hides the backbone and exposes only 1.3 ms (10%); DPara-A10 also keeps the A10-side backbone and hidden-state push inside the verification window, adding only the target-side LM head (0.3 ms) to the critical path. **Right**: Qwen3-8B speedup over AR at batch size 1 with the M-DFlash backbone running on an NVIDIA A10 as the drafting GPU, compared with DSpark and DPara.

4.3.1 Heterogeneous drafting

Engineering adaptation to the heterogeneous placement. DPara-A10 pairs the H800 target with an NVIDIA A10 for drafting (Appendix D), so the drafting pipeline must be partitioned across two unequal GPUs. A naive port keeps the whole pipeline—backbone, LM head, and AR-head sampling—on the A10. Its AR-head sampling then sits on the critical path, where the latency negli-

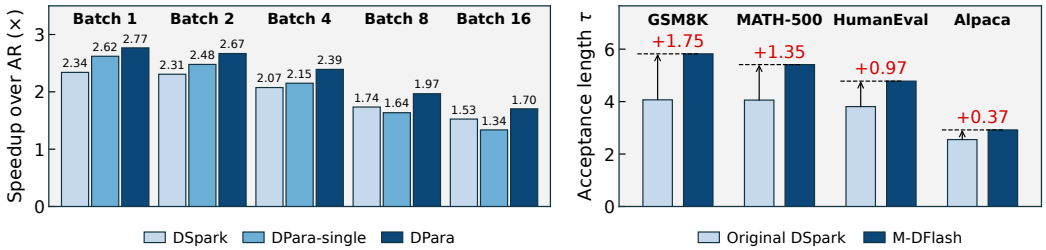

Figure 8: **Left**: Qwen3-8B speedup over AR under DPara-single and DPara with DSpark at different batch sizes; each bar averages the seven benchmarks. **Right**: acceptance length τ gains after finetuning on Qwen3-8B's original DSpark checkpoint; both checkpoints run within DPara with the same frozen AR head, and red labels show absolute gains.

gible on an H800 is magnified on the weaker GPU, motivating a move to the target. But a target-side AR head must be fed full-vocabulary logits, an expensive cross-interconnect transfer. We therefore also move the LM head to the target, so the A10 ships only the backbone hidden states—roughly $1/30$ the size of the logits—which cross the link quickly. The residual cost is one LM-head forward on the target's critical path: we trade a little compute for a large reduction in network transfer.

Impact on end-to-end speed. This partition preserves DPara's overlap structure on the weaker GPU (Figure 7, left): within a single verification window the A10 completes both the drafting backbone forward and the asynchronous transfer of the hidden states of all predicted branches, leaving only the target-side LM head on the critical path. DPara-A10 thus retains about 98% of DPara's speedup on all four benchmarks and stays well ahead of DSpark (Figure 7, right); detailed per-step and per-benchmark measurements are in Appendix D.

4.3.2 Single-GPU colocation

DPara-single executes verification and backbone precomputation on two CUDA streams of one GPU, captured in a single CUDA graph (Appendix D). On Qwen3-8B with GB200, it reaches $2.62\times$ speedup over AR at batch size 1, compared with $2.34\times$ for DSpark and $2.77\times$ for DPara, and retains 90%–95% of DPara's speedup across batch sizes 1–4 while halving the GPU count (Figure 8, left). The advantage over DSpark disappears at batch sizes 8 and 16: decoding there turns compute-bound, and the colocated backbone competes with the target for execution resources and bandwidth (Appendix D). Colocation is therefore a resource-efficient option for small batches, whereas dedicated drafting resources remain preferable as batch size increases.

4.4 Ablation Study

Effectiveness of backbone finetuning. We test whether finetuning only the backbone adapts the drafter to the multiple-anchor, feature-less regime required by DPara (§3.2). We run DPara on Qwen3-8B with the original DSpark checkpoint and the finetuned M-DFlash checkpoint, keeping the AR head and decoding configuration unchanged. Figure 8 (right) shows that finetuning increases τ on all four benchmarks: from 4.07 to 5.82 on GSM8K, 4.06 to 5.41 on MATH-500, 3.81 to 4.78 on HumanEval, and 2.55 to 2.92 on Alpaca. These correspond to relative gains of 43.1%, 33.3%, 25.5%, and 14.4%, respectively; the mean τ across these four datasets rises from 3.62 to 4.73 (30.7%). The consistency of these gains across math, coding, and chat confirms both the necessity of adapting the backbone to multiple feature-less anchors and the reusability of the published AR head without any finetuning.

