QSPAR: CO-DESIGNING SPARSE ATTENTION AND QUANTIZED WEIGHTS FOR LONG-CONTEXT SELF-SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reducing KV-cache traffic alone is insufficient to eliminate drafting overhead in long-context self-speculative decoding. Under a fixed GPU memory budget, longer contexts constrain the feasible batch size, limiting the amortization of weight access across requests. A sparse-attention-based drafter further reduces draft-side KV-cache traffic but leaves repeated access to the target-sized drafter weights unchanged, exposing a substantial weight-side bottleneck. Low-bit weight quantization reduces this cost, but naively combining it with sparse attention degrades acceptance length. We address this runtime–acceptance trade-off through two key observations: token flips concentrate at low-margin positions, and the BF16 target token usually remains among a few top-ranked draft candidates. Based on these observations, QSpar combines a weight-quantized sparse drafter with a candidate Reranker and a margin-aware Reducer. The Reranker recomputes candidate logits using BF16 LM-head weights, while the Reducer selectively retains alternatives under a bounded candidate budget. Final verification uses the original full-attention BF16 model to preserve the exact target-model generation. Under a fixed KV-cache budget, QSpar achieves $1.49\times$ average and $1.53\times$ maximum end-to-end speedups over Vegas, and up to $2.21\times$ over vLLM.

1 INTRODUCTION

Recent advances in reasoning and agentic workloads (Snell et al., 2025; Yang et al., 2024) have led to rapidly growing context lengths in LLM serving. In particular, long-horizon reasoning and multi-turn interactions accumulate previously generated tokens into the decoding context, substantially increasing the amount of context processed by each request (Cai et al., 2025). During autoregressive decoding with full attention, each newly generated token requires access to the accumulated KV cache, causing both per-request KV-cache capacity requirements and memory traffic to increase with context length. This memory-bound behavior can significantly degrade decode throughput in long-context workloads, making efficient context processing a critical challenge for LLM serving.

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) is a promising approach for mitigating the cost of autoregressive generation. A lightweight drafter first generates multiple future tokens, which are then verified in parallel by the target model. With an exact verification procedure, speculative decoding can reduce decoding cost while preserving the target model's generation distribution. However, effective speculation requires the drafter to be both sufficiently fast and well aligned with the target model. This alignment requirement creates an additional model-pair dependency for separately trained drafters (Cai et al., 2024; Li et al., 2025): changes to the target checkpoint or post-training procedure may disrupt the alignment and require further adaptation of the drafter. Self-speculative decoding (Zhang et al., 2024; Xia et al., 2025; Zhao et al., 2025a) alleviates this dependency by constructing the drafter from the target model itself, using approximations such as layer skipping, sparse attention, or reduced precision.

Sparse attention based self speculation (Sadhukhan et al., 2025; Yue et al., 2026; Zhao et al., 2026) is particularly attractive for long-context decoding because a sparse drafter accesses only a subset of the KV cache, substantially reducing draft-side KV-cache traffic. However, sparse attention does not reduce the cost of accessing the drafter's weights. Under a finite GPU memory budget, increasing

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

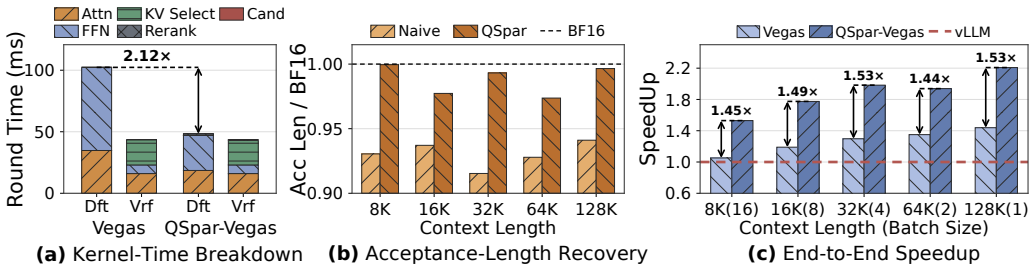

Figure 1: Effect of QSpar on drafter runtime, acceptance length, and end-to-end decode throughput for Qwen3-8B with Vegas on RULER. (a) Kernel-time breakdown at 64K context with a batch size of 2. FFN execution dominates the Vegas drafter, while QSpar achieves a $2.12\times$ drafter-side speedup. (b) Acceptance length normalized to the BF16 drafter across context lengths. Naive AWQ-quantized sparse drafting lowers acceptance length, whereas QSpar recovers most of the loss. (c) End-to-end decode speedup relative to vLLM under a fixed KV-cache budget.

context length reduces the feasible batch size, limiting how effectively weight access can be amortized across requests. Moreover, when the aggregate KV-cache token budget is held fixed, longer contexts do not proportionally increase aggregate KV-cache traffic per draft step because batch size decreases accordingly. Sparse attention further reduces this traffic while leaving weight access unchanged. Figure 1(a) shows that FFN execution dominates a Vegas-based (Yue et al., 2026) sparse drafter on the RULER at 64K context with a batch size of 2, highlighting the substantial weight-side cost that remains after KV-cache traffic is reduced.

This observation motivates combining sparse attention (Xiao et al., 2024; Tang et al., 2024; Zhang et al., 2023) with weight quantization (Lin et al., 2024; Frantar et al., 2023; Shao et al., 2024). Weight quantization directly reduces weight memory traffic of the target-sized drafter, alleviating the remaining weight-side cost. However, naively combining weight quantization with sparse attention decreases the average acceptance length, partially offsetting the reduction in draft latency. Because decode throughput depends on both the cost of each speculative round and the number of tokens it produces, the central challenge is to reduce both KV-cache and weight traffic while recovering acceptance at low additional cost.

