EFFICIENT SPECULATIVE DECODING WITH A FIXED-COST PARALLEL DRAFTER

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates autoregressive inference by verifying multiple draft tokens in a single target forward pass. However, as the context grows, existing state-of-the-art drafters become increasingly expensive, eroding the very efficiency advantage they are designed to provide. We argue that this scaling is unnecessary. A standalone language model must grow with its prefix because it is solely responsible for every token it produces. A drafter, by contrast, only proposes candidates; the target catches and corrects every error before any token is committed. The drafter's decoding cost can therefore be made entirely independent of the prefix length. We introduce LONGSPARK, a block-diffusion drafter that achieves this by extracting fixed-size, multiscale views from the target's verification pass, thereby eliminating the need for a growing persistent state. Extensive evaluations demonstrate that LONGSPARK achieves state-of-the-art end-to-end efficiency across multiple model scales and realistic serving conditions. Notably, it delivers the lowest time-per-output-token on long-context tasks while reducing the drafter's context state by several orders of magnitude.

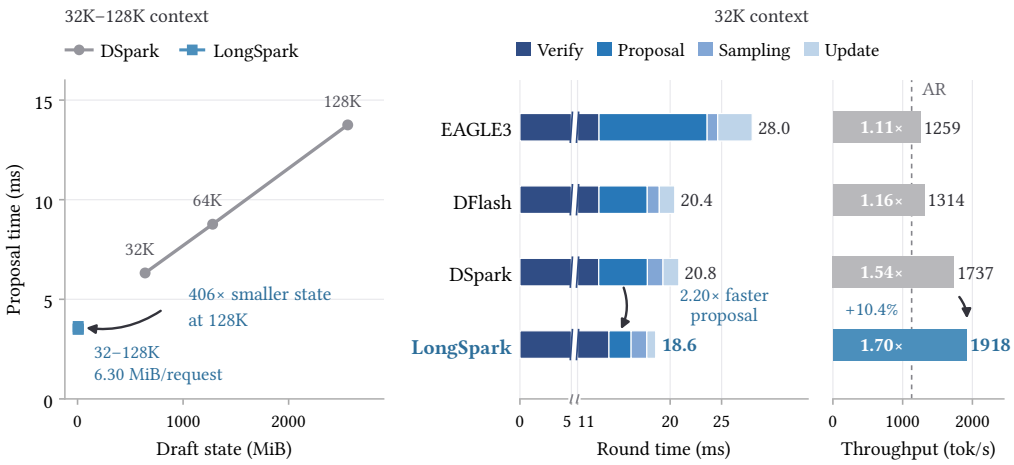

Figure 1: LONGSPARK: Scalable speculative decoding via fixed-cost drafting. Its $\mathcal{O}(1)$ drafting cost keeps proposal latency nearly constant as the prefix grows, with 406× smaller drafter context state than DSpark at 128K tokens (left). Lower drafting overhead improves end-to-end throughput on LongSpec (right). Both panels use Qwen3-8B with concurrency 16.

1 INTRODUCTION

Speculative decoding (Leviathan et al., 2023) accelerates autoregressive inference by delegating token proposal to a lightweight drafter, which allows a target model to verify multiple candidates in a single forward pass. The core premise of this acceleration is that the drafter must be significantly cheaper than the target for the speedup to be meaningful. However, the cost of producing draft candidates typically scales with the context length, eroding the efficiency advantage exactly when it is most needed.

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.

This scaling bottleneck is systemic across current drafting architectures. Autoregressive drafters maintain KV caches that grow linearly with the prefix; while reusing target representations (Li et al., 2024; 2025) can strengthen proposals, it does not eliminate this growing state. Windowed alternatives (Yang et al., 2026) bound their own state but still require full-prefix traversals of the target's context. Even advanced block-diffusion drafters, which predict multiple positions jointly (Chen et al., 2026; Nguyen et al., 2026), typically inject target features from the entire prefix into every draft layer (Chen et al., 2026; Cheng et al., 2026; Huang et al., 2026; Zhang et al., 2026). In each case, the drafter inherits the $\mathcal{O}(t)$ scaling behavior of the target model, transforming the drafter from a lightweight accelerator into a scaling bottleneck in its own right.

But a drafter need not scale this way. A standalone language model must represent the full prefix faithfully because it bears sole responsibility for every token it produces; any information lost from the prefix becomes an error. A drafter, by contrast, operates under a different contract: the target verifies every proposal before any token is committed, catching and correcting errors via rejection sampling. A wrong proposal merely adds a decoding round, never an incorrect output. The drafter can therefore work from a compressed, lossy view of the prefix.

This observation shifts the fundamental design objective for drafting: the true measure of efficiency is the number of accepted tokens produced relative to the drafting overhead. When a drafter maintains sufficient context to generate useful proposals at a constant cost, its per-round overhead decouples from the prefix length—a principle we term *fixed-cost drafting*.

We achieve this with LONGSPARK, a block-diffusion drafter. It extracts all context from the target's verification pass as fixed-size, multiscale views: a single-position anchor at the decoding boundary, a short window of recent token-level detail, and a global context summary that compresses the entire prefix into a fixed number of entries. By integrating these views into a parallel block-prediction framework, LONGSPARK ensures that target verification is the only full-prefix traversal in each round. Every other operation—from context extraction to iterative refinement—remains strictly $O(1)$ relative to the confirmed prefix.

We evaluate LONGSPARK across reasoning, coding, and dialogue benchmarks at multiple model scales, confirming that it achieves the highest end-to-end throughput, delivering 1.88× to 2.13× speedups that grow with target size. Indeed, LONGSPARK outperforms state-of-the-art drafters in throughput even when they achieve longer accepted lengths, validating that strictly bounded overhead is more critical for overall latency than marginal gains in proposal quality. Across three long-context benchmarks up to 128K tokens and all three target scales, LONGSPARK delivers the lowest mean time per output token while reducing drafter state by several orders of magnitude.

The contributions of this work are:

- We identify an asymmetry between drafter and target: because verification guarantees output correctness, the drafter's decoding cost need not scale with the prefix length. We formalize this as *fixed-cost drafting*.
- We introduce LONGSPARK, a block-diffusion drafter that extracts all of its context as fixed-size, multiscale views of the target's verification pass, making every additional per-round operation $O(1)$ with respect to the prefix length.
- We empirically validate fixed-cost drafting across diverse benchmarks and model scales, demonstrating that LONGSPARK improves end-to-end throughput while reducing drafter memory overhead by several orders of magnitude.

2 LONGSPARK: FIXED-COST SPECULATIVE DRAFTING

Fixed-cost drafting requires two ingredients: a context interface that extracts bounded-size views from an unbounded prefix, and a proposal model that produces competitive drafts from these views alone. We formalize both below, after establishing the necessary background (§ 2.1). The context interface (§ 2.2) and proposal model (§ 2.3) are followed by training, inference, and complexity analysis (§ 2.4).

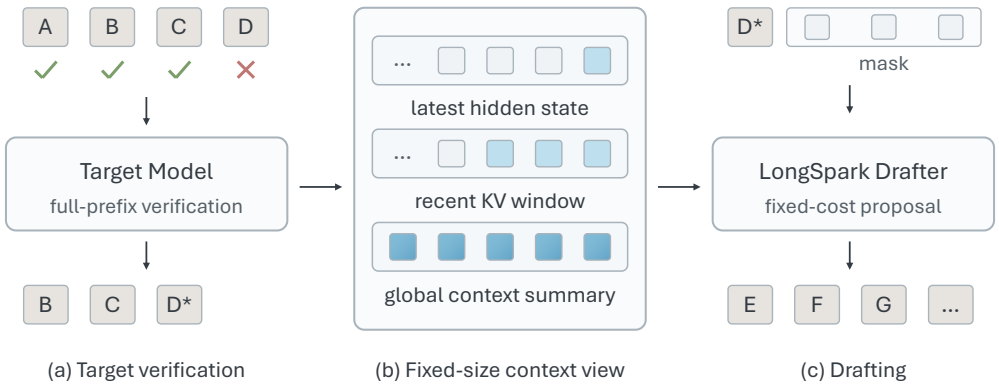

Figure 2: One decoding round of LONGSPARK. (a) The target verifies the proposed tokens. (b) Three fixed-size context views are extracted from the target's state: a boundary state, a recent KV window, and a global context summary. (c) The drafter uses these views to propose the next token block at a cost independent of prefix length.