5 Conclusion

We presented DPara, the first parallel speculative decoding framework powered by DSpark-style parallel drafters. By replacing outcome-dependent speculation with outcome-independent precomputation, DPara eliminates prediction-induced serial backbone fallback entirely while hiding the dominant drafting cost inside the verification window. This decoupling of backbone execution from bonus-conditioned token generation reconciles high-capacity drafting with a minimal critical path, and it translates into state-of-the-art end-to-end decoding latency across math, coding, and chat workloads.

REFERENCES

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

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. *arXiv preprint arXiv:2602.06036*, 2026.

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. *arXiv preprint arXiv:2107.03374*, 2024.

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.

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

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding. *arXiv preprint arXiv:2605.29707*, 2026. doi: 10.48550/arXiv.2605.29707. URL https://arxiv.org/abs/2605.29707.

Tanishq Kumar, Tri Dao, and Avner May. Speculative speculative decoding, 2026. URL https://arxiv.org/abs/2603.03251.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Quentin Lhoest, Albert Villanova Del Moral, Yacine Jernite, Abhishek Thakur, Patrick Von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. Datasets: A community library for natural language processing. In *Proceedings of the 2021 conference on empirical methods in natural language processing: system demonstrations*, pp. 175–184, 2021.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Annual Conference on Neural Information Processing Systems*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Fuliang Liu, Xue Li, Ketai Zhao, Yinxi Gao, Ziyan Zhou, Zhonghui Zhang, Zhibin Wang, Wanchun Dou, Sheng Zhong, and Chen Tian. Dart: Diffusion-inspired speculative decoding for fast llm inference. *arXiv preprint arXiv:2601.19278*, 2026.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. PEARL: Parallel speculative decoding with adaptive draft length. In *International Conference on Learning Representations (ICLR)*, 2025.

Bradley McDanel. Amusd: Asynchronous multi-device speculative decoding for llm acceleration. *arXiv*, 2024.

Yuhao Shen, Junyi Shen, Quan Kong, Tianyu Liu, Yao Lu, and Cong Wang. Speculative decoding via hybrid drafting and rollback-aware branch parallelism. *arXiv*, 2025.

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.

Tengyu Xu, Eryk Helenowski, Karthik Abinav Sankararaman, Di Jin, Kaiyan Peng, Eric Han, Shaoliang Nie, Chen Zhu, Hejia Zhang, Wenxuan Zhou, et al. The perfect blend: Redefining rlhf with mixture of judges. *arXiv preprint arXiv:2409.20370*, 2024.

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. *arXiv preprint arXiv:2505.09388*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

A Latency Breakdown

Profiling setup. We profile per-stage latency on the H800 node used for our main results (§4.1), with Qwen3-14B as the target model on GSM8K: 80 requests, greedy decoding, draft length 7, up to 512 new tokens, thinking mode disabled. We instrument the verifier call and each drafting component with per-phase timers and report the mean interval per speculation round over the 3.6K–4.3K rounds of each run. For DPara, which overlaps drafting with verification, the overlapped intervals are reported against the verification window they hide within and are never summed into the step latency; small residuals between independently timed intervals are folded into the context and scheduling segments. The same pattern holds on the other datasets and on Qwen3-8B.

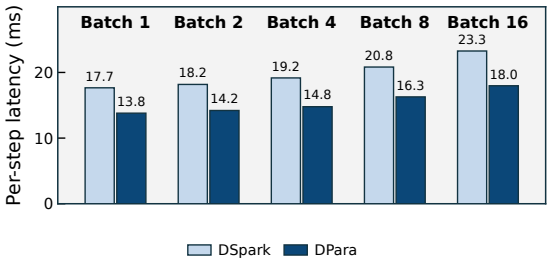

Figure 9: Per-step latency across batch sizes on Qwen3-14B (GSM8K, same profiling setup as Figure 7). DPara is the lower at every batch size; from batch 1 to 16 its step grows 30%, versus 32% for DSpark.