Our analysis identifies two primary sources of acceptance loss in quantized sparse drafting: candidate-ranking errors caused by LM-head quantization and autoregressive error propagation initiated by quantization-induced hidden-state errors accumulated across preceding Transformer blocks (Section 3.2). Importantly, their impact is not uniform across prediction positions. Specifically, token flips, in which the drafter selects a top-1 token different from that selected by the BF16 verifier at the same position, concentrate at positions with small verifier logit margins. Moreover, the BF16 target token usually remains among a few top-ranked draft candidates at flipped positions. Together, these observations motivate correcting candidate rankings and selectively retaining alternatives at uncertain positions while limiting candidate overhead.

Guided by these findings, we propose **QSpar**, a quantized sparse self-speculative decoding framework that retains low draft execution cost while recovering much of the acceptance length lost to quantization. At its core, QSpar comprises two complementary components: a Reranker and a Reducer. The Reranker recomputes BF16 LM-head logits only for the top-K candidates produced by the quantized sparse drafter, correcting candidate-ranking errors without evaluating the full vocabulary in BF16, although hidden-state-induced errors remain. The Reducer uses the reranked candidate margin as an online confidence signal and selectively retains additional candidates at low-margin positions under a bounded candidate budget. These alternatives allow the verifier to recover target tokens that reranking alone does not place first. We focus on greedy decoding and use the original full-attention BF16 target model for final verification.

Figure 1 summarizes the resulting benefits. Figure 1(a) shows that QSpar reduces the dominant FFN time and achieves a 2.12× drafter-side speedup at 64K context, with negligible reranking and candidate-handling overhead. Figure 1(b) shows that QSpar recovers much of the acceptance length loss caused by naive sparse drafting with Activation-aware Weight Quantization (AWQ) (Lin et al., 2024), approaching the BF16 level across context lengths. Together, these gains yield a $1.49\times$

average end-to-end decode speedup over Vegas under a fixed KV-cache budget across 8K–128K contexts, as shown in Figure 1(c). The main contributions of this paper are as follows:

- We characterize a KV-budget-constrained regime for long-context self-speculative decoding and show that sparse attention can expose substantial weight-side draft execution cost after reducing KV-cache traffic.
- We show that token flips concentrate at low-margin positions and that the target token often remains among a small set of draft candidates. Based on these observations, we design a Reranker and a Reducer to recover acceptance length through selective logit recomputation and uncertainty-aware candidate retention.
- We evaluate QSpar across various context lengths, sparse attention methods, quantization methods, model families, and workloads, demonstrating end-to-end decode throughput improvements with final verification by the original BF16 target model.

2 Related Work

Speculative decoding accelerates autoregressive inference by generating candidate tokens with a faster draft model and verifying them in parallel with the target model, while exact acceptance and resampling preserve the target distribution (Leviathan et al., 2023; Chen et al., 2023). Beyond using smaller pretrained models, learned drafters are optimized to better approximate the target distribution (Cai et al., 2024; Li et al., 2025; Chen et al., 2026; Cheng et al., 2026; Li et al., 2024). However, practical speedup requires the drafter to be both sufficiently fast and well aligned with the target, and learned drafters may require additional adaptation after target fine-tuning or RL-based post-training (Lin et al., 2026). To avoid these alignment issues and additional training costs, training-free *self-speculative decoding* derives the drafter directly from the target model through layer skipping (Zhang et al., 2024; Xia et al., 2025; Elhoushi et al., 2024), sparse attention (Sadhukhan et al., 2025; Yue et al., 2026; Zhao et al., 2026), or quantization (Tiwari et al., 2025; Zhao et al., 2025a;b). Layer skipping reduces compute, but aggressive skipping can lower draft-target agreement.

Sparse attention based self speculative decoding reduces drafting cost by accessing only a subset of the target KV cache. MagicDec studies sparse drafting for long-context inference (Sadhukhan et al., 2025), while Vegas and SparseSpec reuse verification-time attention information to guide KV selection (Yue et al., 2026; Zhao et al., 2026). TriForce combines target-weight sharing with retrieved sparse KV cache in a hierarchical speculative pipeline (Sun et al., 2024), and SpecPV similarly exploits partial KV states to reduce verification cost (Tan et al., 2025). These approaches reduce KV cache traffic, but do not reduce the model-weight traffic of each draft forward pass.

Quantization based self speculative decoding instead reduces drafting cost through lower-precision execution while reusing the target model's parameters or states. QuantSpec, QSpec, and SPEQ construct low-precision drafters through quantized weights, activations, or KV cache (Tiwari et al., 2025; Zhao et al., 2025a;b). These methods reduce model-weight traffic and storage duplication, but full-context attention can remain memory-bound unless the KV cache is also quantized, which may sacrifice exact equivalence to the original target. QSpar identifies the bottleneck shift induced by realistic KV-cache budgets and combines sparse KV access with draft-only weight quantization. By mitigating the resulting degradation in accepted length, QSpar improves throughput while preserving exactness with respect to the target model.