2.1 PRELIMINARIES

Speculative decoding. Speculative decoding (Leviathan et al., 2023) pairs a target model with a lightweight drafter. At the start of a round, the target has cached $x_{1:t}$ but has yet to process its latest committed token $\mathtt{a} = x_{t+1}$. Given the committed prefix $s = (x_{1:t}, \mathtt{a})$, the drafter proposes B candidates $y_{1:B}$ by sampling from distributions $q_i(\cdot) = q(\cdot \mid s, y_{<i})$. The target verifies these by processing $[\mathtt{a}, y_{1:B}]$ in a single forward pass to compute $p_i(\cdot) = p(\cdot \mid s, y_{<i})$. Candidates are tested in order, accepting y_i with probability

$$\alpha_i = \min\left(1, \frac{p_i(y_i)}{q_i(y_i)}\right). \tag{1}$$

At the first rejected position i, we discard $y_{i:B}$ and sample a correction token from

$$p_i^{\text{corr}}(v) = \frac{[p_i(v) - q_i(v)]_+}{\sum_u [p_i(u) - q_i(u)]_+}, \qquad [z]_+ = \max(z, 0). \tag{2}$$

Each round adds one token beyond the accepted candidates: a correction token upon rejection, or a sample from $p(\cdot \mid s, y_{1:B})$ when all B candidates are accepted. Accepting r candidates therefore advances the prefix by $r + 1$ tokens while preserving the target distribution. With mean drafting and verification latencies T_{draft} and T_{verify} and mean advancement $\tau = \mathbb{E}[r + 1]$, the average decoding time per token is modeled as

$$L = \frac{T_{\text{draft}} + T_{\text{verify}}}{\tau}. \tag{3}$$

Block-diffusion drafting. Single-step block-diffusion drafters such as DFlash (Chen et al., 2026) reduce T_{draft} by predicting a block of proposal distributions in one denoising pass. Let $C(s)$ denote context features extracted from the target and $\boldsymbol{u}_{1:B}$ denote input representations for the proposal slots, such as mask embeddings or available committed-token embeddings. A bidirectional draft network F_θ produces base logits jointly:

$$\boldsymbol{z}_{1:B}^{(0)} = F_\theta(\boldsymbol{u}_{1:B}; C(s)), \qquad q_i^{(0)} = \text{softmax}(z_i^{(0)}). \tag{4}$$

Sampling each position from these distributions yields the factorized proposal $q^{(0)}(y_{1:B} \mid s) = \prod_{i=1}^{B} q_i^{(0)}(y_i \mid s)$. The backbone mixes information across positions through bidirectional attention, while tokens are sampled independently from its outputs. Semi-autoregressive variants (Cheng et al., 2026) add lightweight token dependencies after this parallel pass. The cost of this pass still grows with prefix length when attention spans the full prefix. The remaining question is whether a fixed-size context view $C(s)$ can retain competitive proposal quality.

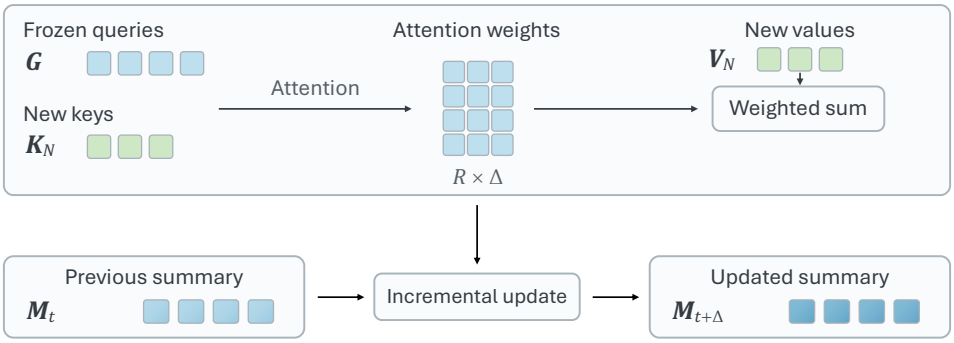

Figure 3: Incremental global context summary during inference. The summary is initialized over the full prefix during target prefill, then updated using only newly retained KV rows N ($\Delta = |N| \leq B+1$).

2.2 Fixed-Cost Context Interface

We instantiate $C(s)$ with three fixed-size views that capture the target's cached state at complementary temporal scales. A *boundary state* anchors the current generation point, fusing target hidden representations $\{\boldsymbol{h}_t^\ell\}_{\ell\in\mathcal{L}}$ at the decoding boundary into a single conditioning vector $\boldsymbol{c}_t$ via learned projection. A *recent KV window* of W entries per selected layer preserves token-level detail in the local neighborhood. A *global context summary* compresses the entire prefix into R entries per layer and head. All three are extracted from a subset $\mathcal{L}$ of evenly spaced target layers (Appendix B.1) and involve only constant-size reads and transformations.

Global context summary. The summary must compress the entire prefix into a fixed number of entries while remaining incrementally updatable as new tokens are committed. To achieve this, each selected layer and attention head maintains a bank of R learned summary queries $\boldsymbol{G} \in \mathbb{R}^{R\times d}$, which are independent of the prefix and held fixed during inference (Appendix B.2). For a head of dimension d and corresponding target keys and values $\boldsymbol{K}_{1:t}$ and $\boldsymbol{V}_{1:t}$, the summary values are computed as

$$\boldsymbol{M}_t = \mathrm{Attn}(\boldsymbol{G}, \boldsymbol{K}_{1:t}, \boldsymbol{V}_{1:t}). \tag{5}$$

Because $\boldsymbol{G}$ remains fixed during inference, historical attention contributions remain valid as the context grows, and only the newly retained target KV rows need to be incorporated. After verification, the summary absorbs at most $B+1$ new rows (Figure 3), keeping update cost and stored state independent of t. The drafter reads $(\boldsymbol{G}, \boldsymbol{M}_t)$ as summary keys and values, respectively: the same queries that produced the summary now serve as retrieval keys for the draft layers. Appendix A.2 gives the incremental update algorithm.

2.3 Proposal Model

We instantiate the parallel backbone in (4) with the fixed-cost interface from § 2.2, then introduce token-level dependencies through a lightweight sequential correction. The result is a D-layer Transformer that processes all B draft positions in parallel, followed by a per-position Markov correction that conditions each token on its predecessor.

The backbone operates on B input slots, where the first encodes the committed token $\mathtt{a}$ and the remaining $B-1$ employ mask-token embeddings. Their embeddings are combined with within-block positional encodings and the boundary state $\boldsymbol{c}_t$ to initialize the draft representations. The output of slot i supplies the base logits for proposal y_i, for $i = 1, \ldots, B$ (Appendix A.1).

These representations are processed by $D = |\mathcal{L}|$ Transformer layers, where each draft layer j is paired with a corresponding selected target layer $\ell(j)$, from which it reads the recent window and global summary. At each layer, attention is computed over a concatenated context:

$$\begin{aligned} \boldsymbol{K} &= [\boldsymbol{K}^{\mathrm{b}};\ \boldsymbol{K}^{\mathrm{w}};\ \boldsymbol{G}], \\ \boldsymbol{V} &= [\boldsymbol{V}^{\mathrm{b}};\ \boldsymbol{V}^{\mathrm{w}};\ \boldsymbol{M}_t], \end{aligned} \tag{6}$$

where superscripts b and w denote the proposal block and recent window, respectively; layer and head indices are omitted for readability. Consequently, all block positions attend to one another through a fixed-size context of at most $B+W+R$ entries per head, ensuring the computation remains independent of the prefix length.

