SCD: ENABLING SPECULATIVE OVERLAP FOR TARGET-CONDITIONED DRAFT MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

A stronger drafter can improve acceptance in speculative decoding, but its extra computation may offset the gain. Target–draft disaggregation places the models on separate devices, yet each request still waits for verification before the next draft. Speculative speculative decoding (SSD) predicts the next starting point to overlap these steps, but target-conditioned drafters must also wait for target hidden states. We propose Staged Conditional Drafting (SCD) to shorten this wait. The first draft layers use earlier target features to compute predicted next-round branches during verification. The remaining layers reuse these intermediate activations and continue with deeper target features. For batched requests, a small drafter handles prediction misses in parallel. We implement SCD with Two-stage DSpark and KDA + Target-KV. Both preserve accepted length on a 4B target; the KDA version also does so on 14B. With the same models, batch, and two-GPU layout, using target features earlier reduces total decode time by 15.47%. Under greedy decoding in native SGLang, two-GPU overlap improves batch-size-one decode rate by 6.86–9.32% over single-GPU serial execution across two draft architectures and three tasks. It also improves throughput in fully occupied batched rounds by 8.50–17.99% over single-GPU DSpark. KDA + Target-KV accommodates 13% more concurrent sessions within the same GPU memory budget, at a 2.17% lower single-GPU decode rate.

1 INTRODUCTION AND RELATED WORK

Speculative decoding accelerates generation by verifying several draft tokens in one target-model pass (Leviathan et al., 2023). Modern parallel drafters use target hidden states to generate these candidates (Chen et al., 2026; Cheng et al., 2026; Inco AI, 2026). Although these drafters have few layers, each layer can still be expensive. DeepSeek-V4-Pro's DSpark, for example, uses the same hidden width, attention dimensions, and MoE expert counts as its target (DeepSeek-AI, 2026). A stronger drafter can reduce the number of verification rounds, but extra computation in each round may offset the saving. In DFlash's depth study, the target accepts more tokens from deeper drafters, but the overall speedup falls (Chen et al., 2026). Draft design therefore needs to consider accepted length together with execution time and memory use.

Target–draft disaggregation places the two models on separate devices so they can run concurrently (vLLM contributors, 2026; SGLang contributors, 2026). MineDraft uses this layout to run drafting for one batch alongside verification of another (Tang et al., 2026). This improves device utilization, but the next draft for each request still waits for that request's current verification. Cross-round speculation instead starts the next draft during the same request's verification (Kumar et al., 2026; Zhang et al., 2025). Mirror-SD similarly starts draft branches from early-exit token candidates while the target finishes verification (Bhendawade et al., 2025). For modern target-conditioned drafters, this requires more than predicting the starting token: their first layer already needs hidden states produced late in the target pass. SSD's EAGLE-3 extension uses draft activations in place of target activations that are not yet available, potentially reducing acceptance (Kumar et al., 2026, Appendix E).

We propose **Staged Conditional Drafting (SCD)** to let the draft start with target hidden states that are already available. The early draft layers use earlier target features and run while the target finishes verification. The later draft layers continue from the saved activations and use deeper target

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.

features. We choose how many layers to run early according to the remaining verification time. A joint predictor supplies possible starting points for the next round, so drafting and verification overlap within the same request and batch. The draft still uses late target information, but less of its computation remains after verification.

We implement SCD with Two-stage DSpark and KDA + Target-KV. Both retain accepted length on the 4B target, and we also evaluate the KDA version on 14B. On the same batch with the same models and two-GPU layout, using target features earlier reduces total decode time by 15.47% (Section 4.4). In native SGLang, two-GPU overlap improves BS1 decode rate by 6.86–9.32% and fully occupied batch throughput by 8.50–17.99% over single-GPU baselines. The KDA version combines recurrent state with target-KV reuse (Kimi Team et al., 2025; Yang et al., 2025b; Liu et al., 2026). It uses less memory for draft history at the cost of additional computation, providing an alternative for memory-limited devices.

2 Background

In each decoding round, the drafter generates L candidate tokens $y_{1:L}$ from a committed prefix P. The target computes their conditional distributions and accepts a consecutive prefix using standard speculative verification (Leviathan et al., 2023). Let $r \in \{0, \dots, L\}$ be the number of accepted draft tokens. If $r < L$, verification produces a correction token; otherwise, it produces a bonus token. We denote either token by a, the next anchor. Unless generation has terminated or exhausted its output budget, (r, a) determines where the next draft starts.

A parallel drafter computes several candidate positions in one backbone pass. A lightweight Markov head models dependencies among positions without running the full backbone once per token (Cheng et al., 2026). The drafter can read history in two forms: *draft context KV*, which it constructs from fused target hidden states and stores itself, or *native target KV*, which the target already maintains. KDA replaces draft context KV with recurrent state and retains access to native target KV.

SSD predicts candidate pairs (r, a) before verification ends and reuses the branch that matches the result. On separate target and draft devices, a branch can start once its predicted prefix, target features, and history state are ready.

For BS1, mean decode time per output token is $t_{\text{token}} = 1000 \sum_i T_{\text{decode},i} / \sum_i N_i$ milliseconds, where i indexes requests and $T_{\text{decode},i}$ is measured in seconds. Batch throughput divides all useful output tokens by elapsed batch decode time, counting overlapping execution only once. Its inverse is not an individual request's TPOT. Mean round time divides decode time by verification rounds. Reducing draft FLOPs lowers latency only when it shortens the time a request waits.

3 Staged conditional drafting

3.1 Two-stage drafting with target hidden states

We divide the draft backbone into two stages that use target features available at different times. Let C_{early} and C_{late} denote their respective inputs:

$$z = F_{\text{early}}(x; C_{\text{early}}), \qquad h = F_{\text{late}}(z; C_{\text{late}}). \tag{1}$$

The late stage starts from z, reusing the early-stage result instead of running the backbone again. This allows the early stage to start before the deeper target features are ready. Section 3.2 explains how we predict its starting point.

Fit the early stage into the remaining verification time. In our five-layer 4B drafter, D0–D2 use fused target features $F[1, 9, 17, 25, 30]$. D3–D4 use the same sources, with H33 replacing H30. Once the early features and branch state are ready, D0–D2 compute possible next-round drafts while the target continues verification. If a predicted boundary and anchor match the result, D3–D4 use the saved activations and late features to finish that draft. We select the early-stage depth to fit the remaining verification time, including prediction, state preparation, and communication. The 14B KDA version uses H35/H37. These splits balance candidate quality and execution time; other splits may also preserve quality.

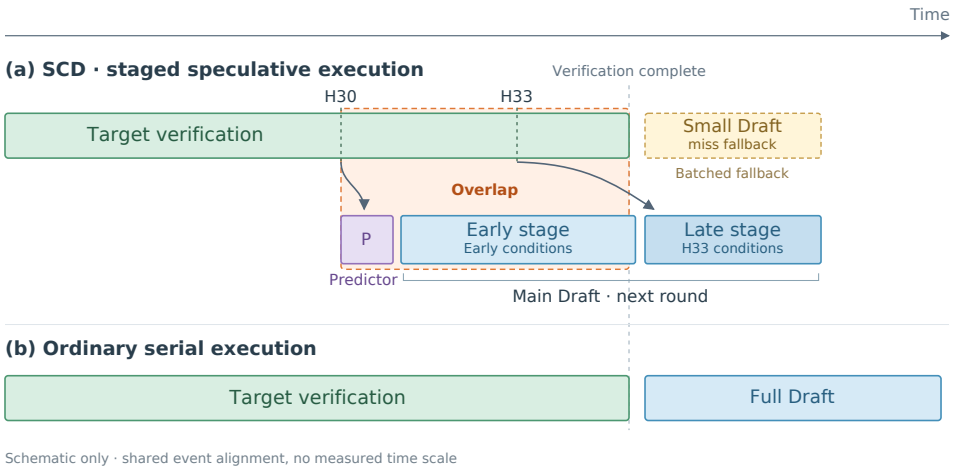

Figure 1: Staged and ordinary serial execution, aligned at verification completion. Earlier target features let prediction and Main's early stage run during verification. Main then continues with deeper features while Small handles misses. Schematic durations are not to scale.

Both draft architectures can use this split. Two-stage DSpark builds context KV from each stage's fused hidden states. In KDA + Target-KV, these hidden states update the stage's recurrent state, and the corresponding target layers supply native KV. The two architectures store history differently, but each stage starts only after the features it needs become available.

3.2 Predicting the next starting point

The next draft needs both the accepted boundary and the anchor token: the boundary gives the prefix position, and the anchor gives the starting token. Our lightweight joint predictor uses available target representations and current draft features Z to select the global Top-B pairs:

$$p_\theta(r, a \mid Z) = p_\theta(r \mid Z)p_\theta(a \mid r, Z), \qquad \mathcal{C}_B = \mathrm{TopB}_{(r,a)}\, p_\theta(r, a \mid Z). \tag{2}$$

Each predicted pair defines a possible prefix $P' = P \oplus y_{1:r} \oplus a$. The predictor has approximately 3.72M trainable parameters. We prepare draft features in advance and project target features as their layers finish. Once H30 is available, the predictor can finish without waiting for H33 or the target's final output. Appendix A.2 describes the architecture and candidate construction.

The confidence head chooses the *current* proposal length L. The joint predictor chooses which *next-round* drafts to compute early. A prediction hits only when both the boundary and anchor are correct. Our decoder uses a fixed number of branches; Appendix B.3 studies probability-based allocation across requests.