3 Preliminaries and Motivation

3.1 Serving-Constrained Self-Speculation

LLM serving must accommodate both model weights and the KV caches of all active requests within a finite GPU HBM budget. Let B denote the batch size, L the average context length per active request, and $m_{\mathrm{KV/token}}$ the KV-cache memory required per token. Ignoring other runtime memory components, the serving memory constraint can be approximated as $M_W + BLm_{\mathrm{KV/token}} \leq M_{\mathrm{HBM}}$, where M_W denotes the memory footprint of the model weights. Under a fixed HBM budget M_{HBM}, increasing L increases the KV-cache capacity required by each request and consequently reduces the feasible batch size B. Long-context serving therefore naturally enters a low-batch regime. Within this regime, the performance of speculative decoding depends on both drafter latency and

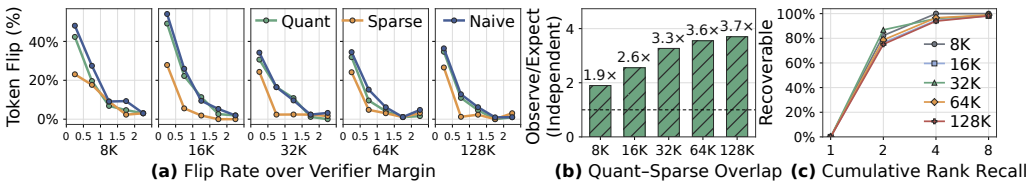

Figure 2: Margin analysis of quantized sparse drafting for Qwen3-8B on RULER with Vegas and AWQ. (a) Token-flip rate versus verifier margin for quantization, sparse attention, and Naive combination; token flips concentrate at low-margin positions. (b) Ratio of the observed Quant–Sparse flip overlap to that expected under independent errors. (c) Cumulative rank recall of the target token among the top-K draft candidates; most target tokens at flipped positions remain within the top-4.

acceptance length. Let T_{draft} and T_{verify} denote the execution times of the draft and verification stages, respectively, and let N_{accepted} denote the number of accepted draft tokens in one speculative iteration. The amortized per-token cost can be approximated as

$$T_{\text{token}} \approx \frac{T_{\text{draft}} + T_{\text{verify}}}{\mathbb{E}[N_{\text{accepted}}] + 1}. \tag{1}$$

Thus, reducing draft latency alone does not guarantee end-to-end speedup, since a reduction in drafter cost can be offset by a decrease in the number of accepted tokens per speculative iteration.

Because draft execution time is influenced by both weight and KV-cache accesses, we decompose the memory traffic of a sparse drafter as $D_{\text{draft}} \approx D_W + \rho D_{\text{KV}}(B, L)$ where D_W denotes model-weight memory traffic, D_{KV} denotes the traffic incurred by dense KV-cache access, and $\rho \in (0, 1]$ denotes the fraction of KV entries accessed by the sparse attention mechanism. Sparse attention reduces the KV-cache component ρD_{KV} but does not directly reduce D_W. Consistent with this decomposition, the weight-intensive FFN runtime dominates the sparse-drafter runtime in Figure 1(a).

3.2 Acceptance Degradation in Quantized Sparse Drafting

We next examine why adding weight quantization to sparse drafting degrades acceptance. We refer to a prediction as a *token flip* when an approximate drafter selects a token different from that selected by the original BF16 target model at the same prediction position. We distinguish Quant flips caused by weight quantization, Sparse flips caused by sparse attention, and Naive combination flips induced by jointly applying both sparse attention and weight quantization.

To identify where these flips occur, we define the verifier margin at position t as $\delta_t^{\text{ver}} = z_{t,(1)}^{\text{ver}} - z_{t,(2)}^{\text{ver}}$, where $z_{t,(1)}^{\text{ver}}$ and $z_{t,(2)}^{\text{ver}}$ denote the largest and second-largest logits produced by the original full-attention BF16 verifier, respectively. A small δ_t^{ver} indicates that the top-1 and top-2 candidates have similar scores and are therefore sensitive to small perturbations. Figure 2(a) confirms that token flips concentrate at low-margin positions across all three settings. Acceptance degradation is therefore driven disproportionately by a small set of uncertain prediction positions.

To isolate the effect of weight quantization, we compare a BF16 sparse drafter with its quantized counterpart under the same sparse attention pattern. Let $h_t^{\text{S,BF16}}$ and $h_t^{\text{S,q}}$ denote respective hidden states, and let $W_{\text{LM}}^{\text{BF16}}$ and W_{LM}^{q} denote LM-head weights. The corresponding logits are $z_t^{\text{S,BF16}} = W_{\text{LM}}^{\text{BF16}} h_t^{\text{S,BF16}}$ and $z_t^{\text{S,q}} = W_{\text{LM}}^{q} h_t^{\text{S,q}}$. Their difference is decomposed as

$$z_t^{\text{S,BF16}} - z_t^{\text{S,q}} = \underbrace{W_{\text{LM}}^{\text{BF16}} \left(h_t^{\text{S,BF16}} - h_t^{\text{S,q}} \right)}_{\epsilon_t^{\text{HS}}} + \underbrace{\left(W_{\text{LM}}^{\text{BF16}} - W_{\text{LM}}^{q} \right) h_t^{\text{S,q}}}_{\epsilon_t^{\text{LM}}}. \tag{2}$$

We refer to ϵ_t^{HS} as the *hidden-state-induced logit error* and ϵ_t^{LM} as the *LM-head-induced logit error*. The former captures errors accumulated layer by layer through the quantized Transformer blocks, whereas the latter captures the direct effect of LM-head weight quantization. Either quantization-induced error can reverse the candidate ordering when the verifier margin δ_t^{ver} is small, explaining why Quant flips in Figure 2(a) concentrate at low-margin positions. ϵ_t^{HS} is a logit deviation at the current position; if it changes the selected token, it can subsequently trigger autoregressive error propagation across later draft positions.