Following the D draft layers, the frozen Target normalization and vocabulary head generate base logits $z_i^{(0)}$ for all B positions in parallel. These are then refined by a low-rank sequential correction (Cheng et al., 2026) that conditions each position on the token sampled at its predecessor. With $y_0 = \texttt{a}$, a correction embedding $\boldsymbol{E}_\mathrm{c}$, and an output mapping $\boldsymbol{W}_\mathrm{c}$, the final logits are given by:

$$z_i = z_i^{(0)} + \boldsymbol{W}_\mathrm{c}\boldsymbol{E}_\mathrm{c}(y_{i-1}), \qquad i = 1, \dots, B. \tag{7}$$

The resulting proposal factorizes as

$$q(y_{1:B} \mid s) = \prod_{i=1}^{B} q_i(y_i \mid s, y_{i-1}), \qquad q_i = \mathrm{softmax}(z_i), \tag{8}$$

with $y_0 = \texttt{a}$. These corrected distributions are used for both sampling and verification in (1)–(2). Only the correction and sampling occur sequentially; the Transformer backbone and base vocabulary projection remain fully parallel.

2.4 Training and Inference

Training. All Target parameters, including the token embedding, normalization, and vocabulary head, remain frozen. We jointly train the draft network, boundary fusion and input mappings, summary queries, and the DSpark-style sequential correction (Cheng et al., 2026). The full-vocabulary distributions p_i and q_i are evaluated under teacher forcing: the target sees the ground-truth prefix $x_{1:t+i}$, and the correction in (7) uses the predecessor x_{t+i}. For ground-truth token x_{t+1+i}, the weighted loss at position i is

$$\mathcal{J}_i = w_i\Big[\lambda\big[-\log q_i(x_{t+1+i})\big] + (1-\lambda)\|q_i - p_i\|_1\Big], \tag{9}$$

where $\lambda = 0.1$ and $w_i = \exp(-(i-1)/4)$. We sum these losses over valid training positions and normalize by the sum of their weights. At inference, the correction instead uses the previously sampled token.

Inference. Verification follows § 2.1, using the corrected proposal distributions from (8). When r candidates are accepted, the prefix advances by $r+1$ positions, and the committed token a along with $y_{1:r}$ enter the Target KV cache. This update naturally drives the incremental maintenance of the interface: their KV rows update the global context summary and advance the recent window, while the hidden states at the new boundary yield the updated $\boldsymbol{c}_t$. The process concludes by setting the correction or bonus token as the next committed token a and discarding all transient draft KV, leaving the updated interface ready for the next round.

Fixed-cost complexity. For fixed model dimensions, each draft layer attends to B block positions, W recent tokens, and R summary entries, costing $O(B^2 + BW + BR)$ per round. Full-prefix draft attention instead costs $O(B^2+Bt)$. Interface maintenance reads a constant-size boundary and window and incorporates at most $B+1$ newly retained KV rows into the summary. With H attention heads of dimension d, the summary occupies $O(|\mathcal{L}|HRd)$ persistent state; the recent window is read directly from the target cache, and block KV is transient. Both the drafter's additional state and its per-round computation are therefore independent of t. Full-prefix processing is confined to target verification, with the global summary initialized once during prefill.

3 Experiments

We organize the evaluation around three questions. Does a fixed-cost context interface outperform drafters that read the entire prefix across model scales, tasks, and serving loads (§ 3.2)? Does its advantage persist as contexts lengthen, with drafting cost independent of the prefix length (§ 3.3)? And how much does each context component contribute (§ 3.4)?

Table 1: Speculative decoding across model scales and tasks. Entries report accepted length τ, with throughput speedup over autoregressive decoding in parentheses. Avg. denotes the mean across eight benchmarks; bold marks the highest speedup.

Target	Drafter	Math			Code			Chat		Overall
		GSM8K	MATH-500	AIME25	MBPP	HumanEval	LCB	MT-Bench	Alpaca	Avg.
Q3-4B	EAGLE-3	5.15 (1.36)	4.60 (1.35)	3.84 (1.24)	3.69 (1.03)	4.15 (1.17)	3.77 (1.14)	2.42 (0.72)	2.26 (0.69)	3.74 (1.09)
	DFlash	5.37 (1.81)	4.89 (1.89)	4.01 (1.72)	4.40 (1.55)	4.74 (1.72)	4.23 (1.61)	3.05 (1.18)	2.94 (1.14)	4.20 (1.58)
	DSpark	6.10 (1.99)	5.72 (2.14)	4.92 (2.05)	5.14 (1.73)	5.44 (1.91)	4.92 (1.79)	3.65 (1.36)	3.55 (1.32)	4.93 (1.79)
	LongSpark	5.90 (**2.08**)	5.52 (**2.28**)	4.67 (**2.17**)	5.01 (**1.84**)	5.12 (**1.97**)	4.63 (**1.86**)	3.52 (**1.46**)	3.46 (**1.42**)	4.73 (**1.88**)
Q3-8B	EAGLE-3	5.26 (1.49)	4.75 (1.50)	4.02 (1.37)	3.92 (1.16)	4.32 (1.30)	4.17 (1.28)	2.67 (0.85)	2.54 (0.82)	3.96 (1.22)
	DFlash	5.36 (1.96)	4.91 (2.02)	4.07 (1.83)	4.36 (1.66)	4.67 (1.82)	4.44 (1.68)	3.09 (1.28)	3.00 (1.24)	4.24 (1.69)
	DSpark	6.17 (2.17)	5.80 (2.32)	4.99 (2.19)	5.18 (1.89)	5.48 (2.06)	5.09 (1.84)	3.69 (1.47)	3.59 (1.42)	5.00 (1.92)
	LongSpark	5.93 (**2.24**)	5.57 (**2.42**)	4.72 (**2.27**)	5.03 (**1.98**)	5.16 (**2.11**)	4.83 (**1.88**)	3.53 (**1.54**)	3.46 (**1.49**)	4.78 (**1.99**)
Q3-14B	EAGLE-3	5.22 (1.69)	4.62 (1.63)	3.82 (1.44)	3.81 (1.29)	4.11 (1.41)	4.01 (1.35)	2.61 (0.93)	2.49 (0.90)	3.84 (1.33)
	DFlash	5.37 (2.16)	4.86 (2.17)	4.01 (1.91)	4.45 (1.85)	4.57 (1.94)	4.35 (1.72)	3.09 (1.38)	2.96 (1.33)	4.21 (1.81)
	DSpark	6.19 (2.43)	5.77 (2.53)	4.93 (2.32)	5.25 (2.12)	5.41 (2.24)	4.97 (1.88)	3.67 (1.61)	3.56 (1.56)	4.97 (2.09)
	LongSpark	6.03 (**2.47**)	5.63 (**2.61**)	4.76 (**2.37**)	5.14 (**2.18**)	5.15 (**2.25**)	4.78 (**1.90**)	3.57 (**1.65**)	3.48 (**1.60**)	4.82 (**2.13**)

3.1 Experimental Setup

Models and baselines. To demonstrate the generalizability and scalability of LongSpark, we evaluate it across three Target scales: Qwen3-4B, 8B, and 14B. We compare our approach against a suite of representative baselines, including the autoregressive EAGLE-3 (Li et al., 2025) and state-of-the-art parallel drafters DFlash (Chen et al., 2026) and DSpark (Cheng et al., 2026), alongside vanilla autoregressive decoding.

Training data. We train LongSpark on Open-PerfectBlend (Labonne, 2024), a high-quality instruction-tuning dataset of approximately 1.42 million samples, for 10 epochs. We use only the prompts and regenerate all responses with the corresponding Target model, ensuring the drafter learns the exact distribution it is intended to accelerate. Training details are provided in Appendix B.2.