Anatomy of one speculation round. Figure 7 (left) decomposes a single round at batch size 1. Under the conventional draft–verify schedule, DSpark executes its entire draft stage on the critical path before verification can start: the diffusion backbone costs 2.7 ms, the AR head 0.3 ms, and context bookkeeping 1.4 ms, exposing 4.4 ms—25% of its 17.7 ms step. Across both target models and all seven benchmarks, this serial draft stage accounts for 24%–32% of the per-step latency of DFlash/DSpark-style parallel drafters, which is exactly the overhead that PSD must hide to pay off. DFlash behaves the same way (4.1 ms, 24% of a 17.3 ms step). DPara precomputes the same diffusion backbone against the verification window and exposes only the bonus-conditioned AR head plus acceptance bookkeeping: 1.3 ms (10%) on top of a 12.5 ms verify, for a 13.8 ms step—just $1.21\times$ the AR per-token forward and the lowest per-step latency among the compared methods. The colocated single-GPU deployment (§4.3) keeps the same structure and still leads DSpark at 15.1 ms per step.

Per-step latency across batch sizes. Figure 9 tracks the same quantities from batch size 1 to 16. DPara is the fastest at every batch size, and its step grows only mildly, from 13.8 to 18.0 ms (+30%), as verification lengthens. DSpark follows a similar slope (17.7 $\rightarrow$ 23.3 ms, +32%) because its serial draft stage also lengthens with the batch, but it never closes the gap: its step stays 28%–30% longer than DPara's at every batch size. Read together with acceptance length, the per-step gap is decisive: at batch size 1, DSpark's higher τ (6.5 vs. 5.7) does not offset its 28% longer step (2.71 vs. 2.41 ms per token).

B SSD Fallback Rate under Batching

Measuring fallback. SSD pre-drafts against a predicted bonus token while the target verifies (§2). When the revealed bonus token of any request in the batch misses the speculation cache, the precomputed continuation is discarded and the drafter re-drafts the whole batch serially, just in time for the next step. Our profiler instruments these serial re-drafting calls on the drafting GPU; we report the fraction of speculation steps that trigger at least one such call, measured on Qwen3-14B with the configuration of Appendix A.

Fallback rate grows with batch size. Figure 10 shows the measured rates. At batch size 1, 8%–32% of steps already fall back, depending on how predictable the workload's bonus token is; by

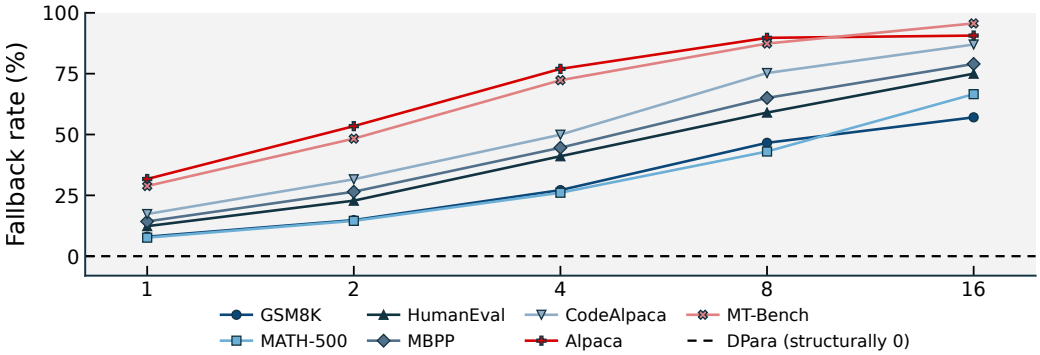

Figure 10: Fraction of speculation steps in which SSD falls back to serial re-drafting, measured on Qwen3-14B (seven benchmarks, batch 1–16, same configuration as Appendix A). A step falls back when the revealed bonus token of any request in the batch misses SSD's speculation cache, forcing the drafter to re-draft the whole batch serially. The rate grows with batch size on every workload and reaches 57%–96% at batch 16. DPara has no such events at any batch size.