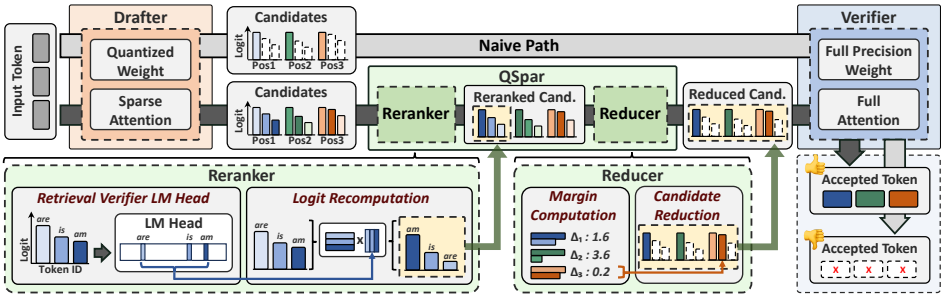

Figure 3: End-to-end pipeline of QSpar. The Naive and QSpar paths use the same quantized sparse drafter; the Naive path forwards the generated candidates directly to the verifier. QSpar instead uses the Reranker to recompute full-precision LM head logits over a small candidate set and the margin-aware Reducer to selectively retain and manage candidate branches before verification.

Figure 2(b) compares the observed Quant–Sparse flip overlap with that expected under independent errors and shows that the two types of flips are not statistically independent, consistent with their shared concentration at low-margin positions. This ratio increases from $1.9\times$ at 8K context to $3.7\times$ at 128K context, indicating that the two approximations are increasingly likely to fail at the same vulnerable positions. The implications of this overlap for acceptance recovery are examined in Section 5.3, where we show that QSpar recovers not only quantization-induced ranking errors but also a subset of tokens that would otherwise be flipped by sparse attention.

Figure 2(c) then examines whether the BF16 target token can be recovered from the draft candidate set at flipped positions. It shows that the BF16 target token remains within the top-2 draft candidates at many flipped positions even when it is no longer ranked first. Thus, many flips reflect candidate misranking rather than removal of the target token from the candidate set. This observation motivates the design of QSpar, which corrects candidate rankings and selectively retains alternatives only at uncertain, low-margin positions rather than maintaining multiple candidates everywhere.

4 QSPAR

Section 3.2 identifies two quantization-induced logit-error components relative to the corresponding BF16 sparse drafter: ϵ_t^{LM} and ϵ_t^{HS}. QSpar uses the Reranker to remove ϵ_t^{LM} over a small candidate set and the Reducer to mitigate autoregressive error propagation caused by the remaining ϵ_t^{HS}. Figure 3 contrasts the Naive and QSpar paths, both of which use the same quantized sparse drafter. The Naive path forwards draft candidates directly to the full-precision verifier without the Reranker or Reducer. In contrast, QSpar reranks the candidates and retains alternatives at uncertain positions, recovering acceptance while preserving the low execution cost of the quantized sparse drafter.

4.1 RERANKER: TOP-K CANDIDATE RERANKING

Figure 2(c) shows that the top-1 token selected by the full-attention BF16 verifier remains among the top-8 candidates of the quantized sparse drafter at nearly all flipped positions. This suggests that many acceptance failures stem from candidate misranking rather than target-token omission, motivating BF16 reranking of a small candidate set rather than full-vocabulary logit recomputation.

The Reranker is inspired by the retrieve-then-rerank paradigm commonly used in Retrieval-Augmented Generation (RAG) systems (Yu et al., 2024; Sun et al., 2025; Glass et al., 2022): a lightweight retrieval stage produces a small candidate set, after which a more accurate scorer reorders those candidates. Similarly, let $\mathcal{C}_t = \{c_{t,1}, \ldots, c_{t,K}\}$ denote the top-K candidates generated by the quantized sparse drafter at position t. Based on Figure 2(c), we set $K_{rerank} = 8$ to retain the BF16 target token with high probability. The Reranker gathers only the LM head rows corresponding to the candidates in $\mathcal{C}_t$ and recomputes their logits using the BF16 LM head. For candidate $c_{t,i}$, the reranked logit is computed as $z_{t,i}^{\mathrm{rerank}} = (W_{c_{t,i}}^{\mathrm{BF16}})^{\top} h_t^q$, where $W_{c_{t,i}}^{\mathrm{BF16}}$ is the corresponding BF16 LM head row and h_t^q is the hidden state produced by the quantized sparse drafter. The candidates are then reordered according to the recomputed logits.

For the selected candidates, this recomputation removes ϵ_t^{LM}, while ϵ_t^{HS} remains because h_t^q is unchanged. This candidate-only recomputation also reduces the additional LM-head scoring cost from $O(|V|d)$ required for full-vocabulary scoring to $O(K_{\mathrm{rerank}}d)$ for the reranked candidate set, where $|V|$ is the vocabulary size and d is the hidden dimension. With $K_{\mathrm{rerank}} = 8$, only eight BF16 LM head rows are loaded. Passing all K_{rerank} candidates to the verifier would increase verification overhead; the Reducer therefore retains additional candidates only at uncertain positions.