Evaluation benchmarks. We subject LongSpark to a comprehensive set of stress tests across eight benchmarks spanning three primary domains: mathematical reasoning, i.e., GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), and AIME25 (MAA, 2025), code generation, i.e., MBPP (Austin et al., 2021), HumanEval (Chen et al., 2021), and LiveCodeBench (LCB) (Jain et al., 2025), and open-ended dialogue, i.e., MT-Bench (Zheng et al., 2023) and Alpaca (Taori et al., 2023). The main long-context evaluation covers continuation tasks from LongSpec's training corpus (Yang et al., 2026) at 32K, our curated CodeSpan at 64K, and LongSWE-Bench (LSWE) (Rando et al., 2025) at 128K. Details of the long-context datasets are provided in Appendix C.1.

Evaluation settings and metrics. We use a target–drafter (TD) disaggregated framework: speculative methods add one dedicated drafter GPU to the Target GPUs used by autoregressive decoding. Main-text results use $T = 1$, thinking mode disabled, and seven draft tokens per verification round for speculative methods. We report throughput speedup over autoregressive decoding and average accepted length τ, with output throughput (TPS) and time per output token (TPOT) for long-context workloads. Detailed metric definitions are provided in Appendix C.2.

3.2 Overall Performance

We first evaluate end-to-end efficiency across model scales and task domains, using target TP1, concurrency 32, and up to 2,048 generated tokens per request. As shown in Table 1, LongSpark achieves the highest average throughput at every scale, delivering 1.88×, 1.99×, and 2.13× speedups from 4B to 14B—with the speedup growing as the target scales. Crucially, it keeps this lead even when DSpark attains longer accepted lengths. This separation exposes the central tension of speculative decoding: end-to-end efficiency depends not on proposal quality alone, but on the balance between accepted tokens and the overhead of producing them. The same ranking holds under greedy

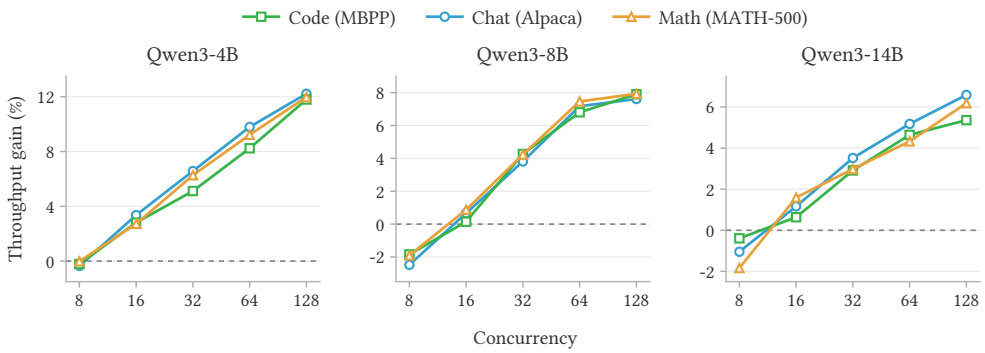

Figure 4: Higher concurrency increases serving load. As this load rises, LONGSPARK's throughput advantage over DSpark grows.

Table 2: Long-context decoding on Qwen3-8B. We report TPS (tokens/s) and TPOT (ms/token), averaged over ten seeds. Bold marks the best mean.

Method	LongSpec 32K		CodeSpan 64K		LSWE 128K	
	TPS ↑	TPOT ↓	TPS ↑	TPOT ↓	TPS ↑	TPOT ↓
Vanilla	1128.9	14.0	884.4	17.8	160.5	97.2
EAGLE-3	1258.5	12.5	1071.8	14.7	158.7	97.4
DFlash	1313.7	12.0	919.1	17.2	163.6	94.6
DSpark	1736.6	9.0	1217.3	13.0	190.9	81.4
LONGSPARK	**1917.6**	**8.2**	**1571.8**	**10.0**	**212.7**	**74.3**

decoding, confirming that the advantage of a fixed-cost drafter is structural rather than sampling-dependent (Appendix D.1).

High-concurrency performance. To examine how the advantage scales with serving load, we vary concurrency from 8 to 128 on Alpaca, MBPP, and MATH-500. As Figure 4 shows, all three target scales exhibit the same pattern: LONGSPARK is marginally behind DSpark at low concurrency, then overtakes it and pulls away steadily as load rises, leading by a clear advantage at concurrency 128. The gap widens on every benchmark despite shorter accepted lengths, revealing that the value of low-overhead drafting compounds under high serving load. Appendix D.4 provides additional concurrency results.

3.3 SCALING WITH CONTEXT LENGTH

Long-context performance. We evaluate contexts from 32K to 128K using Qwen3-8B with target TP4, concurrency 16, and up to 8,192 generated tokens per request. Long-context evaluation uses fixed NTK scaling (bloc97, 2023) with $\alpha = 4$. As shown in Table 2, LONGSPARK achieves the highest mean throughput and lowest mean TPOT across all three workloads. On CodeSpan at 64K, its mean TPS exceeds that of DSpark, the strongest competing method, by 29.1%, while reducing mean TPOT by 23.1%. Results for Qwen3-4B and 14B are provided in Appendix D.2.

Drafting cost and memory. The throughput gains of LONGSPARK stem from its lean drafting overhead. Figure 1 (right) decomposes a decoding round on LongSpec with Qwen3-8B into proposal and verification. Target verification dominates the round, and LONGSPARK nearly halves the proposal component relative to DSpark. Because the round is bounded by verification, this reduction lowers total round latency only modestly; its value lies in repetition, as the saving accumulates over the thousands of rounds needed to generate a long response.

The context-length sweep in Figure 1 (left) confirms that the fixed-size interface decouples the drafter's decoding cost from the prefix length, allowing the advantage to grow with context rather

(a) Accepted length τ.

Variant	Math	Code	Chat	Avg.
LONGSPARK	**5.33**	**4.91**	**3.40**	**4.69**
w/o global summary	4.99	4.50	3.17	4.35
w/o boundary state	4.90	4.42	3.15	4.28
w/o recent KV window	4.17	3.82	2.86	3.71

(b) Training loss.

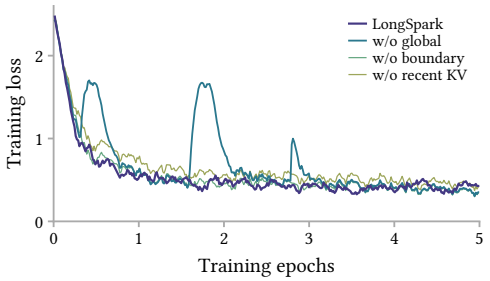

Figure 5: Ablation on Qwen3-8B. (a) Mean accepted length by domain; (b) Training loss curve.

than fade. As the context grows from 32K to 128K, DSpark's proposal latency rises from 6.3 to 13.8 ms and its effective context state grows from 640 to 2560 MiB, since both quantities are tied to the number of retained tokens. LONGSPARK holds both nearly constant: its proposal latency stays near 3.5 ms and its context state remains fixed at 6.3 MiB across the same range, amounting to a 406× reduction in drafter state at 128K.

These measurements isolate where the cost of drafting actually resides. Since target verification remains the only full-prefix operation in each round, the drafter's contribution can be made both small and constant. This is the structural distinction that separates LONGSPARK from prefix-scaling drafters: for them, a longer context translates directly into more work per proposal, whereas here the proposal cost is a fixed tax that never grows.

3.4 ABLATION STUDY

Figure 5(a) isolates each context component by removing it in turn, with up to 2,048 generated tokens per request. The four variants are trained for 5 epochs for a fair comparison. The recent KV window proves the most critical: without it, average accepted length falls from 4.69 to 3.71, confirming that token-level local detail is the primary driver of proposal quality. Removing the global summary or the boundary state causes smaller degradations, to 4.35 and 4.28 respectively, so both are secondary yet not redundant. The training dynamics in Figure 5(b) add a complementary view: removing the global summary produces pronounced transient loss spikes, whereas the full model descends smoothly. The summary thus contributes to training stability as well as proposal quality, since without it a bounded-context drafter has no reliable signal about the global prefix and must reconstruct it from local evidence alone.