3.3 Completing hits and handling misses

After verification, a hit selects the matching early-stage result and runs Main's late stage. A miss discards the predicted branches and generates a new draft from the true prefix. At batch size one, Main reruns the full draft. For batches, a separate Small drafter handles misses while Main completes the hit requests. Following Saguaro's low-latency fallback design (Kumar et al., 2026, Section 4.3), we aim to run Small within the time Main needs for its late stage.

Our two-GPU implementation places Target and Small on one GPU and Main on the other. Small is a two-layer DSpark drafter that uses H33 and starts from the true anchor with its history aligned to the verified prefix. A shared predictor takes the current candidates and their source, Main or Small, to predict the next starting point. We maintain three state rules: public state contains only the committed prefix; each branch's state, visible KV, and positions describe the same prefix; and only a branch that matches verification may continue. KV computed during verification is committed only after acceptance. The target verification rule remains unchanged.

Algorithm 1 Batched SCD with Main continuation and Small fallback

1: Target verifies current proposals and produces features layer by layer.
2: When early features are ready, predict up to B joint pairs per active request.
3: Set up separate history, visible KV, and positions for each branch; run Main's early stage.
4: Obtain true (r^*, a^*); update the committed prefix only from verified tokens.
5: Partition active requests into hits and misses.
    Hits: select a matching branch; finish its early stage and late stage.
    Misses: discard branches; align Small to the real prefix and run Small.
6: Schedule the two paths in parallel when resources permit.
7: Apply confidence truncation; merge proposals for the next target verification.
8: Retire terminated requests and obsolete branches; repeat.

Small reduces fallback time but may produce fewer accepted tokens in the next round. The confidence head truncates low-confidence proposals. We measure both accepted length and complete round time to evaluate the resulting trade-off.

3.4 Choosing hidden layers and the stage split

The target hidden layer determines both the information available to Main and when Main can start. Earlier features leave more time for overlap, but may lower candidate quality. Later features have passed through more target layers, but leave less time before verification finishes. We compare their training and generation quality in Section 4.2, then choose how many draft layers to run within the available time.

Let W be the time from early-feature availability to the verification result, T_{pred} the remaining prediction time, and T_{early} the early-stage time. The part that extends beyond verification is

$$W = t_{\text{truth}} - t_{\text{condition}}, \qquad T_{\text{spill}} = [T_{\text{pred}} + T_{\text{early}} - W]^+. \tag{3}$$

When Main's late stage and Small fallback run in parallel, the remaining time after verification is approximately

$$T_{\text{post}} \approx \max(T_{\text{spill}} + T_{\text{late}}, T_{\text{small}}) + T_{\text{handoff}}. \tag{4}$$

The feature layer sets the available time W; the stage split determines how much draft computation we try to fit into it. Our H30/H33 configuration runs D0–D2 with H30 and D3–D4 with H33. Using earlier features or adding early draft layers is useful only if the accepted length compensates for any extra wait after verification. Branch count and Small's size also affect both candidate quality and round time.

Allocating a fixed branch budget. On Arena BS16, fixed Top4 raises mean round time by 2.20% and lowers decode rate by 2.74% relative to Top2 (Appendix C.5). We also use joint probabilities to allocate branches unequally across requests. Each request keeps Top1; remaining branches are assigned in order of marginal joint probability. Replaying fixed Arena trajectories, this raises Joint Recall from 60.13% to 61.76% with the same 32 branches across 16 slots. Appendix B.3 gives the rule, coverage curve, and component measurements.

3.5 Compact history with KDA + Target-KV

The KDA version uses attention within the candidate block, a fixed-size recurrent state for the prefix, and direct reads from target KV. The state and KV both derive from the target, but retain history in different forms. Each layer first retrieves target KV and adds the gated result to the current representation. It then combines block-local attention with the KDA readout, applies an output projection, and runs the MLP. Appendix B.2 gives the equations.

In the 4B configuration, each of the five draft layers stores eight 128×128 FP32 matrices. These states total 2.5 MiB regardless of prefix length. BF16 draft history KV instead requires 20 KiB per token, or 320 MiB at 16K. These numbers cover draft history; weights, activations, and branch copies require additional memory. A colocated draft can share target KV storage, while a separate draft device needs copies and transfers. Updating KDA state requires computation, and reading target KV costs more as the visible history grows.

3.6 TRAINING

We freeze the target and train the draft, context projections, and adapters in TorchSpec with DSpark's candidate and auxiliary-head losses. The target generates features online and streams them to the draft trainer. We train the joint predictor separately on the boundary and correction/bonus token produced by verification, using each proposal's actual confidence-truncated length. Appendix A.1 lists the settings.

4 EXPERIMENTS

We evaluate draft quality, decode speed, and memory capacity.

4.1 EXPERIMENTAL SETUP

Models and tasks. We use Qwen3-4B and Qwen3-14B targets (Yang et al., 2025a) and compare original DSpark, Two-stage DSpark, and KDA + Target-KV. Drafters have five layers unless stated otherwise, maximum proposal length seven, and Markov and confidence heads. Tasks cover mathematical reasoning (MATH500), code generation (MBPP), and open-ended questions (Arena-Hard-v2) (Hugging Face H4, 2024; Austin et al., 2021; LMArena, 2025). We describe additional comparison models in the relevant experiments.

Training. We train drafts and adapters for ten epochs on OpenPerfectBlend clean-v2 responses regenerated by the frozen target. Training uses non-thinking mode and a 4096-token sequence length. Predictors are trained separately. Appendix A.1 lists checkpoints and hyperparameters.

Evaluation. We assess candidate quality with standard speculative sampling at temperature one and confidence truncation disabled. The metric is accepted length τ per verification round, including correction/bonus. System experiments use native SGLang (Zheng et al., 2024), A100-80GB GPUs, greedy decoding, and confidence 0.5, yielding lengths 1–7. Decode timing includes prediction, communication, state handling, and fallback. Batched experiments use fixed batches without replacing completed requests. As requests finish, fewer slots remain active. The main performance table therefore uses rounds in which every slot is active, including all slow rounds; it does not apply an occupancy correction. Appendix A.7 gives complete-run results and timing boundaries.

4.2 HIDDEN DEPTH AND CANDIDATE QUALITY

Table 1 compares accepted length during generation. Figure 2 then shows how the choice of target hidden states affects Main during training.

Accepted length across architectures and scales. On the 4B target, KDA with H30/H33 matches or exceeds the accepted length of full H33 on all three tasks. Two-stage DSpark stays close to original DSpark. On 14B, KDA with H35/H37 improves accepted length over DSpark by 1.68%, 1.97%, and 1.16%. An eight-layer Three-stage DSpark with H28/H30/H33 also exceeds the original DSpark on all three tasks. Its $3+3+2$ split adds layers before the final stage, keeping the last stage at two layers (Appendix B.5).

How early can Main use target features? Figure 2 compares Main trained with different target hidden states. Full H33 supplies all five layers with $F[1, 9, 17, 25, 33]$. H30/H33 changes the last source to H30 for D0–D2 and keeps H33 for D3–D4. H17/H33 uses the earlier source set $F[1, 9, 17]$ for the first stage. Progressive cumulative conditioning adds deeper sources as the draft layer increases (Appendix B).

Over the common 10k–15k training interval, H17/H33 and progressive cumulative conditioning have higher CE and shorter simulated accepted lengths than full H33 and H30/H33. These earlier-feature configurations fit the training data less well. H30/H33 instead follows full H33 closely, including the final 22k–24.7k interval. The figure also includes the local DSpark training curve as an attention-based reference.

Table 1: Accepted length across draft architectures and target scales. Accepted length τ per verification round includes correction/bonus; higher is better. Arrows show target features used by successive stages: $3+2$ layers for two stages and $3+3+2$ for three.

Target	Drafter	Layers	Conditions	MATH500	MBPP	Arena
4B	DSpark	5	Original	5.74	5.13	3.29
	Two-stage DSpark	5	H30→H33	5.73	5.16	3.23
	KDA + Target-KV	5	H33→H33	5.78	5.16	3.27
	KDA + Target-KV	5	H30→H33	5.85	5.24	3.32
	Three-stage DSpark	8	H28→H30→H33	**5.86**	**5.30**	**3.41**
14B	DSpark	5	Original	5.76	5.27	3.15
	KDA + Target-KV	5	H35→H37	**5.85**	**5.37**	**3.18**

Each model uses its trained checkpoint. Appendix A.1 lists fused feature sources; Appendix B.5 gives the eight-layer configuration. Bold marks the largest value within each target; draft depths differ.