4.2 Reducer: Margin-Aware Candidate Reduction

Because reranking does not remove ϵ_t^{HS}, the reranked top-1 token can still be incorrect and perturb later hidden and KV states. The Reducer limits this propagation by retaining alternative candidates at uncertain positions. The verifier margin δ_t^{ver} in Figure 2(a) is computed from the original full-attention BF16 verifier logits and is used to analyze the relationship between token flips and prediction uncertainty. Because this margin is unavailable before final verification, the Reducer instead uses the reranked candidate margin $\delta_t^{\mathrm{rerank}} = z_{t,1}^{\mathrm{rerank}} - z_{t,2}^{\mathrm{rerank}}$ as an online confidence signal. Figure 2(a) shows that the flip rate is low when $\delta_t^{\mathrm{ver}} \geq 2$. Motivated by this observation, we set $\tau = 2$ as an empirical threshold for the online reranked margin and evaluate its sensitivity in Appendix A.2. Positions satisfying $\delta_t^{\mathrm{rerank}} \geq \tau$ retain only the top-1 candidate, whereas positions satisfying $\delta_t^{\mathrm{rerank}} < \tau$ are considered uncertain and become eligible for an additional candidate.

Figure 2(c) shows that the top-2 candidates provide sufficiently high recall of the BF16 target token. To balance target-token recovery against verification overhead, we retain the reranked top-1 candidate and one alternative at each selected uncertain position, yielding $K_{\mathrm{reducer}} = 2$.

When many low-margin positions occur, retaining additional candidates at all of them could substantially increase the number of tokens passed to the verifier. We therefore impose a global budget M on the total number of additional candidates retained along each draft path. For a draft path of length N_{draft}, we set $M = \lfloor 0.2 N_{\mathrm{draft}} \rfloor$; for example, at most two additional candidates are retained for a path containing ten draft tokens. Let r_t denote the ascending rank of $\delta_t^{\mathrm{rerank}}$ among the margins along the draft path. The reduced candidate set is defined as

$$\mathcal{C}_t^{\mathrm{reducer}} = \begin{cases} \{c_{t,1}, c_{t,2}\}, & \delta_t^{\mathrm{rerank}} < \tau \text{ and } r_t \leq M, \\ \{c_{t,1}\}, & \text{otherwise.} \end{cases} \tag{3}$$

This policy prioritizes positions where ambiguous predictions are most likely to propagate errors while bounding verification overhead. Because the Reducer relies only on the observed candidate margins, it can be combined with different weight-quantization methods.

The retained alternatives and main draft path form a candidate tree, which is passed to the original verifier. To process multiple candidate branches in a single verification pass, QSpar follows the tree-attention design of EAGLE-2 (Li et al., 2024). Within the verifier, QSpar constructs branch-aware attention masks and maintains the corresponding KV cache states for each candidate branch. Each branch attends only to the shared prefix and its own ancestor nodes, preventing states from different branches from being mixed. This structure enables the verifier to evaluate all candidate branches in parallel using the original BF16 model weights and full attention. Algorithm 1 in Appendix B details the complete procedure from candidate-tree construction to alternatives verification.

5 Experiments

5.1 Experimental Setup

Implementation and Evaluation Settings. We implement QSpar by integrating it into vLLM (Kwon et al., 2023) and evaluate all methods within the same vLLM-based serving environment. We use Qwen3-4B, Qwen3-8B, Qwen3-14B, and Qwen3-32B, together with Meta-Llama-3.1-8B-Instruct for RULER and DeepSeek-R1-Distill-Llama-8B for AIME25 (Yang et al., 2025; Grattafiori et al., 2024; Guo et al., 2025). All experiments use a single NVIDIA A100 80GB GPU, except for Qwen3-32B, which uses two NVIDIA GPUs interconnected via NVLink with tensor parallelism. Unless otherwise stated, we use a 7% sparsity ratio, meaning that the drafter accesses 7% of the KV entries, a draft length of 10 tokens, and a temperature of 0. For QSpar, we use $K_{\mathrm{rerank}} = 8$, $\tau = 2$, $K_{\mathrm{reducer}} = 2$, $M = 2$. Appendix A analyzes how these parameters and the draft length affect

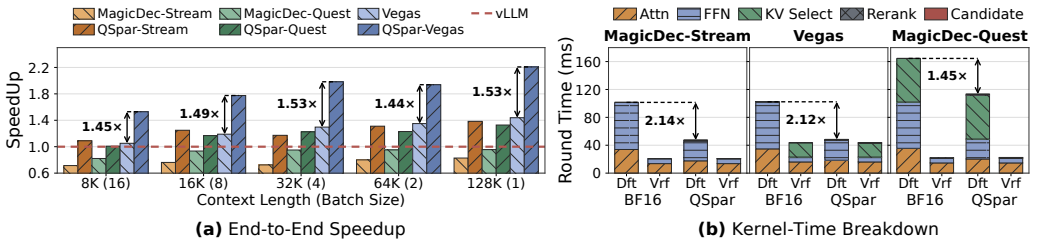

Figure 4: Throughput and kernel-time analysis under a fixed KV-cache token budget. (a) End-to-end decode speedup relative to vLLM as context length increases from 8K to 128K and batch size decreases from 16 to 1. The QSpar variants consistently outperform their corresponding sparse-drafting baselines. (b) Drafter and verifier kernel-time breakdown at 64K context with a batch size of 2. QSpar reduces the weight-intensive drafter runtime across all three sparse attention methods.

acceptance length and throughput. We use AWQ with a group size of 128 for the main results and additionally evaluate Round-to-Nearest (RTN) (Kogan, 2025) in Section 5.3. For Qwen3 beyond 64K context lengths, we use YaRN-based RoPE scaling with a factor of 4 (Peng et al., 2024).