Together, these results reveal a fundamental asymmetry between drafter and target. The target requires a full, precise state to verify; the drafter needs only enough orientation to propose. The recent window supplies immediate local grounding, while the global summary supplies a stable, low-resolution view of the entire prefix. Detailed long-context ablations are provided in Appendix D.3.

4 RELATED WORK

We first review the lossless verification framework that makes drafter quality a pure efficiency concern, then survey how autoregressive, recurrent, and block-diffusion drafters condition on the confirmed prefix.

Lossless Speculative Decoding Stern et al. (2018) first showed that a drafter can propose multiple future tokens at once and that a Target model can verify them, accepting the longest prefix consistent with exact greedy decoding. Speculative sampling extends this to stochastic generation: Leviathan et al. (2023) use rejection and residual correction to preserve the Target distribution exactly. Because the Target's verification preserves the output distribution regardless of proposal quality, the drafter determines only efficiency. How it conditions on the confirmed prefix is therefore a design choice that determines how its cost scales with the prefix length.

Autoregressive Drafters The standard construction uses a smaller autoregressive language model whose own KV cache grows with the prefix. Later methods strengthen proposals by incorporating Target representations: EAGLE and EAGLE-3 reuse Target hidden features (Li et al., 2024; 2025; Hui et al., 2026), and ReDrafter conditions a recurrent proposer on the Target state (Cheng et al., 2024). LongSpec bounds its own draft cache using windowed self-attention, but it still cross-attends to the Target's full KV history (Yang et al., 2026). Bounding the drafter's own cache does not remove this growing read, and these drafters still advance one position at a time.

Block-Parallel and Block-Diffusion Drafting Block-parallel methods take a different approach: instead of drafting tokens one by one, Xiao et al. (2024) and An et al. (2026) predict several future positions at once. Medusa attaches independent prediction heads to the Target for fixed-horizon parallel proposals (Cai et al., 2024; Wertheimer et al., 2024). Each head conditions only on the Target's last hidden state, making its cost prefix-independent but limiting it to single-position predictions that do not attend to the prefix. DFlash replaces these independent heads with a block-diffusion backbone that predicts all masked positions jointly and injects Target features from the entire confirmed prefix into every draft layer (Chen et al., 2026; Li et al., 2026). This yields more expressive proposals, but makes every draft layer process a context sequence that grows with the prefix. Orthrus shares the Target KV cache to eliminate duplicate storage, but its diffusion view still attends to the complete cache, leaving prefix-length context computation intact (Nguyen et al., 2026).

Subsequent work improves along other dimensions while retaining the same growing context interface. DFlare strengthens Target-to-drafter feature fusion (Zhang et al., 2026); DSpark, Domino, JetSpec and xPress restore causal dependence within the proposed block (Cheng et al., 2026; Hu et al., 2026; Huang et al., 2026; Wang et al., 2026a); DDTree organizes the position-wise distributions into a draft tree (Agrawal et al., 2025; Gao et al., 2026; Ringel & Romano, 2026; Wang et al., 2026b); and AdaFlash, like DSpark, schedules the proposal horizon with a dynamic confidence head (Cheng et al., 2026; Qian et al., 2026). Each of these improves feature conditioning, within-block coherence, candidate coverage, or the proposal horizon, yet each keeps the context that proposals read tied to the prefix. LONGSPARK changes this remaining axis: it is, to our knowledge, the first block-diffusion drafter to replace the growing context with a fixed-size, multiscale view of the Target state. Target verification is consequently the only full-prefix traversal in each round; all additional context processing, proposal computation, and drafter state remain fixed as the context grows.

5 CONCLUSION AND FUTURE DIRECTIONS

This work establishes that, during decoding, a speculative drafter's cost can be made entirely independent of the prefix length while maintaining competitive accepted lengths. Because verification guarantees output correctness, the drafter needs only enough context to produce useful proposals, and we show that this context can be extracted from the target's state at fixed cost.

The end-to-end results bear out the value of this fixed cost. LONGSPARK attains the highest throughput across model scales even though the strongest prefix-scaling drafter reaches slightly longer accepted lengths, showing that low proposal overhead outweighs marginal acceptance gains. This advantage grows with context length and serving concurrency because the drafter's cost remains constant while the baselines' costs grow. Ablations further show that a compact, lossy view of the prefix provides sufficient orientation for competitive proposals. Together, these results identify accepted tokens per unit of proposal overhead as the quantity drafting design should optimize.

Future directions include evaluating the fixed-cost interface with larger target models and longer context windows to assess its effectiveness as model scale and context demands increase. The interface developed here offers one viable realization of fixed-cost drafting; alternative interface designs and drafter architectures within this framework may further improve proposal quality while retaining prefix-independent cost. Beyond proposal design, exploring tree-structured and multi-drafter verification schemes could help increase the number of accepted tokens per round while preserving the $O(1)$ drafting cost. On the application side, reinforcement-learning rollouts offer a promising setting for these extensions, as many long trajectories run under tight latency budgets and a constant proposal overhead across trajectory length could compound throughput gains.

AI USE OF GENERATIVE MODELS

We used generative AI tools to assist with the research workflow: developing and debugging code, analysing experimental results, and preparing the manuscript, tables, and figures. The authors reviewed and verified all AI-assisted work and take full responsibility for the final content of this work, including all text, claims, and artefacts.

REFERENCES

Sudhanshu Agrawal, Risheek Garrepalli, Raghavv Goel, Christopher Lott, Fatih Porikli, and Mingu Lee. Structuring the future: Diffusion LLM speculative decoding via calibrated draft graphs. *arXiv preprint arXiv:2509.18085*, 2025.

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. PARD: Accelerating LLM inference with low-cost parallel draft model adaptation. In *The 14th International Conference on Learning Representations*, Rio de Janeiro, Brazil, 2026.

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 preprint arXiv:2108.07732*, 2021.

bloc97. NTK-Aware Scaled RoPE allows LLaMA models to have extended context length, 2023. URL https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/.

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, Vienna, Austria, 2024. PMLR.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, Seoul, South Korea, 2026.

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

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026.

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, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Zipeng Gao, Zhi Zheng, Qingrong Xia, Junda Lin, Ziwei Zhao, Tong Xu, Zhefeng Wang, and Enhong Chen. Unlocking parallelism in autoregressive language models via speculative decoding with progressive tree drafting. In *The 3rd Conference on Language Modeling*, San Francisco, California, United States, 2026.

Lanxiang Hu, Zhaoxiang Feng, Yulun Wu, Haoran Yuan, Yujie Zhao, Yu-Yang Qian, Bojun Wang, Peng Zhao, Daxin Jiang, Yibo Zhu, Tajana Rosing, and Hao Zhang. JetSpec: Breaking the scaling ceiling of speculative decoding with parallel tree drafting. *arXiv preprint arXiv:2606.18394*, 2026.

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.

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

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The 13th International Conference on Learning Representations*, Singapore, 2025.

Maxime Labonne. Open-PerfectBlend, 2024. URL https://huggingface.co/datasets/mlabonne/open-perfectblend.

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, Honolulu, Hawaii, United States, 2023. PMLR.

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. DiffuSpec: Unlocking diffusion language models for speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 20896–20910, San Diego, California, United States, 2026. Association for Computational Linguistics.

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, Vienna, Austria, 2024. PMLR.

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, San Diego, California, United States, 2025.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The 12th International Conference on Learning Representations*, Vienna, Austria, 2024.

MAA. American Invitational Mathematics Examination - AIME, 2025. URL https://maa.org/math-competitions/american-invitational-mathematics-examination-aime.