batch size 16 the rate reaches 57%–96%. The mechanism is the amplification of a per-request bonus-mismatch probability p into a per-step fallback probability $1-(1-p)^B$ for a batch of B requests, since a single mismatch voids the whole batch's precomputed drafts. Chat workloads suffer most: their per-request mismatch rate is highest at batch 1 (29%–32%), and by batch 16 nearly every step is serial (91%–96%), consistent with SSD's weakest speedups on Alpaca and MT-Bench in Table 1 and its steep throughput drop under batching in §4.2. Math workloads, with the most predictable bonus tokens (8% at batch 1), still fall back in over half of all steps at batch 16. DPara records no such events at any batch size: by covering every acceptance boundary instead of betting on a predicted bonus token, it removes the prediction whose failure triggers the fallback, so the rate is structurally zero.

C Results under Sampling

We repeat the batch-size-1 comparison of Table 1 under sampling: temperature 1.0 for both the target model and the drafter, top-p 1.0, draft length 7, the same 80 requests per benchmark, and up to 512 new tokens. Acceptance follows the standard speculative-sampling rejection rule, under which a draft token can be rejected even when it coincides with the target's most likely continuation.

Table 3: DPara against serial speculative decoding (SD) baselines on Qwen3-8B and Qwen3-14B with sampling decoding. We report decoding speedup over the AR baseline and the average acceptance length τ, with batch size 1, temperature 1.0 for both the target and the drafter, up to 512 new tokens, draft length 7.

Model	Method	Math				Coding						Chat				Avg.	
		GSM8K		MATH-500		HumanEval		MBPP		CodeAlpaca		Alpaca		MT-Bench			
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Q3-8B	DFlash	2.71×	5.28	2.66×	5.15	2.39×	4.64	2.31×	4.47	2.30×	4.46	1.56×	2.98	1.65×	3.18	2.23×	4.31
	DSpark	3.06×	6.15	3.00×	5.96	2.78×	5.53	2.64×	5.24	2.63×	5.22	1.80×	3.54	1.89×	3.74	2.54×	5.05
	DPara	**3.45**×	5.29	**3.17**×	4.77	**2.95**×	4.46	**2.83**×	4.28	**2.77**×	4.20	**1.89**×	2.81	**1.97**×	2.93	**2.72**×	4.11
Q3-14B	DFlash	3.29×	5.35	3.13×	5.04	2.86×	4.61	2.76×	4.53	2.73×	4.45	1.85×	2.96	1.94×	3.19	2.65×	4.30
	DSpark	3.74×	6.18	3.61×	5.91	3.33×	5.46	3.13×	5.25	3.15×	5.17	2.16×	3.54	2.26×	3.79	3.05×	5.04
	DPara	**4.00**×	5.31	**3.74**×	4.88	**3.38**×	4.39	**3.14**×	4.17	**3.16**×	4.15	**2.18**×	2.79	**2.28**×	2.93	**3.13**×	4.09

Sampling lowers acceptance for every method. Each method's τ drops relative to greedy decoding (Table 1): for DPara, from 4.43 to 4.11 on Qwen3-8B and from 4.41 to 4.09 on Qwen3-14B; DSpark falls from 5.36 to 5.05 and 5.04, respectively. Average speedups fall correspondingly: DPara

goes from $3.21\times$ to $2.72\times$ on Qwen3-8B and from $3.52\times$ to $3.13\times$ on Qwen3-14B, with similar relative declines for the serial baselines.

DPara retains the best average speedup. The ranking of methods is preserved under sampling (Table 3). DPara keeps the highest average speedup on both models—$2.72\times$ on Qwen3-8B and $3.13\times$ on Qwen3-14B, versus $2.54\times$ and $3.05\times$ for DSpark—and is the fastest on every dataset of both models, including the chat workload where DSpark tied it under greedy decoding (Table 1). DPara's advantage is orthogonal to the acceptance rule: its backbone stays hidden inside verification whether tokens are accepted greedily or by sampling, so the per-step latency gap over serial drafting (Appendix A) carries over unchanged and only the acceptance length moves.

D Deployment Details for Resource-Constrained Drafting

This appendix expands on the two deployments of §4.3.

