SAS: SCALING SPECULATIVE DRAFTING FOR EFFICIENT 1T LLM SERVING WITH SEMI-AUTOREGRESSIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates LLM serving by verifying multiple tokens in parallel through a drafter model. When target verification is much more expensive than drafting, there is room to scale draft computation for longer accepted prefixes. We study depth scaling, which increases backbone capacity by adding layers, and round scaling, which reapplies a shared backbone after extending the selected prefix. In our study, round scaling improves acceptance length more than depth scaling, at the cost of multiple sequential forwards. We introduce Semi-Autoregressive Scaling (SAS) to turn the acceptance gains of round scaling into faster generation while controlling sequential overhead. Its *in-block draft* combines shallow readout and candidate hedge to produce multiple tokens per forward with early predecessor conditioning. Dynamic partition training supports different round schedules with shared weights, while execution specialization compacts layouts and fuses state handling. On the open-weight 1T-parameter MiMo-v2.5-Pro target evaluated on 32 NVIDIA H20 GPUs across four nodes with 48 concurrent requests in total, the two SAS schedules improve overall throughput by 14.56%–17.08% over native multi-token prediction (MTP), 8.11%–9.88% over DFlash, and 4.17%–7.92% over DSpark across both greedy decoding and sampling, reaching up to $2.30\times$ autoregressive throughput.

1 INTRODUCTION

Speculative decoding accelerates LLM generation by pairing a lightweight drafter with a large target model (Leviathan et al., 2023; Chen et al., 2023). The drafter proposes a block of tokens, which the target verifies in one forward pass. Rejection sampling preserves the target distribution while amortizing an expensive target call over multiple output tokens. Large models such as DeepSeek-V3 and GLM-5.2 adopt this approach through native multi-token prediction (MTP) modules (DeepSeek-AI, 2024; Z.ai, 2026; LLM-Core Xiaomi, 2026).

Longer accepted prefixes amortize verification over more output tokens, but improving draft predictions can require extra computation. For models with very large parameter counts, each target verification can be substantially more expensive than lightweight drafting. We profile an approximately 1T-parameter open-weight model, MiMo-v2.5-Pro (Xiaomi MiMo Team, 2026), and observe this cost asymmetry: optimized drafting takes approximately 1–3 ms, only about 3%–7% of target-verification time (Section 5.4). This gap motivates investing more computation in draft prediction, provided that accepted progress increases proportionally more than total step time.

Existing drafter designs prioritize better predictions while keeping draft overhead small (Zhou et al., 2024; Liu et al., 2024; Zhang et al., 2024; Elhoushi et al., 2024). Iterative causal drafters, such as EAGLE-3 and native MTP modules, use compact prediction paths, while DFlash and DART reduce serial work through parallel block prediction (Li et al., 2025b; Chen et al., 2026; Liu et al., 2026). Hybrid methods add lightweight causal correction, convolution, or candidate-path selection within this low-overhead design approach (Huang et al., 2026; Cheng et al., 2026; Inco AI, 2026). To exploit the potential headroom of large targets, we take DFlash, a strong parallel drafter, as our architectural starting point and investigate how to use the additional computation budget available with large targets. *Depth scaling* increases drafter capacity by adding independently parameterized layers within

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.

a single forward. *Round scaling* increases inference computation by applying a fixed-size backbone multiple times, reusing its parameters while extending the selected prefix between rounds.

Our motivation study (Section 3) finds that round scaling improves acceptance length by up to 19.6%, compared with 5.7% for depth scaling, relative to a five-layer, single-pass DFlash baseline. Depth scaling adds capacity while keeping the conditioning inputs of the parallel pass fixed. Causal rounds expose later predictions to selected predecessor tokens; their larger gains at later positions are consistent with the value of this additional conditioning. However, obtaining this information requires additional sequential draft forwards, whose overhead can offset the acceptance gain.

We introduce Semi-Autoregressive Scaling (SAS) to make causal round scaling efficient. SAS uses *in-block draft* through shallow readout to propose predecessor candidates early enough for the remaining layers to compute their conditional predictions within the same forward. *Candidate hedge* selects the continuation matching the full-depth predecessor prediction, with a fallback for a missed candidate. In-block draft thus enables each round to produce multiple tokens with sufficient causal information. SAS further incorporates *Dynamic partition training* to make the tradeoff between prefix conditioning and serial rounds adjustable with shared weights. To minimize overhead of causal computing, *execution specialization* exploits the fixed row roles of a chosen schedule to compact round layouts and fuse readout and prefix updates.

On the 1T-parameter MiMo-v2.5-Pro evaluated on 32 NVIDIA H20 GPUs across four nodes with 48 concurrent requests in total, SAS reaches up to $2.30\times$ the overall throughput of target-only autoregressive decoding. Across greedy decoding and sampling, its two schedules improve overall throughput by 14.56%–17.08% over the target's native MTP, 8.11%–9.88% over DFlash, and 4.17%–7.92% over DSpark. We further demonstrate that round scaling can build on improvements to the drafter itself: retaining DFlash-2's dynamic convolution and candidate-path selector while applying causal rounds on Qwen3.8-27B (Qwen Team, 2026) raises overall acceptance length by 9.37%.

Our contributions are:

- We study how additional draft computation can improve acceptance through depth and causal rounds, motivating the design of efficient round scaling.
- We enable efficient causal round scaling with an adjustable computation budget through in-block draft, dynamic partition training, and execution specialization.
- We demonstrate end-to-end gains for a 1T LLM under greedy decoding and sampling, and additional acceptance gains when composing round scaling with other advanced drafter designs.

2 Background and Related Work

Let c denote the verified context entering a speculative-decoding step and x_0 its final token. The root-inclusive block budget B comprises this anchor and $m = B - 1$ draft positions $x_{1:m}$. The drafter constructs these proposals for target verification (Leviathan et al., 2023; Chen et al., 2023; Miao et al., 2024). Existing drafting methods differ in the information available when predicting each x_i (Xia et al., 2024). Figure 1(a) contrasts the iterative causal and parallel block paths described below.