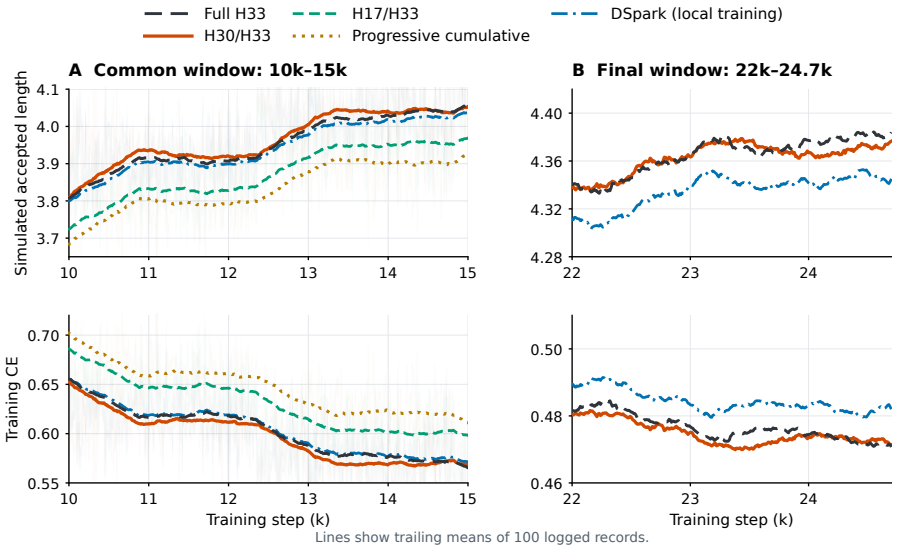

Figure 2: Draft quality with target features from different layers. A: common steps 10k–15k for all five configurations. B: steps 22k–24.7k for the three completed runs. Rows show simulated accepted length and cross-entropy (CE) during training. Faint curves are logged values; lines are trailing means of 100 records.

We therefore use H30 for the first three draft layers and retain H33 for the last two. Compared with using H33 throughout, this split preserves accepted length in the tested KDA model (Table 1) and lets the first three layers run during target verification. Section 4.4 shows the resulting overlap and reduction in decode time.

4.3 Decode Performance

The benefit of starting earlier. Using target features before verification ends reduces total decode time by 15.47% on one Arena batch of 32 requests. Both configurations use the same models, two-GPU layout, and branch computation. We combine four paired runs (Section 4.4, Appendix A.8).

With Target and Main on separate GPUs, KDA + Target-KV overlap increases decode rate by 16.83% on MATH500 and 9.39% on MBPP over serial execution without prediction or extra branches. Neither uses Small. Each configuration runs once, excluding the first decode and final drain (Appendix A.3). Tables 2 and 3 compare single-GPU serial with two-GPU overlap, so their gains include the additional GPU.

Table 2: BS1 deployment comparison: single-GPU serial versus two-GPU overlap. Serial colocates Target and Main; overlap places them on separate GPUs. Each row uses the same drafter.

Drafter	Task	Rate (token/s)		Time (ms/round)		Gain
		Serial	Overlap	Serial	Overlap	
KDA + Target-KV	MATH500	536.59	**586.58**	12.050	11.019	+9.32%
	MBPP	467.14	**506.87**	11.959	11.022	+8.51%
	Arena	310.71	**332.02**	12.017	11.234	+6.86%
Two-stage DSpark	MATH500	547.41	**594.10**	11.711	10.779	+8.53%
	MBPP	471.82	**507.14**	11.544	10.740	+7.49%
	Arena	324.93	**348.56**	11.592	10.767	+7.27%

Complete EOS/4096 decoding, with no latency filtering. Runs are collected separately; Appendix A.3 defines timing and deployment scope.

Table 3: Performance in fully occupied rounds: single-GPU DSpark versus two-GPU overlap (labeled Overlap). Target and Small share one GPU; Main runs on the other. Rates use actual output tokens divided by complete cycle times.

Task	BS	Method	Small (%)	τ	Accept (%)	ms/round↓	token/s↑	Gain
MATH500	16	DSpark	—	6.261	77.82	16.84	5946	—
	16	Overlap	11.85	6.189	77.10	14.20	**6970**	**+17.22%**
	32	DSpark	—	6.271	77.77	24.73	8114	—
	32	Overlap	12.19	6.168	76.69	20.80	**9486**	**+16.91%**
MBPP	16	DSpark	—	5.483	67.47	15.62	5615	—
	16	Overlap	18.92	5.401	67.27	13.17	**6557**	**+16.78%**
	32	DSpark	—	5.466	67.20	22.36	7822	—
	32	Overlap	18.64	5.417	67.26	18.77	**9230**	**+17.99%**
Arena	16	DSpark	—	3.691	59.54	16.78	3519	—
	16	Overlap	35.43	3.547	60.03	14.41	**3939**	**+11.94%**
	32	DSpark	—	3.720	60.02	28.47	4181	—
	32	Overlap	35.15	3.537	59.57	24.95	**4536**	**+8.50%**

τ includes bonus; draft-token acceptance excludes bonus. Gains use unrounded aggregates. Both arms use the same inputs but their own batch partitions. Appendix A.7 gives complete-run results and retained-round counts.

Single-request execution across draft architectures. Table 2 compares single-GPU serial and two-GPU overlap with the same draft weights and generation settings. Two-GPU overlap predicts joint Top2 branches and reruns the full Main on a miss; BS1 does not use Small. Across math, code, and chat, decode rate increases by 6.86–9.32% for KDA + Target-KV and 7.27–8.53% for Two-stage DSpark. Timing includes initial drafting, prediction, communication, state handling, fallback, and final drain. MATH500 answer accuracy is 82.6/82.4% for KDA serial/overlap and 83.2/83.2% for Two-stage DSpark (Appendix A.6).

Two-GPU overlap under batching. For batched execution, Main finishes the hit branches while Small generates new drafts from the true prefixes for misses. Table 3 compares two-GPU overlap with single-GPU DSpark on the same prompt sets, with each method using its own batch partitions.

Small handles 11.85–35.43% of active request-rounds in the fully occupied subset. Although accepted length falls by 0.91–4.93%, mean round time falls by 12.38–16.02%, giving an 8.50–17.99% throughput increase. Over complete fixed-batch runs, BS16/32 gains are 16.54/12.10% on MATH500, 1.63/13.15% on MBPP, and −1.16/28.17% on Arena. Complete runs include low-occupancy tails; Appendix A.7 gives their rates and occupancy.

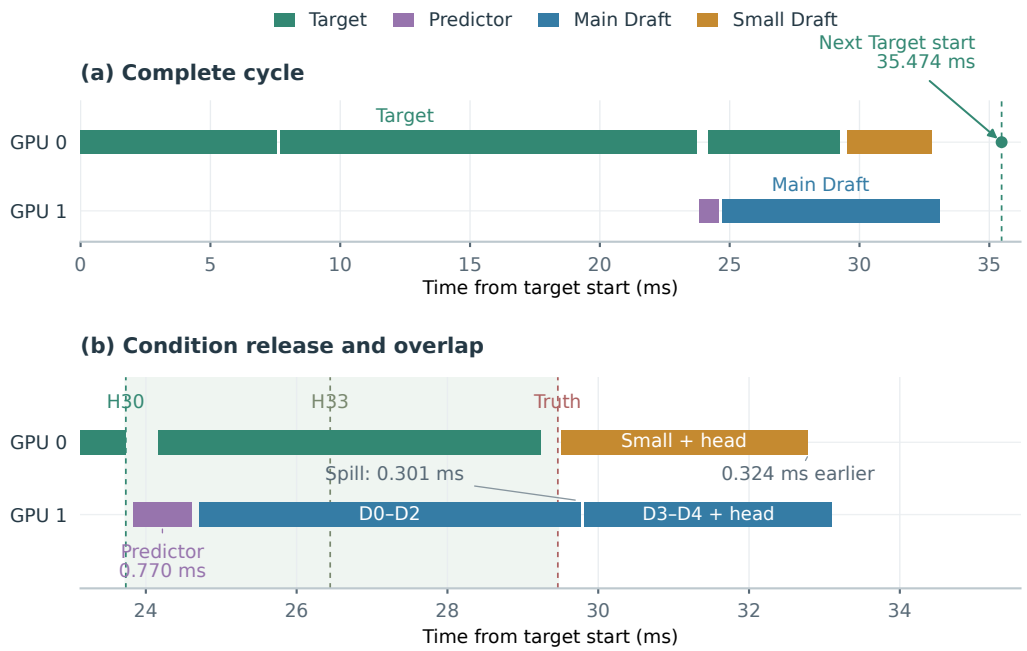

Figure 3: A separately profiled two-GPU cycle, with a zoom showing when target features become available. The arrow marks the start of the next target verification. Prediction and D0–D2 run during verification; Main's late stage and Small fallback overlap afterward. Intervals share one Nsight clock and include scheduling and instrumentation overhead.

4.4 Execution mechanism: overlap, prediction, and fallback

The effect of starting earlier. We vary when the draft starts using target hidden states, using one fixed batch of 32 Arena requests. Both configurations use the same models, two-GPU layout, and Main + Small computation. One starts using H25/H30/H33 as they become available; the other waits for the verification result. Four independently initialized pairs alternate execution order. Across the four pairs, early feature use reduces total decode time from 39.982 s to 33.798 s (15.47%), with the same output tokens, acceptance, and routing within each pair. In the matched fully occupied rounds, using hidden states earlier reduces mean round time by 17.79% and raises throughput by 21.64% (Table 4). Appendix A.8 gives complete-run results and variation across repeats.

Table 4: Ablating early use of target hidden states. The same Arena batch, models, and GPU layout are used in four independent pairs. Outputs, acceptance, and routing match within each pair. Each configuration includes the same 300 fully occupied rounds, including all slow rounds.

Condition use	Mean round (ms)↓	Rate (token/s)↑
After verification	24.94	4,585.36
As hidden states become ready	**20.50**	**5,577.82**

Balancing candidate quality and overlap. The $3 + 2$ split keeps candidate quality while running most of the first three layers during verification. Section 4.2 shows similar training quality and accepted length for H30/H33 and full H33. Figure 3 shows how the computation fits: after H30 is ready, the target has 5.73 ms of verification left. The predictor and Main's first three layers run during this interval and finish only 0.30 ms after the verification result. The last two layers then continue with H33. This keeps late target features in the draft while reducing the computation that follows verification.