Baselines and Benchmarks. We compare QSpar against an autoregressive generation baseline (vLLM), Vegas (Yue et al., 2026), MagicDec-Stream (Sadhukhan et al., 2025; Xiao et al., 2024), MagicDec-Quest (Tang et al., 2024), and naive quantized sparse drafting without candidate reranking or reduction. The sparse attention baselines use different KV-selection policies: MagicDec-Stream applies static selection, MagicDec-Quest dynamically selects KV entries at each drafting step, and Vegas uses verifier attention information to guide selection. We exclude the KV-cache-quantization baseline QuantSpec (Tiwari et al., 2025) because it quantizes the verifier's KV cache and therefore does not preserve lossless target-model generation whereas QSpar retains the original BF16 full-attention verifier. We evaluate long-context workloads using RULER (Hsieh et al., 2024) with context lengths from 8K to 128K tokens and long-generation reasoning using AIME25 (Zhang & Team, 2025), where the generated responses average approximately 19.8K tokens. For AIME25, we enable thinking mode for the Qwen3 models and DeepSeek-R1-Distill-Llama-8B.

5.2 Throughput Analysis

Figure 4 evaluates whether reducing drafter-side execution cost leads to actual throughput gains in a realistic serving regime with a fixed KV-cache budget. Specifically, Qwen3-8B is evaluated on RULER while the context length and batch size vary inversely: $(8\text{K}, 16)$, $(16\text{K}, 8)$, $(32\text{K}, 4)$, $(64\text{K}, 2)$, and $(128\text{K}, 1)$. Because $B \times L = 128\text{K}$ for all configurations, the experiment keeps the aggregate KV-cache token budget fixed while transitioning from a high-batch to a low-batch regime.

Under this setting, Figure 4(a) shows that existing sparse attention methods do not consistently outperform the AR baseline. Although sparse attention baselines reduce KV-cache traffic, the cost of repeatedly accessing the drafter's model weights remains substantial and can offset the benefit of speculative decoding. In contrast, QSpar consistently achieves higher throughput across all context lengths. Compared with the AR baseline, QSpar achieves up to $2.21\times$ end-to-end speedup. These results indicate that reducing KV-cache traffic alone is insufficient; the weight-side cost of the drafter must also be reduced to achieve end-to-end gains.

To explain the source of this improvement, we use Nsight Systems (NVIDIA, 2024) to obtain a GPU kernel-time breakdown for the $(64\text{K}, 2)$ setting, which represents a low-batch serving regime. As shown in Figure 4(b), drafting accounts for most of the execution time in self-speculative decoding. After sparse attention reduces KV-cache traffic, the FFN becomes the dominant drafter component because each drafting step must still stream the target-sized model weights. Weight quantization reduces this weight traffic, producing substantial drafter-side speedups. Across MagicDec-Stream, MagicDec-Quest, and Vegas, QSpar achieves approximately $2.14\times$, $1.45\times$, and $2.12\times$ drafter-side speedups, respectively. Reranking adds drafter-side work, while verifying the retained candidate branches adds verifier-side work; both account for only a negligible fraction of the corresponding execution time. The smaller gain for MagicDec-Quest is attributable to its larger KV-selection overhead, which remains a significant portion of drafter execution even after weight quantization.

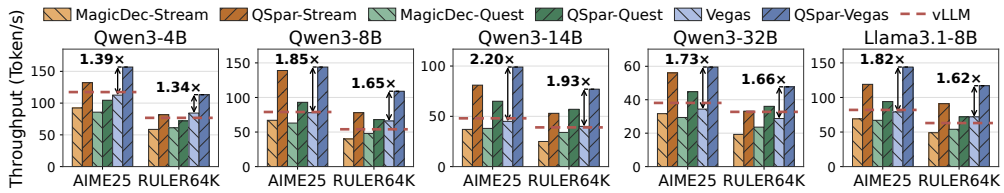

Figure 5: End-to-end decode throughput across model scales, model families, and workloads. Results are reported for Qwen3 from 4B to 32B, Llama3.1-8B on RULER, and DeepSeek-R1-Distill-Llama-8B on AIME25; dashed lines indicate the corresponding vLLM throughput. QSpar improves throughput over the corresponding sparse-drafting baselines across all configurations.

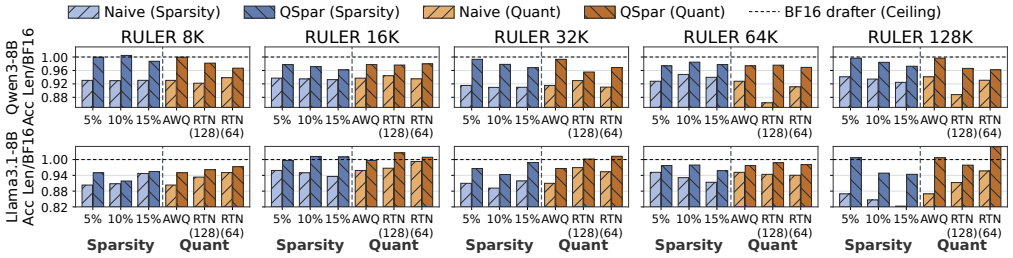

Figure 6: Acceptance length normalized to the BF16 sparse drafter across sparsity and quantization configurations for Qwen3-8B and Llama3.1-8B on RULER (8K–128K); dashed lines denote the BF16 reference. The sparsity sweep varies the sparsity ratio from 5% to 15% under AWQ, while the quantization sweep fixes it at 5% and compares AWQ with RTN using group sizes of 128 and 64. Across both sweeps, QSpar recovers the acceptance loss from naive quantized sparse drafting, approaching or exceeding the BF16 reference.