2.1 Iterative causal drafting

An iterative drafter constructs the proposal from left to right:

$$q_{\text{causal}}(x_{1:m} \mid c) = \prod_{i=1}^{m} q_i(x_i \mid c, x_{<i}), \qquad x_{<i} = (x_1, \ldots, x_{i-1}). \tag{1}$$

Early speculative decoding uses a smaller autoregressive model for this purpose. EAGLE builds a single-layer drafter on target hidden features, and EAGLE-3 improves prediction through multi-level features and training-time rollout (Li et al., 2024a; 2025b). Native MTP modules in DeepSeek-V3 and GLM-5.2 provide similar causal drafters trained with the target (DeepSeek-AI, 2024; Z.ai, 2026). In causal drafting, each later prediction benefits from conditioning on realized predecessor tokens at the cost of successive draft forwards, motivating lightweight per-step computation (Li et al., 2024b; Cheng et al., 2024).

2.2 PARALLEL BLOCK DRAFTING

A parallel block drafter predicts all positions from the same verified context (Gloeckle et al., 2024):

$$q_{\text{par}}(x_{1:m} \mid c) = \prod_{i=1}^{m} q_i(x_i \mid c). \tag{2}$$

Blockwise decoding and Medusa use parallel prediction heads (Stern et al., 2018; Cai et al., 2024); DART and DFlash use masked block prediction conditioned on target features (Liu et al., 2026; Chen et al., 2026). In parallel drafting, one forward produces several proposals, shortening the serial path, but later predictions cannot observe discrete predecessor tokens selected within that block.

2.3 HYBRID DRAFTING

Hybrid designs seek richer token dependencies while retaining a lightweight draft path (Li et al., 2025a; Shi et al., 2026). Hydra introduces sequentially dependent heads, while Domino and DSpark add lightweight post-backbone causal correction (Ankner et al., 2024; Huang et al., 2026; Cheng et al., 2026). DFlash-2 combines dynamic convolution within the parallel backbone with a subsequent candidate-path selector (Inco AI, 2026). These mechanisms enrich predictions through compact modules or local interactions within the draft path that limit the prediction capability.

Our study asks how a larger draft-compute budget can improve predictions through larger backbone capacity or repeated causal rounds when target verification is relatively expensive (Section 3).

3 MOTIVATION: DEPTH SCALING VS. ROUND SCALING

A larger draft-compute budget can support a larger backbone or more inference steps with a fixed-size backbone. *Depth scaling* increases capacity by adding independently parameterized layers within one forward; *round scaling* reuses the backbone across multiple forwards, extending the selected prefix between rounds. Figure 1(a) illustrates the causal and parallel drafting paths underlying these choices. We measure their effect on acceptance at each draft position on Qwen3-8B (Qwen Team, 2025).

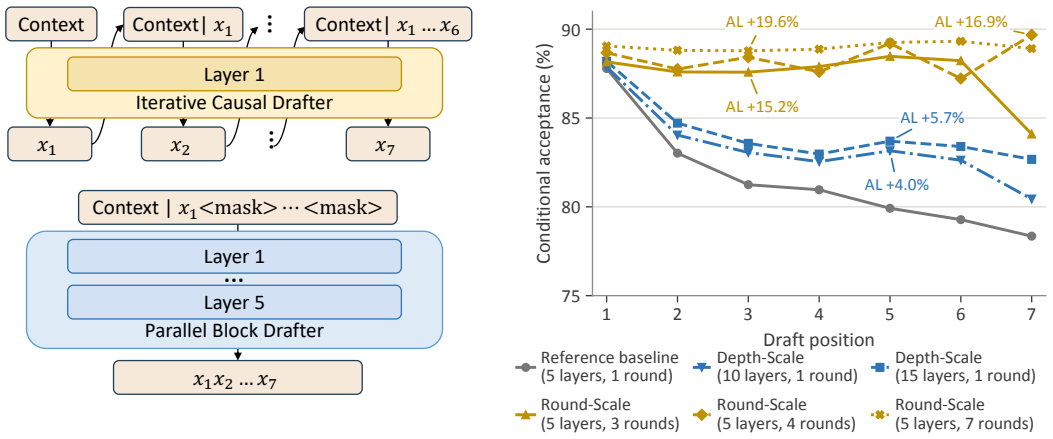
We use a 5-layer, single-pass DFlash drafter as the reference baseline. *Depth-Scale* increases its depth to 10 or 15 layers, while *Round-Scale* uses 5-layer drafters with 3, 4 or 7 rounds. All configurations use $B = 8$: one anchor and seven proposal positions. The 3, 4-round schedules $(2, 2, 3)$ and $(2, 2, 2, 1)$ divide these seven positions into groups with nearly equal token counts. Each group can condition on tokens selected in earlier groups. 7-round schedule generates one token per forward before a single final verification. All variations are trained using the same recipe.

Let A be the accepted draft-prefix length excluding the target bonus. We plot $a_i = \Pr(A \geq i \mid A \geq i - 1)$ at position $i \in \{1, \ldots, 7\}$: the probability of accepting the next proposal conditional on accepting every earlier one. Figure 1(b) summarizes the results. Appendix C provides the detailed experimental settings, metric calculation and per-dataset acceptance length (AL) results.

Depth-Scale. Increasing depth to 10 or 15 layers changes first-position acceptance little, from the reference's 87.8% to 87.9% or 88.2%. At the seventh position, acceptance increases from 78.3% to 80.4% or 82.7%. AL rises from the reference's 4.603 to 4.786 ($+4.0\%$) or 4.866 ($+5.7\%$). Additional layers improve predictions but increase computation throughout the block without supplying the realized predecessor tokens unavailable to a single parallel pass.

Round-Scale. All three configurations improve acceptance at positions 2–7 over the reference and both Depth-Scale configurations. AL reaches 5.30, 5.38, and 5.50 for 3, 4, and 7 rounds: gains of 15.2%, 16.9%, and 19.6% over the baseline. 7-round acceptance stays within 88.8%–89.3% across positions, yielding the highest overall AL and supporting the value of selected-prefix conditioning.