Table 5: History storage, resident sessions, and single-GPU decode rate. Capacity uses 16K paused sessions; decode rate comes from a separate full-MATH500 workload.

Drafter	Owned state (MiB)↓	Sessions↑	Rate (token/s)↑
DSpark	320	22	**560.83**
KDA + Target-KV	**2.5**	**25**	548.68
Relative change	−99.22%	+13.64%	−2.17%

Main's last two layers take 3.279 ms and Small takes 3.273 ms. Small starts earlier and finishes 0.324 ms before Main, so it does not delay candidate completion in this cycle.

Prediction accuracy and accepted length. At batch size 32, Joint@2 for Main-generated proposals is 90.06% on MATH500, 82.41% on MBPP, and 69.16% on Arena over complete trajectories (Appendix C.1). Small handles 12.19%, 18.64%, and 35.15% of the fully occupied request-rounds, respectively. Arena has more prediction misses, uses Small more often, and loses more accepted length (4.93%). Shorter rounds (12.38%) improve throughput, while lower accepted length offsets part of that gain; throughput increases by 8.50%.

Appendix C.2 compares Main and Small on the same prefixes after prediction hits and misses.

4.5 History storage and decode speed

At 16K, KDA reduces draft-owned history from 320 to 2.5 MiB (Table 5). After accounting for weights, target KV, and runtime buffers, the same device holds 25 sessions instead of 22 (+13.64%). In a separate full-MATH500 single-GPU test, KDA decodes 2.17% slower than optimized DSpark. It therefore stores more sessions at a small cost in decode speed.

5 Discussion and conclusion

SCD starts the next draft during the current target verification. The first layers use earlier target hidden states; a matching branch then continues with deeper features. Small handles batched misses while Main finishes the hits. This preserves accepted length in the tested models and shortens decode time. Across four paired runs on the same batch, starting earlier reduces total decode time by 15.47% while keeping the models, two-GPU layout, and branch computation unchanged.

When memory is limited. Native Target-KV offers a simple way to save memory for personal agents. On the same GPU, the draft reads the target's existing KV, so it needs neither a separate history cache nor prefill to build that cache. This can help agents retain conversations and tool results on memory-limited devices. KDA + Target-KV additionally maintains a small recurrent state. Our KDA configuration has longer accepted sequences than Native Target-KV (Table 19), although their training and visible history also differ.

Experimental scope. We measure speed on 4B/A100 and candidate quality on both 4B and 14B. When similarly configured draft and target layers scale together, the verification tail can accommodate a similar number of draft layers (Appendix B.4). Larger-model speedups remain unmeasured. Batched results use fixed groups without replacing completed requests.

Further opportunities. The eight-layer draft improves accepted length while keeping the last stage at two layers. The extra layers go into the first two stages, where they can run during target verification. If these stages finish before verification ends and the prediction hits, only the last two layers and handoff remain (Appendix B.5). D/T disaggregation also allows several replicas of the same target to share a draft service, as proposed in M:N deployments (vLLM contributors, 2026; SGLang contributors, 2026). This could spread draft GPU costs across target replicas instead of reserving a draft GPU for each. For SCD, shared drafting and feature transfer would still need to finish within each request's verification window; we have not measured this pooled deployment.

AI USE STATEMENT

Generative AI tools assisted research and experimental-design discussions, implementation and code review, data analysis, literature discovery, manuscript drafting, Chinese-to-English translation, and LaTeX preparation. Target models generated training responses as described in Appendix A.1. The authors take responsibility for the final manuscript, experimental claims, code, and artifacts.

ETHICS STATEMENT

This work studies inference efficiency using existing language models and evaluation datasets. Faster decoding neither improves nor replaces the underlying models' safety mechanisms. Deployment must respect the licenses and usage requirements of the models and data. The work does not introduce human-subject experiments.

REPRODUCIBILITY STATEMENT

The appendices describe model and training settings (Appendix A.1), predictor metrics (Appendix A.2), timing and aggregation (Appendices A.3, A.7, and A.8), and implementation checks (Appendix A.5). Experiment records retain checkpoint and source hashes, per-request results, and execution traces.

REFERENCES

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

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. arXiv:2602.06036, 2026. URL https://arxiv.org/abs/2602.06036.

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. Technical report, 2026.

DeepSeek-AI. DeepSeek-V4-Pro-DSpark: Model configuration, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro-DSpark/blob/main/config.json. Accessed September 25, 2026.

Hugging Face. Math-Verify, 2025. URL https://github.com/huggingface/Math-Verify. Version 0.9.0.

Hugging Face H4. MATH-500: Dataset card, 2024. URL https://huggingface.co/datasets/HuggingFaceH4/MATH-500.

Inco AI. DFlash 2: Keep drafting parallel. Technical blog, 2026. URL https://inco.ai/blog/dflash2/.

Kimi Team et al. Kimi linear: An expressive, efficient attention architecture. arXiv:2510.26692, 2025. URL https://arxiv.org/abs/2510.26692.

Tanishq Kumar, Tri Dao, and Avner May. Speculative speculative decoding. arXiv:2603.03251, 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*, 2023. URL https://arxiv.org/abs/2211.17192.

Tianyu Liu, Yuhao Shen, Xinyi Hu, Baolin Zhang, Hengxin Zhang, Jun Dai, Jun Zhang, Shuang Ge, Lei Chen, Yue Li, and MingCheng Wan. When hidden states drift: Can KV caches rescue long-range speculative decoding? arXiv:2604.26412, 2026. URL https://arxiv.org/abs/2604.26412.

LMArena. Arena-Hard-Auto and Arena-Hard-v2, 2025. URL https://github.com/lmarena/arena-hard-auto.

SGLang contributors. Parallel speculative decoding roadmap. Issue 27462, 2026. URL https://github.com/sgl-project/sglang/issues/27462.

Zhenwei Tang, Arun Verma, Zijian Zhou, Zhaoxuan Wu, Alok Prakash, Daniela Rus, and Bryan Kian Hsiang Low. MineDraft: A framework for batch parallel speculative decoding. arXiv:2603.18016, 2026. URL https://arxiv.org/abs/2603.18016.

vLLM contributors. Disaggregated speculative decoding with standalone parallel draft model. RFC 42109, 2026. URL https://github.com/vllm-project/vllm/issues/42109.

An Yang et al. Qwen3 technical report. arXiv:2505.09388, 2025a. URL https://arxiv.org/abs/2505.09388.

Penghui Yang, Cunxiao Du, Fengzhuo Zhang, Haonan Wang, Tianyu Pang, Chao Du, and Bo An. LongSpec: Long-context lossless speculative decoding with efficient drafting and verification. arXiv:2502.17421, 2025b. URL https://arxiv.org/abs/2502.17421.

Ziyi Zhang, Ziheng Jiang, Chengquan Jiang, Menghan Yu, Size Zheng, Haibin Lin, Henry Hoffmann, and Xin Liu. SwiftSpec: Ultra-low latency LLM decoding by scaling asynchronous speculative decoding. arXiv:2506.11309, 2025. URL https://arxiv.org/abs/2506.11309.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2312.07104.

A EXPERIMENTAL SETTINGS AND MEASUREMENT CONVENTIONS

A.1 MODELS AND TRAINING

The 4B candidate-quality checkpoint is step 24700 (iteration 24701); 4B KDA system and capacity results use step 23400 (iteration 23401). The 14B instance uses step 24600 (iteration 24601), and its DSpark-architecture control uses step 24920 (iteration 24921). Targets are frozen. The five-layer Main drafters use block size seven, train for ten epochs at maximum length 4096, and use a cosine learning-rate schedule. Peak learning rate is 6×10^{-4}, warmup fraction 0.04, weight decay zero, and training seed 42. The 14B configuration uses Adam $\beta_2 = 0.95$. The inherited CE/L1 coefficients are 0.1/0.9 and loss-decay gamma is four; the complete objective includes the configured auxiliary heads. Data are target-regenerated OpenPerfectBlend clean-v2 responses.

Table 6: Fused target-hidden sources and native KV mapping for five-layer drafters. Interface conventions distinguish hidden and KV indices.

Target	D0–D2 hidden sources	D3–D4 hidden sources	Target KV sources
4B	1, 9, 17, 25, 30	1, 9, 17, 25, 33	2, 10, 18, 26, 34
14B	1, 10, 19, 28, 35	1, 10, 19, 28, 37	2, 11, 20, 29, 38

Each KV adapter adds retrieved information before its draft layer and reads only tokens before the anchor. Candidate-quality tests and system timings use different checkpoints, so acceptance differences between these experiments do not explain their speed differences.

Main, Small, and the shared predictor. The two-GPU implementation in Figure 3 uses a five-layer Two-stage DSpark Main and a separately trained two-layer DSpark Small. Small has hidden width 2560, FFN width 9728, 32 query heads, eight KV heads, head dimension 128, block size seven, and Markov and confidence heads. Both Small layers use $F[1, 9, 17, 25, 33]$. Its step-24700 checkpoint completes ten training epochs. Target and Small share one GPU; Main runs on the other. A shared predictor uses H30 target features, proposal-specific statistics, and a learned source embedding for Main or Small. Predictor training balances the two candidate sources while keeping the drafters frozen; source-specific Markov weights construct the corresponding candidate distributions.