Figure 5 evaluates whether the throughput gains of QSpar generalize across model scales, model families, and workload types under a batch size $B = 1$. Across Qwen3 models ranging from 4B to 32B parameters and Llama-based models, QSpar consistently achieves higher throughput than the corresponding sparse-drafting baselines on both RULER and AIME25.

For Qwen3 models, the speedup relative to Vegas increases as the model scale grows from 4B to 14B. Specifically, the speedup over Vegas rises from $1.39\times$ to $2.20\times$ on AIME25 and from $1.34\times$ to $1.93\times$ on RULER. This trend results from the larger weight-memory traffic incurred by larger Qwen3 drafters. As model size increases, weight loading accounts for a larger fraction of drafter execution, so weight quantization can reduce a larger portion of the dominant runtime cost and thereby produce a larger throughput gain over Vegas. For Qwen3-32B, the speedup is lower than that of Qwen3-14B, reaching $1.73\times$ on AIME25 and $1.66\times$ on RULER, because it requires tensor parallelism across two GPUs. Distributing the model weights across GPUs increases the aggregate model-loading bandwidth and reduces the relative benefit of weight quantization, while synchronization overhead from inter-GPU communication makes attention a larger fraction of decoding time, thereby accounting for the reduced speedup.

The Llama-based models show gains comparable to those of Qwen3-8B: DeepSeek-R1-Distill-Llama-8B achieves a $1.82\times$ speedup on AIME25, while Llama3.1-8B achieves a $1.62\times$ speedup on RULER. These results indicate that the benefit of QSpar is not specific to the Qwen3 architecture. The baselines also exhibit workload-dependent behavior. MagicDec-Stream provides a smaller benefit on RULER because its static recent-token selection is poorly matched to retrieval tasks that require distant context, resulting in shorter accepted spans. On AIME25, generations depend more heavily on recent context and therefore achieve longer accepted spans. This behavior highlights the coupling between KV selection and acceptance length in determining end-to-end throughput.

5.3 Acceptance-Length Recovery

Robustness across Sparsity and Quantization Configurations. Figure 6 evaluates whether QSpar recovers the acceptance-length degradation caused by quantized sparse drafting across different

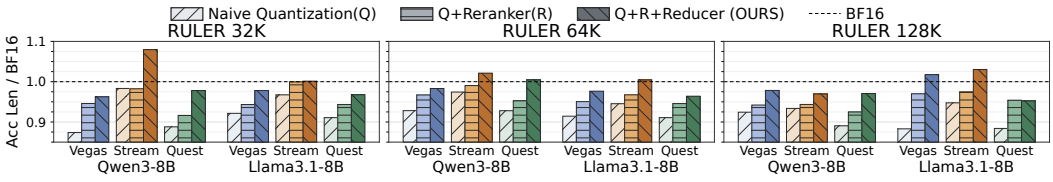

Figure 7: Component ablation of QSpar across sparse attention methods, models, and context lengths. Acceptance length is normalized to the BF16 sparse drafter for Qwen3-8B and Llama3.1-8B on RULER (32K–128K); dashed lines indicate the BF16 reference. Reranker recovers much of the lost acceptance, while candidate retention using Reducer provides further gains.

sparsity and quantization. All acceptance lengths are normalized to those of the corresponding BF16 sparse drafter. We define the acceptance recovery rate as the fraction of the gap between naive quantized sparse drafting and the BF16 sparse drafter reference that is closed by QSpar. Reported values are averaged across Vegas, MagicDec-Stream and MagicDec-Quest, with per-method results provided in Appendix C. We consider two complementary settings: a sparsity sweep that uses AWQ and varies the sparsity ratio from 5% to 15% and a quantization sweep that fixes the ratio at 5% and compares AWQ with a group size of 128 against RTN with group sizes of 128 and 64.

Across the sparsity sweep, naively quantizing the sparse drafter consistently reduces its acceptance length relative to the BF16 reference. The normalized acceptance length falls to as low as 0.85 for Qwen3-8B and approximately 0.82 for Llama3.1-8B. In contrast, QSpar remains close to or above the BF16 reference, reaching approximately 0.96–1.01 for Qwen3-8B and exceeding the reference in several Llama3.1-8B settings, indicating robustness across the evaluated sparsity range.

Figures 2(b) and (c) help explain why QSpar can occasionally achieve a higher acceptance length than the BF16 sparse drafter reference. Quantization- and sparsity-induced flips frequently overlap, while the token selected by the full-attention BF16 verifier often remains in the draft candidate set. The Reducer can therefore preserve this token as an alternative even when the single-path BF16 sparse drafter ranks another token first, allowing candidate branching to surpass the BF16 sparse drafter reference.

The quantization sweep shows a consistent acceptance-recovery trend across different quantizers and group sizes. At 128K context, QSpar closes approximately 50–97% of the acceptance gap between naive quantized sparse drafting and the corresponding BF16 sparse drafter reference for Qwen3-8B. For Llama3.1-8B, QSpar recovers nearly all of the loss across the three quantization settings and occasionally exceeds the BF16 reference. Similar behavior across context lengths indicates that the recovery mechanism is not tied to a particular quantizer or group size.

Component Ablation. Figure 7 isolates the contributions of the Reranker and Reducer to acceptance recovery. We compare naive quantized sparse drafting, the same drafter with candidate reranking, and the complete QSpar pipeline across sparse attention methods, models, and context lengths. For Qwen3-8B with Vegas at 32K context, the Reranker closes 54% of this gap, indicating that many failures arise from candidate misranking rather than the complete removal of the target token. Candidate-only full-precision LM head recomputation therefore recovers substantial acceptance without evaluating the full vocabulary. Adding the Reducer increases the recovered fraction to 70% by retaining alternatives at low-margin positions. For Llama3.1-8B with MagicDec-Stream, the complete pipeline fully closes the gap across all evaluated context lengths.