In summary, **at this fixed proposal length, adding causal rounds yields larger acceptance gains than adding backbone depth in our study.** Turning these gains into faster generation requires controlling the cost of serial draft forwards. Section 4 develops in-block draft to produce multiple tokens per forward and uses execution specialization to reduce multi-round execution overhead.



(a) Drafting paradigms.

(b) Conditional acceptance.

Figure 1: Drafting paradigms and scaling behavior. (a) Causal drafting conditions on selected predecessors across forwards; parallel drafting predicts a chunk in one forward. (b) Qwen3-8B conditional acceptance for seven proposals ($B = 8$).

4 METHOD

SAS makes round scaling efficient through *in-block draft*, which uses early predecessor candidates to produce a group of tokens in each forward. Group sizes determine the tradeoff between selected-prefix updates and sequential draft work; *dynamic partition training* makes this choice adjustable with shared weights. We then use *execution specialization* to reduce the chosen schedule's GPU execution overhead. Training settings and pseudocode are listed in Appendix A and B.

4.1 IN-BLOCK DRAFT

Section 3 motivates scaling causal rounds: preceding token embeddings carry semantic information that improves later predictions. A naive token-wise drafter makes this information available by completing each prediction before starting the next. For $B-1$ proposal positions, this requires $B-1$ sequential full draft forwards, making causal conditioning expensive.

To reduce forward invocations while injecting consistent causal information before the compute-intensive Transformer layers, our key insight is to predict a predecessor early enough for the remaining draft layers to use it. We propose several possible predecessor tokens through a *shallow readout* after the first layer, then use the full-depth prediction to select the matching predecessor and continuation. We describe this computation for a general token group and use the two-token round in Figure 2 to illustrate each step.

Multi-round drafting. SAS partitions the $m = B-1$ proposal positions into J ordered contiguous groups $\mathcal{G}_{1:J}$ and applies the same N-layer stack $\mathcal{D}_{1:N}$ to each group. The schedule is $\pi = (|\mathcal{G}_1|, \dots, |\mathcal{G}_J|)$, with $\sum_j |\mathcal{G}_j| = m$. Each group contains at least two positions, giving $J \le \lfloor m/2 \rfloor$. Round j receives the verified context c and the selected prefix P_{j-1} from earlier rounds, and generates all positions in $\mathcal{G}_j$ in one forward. The following equations apply to every group; the running example uses $\mathcal{G}_1 = \{1, 2\}$ and $K = 2$, as in the right panel of Figure 2.

Round inputs. The first position $s_j = \min \mathcal{G}_j$ is the *leader*; the remaining positions $\mathcal{F}_j = \mathcal{G}_j \setminus \{s_j\}$ are *followers*. The leader query starts from $E(x_{s_j-1})$, while every follower $i \in \mathcal{F}_j$ starts from a learned mask embedding. Both are fused with the target context feature h_c. At layer 1, each query reads the verified context and the selected prefix P_{j-1}, together with its own state; tokens from the current group are not yet available. Let h_i denote its layer-1 output. In the two-token example, $E(x_0)$

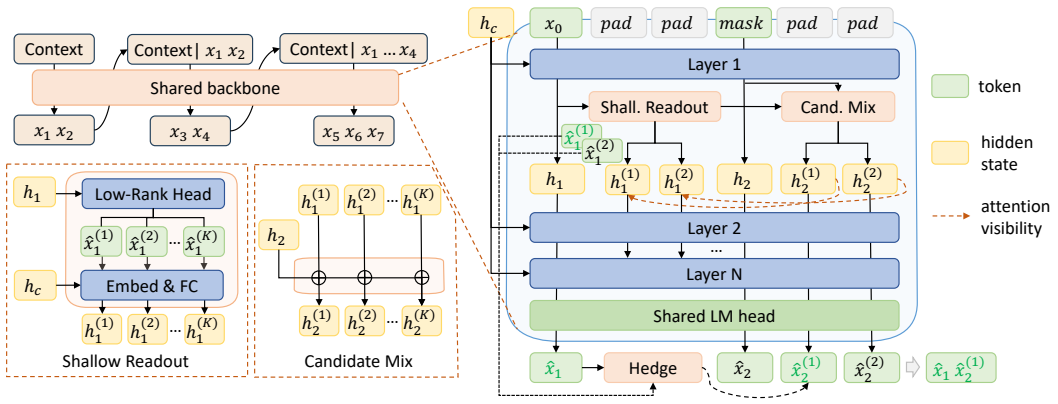


Figure 2: SAS data flow. Upper left: each round applies in-block draft to a token group using the shared backbone. Lower left: shallow readout supplies candidate states for residual mixing. Right: one two-token round with $K = 2$, illustrating selection of the first candidate branch. Context projections and committed prefix slots are omitted.

initializes the leader that produces h_1, while the masked follower produces h_2. These states support early leader prediction and preparation of its possible continuations.

Shallow readout. The leader state produces K possible leader tokens as *candidates*, using a normalized low-rank head followed by TopK:

$$\begin{aligned} \mathrm{ShHead}(h_i) &= W_{\mathrm{up}}\,\mathrm{SiLU}(W_{\mathrm{down}} N_{\mathrm{sh}}(h_i))\,, \\ (\hat{x}_{s_j}^{(1)}, \ldots, \hat{x}_{s_j}^{(K)}) &= \mathrm{TopK}\big(\mathrm{ShHead}(h_{s_j}), K\big)\,. \end{aligned} \tag{3}$$

Here N_{sh} is the shallow normalization, and the learned projections use a bottleneck width r much smaller than the hidden size d. The shallow head acts as a *draft for the draft*: its candidate set only needs to contain the eventual full-depth leader. These candidates supply predecessor embeddings to the remaining layers while the leader continues toward its final prediction.