Predictor training and checkpoint selection. We use two deployed heads. P_K is the H30 head for the BS1 KDA instance, after one final Top1024 fine-tuning epoch (18,177 updates, global batch 64); its initialization is the fixed-length Top1024 checkpoint `continue40kbest-r2`, identified by SHA-256 prefix `c96946278ffae49b7`. The accompanying configuration files give its full path and hash. P_S initializes from P_K and trains for four epochs on balanced KDA-Main and two-layer DSpark-Small records, using Top512 candidate lists (145,420 updates, global batch 64). A zero-initialized source embedding distinguishes the two inputs. The selected shared checkpoint is epoch four, which maximizes mean held-out Joint@2 across sources; it is deployed with Two-stage DSpark and with batched Main + Small. Thus the shared head trains on KDA-Main candidates and later predicts Two-stage DSpark candidates at deployment.

Both heads train on target-generated OpenPerfectBlend records. We split source records by hashing their group IDs with seed 20260908 and holding out one hash bucket out of 20. This gives 1,163,380 training groups and 61,410 held-out groups. Anchors from the same source record remain in the same split across epochs, and the shared trainer checks that Main and Small use the same group IDs. Evaluation uses the anchor schedule from data epoch one. All cycles from a source record therefore stay in the same split. The loss sums boundary negative log-likelihood, token cross-entropy for labels in the candidate set, and a joint Top2 hinge with coefficient 0.25 and margin 0.2. Labels outside the candidate set do not contribute to token or hinge loss. We use learning rate 10^{-5}, 4% warmup, and cosine decay. Downstream benchmarks are evaluated separately.

A.2 CANDIDATE AND PREDICTOR METRICS

Candidate-quality evaluation uses temperature one, confidence zero, seed 980406, and non-thinking generation to EOS or a 2048-token budget. Accepted length divides the total recorded committed

tokens by verification rounds, including actual correction/bonus tokens. If an accepted token reaches EOS, no extra bonus is counted. The evaluator clips the returned output at the token budget but keeps the full final-round verification record. Accepted length can therefore differ from returned tokens divided by rounds. System rate instead uses the actual scored output defined below. Request-macro metrics give each request equal weight after computing its metric separately. Table 1 reports local measurements; the 14B DSpark control was trained with the official architecture.

Joint head. The H30 predictor reads hidden states from target layers 1, 9, 17, 25, and 30. Independent low-rank adapters project each source, followed by channel-wise sigmoid-gated fusion. Draft hidden states and candidate embeddings are projected separately. Six statistics—candidate log probability, normalized entropy, top-1/top-2 log-probability gap, confidence, normalized position, and whether the candidate is argmax—are encoded by a two-layer SiLU MLP. Position encodings and a bonus marker are added before a two-layer, four-head, pre-norm Transformer with twice-width feed-forward layers, GELU, zero dropout, and final RMSNorm.

The boundary head combines position scores with a 128-dimensional SiLU MLP that adjusts the first-rejection hazard. Positions after actual length L are masked, and survival through L defines bonus. Token scoring combines a residual H30 representation with the position representation and applies the frozen target LM head over the candidate union. Rejection positions additionally rerank Draft Top8 using a one-layer, 128-dimensional, four-head candidate Transformer and a learned gate; bonus uses the same union. The deployed head does not use the optional token refiner. Labels appear only in supervision.

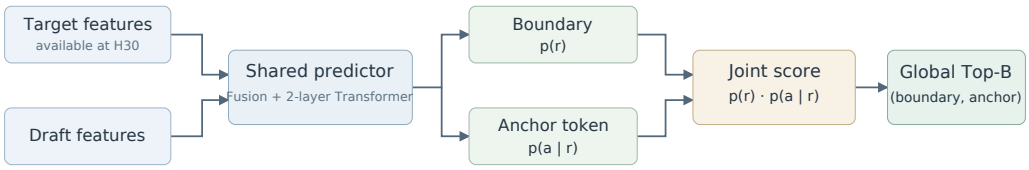

Figure 4: Joint prediction architecture. Boundary and anchor distributions define global joint scores over next-round starting points.

The union combines the first L positions' Draft Top1024 candidates for P_K (Top512 for P_S) with original per-position mixture Top8, sorted and deduplicated. The offline heads in Table 7 use Top1024. Bonus uses the same union; ground truth is not inserted. Confidence 0.5 retains the first low-confidence position and defines actual $L \in \{1, \ldots, 7\}$. Joint@B tests whether the true boundary–token pair is in global Top-B; rejection and bonus metrics use their respective record counts. Coverage measures whether the candidate union contains the label.

Table 7: Best held-out joint heads under variable-length confidence-0.5 evaluation. Each layer uses its best held-out checkpoint.

Metric	H27	H28	H30
Joint@1 (%)	67.49	69.20	73.94
Joint@2 (%)	78.17	79.93	84.40
Reject Joint@2 (%)	62.43	64.99	71.44
Bonus Joint@2 (%)	88.51	89.74	92.91
Candidate coverage (%)	98.90	98.90	98.90
Records	61,424	61,424	61,412
Training data epoch	10	9	4

H27/H28 have 24,344 rejection and 37,080 bonus records; H30 has 24,338 and 37,074. Evaluation data epoch is fixed to one. Average proposal length is 6.4053, target verification rows 7.4053, and trainable parameters 3,718,286. Online prediction statistics for the measured BS1 and batched deployments appear in Appendices A.3 and C.1.

A.3 DECODE TIMING AND STATISTICAL SCOPE

Table 2 uses complete BS1 decoding on 500 MATH500, 256 MBPP, and 750 Arena requests per arm. Each request is fully warmed before generating to EOS or 4096 output tokens. Confidence is 0.5; verification processes the anchor plus the actual 1–7 proposals. We divide total decode tokens by the sum of request decode times; we do not add rates across replicas. Timing includes initial drafting and the final GPU drain, and excludes prefill and warmup; the prefill-generated token is excluded. Average round time is the same complete decode time divided by the number of verification rounds. Timing includes all intermediate work and slow rounds. No new CUDA graphs are captured during measured decoding.

Table 8: Complete BS1 decode accounting for Table 2. S: single-GPU serial; O: two-GPU overlap.

Drafter	Task	Mode	Tokens	Decode (s)	Rounds	EOS / cap
KDA + Target-KV	MATH500	S	466,764	869.867	72,191	470 / 30
	MATH500	O	470,333	801.825	72,770	468 / 32
	MBPP	S	92,487	197.985	16,555	253 / 3
	MBPP	O	92,487	182.465	16,555	253 / 3
	Arena	S	884,926	2,848.098	237,015	702 / 48
	Arena	O	884,635	2,664.397	237,173	702 / 48
Two-stage DSpark	MATH500	S	477,467	872.235	74,478	472 / 28
	MATH500	O	477,542	803.809	74,569	469 / 31
	MBPP	S	89,033	188.703	16,347	254 / 2
	MBPP	O	89,033	175.560	16,346	254 / 2
	Arena	S	891,547	2,743.815	236,707	701 / 49
	Arena	O	892,619	2,560.872	237,837	701 / 49

Serial execution places Target and Main on one GPU; overlap places them on separate GPUs. Both run the corresponding drafter without Small fallback. Each configuration processes its full prompt set once, on separate A100 allocations. The KDA overlap result combines disjoint local and remote request shards. Pauses caused by recorded CPU throttling remain in the measured time. These rates compare the stated deployments, with gain defined as $R_{\text{overlap}}/R_{\text{serial}} - 1$.

Online prediction. Joint Top2 recall is 88.67%, 79.61%, and 63.01% for KDA + Target-KV on MATH500, MBPP, and Arena, respectively. Two-stage DSpark achieves 89.19%, 81.50%, and 64.11%. Each value uses that deployment's predictor and complete generated trajectory. Accepted length includes bonus tokens, whereas draft-token acceptance rate excludes them. Cropping at EOS or the output cap can make the returned token count differ from the verifier's accepted count.

Two-GPU serial versus overlap. A separate KDA + Target-KV control places Target and Main on separate GPUs in both arms. The serial arm waits for target verification to finish, then runs the full Main draft on the true prefix, without a predictor or extra branches. The overlap arm uses the joint predictor at H30 to start Main's first three layers; a hit continues with the last two layers, while a miss reruns the full Main draft. Neither arm uses Small. This comparison measures how much overlap speeds up decoding when Target and Main already run on separate GPUs. MATH500 serial and overlap rates are 537.29 and 627.72 token/s (+16.83%); MBPP rates are 492.37 and 538.59 (+9.39%). Each arm runs once. MBPP alternates arms per prompt on the same GPU pair; its request-cluster bootstrap interval is [2.05%, 13.90%]. This control excludes the first decode and its tokens, prefill, and final drain, and stops at EOS or the context-20000/reserve-16 boundary. Table 2 uses the complete-decode protocol above.

A.4 CAPACITY, MODEL SIZE, AND COMPUTE

Both capacity tests run Target and Draft on one A100-80GB with an 85% static-memory budget and one active request. Each session pauses after generating 32 tokens. We retain its state and reserve enough memory to resume it. We count sessions under this policy rather than filling memory to physical OOM. Target KV is shared without an extra copy, and the budget includes weights, target pools, activations, and runtime buffers.

Table 9: System-model parameter accounting. Vocabulary and frozen target projections are excluded from the non-vocabulary comparison.

Elements / parameters	KDA instance	DSpark
Backbone, context projections, norms	540,238,920	537,427,200
Markov head	77,791,232	77,791,232
Confidence head	2,817	2,817
Additional trainable adapter	85.24M	None
Compared non-vocabulary total	703.27M	615.22M

The adapter adds parameters equal to 13.85% of the DSpark comparison total. Including it, KDA has 14.31% more non-vocabulary parameters. KDA shares its 388,956,160-element embedding with the output head, while native DSpark uses Target's vocabulary modules at runtime. GPU pairs connect through NV12, and per-file SHA-256 hashes identify the runtime source versions.