Chien Van Nguyen, Chaitra Hegde, Van Cuong Pham, Ryan A. Rossi, Franck Dernoncourt, and Thien Huu Nguyen. Orthrus: Memory-efficient parallel token generation via dual-view diffusion. *arXiv preprint arXiv:2605.12825*, 2026.

Yu-Yang Qian, Hao-Cong Wu, Chen Chen, Jiacheng Sun, Zhenhua Dong, Peng Zhao, and Zhi-Hua Zhou. AdaFlash: Adaptive speculative decoding via on-policy distilled diffusion drafters. *arXiv preprint arXiv:2607.19223*, 2026.

Stefano Rando, Luca Romani, Alessio Sampieri, Luca Franco, John Yang, Yuta Kyuragi, Fabio Galasso, and Tatsunori Hashimoto. LongCodeBench: Evaluating coding LLMs at 1M context windows. In *The 2nd Conference on Language Modeling*, Montreal, Canada, 2025.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. In *The 3rd Conference on Language Modeling*, San Francisco, California, United States, 2026.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, pp. 10107–10116, Montreal, Canada, 2018.

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

Zheng Wang, Davis Wertheimer, Yu Chin Fabian Lim, Mudhakar Srivatsa, Raghu K. Ganti, Minjia Zhang, and Naigang Wang. xPress: Parallel refinement for diffusion drafters in speculative decoding. *arXiv preprint arXiv:2608.02438*, 2026a.

Zheng Wang, Zhifan Ye, Qi Cheng, Yonggan Fu, Ziyan Wang, Feng Zhu, Haozhe Zhao, Jan Kautz, Pavlo Molchanov, Humphrey Shi, and Minjia Zhang. PRESTO: Prefix-aligned tree drafting for diffusion speculative decoding. *arXiv preprint arXiv:2607.22634*, 2026b.

Davis Wertheimer, Joshua Rosenkranz, Thomas Parnell, Sahil Suneja, Pavithra Ranganathan, Raghu Ganti, and Mudhakar Srivatsa. Accelerating production LLMs with combined token/embedding speculators. *arXiv preprint arXiv:2404.19124*, 2024.

Zilin Xiao, Hongming Zhang, Tao Ge, Siru Ouyang, Vicente Ordonez, and Dong Yu. ParallelSpec: Parallel drafter for efficient speculative decoding. *arXiv preprint arXiv:2410.05589*, 2024.

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. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1826–1844, San Diego, California, United States, 2026. Association for Computational Linguistics.

Jiebin Zhang, Zhenghan Yu, Song Liu, Eugene J. Yu, Zheng Li, Dawei Zhu, Jiangshan Duo, Weimin Xiong, Yifan Song, Guanghua Yu, Jianchen Zhu, and Sujian Li. DFlare: Scaling up draft capacity for block diffusion speculative decoding. *arXiv preprint arXiv:2606.02091*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, New Orleans, Louisiana, United States, 2023.

APPENDIX

A METHOD AND ALGORITHMIC DETAILS

This section details the proposal computation and the incremental maintenance of the global summary.

A.1 DRAFTER FORWARD PASS

We first describe how the drafter turns the three fixed-size context views into one block proposal.

Figure 6 summarizes one proposal step after the Target has cached $x_{1:t}$, with $\mathtt{a} = x_{t+1}$ as the next committed token. The drafter takes the selected-layer boundary states $\{\boldsymbol{h}_t^{\ell}\}_{\ell \in \mathcal{L}}$, recent Target KV windows, and global summary pairs $(\boldsymbol{G}, \boldsymbol{M}_t)$, all ordered by selected target layer.

```
def propose_block(committed_token, boundary_states, recent_windows, global_summaries):
    slots = [committed_token] + [MASK] * (B - 1)
    slot_inputs = target_embedding(slots) + block_positions(B)
    block_states = project_inputs(slot_inputs, fuse_boundary_states(boundary_states))

    # Parallel block pass
    for layer_idx, layer in enumerate(draft_layers):
        attention_sources = (global_summaries[layer_idx], recent_windows[layer_idx],
                             layer.local_kv(block_states))
        block_states = layer(block_states, attention_sources)
    base_logits = target_head(target_norm(block_states))

    # Sequential Markov correction
    previous_token = committed_token
    draft_tokens, proposal_distributions = [], []
    for slot_logits in base_logits:
        proposal_dist = softmax(slot_logits + markov_correction(previous_token))
        previous_token = sample(proposal_dist)
        draft_tokens.append(previous_token)
        proposal_distributions.append(proposal_dist)
    return draft_tokens, proposal_distributions
```

Figure 6: One block proposal by the LONGSPARK drafter.

Input slot 0 supplies the base logits for y_1. All slots pass through the draft layers in parallel, with one attention normalization across the three sources and bidirectional attention among local slots. The Target embedding, final normalization, and vocabulary head remain frozen. Local block KV is discarded after drafting; only the Markov correction and sampling proceed sequentially. The returned distributions are used in verification, after which retained Target KV rows update the summary as described next.

A.2 INCREMENTAL GLOBAL CONTEXT SUMMARY

We then derive the update that folds newly retained rows into the global summary at constant cost.

For a fixed summary query $\boldsymbol{g}$, let $s_j = \boldsymbol{g}^\top \boldsymbol{k}_j / \sqrt{d}$ be its attention score for Target key $\boldsymbol{k}_j$. For any nonempty set of KV rows A, define its summary value and log-normalizer as

$$\eta_A = \log \sum_{j \in A} \exp(s_j), \qquad \boldsymbol{m}_A = \sum_{j \in A} \exp(s_j - \eta_A) \boldsymbol{v}_j. \tag{10}$$

Let A contain the previously processed prefix and N the newly retained rows. Their attention statistics merge as

$$\begin{aligned} \eta &= \log\big(\exp(\eta_A) + \exp(\eta_N)\big), \\ \boldsymbol{m} &= \exp(\eta_A - \eta)\boldsymbol{m}_A + \exp(\eta_N - \eta)\boldsymbol{m}_N. \end{aligned} \tag{11}$$

The merged value $\boldsymbol{m}$ equals attention over $A \cup N$; an empty new set leaves the state unchanged. The log-sum-exp operations are evaluated with max subtraction for numerical stability. Applying this merge independently to all summary queries recovers (5) while storing only one value vector and one log-normalizer per query. Since the queries and historical Target KV remain fixed, only the statistics over N need to be computed in each round.

B Model Architecture and Training

This section records the architecture and training settings shared by all reported drafters.

B.1 Drafter Architecture

We specify the drafter depth, its context-view sizes, and the Target layers it reads.

LongSpark employs a lightweight decoder comprising 5 Qwen3-style blocks that operate in parallel to propose 7 tokens. To maintain architectural consistency, all drafters adhere to the standard Qwen3 hyperparameters for a given hidden dimension d (e.g., head dimension and intermediate width). The architecture is designed as an efficient distilled version of the Target, integrating a global context summary, a recent KV window, and a small local workspace. We extract context views from Target layers $\{0, 9, 18, 26, 35\}$ for Qwen3-4B and 8B, and $\{0, 10, 20, 29, 39\}$ for Qwen3-14B. Across all scales, the global context summary contains $R = 16$ entries per selected layer and attention head, and the recent KV window contains $W = 256$ entries per selected layer. We reuse the Target's embedding and final LM head to ensure vocabulary consistency.

B.2 Training Procedure

We list the data, objective, and optimization settings used to train each drafter.

We train the LongSpark drafters with the objective in (9) on the Open-PerfectBlend dataset (Labonne, 2024), which contains approximately 1.42M sessions. For each target scale, we regenerate the assistant responses in a non-thinking mode using the frozen Target model (temperature $= 0.7$, top-p $= 0.8$, top-k $= 20$). During training, we limit the total sequence length to 4096 tokens and randomly sample up to 512 anchors per sequence to optimize the distillation loss across multiple positions in each generated response.