Candidate mix. To let the remaining layers use these tokens immediately, each candidate defines a branch containing all followers in $\mathcal{F}_j$. Candidate $\hat{x}_{s_j}^{(k)}$ is converted into a token-conditioned state through embedding lookup E and input fusion ϕ. This state has two uses: it initializes a *committed slot* that every follower in that branch can attend to, and it is added to each follower's layer-1 state:

$$h_{s_j}^{(k)} = \phi\Big(E(\hat{x}_{s_j}^{(k)}), h_c\Big)\,, \qquad h_i^{(k)} = h_i + h_{s_j}^{(k)}, \quad i \in \mathcal{F}_j. \tag{4}$$

Thus candidate branch k contains $(h_i^{(k)})_{i\in\mathcal{F}_j}$, all conditioned on the same leader candidate. A *fallback branch* retains $(h_i)_{i\in\mathcal{F}_j}$ without the new leader candidate, preserving the previously selected prefix. In the two-token example, these are $h_2^{(1)}$, $h_2^{(2)}$, and fallback h_2. Candidate committed states start at the layer-1/2 boundary, while followers retain their layer-1 computation.

Deep computation and candidate hedge. The leader and all follower branches pass through layers 2:N together. At each deep layer, follower $i \in \mathcal{F}_j$ reads its matching candidate committed state, the available prefix, itself, and earlier follower query states at positions $\{i' \in \mathcal{F}_j : i' < i\}$ within the same branch. These attention edges pass continuous predecessor information through the layer stack; they do not require earlier followers' final token selections. Different candidate branches remain isolated, and fallback uses the same follower connections without the new leader candidate. The shared LM head produces the full-depth leader prediction $\hat{x}_{s_j}$ and the follower predictions $\hat{x}_i^{(k)}$ and $\hat{x}_i$. *Candidate hedge* selects the entire follower group using the leader's match:

$$x_{s_j} = \hat{x}_{s_j}, \qquad x_{\mathcal{F}_j} = \begin{cases} (\hat{x}_i^{(k)})_{i\in\mathcal{F}_j}, & \hat{x}_{s_j} = \hat{x}_{s_j}^{(k)} \text{ for some } k, \\ (\hat{x}_i)_{i\in\mathcal{F}_j}, & \text{otherwise.} \end{cases} \tag{5}$$

A match retains one candidate branch for all followers in position order; a miss uses the fallback group. With a single follower, the set of earlier follower positions is empty and this reduces to the two-token computation in Figure 2. Its green example selects $(x_1, x_2) = (\hat{x}_1, \hat{x}_2^{(1)})$. Each round updates $P_j = P_{j-1} \parallel x_{\mathcal{G}_j}$. After all rounds, the target verifies one linear proposal $x_{1:m}$.

Larger groups reduce the number of sequential draft forwards, but more predictions rely on within-round follower states. The preferred schedule therefore depends on whether the extra conditioning compensates for its execution cost. We next make this tradeoff selectable by training the same weights across partitions.

4.2 Dynamic Partition Training

To support different choices along this tradeoff, we train shared parameters across the schedules π. The round partition can then be selected for the deployment scenario without retraining the drafter.

Sampling causal layouts. At each training forward, we sample $\pi \sim U(\Pi)$ from a set of valid partitions. The sampled groups $\mathcal{G}_{1:J}$ determine their leaders s_j, followers $\mathcal{F}_j$, and available prefixes P_{j-1}. All roles, attention dependencies, and candidate-injection positions follow the construction in Section 4.1. Varying the partition exposes the shared model to different prefix lengths and leader/follower assignments.

Training shared predictions. The frozen target supplies hidden features and next-token distributions on training sequences. Teacher forcing instantiates P_{j-1} with known tokens and supplies the true leader embedding at the injection boundary in equation 4. We jointly supervise the groups under their corresponding attention masks, retaining the same within-round follower dependencies as at inference. For the shared trainable parameters θ, we optimize

$$\mathcal{L} = \mathbb{E}_{\pi \sim U(\Pi)} \left[\ell_{\text{deep}}(\theta; \pi) + \lambda_{\text{sh}} \ell_{\text{sh}}(\theta; \pi) \right], \tag{6}$$

Here ℓ_{deep} supervises valid proposal positions in $\bigcup_j \mathcal{G}_j$, and ℓ_{sh} supervises the current leaders $\{s_j\}_{j=1}^{J}$. For readout $u \in \{\text{deep}, \text{sh}\}$, let $\Omega_u(\pi)$ be its valid supervision positions, p_i the target distribution, and $q_{\theta,i}^u(\pi)$ the draft distribution. We use the weighted squared Hellinger objective

$$\ell_u(\theta; \pi) = \frac{\sum_{i \in \Omega_u(\pi)} w_i H^2\Big(p_i, q_{\theta,i}^u(\pi)\Big)}{\sum_{i \in \Omega_u(\pi)} w_i}, \tag{7}$$

where H^2 denotes the squared Hellinger distance (Hellinger, 1909).

4.3 Efficient Execution of Multi-Round Drafting

For a schedule π supported by dynamic partition training, additional causal rounds must repay their execution cost. Although candidate identities vary, π fixes their row roles and consumers. Our *execution specialization* exploits this structure to reduce the construction and state-transfer work repeated across rounds.

Compact round layouts. Each round uses a compact layout for P_{j-1} and its candidate branches, avoiding projection and feed-forward work on inactive rows that attention masking alone would retain. Shapes are fixed per round and batch bucket for graph replay, but can differ across rounds. Precomputed source/destination maps assemble prefix, leader, and follower rows directly. For two-token rounds, shallow leader preparation fuses residual addition, normalization, and position modulation. A mapped write then injects candidate embeddings into committed slots and follower residuals, preserving layer-1 states while avoiding repeated full-tensor operations.

Fused readout and prefix updates. Only leader and continuation rows need final normalization. Let H_j, U_j be the final hidden and residual matrices and S_j gather the readout rows. Row-wise RMSNorm gives

$$S_j(\text{RMSNorm}(H_j + U_j)) = \text{RMSNorm}(S_j(H_j) + S_j(U_j)). \tag{8}$$

We gather first and fuse residual addition, normalization, and packing into the head input, avoiding unused full-state writes while retaining all continuations needed by hedge. The existing draft-local collective exchanges scores and token ids; fused winner selection and hedge matching write P_j directly into the next round's GPU buffers.