Heterogeneous role split. The naive port of DPara's two-GPU role split places the entire drafting side, including the M-DFlash LM head and the AR head, on the drafting GPU. On an A10 this hurts twice: the AR head, which sits on the critical path, adds substantial latency on the weak GPU, and shipping full-vocabulary logits crosses the slow interconnect. DPara-A10 therefore narrows the drafting GPU to the diffusion backbone alone: verification, branch selection, the LM head, and AR-head sampling of the next draft tokens all run on the H800 target. The two machines are connected over Ethernet (NCCL data plane on TCP sockets, RTT ~ 0.8 ms, measured streaming bandwidth 3.9–4.0 GB/s). The A10 pushes only the final hidden states of all $d+1$ branches across d positions (573 KB for Qwen3-14B, about $1/30$ of the corresponding logits, ≈ 0.14 ms to transfer), and the H800 runs the LM head itself (≈ 0.3 ms of extra target-side compute) before the AR head. Trading cheap target-side compute for network transfer keeps every network delay off the critical path.

Asynchronous cross-machine transfer. Decoding proceeds in verification windows. While the H800 verifies, the A10 runs the backbone for the next round and asynchronously pushes the branch hidden states; under full overlap they have all arrived by the time verification returns, so the H800 can immediately run the LM head and AR head. It then packs the accepted length and the target features at the accepted positions into a single asynchronous message back to the A10, which advances its drafting context and starts the next backbone pass. Both sides replay captured CUDA graphs to eliminate scheduling overhead.

Per-step anatomy and end-to-end cost. Figure 7 (left) dissects one speculation round on Qwen3-14B (GSM8K, batch 1). DSpark serializes its 4.4 ms draft stage—25% of the step—before verification, whereas DPara hides the backbone entirely and exposes only 1.3 ms (10%). On the A10, the backbone pass takes 7.3 ms and the asynchronous hidden-state push 1.5 ms; both complete within the 12.5 ms verification window, and the target-side LM head adds only 0.3 ms to the critical path, for a 14.3 ms step versus 13.8 ms for DPara. End to end, DPara-A10 retains about 98% of DPara's speedup on all four benchmarks; averaged over all seven benchmarks, it reaches $3.14\times$ on Qwen3-8B and $3.43\times$ on Qwen3-14B, versus $3.21\times$ and $3.52\times$ for DPara.

Single-GPU execution. We colocate the target and drafter in one process on the same GPU and execute verification and backbone precomputation on two CUDA streams. The branch-selection and bonus-conditioned AR-head logic is unchanged. The head and both forward passes are captured in a single CUDA graph, and acceptance checks and request scheduling remain outside the graph. Incremental projection of cached history features avoids repeated work, while per-request branch state and fixed buffers support safe graph reuse. Unlike the two-GPU deployment, however, the two streams share compute and memory bandwidth: logical concurrency does not guarantee that backbone latency is fully hidden.

Why colocation loses ground at larger batches. The single-GPU advantage over DSpark disappears at batch sizes 8 and 16, and its gap to DPara widens. This trend is consistent with a shift from memory-bound toward compute-bound decoding: larger batches amortize weight reads across more requests. At small, memory-bound batches, spare compute capacity allows useful overlap between

drafting and verification. As compute demand grows, the colocated backbone competes with the target for the same execution resources and bandwidth, increasing the effective per-step cost; separate GPUs avoid this direct competition.

E TRAINING RECIPE

All M-DFlash backbones are finetuned for one epoch with AdamW under a linear-warmup cosine schedule (learning rate 3e-5, 4% warmup). The effective global batch is 128 (micro-batch 1 per GPU with gradient accumulation); each text is truncated to 4096 tokens and sampled with up to 256 prediction windows that balance all legal anchor counts $k \in \{1, \ldots, d+1\}$. The loss follows §3.2, and target features and supervision are produced online by the corresponding frozen target model. We construct a shared training set from responses generated by Qwen3-8B on prompts from Open-PerfectBlend (Xu et al., 2024), and use it to finetune the M-DFlash drafters for Qwen3-8B and Qwen3-14B. We initialize from the corresponding official DSpark checkpoints and keep the AR heads frozen.

F LLM USAGE

We employed LLM-based tools solely during manuscript preparation to refine language and improve readability. The research questions, technical approach, experimental methodology, interpretation of results, and conclusions were developed independently by the authors. The authors reviewed the complete manuscript and remain fully responsible for its content and accuracy.