For a visible prefix of 16,423 tokens, we estimate the cost of one seven-token candidate pass with a single branch as approximately 21.93 GFLOPs for DSpark and 23.90 GFLOPs for KDA including the full-vocabulary projection (+9.0%). A multiply-add counts as two operations. With $d = 2560$, total query dimension $q = 4096$, KV dimension $v = 1024$, MLP width $m = 9728$, block length $b = 7$, and vocabulary $V = 151936$, per-layer backbone linear cost is $2b(2dq + 2dv + 3dm)$, history attention $4bnq$, and local attention $4b^2q$. KDA adds adapter terms $2b(2dq + d^2 + 2d)$ and read/gate terms $2b \cdot 32 \cdot 128^2 + 2bd \cdot 32$. Both add $2bdV$. Five-layer totals before vocabulary projection are 16.49 and 18.46 GFLOPs; vocabulary projection adds 5.45 GFLOPs. These estimates exclude history commit, Markov/confidence, prediction, extra branches, and elementwise work, and are not complete speculative-round FLOPs.

A.5 Implementation checks

For BS1, we check input IDs, prompt hashes, output counts, termination, and the absence of cold graph captures. The optimized KDA overlap matches the previous overlap implementation on all 1,506 requests: output IDs, candidate and acceptance traces, and predictor decisions are identical. This comparison checks the optimizations to the overlap implementation.

On 20 paired requests, we check state handoff by comparing accepted counts, committed tokens, target rows, and the final five-layer state, hidden, logits, and confidence. Eight further request pairs compare hit/miss decisions and final state across 76 rounds per version. EOS and capacity-boundary checks also cover the last state commit, when no further proposal is needed.

A.6 Independent task-quality evaluation

Table 10: MATH500 accuracy from the current complete outputs in Table 2. All 500 requests per arm are scored, including capped outputs.

Drafter	Execution	Correct / total	Accuracy
KDA + Target-KV	Serial	413 / 500	82.6%
	Overlap	412 / 500	82.4%
Two-stage DSpark	Serial	416 / 500	83.2%
	Overlap	416 / 500	83.2%

We score saved complete responses with Math-Verify 0.9.0, using boxed-first extraction and first-match equivalence checking against official MATH500 answers (Hugging Face, 2025). Responses include the initial output token; no generation is repeated for scoring. Within each drafter, serial and overlap differ by one correct answer for KDA and zero for Two-stage DSpark. Both paired exact McNemar tests give $p = 1.0$. This task-level check does not establish decoding equivalence.

A.7 FIXED-BATCH MEASUREMENTS AND OCCUPANCY VIEWS

Table 3 compares two-GPU overlap with single-GPU DSpark. In two-GPU overlap, Target and Small share one GPU and Main runs on the other. Table 3 uses rounds that start with a real active request in every slot. All slow rounds in this set are retained. Throughput divides useful output tokens by the total time of the selected rounds. Mean round time uses that same total time. Timing starts after prefill and initial drafting; it includes subsequent submission, communication, state handling, waiting, and final drain. All CUDA graphs used during decoding are warmed before measurement.

Both methods process the same 500 MATH500, 256 MBPP, and 750 Arena prompts with confidence 0.5 and EOS/4096 termination. Completed slots are not refilled, and padding is masked. Each method groups requests using its own historical output lengths. Fully occupied rounds can therefore still differ in batch membership and prefix lengths.

Table 11: Sensitivity to occupancy selection. Decode rate (token/s) for single-GPU DSpark and two-GPU overlap, with gain over DSpark. Every view retains all slow rounds. Table 3 contains the fully occupied rounds.

		Complete run			$\geq 80\%$ active slots		
Task	BS	DSpark	Overlap	Gain	DSpark	Overlap	Gain
MATH500	16	2928.7	3413.1	+16.54%	5722.2	6603.4	+15.40%
	32	3450.8	3868.2	+12.10%	7860.5	8998.3	+14.47%
MBPP	16	3147.0	3198.1	+1.63%	5479.4	6375.7	+16.36%
	32	3171.1	3588.1	+13.15%	7596.6	8852.0	+16.53%
Arena	16	1649.6	1630.4	-1.16%	3398.1	3763.9	+10.76%
	32	1386.9	1777.6	+28.17%	4026.8	4378.2	+8.73%

At least 80% occupancy means 13/16 or 26/32 active real requests at round start. Rates are pooled token/time ratios within each view, not occupancy-normalized estimates.

Table 12: Fully occupied rounds / all rounds (percentage). Rounds are selected by the number of active requests, regardless of duration.

Task	BS	Single-GPU DSpark	Two-GPU overlap
MATH500	16	2,800 / 8,644 (32.39%)	2,672 / 8,926 (29.94%)
	32	1,199 / 4,642 (25.83%)	1,065 / 4,987 (21.36%)
MBPP	16	727 / 1,763 (41.24%)	761 / 2,018 (37.71%)
	32	348 / 1,112 (31.29%)	327 / 1,149 (28.46%)
Arena	16	7,058 / 28,593 (24.68%)	7,245 / 32,449 (22.33%)
	32	3,281 / 18,043 (18.18%)	3,275 / 16,860 (19.42%)

Table 11 gives complete-run rates and rates for rounds with at least 80% active slots. Complete-run gains range from -1.16% on Arena B16 to $+28.17\%$ on Arena B32; these include both execution speed and differences in which requests remain active. Arena B32 has 13.41 versus 11.50 time-weighted active slots. We keep all rounds that meet each active-slot threshold, including SCD's 592.38 ms Arena B32 round and the 426.51/446.48 ms MATH500 B16/B32 rounds.

Small usage divides the active request-rounds handled by Small by all active request-rounds. Accepted length includes bonus tokens; draft-token acceptance rate divides accepted draft tokens by candidates submitted after confidence truncation. The final verification record can include tokens beyond the returned EOS/cap output, so acceptance and throughput use different token counts. Appendix C.1 includes the partially occupied tails.

A.8 EARLY VERSUS DELAYED FEATURE USE ON THE SAME BATCH

Before timing, we choose one batch of 32 Arena requests using median historical length. The two configurations keep models, devices, CPU affinity, confidence, graph preparation, and Main + Small computation fixed. One uses H25/H30/H33 as soon as they are available. The other saves these

snapshots and uses them only after the verification result arrives. Draft input preparation and KV transport are unchanged. Because the delayed configuration still computes the same branches, this comparison tests when features are used, rather than an optimized ordinary serial implementation.

Four independently initialized pairs use early/delayed, delayed/early, early/delayed, and delayed/early order. Each arm is warmed separately and runs to EOS/4096. All 32 requests reach EOS in 367 rounds with 15,771 decode tokens in every run. Within each pair, output IDs, active masks, proposal lengths, accepted counts, predictor hits, and candidate sources match. Timing excludes prefill and initial drafting and includes subsequent work and final drain. No slow rounds are removed, and no cold graph capture occurs during measured decoding. The earlier exploratory single pair is excluded.

Each configuration generates 63,084 decode tokens across the four repeats. Total decode time falls from 39.982 s with delayed feature use to 33.798 s with early use. With the same token count, this is a 15.47% time reduction, equivalent to an 18.29% increase in decode rate. The four rate gains are 19.92%, 15.79%, 16.99%, and 20.52%, with mean 18.30%, median 18.46%, and sample standard deviation 2.28 percentage points. Each paired process restart is an independent repeat; rounds within a run are not. These repeats test stability on the selected batch, not variation across batches or the separate BS1 deployments.

At each active-slot threshold, we select the same rounds from both runs in a pair. At least 26 active slots and all 32 active slots give rate gains of 22.19% and 21.64%, respectively. Their pooled counts are 372 and 300 rounds per arm. Selection depends only on active-request counts, with no latency filtering or occupancy correction. Over complete runs (1,468 rounds per arm), mean round time is 23.02 versus 27.24 ms and rate is 1,866.47 versus 1,577.82 token/s (early versus delayed). With at least 26 active requests, the corresponding values are 20.62 versus 25.20 ms and 5,438.20 versus 4,450.50 token/s. Table 4 gives the fully occupied view.

B Condition design and execution analysis

B.1 Condition configurations

Table 13: 4B condition configurations. F denotes fusion of the listed hidden sources.

Configuration	D0–D2	D3–D4
Full H33	$F[1, 9, 17, 25, 33]$	$F[1, 9, 17, 25, 33]$
H30/H33	$F[1, 9, 17, 25, 30]$	$F[1, 9, 17, 25, 33]$
H17/H33	$F[1, 9, 17]$	$F[1, 9, 17, 25, 33]$
Progressive sources	$F[1]$; $F[1, 9]$; $F[1, 9, 17]$	$F[1, 9, 17, 25]$; $F[1, 9, 17, 25, 33]$

H17 changes both the depth and the number of target feature sources. H30/H33 and full H33 keep the same late-stage inputs, allowing us to compare accepted length when D0–D2 use earlier features.