Shared state and graph execution. All rounds reuse the verified context's per-layer draft keys and values, which depend only on target features and absolute positions. Stable GPU slots retain this state as the prefix grows and receive newly committed features; a captured input stage gathers it directly into block-graph buffers. We compute the block-constant context term of ϕ once and cache token-embedding projections. One CUDA Graph per batch bucket captures all rounds, candidate selection, and prefix updates, eliminating host dispatch between rounds.

5 Evaluation

5.1 Experimental Setup

Implementation and hardware. We implement SAS in SGLang (Zheng et al., 2024) and evaluate MiMo-v2.5-Pro. This mixture-of-experts target has 1.02T total parameters and 42B active parameters per token (Xiaomi MiMo Team, 2026). All methods use the same target model and tokenizer. The deployment uses four nodes with eight NVIDIA H20 GPUs each. Attention uses four data-parallel groups with eight-way tensor parallelism within each group (DP4 $\times$ TP8), while the MoE layers use expert parallelism across all 32 GPUs (EP32). CUDA Graphs are enabled.

Methods. We compare autoregressive decoding (AR), native MTP, DFlash (Chen et al., 2026), DSpark (Cheng et al., 2026) with its Markov autoregressive head, and SAS. Native MTP sequentially applies the target's three single-layer prediction modules to propose three tokens ($B = 4$). DFlash, DSpark, and SAS use $N = 5$, $B = 8$ and are trained with the same configuration. For SAS, we set $r = 256$ and $K = 4$, and evaluate $\pi = (2, 2, 3)$ with $J = 3$ and $\pi = (2, 5)$ with $J = 2$. The two schedules share the parameters θ through dynamic partition training as described in Section 4.2.

Workloads. We cover mathematical reasoning with GSM8K (Cobbe et al., 2021) and MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024), code generation with HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021), and dialogue with MT-Bench (Zheng et al., 2023). We also include 100 tasks of Web development (labeled as Webdev), forming 2,420 requests in total. We evaluate greedy decoding (top-$k = 1$, top-$p = 1$) and nucleus sampling (temperature 1.0, top-$p = 0.95$). Both enable thinking and stop at natural EOS or 4,096 generated tokens. We maintain 48 concurrent requests across the deployment by admitting a new request when one finishes. Each attention DP group allows at most 12 running requests; the actual per-step batch size varies with continuous batching.

Measurement. We repeat each main-table configuration three times per workload and decoding setting. Steady-state throughput is the number of client-received output tokens divided by elapsed time in the sustained-load window, including prefill, decoding, and serving overhead. We report its duration-weighted mean across repetitions. Acceptance length $\mathrm{AL}_{\mathrm{req}}$ is the mean per-request ratio of output tokens to target-verification steps, including the target bonus. Position-wise conditional acceptance a_i follows Section 3 and is computed from verification acceptance counts.

5.2 End-to-End Performance

Comparison with baselines. Both SAS schedules outperform native MTP, DFlash, and DSpark in throughput on every workload under both decoding settings (Table 1). The overall gains are 14.56%–17.08%, 8.11%–9.88%, and 4.17%–7.92%, respectively, reaching up to $2.30\times$ AR throughput. At the same $B = 8$ budget, both schedules also improve AL over DFlash and DSpark on every workload. Results across serving concurrency levels are provided in Appendix D.

Effect of decoding strategy. Greedy decoding yields higher acceptance length and absolute throughput than nucleus sampling for both SAS schedules. Relative gains over DSpark, however, are larger under sampling: 6.18%–7.92%, compared with 4.17%–4.29% under greedy decoding. Causal drafting remains effective under stochastic generation, with a larger relative advantage over DSpark despite lower absolute throughput.

Table 1: Steady-state end-to-end throughput (tok/s) and request-mean acceptance length (AL) on MiMo-v2.5-Pro at 48 concurrency. Native MTP uses $B = 4$; the other speculative methods use $B = 8$. Both SAS schedules share draft weights and use $K = 4$.

Method	GSM8K		MATH-500		HumanEval		MBPP		Webdev		MT-Bench		Overall	
	Tok/s	AL	Tok/s	AL	Tok/s	AL	Tok/s	AL	Tok/s	AL	Tok/s	AL	Tok/s	AL
(a) Greedy: top-$k = 1$, top-$p = 1$														
AR	1194.2	–	1374.8	–	1346.4	–	1303.3	–	1548.4	–	1321.2	–	1335.4	–
Native MTP	1973.8	3.57	2912.2	3.51	2576.4	3.39	2507.2	3.42	3885.8	3.73	2262.3	3.04	2626.1	3.52
DFlash	2164.7	5.17	3023.3	4.75	2604.6	4.38	2737.1	4.57	4734.6	6.13	2226.5	3.56	2831.3	4.95
DSpark	2315.9	5.36	3124.5	4.97	2710.8	4.60	2777.9	4.81	4689.1	6.30	2285.8	3.75	2948.2	5.15
SAS 2+2+3	2374.4	**5.63**	3286.3	**5.24**	2838.3	**4.97**	**2919.5**	**5.16**	4930.8	**6.54**	**2384.2**	**4.01**	3071.1	**5.44**
SAS 2+5	**2376.1**	5.55	**3296.3**	5.19	**2928.8**	4.88	2911.0	5.07	**4941.5**	6.44	2374.7	3.99	**3074.5**	5.37
(b) Sampling: temperature 1, top-$p = 0.95$														
AR	1153.1	–	1314.3	–	1261.7	–	1287.3	–	1488.0	–	1278.3	–	1285.2	–
Native MTP	1922.9	3.47	2772.5	3.36	2464.9	3.24	2418.7	3.34	3774.5	3.63	2237.2	2.92	2517.4	3.41
DFlash	2107.3	5.00	2759.5	4.56	2459.9	4.18	2599.2	4.40	4459.7	5.79	2143.3	3.37	2667.6	4.77
DSpark	2137.9	5.17	2877.6	4.76	2469.4	4.44	2597.0	4.64	4389.6	5.99	2069.2	3.59	2716.1	4.96
SAS 2+2+3	2272.4	**5.46**	3023.1	**4.97**	2516.8	**4.75**	2803.0	**4.96**	**4823.7**	**6.28**	2295.9	**3.81**	2883.9	**5.24**
SAS 2+5	**2296.0**	5.38	**3105.5**	**4.97**	**2673.8**	4.70	**2841.6**	4.90	4732.9	6.27	**2352.1**	3.79	**2931.3**	5.18

Table 2: Acceptance length (AL) on Qwen3.8-27B under greedy decoding. The overall value is the unweighted mean across workloads.