Summary queries. Each selected layer and attention head has an independent bank of R summary queries, initialized from $\mathcal{N}(\mathbf{0}, d^{-1}\boldsymbol{I})$ and jointly optimized with the drafter under the same training objective, where d is the head dimension. The queries are shared across examples and remain fixed during inference, enabling the incremental summary updates in Appendix A.2.

The main drafters are trained for 10 epochs and the ablation variants for 5 epochs, all with a global batch size of 512. We use the AdamW optimizer with BF16 mixed-precision and FP32 master weights, a peak learning rate of 6×10^{-4}, and a cosine decay scheduler with a 4% linear warmup. For all model scales, we employ a data-parallel (DP) degree of 8 with gradient accumulation steps of 64.

C Evaluation Protocol and Dataset Details

This section documents the long-context workloads and the measurement protocol behind the reported numbers.

C.1 Long-Context Evaluation Datasets

We describe the composition of the three long-context workloads.

LongSpec. We construct continuation tasks from the code, arXiv, and book subsets of LongSpec's released training corpus (Yang et al., 2026), using 32,768-token inputs. These subsets form the LongSpec 32K evaluation workload. Our drafting-cost analysis uses a prefix from the code subset.

CodeSpan. CodeSpan comprises 32 source-code continuation examples drawn from 32 distinct files across 17 open-source projects, including LLVM, GCC, Linux, and PostgreSQL. We select sufficiently long files, with at most four files per project. The collection is predominantly C/C++ (28 files), with additional Python, TypeScript, and Yacc grammar files. Each example uses a contiguous prefix of a single file, without concatenation or repetition. Using the Qwen3-8B tokenizer, we choose the prefix length so that the complete input, including the continuation instruction and chat template, contains 65,536 tokens.

LongSWE-Bench. LongSWE-Bench (Rando et al., 2025) evaluates repository-level bug fixing with long code contexts. We select inputs that fit within a 128K total context budget while reserving space for up to 8,192 generated tokens. The same evaluation inputs and ordering are shared across methods and random seeds.

C.2 Measurement Protocol

We define how throughput, accepted length, and time per output token are measured.

Throughput and accepted length. We measure steady-state serving performance over a 120-second interval after a 30-second warm-up, continuously admitting new requests as others finish. Each method samples its own responses independently. Throughput counts all output tokens emitted during this interval. For each seed, τ is computed by dividing the number of emitted tokens after the first token by the number of verification request-rounds, including bonus or residual tokens and accounting for sequence termination. We take arithmetic means over seeds, then compute speedup as the ratio of a method's mean throughput to the autoregressive baseline's mean throughput. For the long-context component ablations in Table 5, drafters are trained for five epochs. Entries report the mean and standard deviation over ten seeds, with variants evaluated on the same host within each seed.

Time per output token. TPOT is the total post-first-token request time overlapping the measurement interval divided by the number of subsequent output tokens emitted within that interval. The numerator includes the observed time of requests still active at the end of measurement. This definition excludes time to first token while retaining scheduling and concurrent prefill effects.

D Additional Decoding Results

This section collects results that complement the main evaluation.

D.1 Greedy-Decoding Results

We first check whether the efficiency advantage depends on sampling.

Table 3 presents greedy-decoding results ($T = 0$) with target TP1, concurrency 32, and up to 2,048 generated tokens per request, complementing the sampling-based evaluation in Table 1. The performance trends here closely mirror those observed at $T = 1$, with LongSpark consistently achieving the highest speedups across all model scales and benchmarks. This consistency demonstrates that the efficiency gains of LongSpark are a structural advantage independent of the decoding strategy, further validating that the $O(1)$ proposal cost is the primary driver of end-to-end performance regardless of the temperature setting.

Table 3: Greedy decoding across model scales ($T = 0$). Entries report accepted length and throughput speedup using the conventions of Table 1.

Target	Drafter	Math			Code			Chat		Overall
		GSM8K	MATH-500	AIME25	MBPP	HumanEval	LCB	MT-Bench	Alpaca	Avg.
Q3-4B	EAGLE-3	5.46 (1.55)	5.23 (1.68)	4.72 (1.68)	4.12 (1.24)	4.56 (1.40)	4.92 (1.60)	2.70 (0.89)	2.48 (0.82)	4.27 (1.36)
	DFlash	5.93 (2.21)	5.87 (2.54)	5.28 (2.57)	4.88 (1.90)	5.21 (2.12)	5.34 (2.25)	3.44 (1.51)	3.30 (1.43)	4.90 (2.07)
	DSpark	6.33 (2.26)	6.30 (2.64)	5.77 (2.73)	5.38 (2.01)	5.68 (2.24)	5.71 (2.33)	3.84 (1.62)	3.67 (1.53)	5.34 (2.17)
	LONGSPARK	6.14 (**2.41**)	6.07 (**2.85**)	5.44 (**2.94**)	5.27 (**2.17**)	5.37 (**2.37**)	5.44 (**2.47**)	3.70 (**1.76**)	3.60 (**1.68**)	5.13 (**2.33**)
Q3-8B	EAGLE-3	5.58 (1.68)	5.34 (1.79)	4.88 (1.78)	4.30 (1.35)	4.68 (1.51)	5.11 (1.66)	2.92 (1.00)	2.75 (0.94)	4.44 (1.46)
	DFlash	6.00 (2.37)	5.91 (2.66)	5.41 (2.69)	4.89 (2.00)	5.27 (2.25)	5.34 (2.19)	3.41 (1.56)	3.32 (1.51)	4.94 (2.15)
	DSpark	6.46 (2.50)	6.38 (2.82)	5.89 (2.88)	5.47 (2.17)	5.81 (2.41)	5.77 (2.30)	3.82 (1.70)	3.73 (1.65)	5.42 (2.30)
	LONGSPARK	6.24 (**2.59**)	6.14 (**2.96**)	5.57 (**3.01**)	5.32 (**2.28**)	5.50 (**2.49**)	5.53 (**2.37**)	3.65 (**1.78**)	3.60 (**1.74**)	5.19 (**2.40**)
Q3-14B	EAGLE-3	5.55 (1.90)	5.23 (1.97)	4.69 (1.89)	4.09 (1.47)	4.51 (1.63)	4.90 (1.71)	2.82 (1.08)	2.68 (1.03)	4.31 (1.59)
	DFlash	6.01 (2.59)	5.90 (2.85)	5.33 (2.77)	4.88 (2.19)	5.22 (2.39)	5.21 (2.16)	3.42 (1.67)	3.32 (1.62)	4.91 (2.28)
	DSpark	6.47 (2.70)	6.37 (2.99)	5.81 (2.94)	5.47 (2.36)	5.77 (2.54)	5.66 (2.26)	3.89 (1.83)	3.76 (1.77)	5.40 (2.43)
	LONGSPARK	6.32 (**2.74**)	6.21 (**3.10**)	5.54 (**3.00**)	5.39 (**2.46**)	5.52 (**2.57**)	5.46 (**2.29**)	3.79 (**1.89**)	3.68 (**1.83**)	5.24 (**2.49**)

D.2 LONG-CONTEXT PERFORMANCE ACROSS MODEL SCALES

We then check whether the long-context advantage is specific to the 8B target.

Table 4 extends the Qwen3-8B evaluation to 4B and 14B. Across all three model scales, LONGSPARK achieves the highest average TPS and lowest average TPOT on each benchmark.

Table 4: Long-context decoding on Qwen3-4B and 14B, using the evaluation protocol and reporting conventions of Table 2.