B.2 KDA layer computation and condition transport

For current block representation x and state S of the verified prefix, a KDA layer can be summarized as

$$\widetilde{x} = x + \gamma \odot \mathrm{Retrieve}(x, K_T, V_T), \tag{5}$$

$$z = \widetilde{x} + W_O \left[\mathrm{Local}(\widetilde{x}) + \eta \odot \mathrm{Memory}(\widetilde{x}, S)\right], \tag{6}$$

$$x' = z + \mathrm{MLP}(\mathrm{Norm}(z)). \tag{7}$$

Each function includes its normalization and projections. Retrieve builds a trainable query using the target layer's head dimensions, query normalization, and positional encoding. It reads native KV and projects the result into draft space, with γ controlling the residual contribution. Local applies attention within the candidate block. Memory reads recurrent history through a per-head gate η. Both operate on the representation after target-KV retrieval. Each KDA layer updates its state from the corresponding fused target hidden states and commits only verified prefix tokens. Candidate

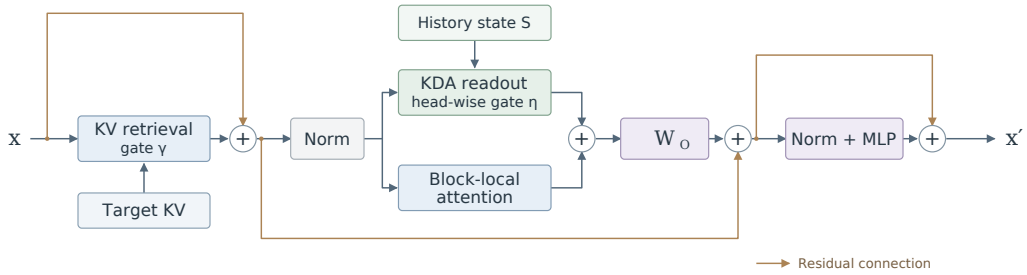

Figure 5: A KDA + Target-KV layer. It combines target-KV retrieval, attention within the candidate block, and recurrent memory. Brown connections are residuals.

generation reads branch state without writing unverified tokens into public history. Target KV reads are restricted to positions before the anchor. Native KV can be prepared as soon as its target layer finishes; KV2, KV10, and KV18, for example, can transfer before H30. The KDA update still waits for the fused hidden features. Prefill transfers the required history, and decode transfers only new entries. Two-stage DSpark instead constructs its own context KV from fused hidden states and does not use this native-KV transfer path.

B.3 Offline prediction coverage under a fixed branch budget

The two-stage draft can begin computing before verification ends; branch allocation determines which candidates to compute in that time. A probability threshold for each request allows the total branch count to vary. Here we instead fix the batch's total budget and distribute branches across requests using their joint probabilities.

Objective and allocation rule. For m active requests, let $p_{i1} \geq \cdots \geq p_{i4}$ be their sorted joint probabilities over mutually exclusive boundary–anchor pairs. Under an equal-cost approximation and a mandatory Top1 branch per request, choose

$$\max_{k_1,\ldots,k_m} \sum_{i=1}^{m} \sum_{j=1}^{k_i} p_{ij}, \qquad \sum_{i=1}^{m} k_i \leq N, \quad k_i \in \{1,2,3,4\}, \quad m \leq N \leq 4m. \tag{8}$$

Assign Top1 to every request, then repeatedly choose the largest remaining marginal probability. Because probabilities decrease within each request, this greedy rule is optimal under the stated branch-count constraint; ties can be resolved in prefix order. By linearity of expectation, the objective is the *model-predicted expected number of hit requests*. Dividing by m gives predicted average recall. We keep the original joint probabilities over boundaries and candidate tokens, without renormalizing Top4.

Paired replay and equal-budget baseline. We select 100 Arena requests before inspecting predictions and record Top4 probabilities during Top2 execution for up to 16 cycles. Replay uses six complete groups of 16 requests: 96 requests and 1,535 active request-rounds. The remaining four requests are excluded from this fixed-capacity comparison. Budgets apply to 16 physical slots, not to a batch size of N. In the only round with an ended slot, that slot retains a zero-probability placeholder and does not count toward recall. All policies are evaluated on the same recorded states rather than new trajectories generated under each policy.

We also compare fixed Top1–Top4 with a balanced allocation that ignores probabilities. Each slot receives $\lfloor N/16 \rfloor$ branches, and a uniformly selected subset receives the remaining branches. We compute expected recall exactly over these assignments. At $N = 56$, eight slots use Top3 and eight use Top4. This tests whether probabilities improve allocation at the same budget, rather than merely showing that the last few branches contribute little.

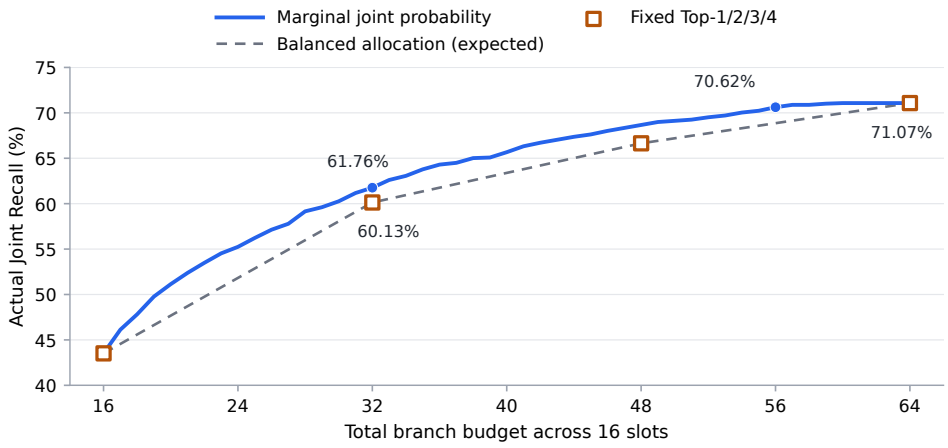

Figure 6: Offline Joint Recall versus branch budget on fixed Top2 Arena trajectories: six 16-slot groups, 96 requests, and 1,535 active request-rounds. Fixed Top-K uses $N = 16K$ branches. The balanced curve averages over probability-independent extra-branch assignments; the marginal curve allocates by joint probability.

At 32 branches, marginal allocation improves recall from 60.13% to 61.76%. At 56 branches it reaches 70.62%, versus 68.86% for balanced allocation at the same budget and 71.07% for fixed Top4 at 64 branches. Thus 12.5% fewer branches cost 0.46 percentage points of coverage. The paired differences and their intervals appear in Table 14. Intervals use request-cluster resampling of the recorded Top2 trajectories, keeping each request's cycles and paired policy outcomes together.

Table 14: Paired coverage differences in percentage points. Intervals use 10,000 request-cluster bootstrap samples (seed 20260924), retaining each request's observed cycles and paired policy outcomes. Batch groupings and branch assignments are held fixed during resampling.

Contrast	Difference	95% interval
Marginal 32 − fixed Top2 (32)	$+1.63$	$[+0.20, +3.06]$
Marginal 56 − balanced (56)	$+1.76$	$[+1.27, +2.25]$
Marginal 56 − fixed Top4 (64)	-0.46	$[-0.78, -0.13]$

GPU allocation. The graph-based allocator selects and packs branches in 26–42 μs, using median times across the tested capacities. After changing the inputs, we verify that it respects the budget, maps branches to the correct rows, and maximizes the probability objective. The recall results evaluate allocation on recorded trajectories; this component test measures selection and packing time.

B.4 Layer-cost matching across target scales

How many draft layers can run early depends on their cost relative to the target layers that remain. In our DSpark configurations, draft and target have the same hidden width, feed-forward width, attention-head configuration, and basic decoder-block structure; the draft has fewer blocks (Cheng et al., 2026). In the measured 4B models, these are width 2560, FFN width 9728, and 32 query/8 KV heads with head dimension 128. If both models scale in similar ways, this correspondence can preserve the time available for early draft layers.

Ignoring auxiliary overheads, an early stage of k layers and a target tail of m layers have the approximate ratio

$$\frac{T_{\text{early}}}{W} \approx \frac{k}{m}\frac{c_D}{c_T}, \tag{9}$$

where c_D and c_T are layer times at the actual branch count, batch shape, and prefix length. If scaling changes both by similar factors, their ratio and the number of layers that fit remain comparable. Prediction, feature transfer, state updates, and different attention workloads also take time. We therefore choose the split from the available window rather than always using H30 or three layers. The mechanism can extend to larger width-matched targets; our 14B results check accepted length, while our timings use 4B/A100.

B.5 Target feature layers and draft depth

An eight-layer draft with three stages. We train an eight-layer Softmax DSpark draft for Qwen3-4B, without KDA or a Target-KV adapter. Layers 1–3 use $F[1, 9, 17, 25, 28]$, layers 4–6 use $F[1, 9, 17, 25, 30]$, and layers 7–8 use $F[1, 9, 17, 25, 33]$. Hidden width is 2560, FFN width is 9728, and block size is seven. Training uses the full OpenPerfectBlend clean-v2 data for ten epochs; evaluation uses the final step-24700 checkpoint (`iter_0024701`).

Evaluation follows Appendix A.2: temperature one, confidence truncation disabled, non-thinking generation, seed 980406, and EOS/2048 termination. MATH500, MBPP, and Arena use 500, 256, and 500 requests. Accepted lengths, including correction/bonus, are 5.86, 5.30, and 3.41 (Table 1). This gives the deeper draft access to progressively later target features and improves accepted length over the five-layer DSpark baseline on all three tasks.

The $3 + 3 + 2$ split puts the extra layers before the final stage. If the first two stages finish during target verification and the predicted branch matches, only the final two layers and handoff remain.