Method	GSM8K	MATH-500	HumanEval	MBPP	MT-Bench	Overall
DFlash-2	5.68	5.51	4.79	5.02	4.24	5.05
DFlash-2 + SAS	**6.48**	**5.96**	**5.20**	**5.38**	**4.59**	**5.52**
Gain (%)	+13.99	+8.32	+8.55	+7.06	+8.23	**+9.37**

Effect of the round partition. With the same learned parameters, $\pi = (2, 2, 3)$ refreshes the selected prefix before position five, whereas $\pi = (2, 5)$ continues within the current round. The first four positions have similar conditional acceptance, but the additional prefix update raises a_i at positions 5–7 by 1.52–2.48 percentage points across the two decoding settings. This improves AL on every workload. Nevertheless, reducing J from 3 to 2 gives 1.65% higher overall throughput under sampling, while greedy throughput is nearly unchanged ($+0.11\%$). These operating points expose the tradeoff between stronger prefix conditioning and serial execution cost, which dynamic partition training makes adjustable without retraining.

5.3 Composition with Drafter Optimizations

We examine whether SAS remains useful when the drafter already uses stronger mechanisms for modeling token dependencies. We evaluate DFlash-2 (Inco AI, 2026), another hybrid drafter that combines dynamic convolution within its backbone with a lightweight selector for coherent candidate paths. On Qwen3.8-27B (Qwen Team, 2026), we compare the released DFlash-2 model with DFlash-2 + SAS. The composed drafter retains both mechanisms and applies a shared stack with $N = 5$ over $J = 3$ causal rounds. Both methods use the same target, tokenizer, $B = 8$ and greedy decoding. Each method completes 2,320 requests across the five public benchmarks.

Relative to DFlash-2, DFlash-2 + SAS increases overall AL by 9.37% and improves AL on every benchmark (Table 2). The gains across reasoning, code generation, and dialogue show that additional causal rounds can complement improvements to the predictions produced within each draft pass.

Table 3: Time breakdown for the joint ablation of execution specialization. Cells show milliseconds and shares (in parentheses). Coarse shares normalize draft, verification, and feedback times; fine shares normalize the draft total. The two breakdowns are measured separately.

	Draft–verify breakdown			Within-draft breakdown								
Execution specialization	Draft	Verify	Feedback	Input prep.	Row assembly	Backbone	Shallow readout	Candidate injection	Final readout	Prefix update	Other	Draft total
Disabled	3.570 (7.4%)	44.152 (92.0%)	0.272 (0.6%)	0.467 (12.2%)	0.105 (2.8%)	2.514 (65.7%)	0.286 (7.5%)	0.125 (3.3%)	0.248 (6.5%)	0.067 (1.8%)	0.015 (0.4%)	3.827 (100.0%)
Enabled	**2.903** (6.1%)	44.161 (93.3%)	0.271 (0.6%)	0.217 (7.5%)	0.069 (2.4%)	2.120 (73.5%)	0.210 (7.3%)	0.043 (1.5%)	0.192 (6.7%)	0.018 (0.6%)	0.015 (0.5%)	**2.884** (100.0%)

5.4 Ablation of Execution Specialization

We jointly ablate the compact round layouts and fused state handling of *execution specialization* (Section 4.3). Both variants use $N = 5$, $B = 8$, $\pi = (2, 2, 3)$, $K = 4$ and the same weights. The ablated variant uses padded layouts and separate tensor operations.

Table 3 presents coarse draft–verify and fine within-draft time breakdowns, measured separately with CUDA events after warmup. Coarse timings are medians of per-step means across groups, measured on 164 HumanEval prompts at concurrency 48 with nucleus sampling. Fine timings average 20 paired measurements on fixed synthetic inputs with batch size 12 and context length 1,024. Each breakdown reports component shares using its own normalization.

Draft–verify balance. Verification accounts for 92.0%–93.3% of the reported coarse component time. Execution specialization reduces drafting from 3.570 to 2.903 ms per step (-18.69%), lowering its share from 7.4% to 6.1%, while verification and feedback remain nearly unchanged.

Within-draft costs. The backbone is the largest draft component. Execution specialization reduces its time by 15.7%; input preparation, row assembly, and candidate injection together decrease by 52.8%; shallow/final readout and prefix updates decrease by 30.1%. These changes are consistent with compact round layouts removing inactive-row computation and fusion reducing repeated state handling. Because state handling shrinks more, the backbone's share rises from 65.7% to 73.5% despite lower absolute time. These changes describe the combined effect of execution specialization.

6 Limitations and Discussion

Our 1T-scale end-to-end evaluation focuses on MiMo-v2.5-Pro because limited GPU resources and lengthy training of target-specific drafters constrain broader coverage. SAS is most useful when $\tau_{\text{verify}} \gg \tau_{\text{draft}}$, since its additional rounds must repay their execution cost. We expect this imbalance in many trillion-parameter deployments, although target architecture, hardware, and serving load determine the available headroom.

Dynamic partition training makes the schedule selectable with shared weights: $\pi = (2, 2, 3)$ offers higher acceptance in our experiments, while $\pi = (2, 5)$ reduces serial rounds. Future work can adapt schedules online to serving load and further reduce state-handling overhead between rounds.