Target	Method	LongSpec 32K		CodeSpan 64K		LSWE 128K	
		TPS ↑	TPOT ↓	TPS ↑	TPOT ↓	TPS ↑	TPOT ↓
Q3-4B	Vanilla	1250.1	12.7	864.4	18.2	185.1	83.0
	EAGLE-3	1810.5	8.7	1197.2	13.1	202.0	77.4
	DFlash	1453.1	10.9	748.9	21.1	175.6	88.7
	DSpark	1761.8	8.9	884.0	17.9	184.0	83.8
	LONGSPARK	**2274.6**	**6.9**	**1497.0**	**10.4**	**233.2**	**66.2**
Q3-14B	Vanilla	951.8	16.6	615.7	25.5	115.1	133.6
	EAGLE-3	964.4	16.4	628.1	25.0	112.0	137.0
	DFlash	1248.8	12.5	832.9	18.8	112.3	136.4
	DSpark	1416.2	11.0	945.1	16.5	121.9	127.8
	LONGSPARK	**1464.2**	**10.7**	**1002.8**	**15.5**	**133.5**	**115.3**

D.3 LONG-CONTEXT COMPONENT ABLATIONS

We evaluate the contribution of each context component on workloads spanning 32K–128K tokens. Table 5 reports accepted length and serving performance for the full drafter and variants that remove one component at a time.

Table 5: Long-context component ablations. All other settings follow Table 2.

(a) Accepted length τ ↑

Variant	LongSpec 32K	CodeSpan 64K	LSWE 128K
LONGSPARK	**2.88** ± 0.11	**3.40** ± 0.11	**2.53** ± 0.22
w/o global summary	2.70 ± 0.09	3.18 ± 0.11	2.25 ± 0.06
w/o boundary state	2.44 ± 0.06	2.82 ± 0.10	2.08 ± 0.10
w/o recent KV window	1.90 ± 0.07	1.77 ± 0.04	1.76 ± 0.08

(b) Serving performance

Variant	LongSpec 32K		CodeSpan 64K		LSWE 128K	
	TPS ↑	TPOT ↓	TPS ↑	TPOT ↓	TPS ↑	TPOT ↓
LONGSPARK	**1597.3** ± 77.0	**9.79** ± 0.46	**1527.2** ± 103.8	**10.26** ± 0.67	**208.4** ± 45.3	**75.39** ± 13.98
w/o global summary	1514.4 ± 77.8	10.36 ± 0.52	1462.3 ± 90.5	10.74 ± 0.65	183.5 ± 10.6	82.90 ± 4.89
w/o boundary state	1460.5 ± 76.9	10.75 ± 0.55	1391.6 ± 73.8	11.29 ± 0.58	177.4 ± 19.2	86.44 ± 8.85
w/o recent KV window	1146.3 ± 58.9	13.74 ± 0.71	976.3 ± 48.5	16.20 ± 0.79	178.1 ± 23.9	86.88 ± 11.96

D.4 PERFORMANCE ACROSS CONCURRENCY LEVELS

We next check how the advantage changes with serving load.

Table 6 extends the Qwen3-8B evaluation to concurrency levels 8, 32, and 128, using target TP1, $T = 1$, and up to 2,048 generated tokens per request. The concurrency-32 results reproduce the corresponding entries in Table 1. We observe a clear trend: while DSpark is competitive at low concurrency (e.g., $C = 8$), LONGSPARK becomes increasingly dominant as the number of concurrent requests grows. At $C = 128$, LONGSPARK maintains a significant throughput lead over all baselines. This scaling behavior stems from the minimal state overhead of LONGSPARK. Unlike existing drafters that must manage and access large KV caches for each concurrent request, LONGSPARK's fixed-cost drafting mechanism avoids the memory-bandwidth bottleneck associated with scaling the drafter's state. Consequently, LONGSPARK is exceptionally well-suited for high-concurrency serving environments where maximizing total throughput is critical.

Table 6: Concurrency scaling on Qwen3-8B (C: concurrency). Entries follow Table 1, with speedups relative to autoregressive decoding at the same concurrency.

C	Drafter	**Math**			**Code**			**Chat**		**Overall**
		GSM8K	MATH-500	AIME25	MBPP	HumanEval	LCB	MT-Bench	Alpaca	Avg.
8	EAGLE-3	5.23 (1.92)	4.70 (1.82)	4.01 (1.61)	3.91 (1.48)	4.34 (1.65)	4.18 (1.58)	2.72 (1.06)	2.54 (0.99)	3.95 (1.51)
	DFlash	5.38 (2.68)	4.92 (2.64)	4.07 (2.27)	4.37 (2.25)	4.68 (2.44)	4.50 (2.27)	3.09 (1.66)	2.99 (1.60)	4.25 (2.23)
	DSpark	6.16 (**2.98**)	5.78 (**3.03**)	5.00 (**2.72**)	5.19 (**2.59**)	5.47 (**2.78**)	5.12 (**2.49**)	3.67 (**1.92**)	3.60 (**1.87**)	5.00 (**2.55**)
	LONGSPARK	5.96 (2.96)	5.57 (2.98)	4.69 (2.63)	5.03 (2.54)	5.14 (2.69)	4.86 (2.44)	3.51 (1.89)	3.46 (1.82)	4.78 (2.49)
32	EAGLE-3	5.26 (1.49)	4.75 (1.50)	4.02 (1.37)	3.92 (1.16)	4.32 (1.30)	4.17 (1.28)	2.67 (0.85)	2.54 (0.82)	3.96 (1.22)
	DFlash	5.36 (1.96)	4.91 (2.02)	4.07 (1.83)	4.36 (1.66)	4.67 (1.82)	4.44 (1.68)	3.09 (1.28)	3.00 (1.24)	4.24 (1.69)
	DSpark	6.17 (2.17)	5.80 (2.32)	4.99 (2.19)	5.18 (1.89)	5.48 (2.06)	5.09 (1.84)	3.69 (1.47)	3.59 (1.42)	5.00 (1.92)
	LONGSPARK	5.93 (**2.24**)	5.57 (**2.42**)	4.72 (**2.27**)	5.03 (**1.98**)	5.16 (**2.11**)	4.83 (**1.88**)	3.53 (**1.54**)	3.46 (**1.49**)	4.78 (**1.99**)
128	EAGLE-3	5.24 (1.10)	4.73 (1.06)	4.02 (1.02)	3.91 (0.82)	4.33 (0.92)	4.14 (0.99)	2.69 (0.60)	2.54 (0.58)	3.95 (0.88)
	DFlash	5.37 (1.33)	4.92 (1.30)	4.09 (1.22)	4.37 (1.07)	4.67 (1.16)	4.46 (1.19)	3.09 (0.80)	3.00 (0.79)	4.25 (1.11)
	DSpark	6.17 (1.49)	5.80 (1.50)	4.98 (1.45)	5.19 (1.25)	5.48 (1.32)	5.11 (1.29)	3.68 (0.93)	3.59 (0.93)	5.00 (1.27)
	LONGSPARK	5.95 (**1.63**)	5.57 (**1.62**)	4.72 (**1.58**)	5.03 (**1.35**)	5.16 (**1.41**)	4.84 (**1.36**)	3.52 (**1.01**)	3.46 (**1.00**)	4.78 (**1.37**)

D.5 FIXED-REQUEST END-TO-END PERFORMANCE

We finally report end-to-end completion time for a fixed set of requests.

Table 7: Mean end-to-end completion time for 32 requests at concurrency 16, averaged over three seeds (seconds; lower is better).

Method	LongSpec 32K	CodeSpan 64K	LSWE 128K
EAGLE-3	82.61	175.67	127.39
DFlash	71.11	202.01	138.75
DSpark	63.66	153.60	116.23
LONGSPARK	**56.14**	**106.77**	**99.71**

We complement the steady-state throughput measurements with the time required to complete a fixed set of 32 requests on each long-context benchmark. All requests arrive together and are served with concurrency 16 using Qwen3-8B, Target TP4, $T = 1$, and a maximum of 8,192 generated tokens. Elapsed time includes prefill, queuing, decoding, and completion of the final requests, excluding model loading and warm-up. Table 7 reports mean completion times. Responses are sampled independently, so completion times reflect both serving efficiency and variation in generated response length. LONGSPARK achieves the lowest mean completion time on all three workloads.