When extra draft layers help. The feature layer and draft depth can be selected together. An earlier target feature leaves more time for a deeper draft prefix, while the late stage can still use deeper features. The additional layers can then run during verification instead of extending the wait afterward. In DFlash on MATH500, increasing the draft from five to eight layers raises accepted length from 5.99 to 6.33 but reduces speedup from $4.71\times$ to $4.64\times$ (Chen et al., 2026, Table 6). For condition layer h and early-stage depth d, let $\tau(h, d)$ denote effective tokens per round and $C(h, d)$ the complete mean cycle cost, including prediction, fallback, and communication. For a fixed output budget N in approximately steady decoding, the latency estimate and condition for a faster configuration are

$$T_{\text{decode}}(h, d) \approx \frac{N\, C(h, d)}{\tau(h, d)}, \qquad \frac{\Delta\tau}{\tau} > \frac{\Delta C}{C} \tag{10}$$

respectively, with changes measured relative to the reference configuration. Extra early computation need not increase the hit-path wait if it fits in the enlarged window and other costs remain unchanged. Earlier features can also change accepted length and joint recall, affecting both τ and C. A deeper drafter helps when the relative increase in accepted length exceeds the relative increase in round time.

C Detailed branch and deployment results

C.1 Complete-trajectory prediction and acceptance

We recompute the following statistics from the complete September 24 decode records, excluding ended requests and padding but retaining partially occupied rounds. This differs from the fully occupied subset in Table 3. Joint recall requires both boundary and anchor to be correct and is grouped by whether Main or Small generated the current proposal.

We compute mean accepted length over active request-rounds and acceptance rate over submitted candidate tokens. The final EOS round retains its full verification record for acceptance statistics, whereas throughput counts output only through EOS. Because these counts differ, acceptance and round time do not give an exact decomposition of throughput.

C.2 Same-prefix conditional candidate quality

We take the first 16 existing requests from each of MATH500, MBPP, and Arena and inspect 16 consecutive rounds per request, giving 48 requests and 768 paired request-rounds. We reuse the five-

Table 15: Shared predictor and source usage over the complete September 24 trajectories (%).

Task	BS	Main Joint@2	Small Joint@2	Main use	Small use
MATH500	16	90.18	82.72	89.54	10.46
	32	90.06	82.34	89.39	10.61
MBPP	16	82.22	76.40	81.48	18.52
	32	82.41	75.99	81.54	18.46
Arena	16	69.54	54.76	64.42	35.58
	32	69.16	54.55	64.04	35.96

Table 16: Complete-trajectory accepted length / draft-token acceptance rate (%). Length includes bonus; rate excludes bonus.

Task	BS	Main	Small	DSpark	Combined SCD
MATH500	16	6.575 / 82.05	4.652 / 59.91	6.407 / 80.08	6.374 / 79.95
	32	6.561 / 81.89	4.627 / 59.91	6.389 / 79.80	6.356 / 79.79
MBPP	16	5.719 / 70.87	4.205 / 54.65	5.517 / 68.32	5.439 / 68.17
	32	5.625 / 69.57	4.155 / 54.22	5.456 / 67.51	5.354 / 67.03
Arena	16	4.317 / 67.42	2.347 / 47.44	3.780 / 62.53	3.616 / 62.59
	32	4.316 / 67.44	2.329 / 47.13	3.721 / 61.88	3.602 / 62.49

layer Main, two-layer Small DSpark, shared joint predictor, and Qwen3-4B target without training or changing the running decode trajectory. We group examples by whether the joint prediction hits at round t. At round $t+1$, both drafters generate candidates from the resulting *same true prefix and anchor* for comparison.

Table 17: Same-prefix paired candidate quality, grouped by the preceding joint prediction. Length includes actual bonus; draft-token acceptance rate excludes bonus. Δ denotes Main minus Small accepted length.

Prior prediction	Paired rounds	Main length / rate	Small length / rate	Truncation Main / Small	Δ [95% CI]
Hit	607	5.353 / 68.02%	4.997 / 65.85%	6.399 / 6.069	0.356 [0.215, 0.501]
Miss	161	3.677 / 50.83%	3.385 / 50.66%	5.267 / 4.708	0.292 [−0.017, 0.626]

Confidence 0.5 uses the native truncation rule. Acceptance rate divides total accepted draft tokens by total submitted candidates after truncation. Confidence intervals use 10,000 request-cluster bootstrap resamples, stratified by dataset, with seed 20260923; all paired rounds within a request remain together. Hit samples cover 48 requests and miss samples cover 45.

The target accepts 1.676 fewer tokens from Main after a miss than after a hit in the pooled sample (miss minus hit 95% CI $[-2.172, -1.174]$). However, the Main–Small gap changes by only -0.064 token, with a 95% CI of $[-0.373, 0.258]$. The interval includes zero, so we cannot conclude that Small loses less quality after misses. The direction differs by dataset (Table 18), and the pooled means also depend on dataset composition. These measurements link prediction misses to harder drafting rounds; they do not show that a miss causes the quality loss.

Each drafter recomputes candidates using private KV, planner, and output buffers. Target scoring reads prefix KV and uses a private temporary suffix, then restores mappings without committing state. This recomputation does not change the tokens generated by the running decoder. All 720 shadow/live comparisons agree on accepted counts; isolated recomputation differs from the selected branch on 2/768 pairs, by -3 and $+1$ token.

Table 18: Dataset-level paired accepted lengths and change in the Main–Small gap. The last column is $\Delta_{\text{miss}} - \Delta_{\text{hit}}$ with a request-cluster 95% interval.

Task	Hit: Main / Small	Miss: Main / Small	Gap change [95% CI]
MATH500	6.342 / 6.098	5.955 / 4.045	1.666 [0.330, 3.189]
MBPP	5.426 / 5.000	4.596 / 4.277	-0.107 [-0.654, 0.408]
Arena	3.848 / 3.421	2.663 / 2.772	-0.536 [-0.945, -0.140]

Table 19: Accepted length of separately trained history variants. The configurations differ, so these are architecture comparisons rather than strictly matched component ablations.

Target	Drafter	MATH500	MBPP	Arena
4B	DSpark	5.74	5.13	3.29
	Pure KDA	5.62	5.05	3.01
	Native Target-KV	5.62	5.02	3.05
	KDA + Target-KV	5.85	5.24	3.32
14B	DSpark	5.76	5.27	3.15
	KDA + Target-KV	5.85	5.37	3.18

C.3 History-structure controls

Evaluation uses 500/256/500 requests. Native Target-KV reads a 4K window; the combined model reads full Target KV. The single-path models also differ in training-data processing. The table compares the trained models with these differences in settings.

C.4 Additional capacity and single-GPU results

Table 20: Draft-owned state and resumable-session counts under the same static budget.

Drafter	State at 8K / 16K (MiB)	8K sessions	16K sessions
DSpark	160 / 320	45	22
KDA + Target-KV	2.5 / 2.5	50	25
Capacity gain	—	+11.11%	+13.64%

The 8K/16K short-output workload gives DSpark rates of 358.42/374.30 and KDA rates of 357.32/376.67 token/s, counting 31 tokens after the first output. In a separate full-MATH500 single-GPU comparison, DSpark reaches 560.83 token/s (1.783 ms/token) and KDA reaches 548.68 (1.823 ms/token).

C.5 Execution cost of Top2 and Top4

We compare fixed Top2 and Top4 on Arena-Hard-v2 at batch size 16, with 750 requests per configuration. Both runs continue to EOS or 4096 tokens, exclude prefill and the initial draft, and retain all decode rounds without latency filtering. These are complete fixed-batch runs, including partially occupied tails.

Top4 raises mean round time by 2.20% and lowers decode rate by 2.74%. The time-weighted active-slot count also falls by 2.38%, while rate per active slot falls by 0.37%. Thus the complete-run difference includes both execution cost and occupancy changes. The 5.16% increase in total decode time also includes 2.28% more generated tokens.

Most of the extra GPU time comes from matrix multiplications in D0–D2. D0–D2 takes 0.715 ms longer, including 0.635 ms more cumulative GEMM time. The predictor, attention kernels, Main late stage, and Small take nearly the same time. Main's late stage and Small each take about 1.58 ms in this profile. When both paths are active, the next verification waits for the slower one; reducing

Table 21: Top2 versus Top4 on complete Arena BS16 runs. Active slots are time-weighted. Rate per active slot divides actual output by accumulated active-slot time; it is not a full-occupancy throughput measurement.

Metric	Top2	Top4
Decode rate (token/s)	1630.44	1585.78
Mean complete round (ms)	16.974	17.346
Decode time (s)	550.78	579.18
Generated tokens	898,018	918,450
Time-weighted active slots	7.094	6.925
Rate per active slot (token/s)	229.84	229.00

Table 22: Independent GPU profile of Top2 and Top4 on the same 16-request Arena batch. Values are milliseconds in round 16; both captured rounds (15 and 16) have all slots active. GEMM and attention rows are kernel-time sums within D0–D2, not additional stages.

GPU stage / kernel group	Top2	Top4
Predictor	0.611	0.611
Early stage D0–D2	2.456	3.171
GEMM kernel sum	1.017	1.652
Attention kernel sum	1.215	1.213
Main D3–D4 + head	1.572	1.574
Small D0–D1 + head	1.584	1.582

misses may therefore leave that wait unchanged. These component times cannot be added to explain the full-run latency difference: stages overlap, trajectories differ, and profiling affects CPU submission. Top2 is faster in this configuration; Appendix B.3 considers assigning branches by marginal probability under a fixed budget.