7 Conclusion

We study how additional draft computation can improve speculative decoding through depth and causal round scaling. SAS combines in-block draft, dynamic partition training, and execution specialization to make round scaling efficient and adjustable. On the 1T-parameter MiMo-v2.5-Pro under greedy decoding and sampling, overall throughput reaches up to $2.30\times$ AR, with gains of up to 9.88% over DFlash and 7.92% over DSpark. On Qwen3.8-27B, composition with DFlash-2 further improves overall acceptance length by 9.37%.

AI USE STATEMENT

The authors used generative AI tools for language polishing and LaTeX proofreading. All technical claims, equations, citations, source references, experimental numbers, and final wording were checked and approved by the authors.

REPRODUCIBILITY STATEMENT

Section 4 describes the algorithm, training objective, and execution design, with inference pseudocode in Appendix B. Training settings are summarized in Appendix A. Section 5.1 specifies the target model, hardware configuration, workloads, decoding settings, and measurement protocol. Additional experimental details and sensitivity analyses are provided in Appendices C–E.

REFERENCES

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for Medusa decoding. arXiv preprint arXiv:2402.05109, 2024.

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.

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

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. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. arXiv preprint arXiv:2607.05147, 2026.

Yunfei Cheng, Aonan Zhang, Xuanyu Zhang, Chong Wang, and Yi Wang. Recurrent drafter for fast speculative decoding in large language models. arXiv preprint arXiv:2403.09919, 2024.

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

DeepSeek-AI. DeepSeek-V3 technical report. arXiv preprint arXiv:2412.19437, 2024.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15706–15734. PMLR, 2024.

Ernst Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 136:210–271, 1909. doi: 10.1515/crll.1909.136.210.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 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.

Inco AI. DFlash 2: Keep drafting parallel. Inco AI technical blog, August 2026.

Maxime Labonne. Open-PerfectBlend. Hugging Face dataset, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023.

Jinze Li, Yixing Xu, Haiduo Huang, Xuanwu Yin, Dong Li, Edith Ngai, and Emad Barsoum. Gumiho: A hybrid architecture to prioritize early tokens in speculative decoding. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 2024a.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, pp. 136737–136756. Curran Associates, Inc., 2025b. doi: 10.52202/085713-4562.

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 *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

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.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 31131–31146. PMLR, 2024.

LLM-Core Xiaomi. MiMo-V2-Flash technical report. arXiv preprint arXiv:2601.02780, 2026.

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

Qwen Team. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL `https://qwen.ai/blog?id=qwen3.8`.

Weijie Shi, Qiang Xu, Fan Deng, Yaguang Wu, Jiarun Liu, Yehong Xu, Hao Chen, Jia Zhu, Jiajie Xu, Xiangjun Huang, et al. SpecBlock: Block-iterative speculative decoding with dynamic tree drafting. arXiv preprint arXiv:2605.07243, 2026.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.456.

Xiaomi MiMo Team. MiMo-V2.5-Pro. Hugging Face model card, 2026. URL `https://huggingface.co/XiaomiMiMo/MiMo-V2.5-Pro`.

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.

Z.ai. GLM-5.2: Built for long-horizon tasks. Z.ai technical blog, June 2026.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.607.

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. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023. doi: 10.52202/075280-2020.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations*, volume 2024, pp. 32011–32050, 2024.

A Training Setup

We train the drafter in BF16 with Muon (Liu et al., 2025), using AdamW for trainable normalization parameters. The target model, token embeddings, and final LM head remain frozen. We use a global batch size of 64, peak and minimum learning rates of 10^{-3} and 10^{-5}, and gradient clipping at 1.0. The shallow loss weight is $\lambda_{\mathrm{sh}} = 0.3$. Padding and unsupervised positions are excluded from the loss. Dynamic partition training samples uniformly from

$$\Pi = \{(2,2,3), (2,3,2), (3,2,2), (2,5), (3,4), (4,3), (5,2)\}.$$

B Inference Algorithm

Algorithm 1 summarizes one speculative step using the notation of Section 4.1. Each round selects its leader and follower group before extending P_j; the target verifies the completed linear proposal once.

Algorithm 1: Inference with in-block draft.

Require: Verified context c; drafter $\mathcal{D}_{1:N}$; target $\mathcal{M}$; round groups $\mathcal{G}_{1:J}$; candidate width K
Ensure: Tokens committed by one speculative-decoding step

$P_0 \leftarrow [\,]$
for $j = 1$ to J **do**
  $s_j \leftarrow \min \mathcal{G}_j$; $\mathcal{F}_j \leftarrow \mathcal{G}_j \setminus \{s_j\}$
  Run layer 1 on the prefix, leader, and masked followers to obtain h_i
  $(\hat{x}_{s_j}^{(1)}, \ldots, \hat{x}_{s_j}^{(K)}) \leftarrow \mathrm{TopK}(\mathrm{ShHead}(h_{s_j}), K)$
  Initialize $h_{s_j}^{(k)}$ and mix $h_i^{(k)}$ using equation 4; retain fallback h_i
  Run layers $2{:}N$ on the prefix, leader, candidate slots, and follower branches
  Apply final normalization, LMHead, and Select to query rows only
  Select $x_{\mathcal{G}_j}$ from the resulting tokens using equation 5
  $P_j \leftarrow P_{j-1} \,\|\, x_{\mathcal{G}_j}$
end for
$(a, b) \leftarrow \mathrm{Verify}_{\mathcal{M}}(c, P_J)$ *accepted draft prefix and target bonus*
return $a \,\|\, [b]$

C Motivation Study Details

Experimental settings. All scaling configurations in Section 3 use Qwen3-8B and are trained on the same Open-PerfectBlend corpus (Labonne, 2024; Xu et al., 2024) for six epochs, with a maximum training sequence length of 4,096 tokens. Evaluation uses one NVIDIA H20 GPU with TP1, BF16, and concurrency 1. We use the first 50 prompts from each of MATH-500, HumanEval, and MT-Bench (first turn), greedy decoding (top-$k = 1$, top-$p = 1$), thinking enabled, and a 512-token generation cap. All configurations use $B = 8$.

Metric calculation. For Figure 1(b), let n_i count recorded verification events accepting at least i draft tokens, and let n_0 be the total event count. We pool these counts across workloads and estimate conditional acceptance as

$$\widehat{a}_i = \frac{n_i}{n_{i-1}}, \qquad i = 1, \ldots, m. \tag{9}$$

The corresponding histogram estimate of acceptance length is $1 + \sum_{i=1}^{m} n_i/n_0$, where the added one counts the target bonus.

Per-dataset results. All three Round-Scale configurations outperform the 5-layer reference baseline and the 10-/15-layer Depth-Scale configurations on all three datasets. The 7-round model achieves the highest AL on each dataset: 6.172 on MATH-500, 5.583 on HumanEval, and 4.858 on MT-Bench, improving over the reference by 16.3%, 19.7%, and 22.1%, respectively. On HumanEval, increasing depth to 10 or 15 layers instead raises AL from 4.663 to 4.913 ($+5.4\%$) or 4.962 ($+6.4\%$). Table 4 also includes separate reference measurements for DSpark, DFlash-2, and EAGLE-3, which yield lower AL than the multi-round configurations on these tasks.

Table 4: Per-dataset acceptance lengths on Qwen3-8B. The upper panel reports pooled client ratios (total output tokens / total verification steps) for the configurations in Figure 1(b); the lower panel reproduces separate reference measurements.

Method	MATH-500	HumanEval	MT-Bench
Reference baseline (5 layers, 1 round)	5.307	4.663	3.978
Depth-Scale (10 layers, 1 round)	5.544	4.913	4.067
Depth-Scale (15 layers, 1 round)	5.601	4.962	4.186
Round-Scale (5 layers, 3 rounds)	5.974	5.401	4.635
Round-Scale (5 layers, 4 rounds)	6.124	5.507	4.668
Round-Scale (5 layers, 7 rounds)	6.172	5.583	4.858
DFlash-2 (5 layers, 1 round + causal head)	5.818	5.190	4.222
DSpark (5 layers, 1 round + causal head)	5.500	4.931	3.982
EAGLE-3 (1 layer, 7 rounds)	2.059	2.137	2.013

D CONCURRENCY SENSITIVITY

We vary total serving concurrency from 4 to 48 for DFlash and SAS with $\pi = (2, 2, 3)$, keeping the draft weights, 32-GPU deployment, six workloads, and decoding settings fixed. Client concurrency and the server request limit change together. Table 5 reports overall throughput and request-mean AL.

Table 5: Concurrency sensitivity of DFlash and SAS with $\pi = (2, 2, 3)$. Throughput (tok/s) aggregates steady-state output tokens and elapsed time across all six workloads; AL is the request mean. C denotes total serving concurrency.

C	DFlash		SAS		Gain (%)	$\Delta\mathrm{AL}_{\mathrm{req}}$
	Tok/s	AL	Tok/s	AL		
Greedy						
4	366.4	4.933	399.6	5.426	+9.05	+0.4934
8	730.4	4.947	788.5	5.433	+7.95	+0.4864
16	1226.9	4.946	1323.6	5.446	+7.88	+0.4995
32	2140.7	4.932	2305.6	5.437	+7.70	+0.5049
48	2831.3	4.952	3071.1	5.438	+8.47	+0.4858
Sampling						
4	335.6	4.757	363.6	5.217	+8.33	+0.4608
8	673.0	4.793	724.7	5.240	+7.69	+0.4469
16	1161.7	4.771	1223.6	5.221	+5.33	+0.4503
32	1987.4	4.768	2084.2	5.217	+4.87	+0.4485
48	2667.6	4.771	2883.9	5.237	+8.11	+0.4662

SAS achieves higher overall throughput at every tested concurrency, with gains of 4.87%–8.33% under nucleus sampling and 7.70%–9.05% under greedy decoding. Its AL advantage remains within 0.447–0.466 tokens for sampling and 0.486–0.505 tokens for greedy decoding. Longer accepted prefixes thus accompany throughput gains across the tested serving loads. The relative throughput gain varies more than the acceptance advantage and is not monotonic in concurrency. Under greedy decoding, it is 9.05% at concurrency 4, 7.70% at concurrency 32, and 8.47% at concurrency 48. Under sampling, it is 4.87% at concurrency 32 and 8.11% at concurrency 48.

E CANDIDATE WIDTH SENSITIVITY

We vary candidate width $K \in \{1, 4, 8\}$ with fixed weights θ, $N = 5$, $B = 8$, and $\pi = (2, 2, 3)$ at concurrency 48. All widths share a common execution configuration. Performance measurements are repeated twice per configuration. $K = 4$ and $K = 8$ are measured on the same node allocation; $K = 1$ is measured separately.

Table 6: Candidate-width sensitivity: coverage (Hit@K, %), overall steady-state throughput (tok/s), and request-mean acceptance length (AL). Coverage uses shared $K = 8$ diagnostic trajectories.

K	Greedy			Sampling		
	Hit@K	Tok/s	AL	Hit@K	Tok/s	AL
1	84.87	3113.4	5.29	83.38	2759.2	5.09
4	95.24	3110.5	5.43	94.58	2930.5	5.22
8	96.82	3054.8	5.43	96.37	2944.3	5.24

Candidate coverage. Hit@K measures how often the shallow top-K candidates contain the full-depth leader prediction, pooling counts over all valid leader events. We compute Hit@1/4/8 on shared $K = 8$ diagnostic trajectories, separately from the performance runs. Table 6 shows that four candidates raise coverage from 83.38%–84.87% to 94.58%–95.24%. Increasing K to 8 adds only 1.58–1.78 percentage points. Thus, four shallow candidates already allow candidate hedge to select a matching branch in the large majority of rounds.

Width sensitivity. Most of the observed AL increase occurs between $K = 1$ and 4; the matched comparison between $K = 4$ and 8 adds only 0.12%–0.32%. Throughput depends on the decoding setting: $K = 8$ is 1.79% slower than $K = 4$ under greedy decoding and 0.47% faster under sampling. Together with the high Hit@4 coverage, these results support $K = 4$ as a practical tradeoff between candidate coverage and execution cost.