6 Conclusion

We presented QSpar, a self speculative decoding framework that jointly addresses KV-cache and weight-memory traffic for long-context inference. By combining a weight quantized sparse drafter with candidate Reranker and margin-aware candidate Reducer, QSpar reduces drafting cost while recovering much of the acceptance length lost to quantization. Across 8K–128K contexts and 4B–32B models, QSpar consistently improves throughput over sparse attention based self speculative decoding and vLLM. These results demonstrate that sparse attention and weight quantization should be co-designed to improve long-context decoding under constrained GPU memory.

AI USE STATEMENT

We used generative AI tools to assist with code implementation and debugging, as well as manuscript language editing and polishing. All AI-assisted outputs were reviewed and verified by the authors. The authors take full responsibility for the final content of this work, including its text, claims, analyses, and code.

ETHICS STATEMENT

This work evaluates inference efficiency on the RULER and AIME25 benchmarks using publicly available language models. The experiments do not involve human subjects or the collection or release of personal data. This study does not separately evaluate bias, fairness, or potentially harmful outputs of the underlying language models, as these issues are outside the scope of our inference-efficiency evaluation.

REPRODUCIBILITY STATEMENT

We document the method and experimental settings in the main paper and appendices to support reproducibility. Section 3 defines token flips and presents the verifier-margin analysis. Section 4 describes the quantized sparse drafter, Reranker, Reducer, and candidate-tree verification procedure. Section 5.1 specifies the models, benchmarks, baselines, hardware, quantization and sparsity settings, and decoding configuration. Appendix A analyzes the effects of key hyperparameters on acceptance length and throughput. Appendix B details the candidate-tree construction and verification procedure, while Appendix C reports per-method results for the evaluated sparse-attention methods. The source code, configuration files, and evaluation scripts will be released to facilitate reproduction of the reported results.

REFERENCES

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

Zefan Cai, Wen Xiao, Hanshi Sun, cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Animashree Anandkumar, Abedelkadir Asi, and Junjie Hu. R-kv: Redundancy-aware kv cache compression for reasoning models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 60980–61005. Curran Associates, Inc., 2025. doi: 10.52202/085713-2038. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/57e0358aed314784cad14a26e1eba642-Paper-Conference.pdf.

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. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Oz335dV48X.

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. URL https://arxiv.org/abs/2607.05147.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=tcbBPnfwxS.

Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Rajaram Naik, Pengshan Cai, and Alfio Gliozzo. Re2G: Retrieve, rerank, generate. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2701–2715, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.194. URL https://aclanthology.org/2022.naacl-main.194/.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Alex Kogan. Is (selective) round-to-nearest quantization all you need? *arXiv preprint arXiv:2505.15909*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

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. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/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 *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=4exx1hUffq.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In P. Gibbons, G. Pekhimenko, and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 87–100, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf.

Luxi Lin, Zhihang Lin, Zhanpeng Zeng, Yuhao Chen, Qingyu Zhang, Jixiang Luo, Xuelong Li, and Rongrong Ji. Efficiently Aligning Draft Models via Parameter- and Data-Efficient Adaptation. *arXiv preprint arXiv:2603.09527*, 2026. URL https://arxiv.org/abs/2603.09527.

NVIDIA. Nsight systems, 2024. URL https://developer.nvidia.com/nsight-systems.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CS2JWaziYr.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=8Wuvhh0LYW.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=HVK6nl3i97.

Jiashuo Sun, Xianrui Zhong, Sizhe Zhou, and Jiawei Han. Dynamicrag: Leveraging outputs of large language model as feedback for dynamic reranking in retrieval-augmented generation. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 168661–168692. Curran Associates, Inc., 2025. doi: 10.52202/085713-5620. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/f69d2246a85e28c3e63dee3daf8d4af8-Paper-Conference.pdf.

Zhendong Tan, Xingjun Zhang, Chaoyi Hu, Junjie Peng, and Kun Xia. SpecPV: Improving Self-Speculative Decoding for Long-Context Generation via Partial Verification. *arXiv preprint arXiv:2512.02337*, 2025. URL https://arxiv.org/abs/2512.02337.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911. PMLR, 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

Rishabh Tiwari, Haocheng Xi, Aditya Tomar, Coleman Richard Charles Hooper, Sehoon Kim, Maxwell Horton, Mahyar Najibi, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. QuantSpec: Self-speculative decoding with hierarchical quantized KV cache. In *Proceedings*

of the 42nd International Conference on Machine Learning, volume 267 of *Proceedings of Machine Learning Research*, pp. 59668–59686. PMLR, 2025. URL https://proceedings.mlr.press/v267/tiwari25b.html.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: On-the-Fly Self-Speculative Decoding for LLM Inference Acceleration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EKJhH5D5wA.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=mXpq6ut8J3.

Yue Yu, Wei Ping, Zihan Liu, Boxin Wang, Jiaxuan You, Chao Zhang, Mohammad Shoeybi, and Bryan Catanzaro. RankRAG: Unifying context ranking with retrieval-augmented generation in LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=S1fc92uemC.

Yikang Yue, Yuqi Xue, and Jian Huang. Vegas: Self-speculative decoding with verification-guided sparse attention. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=vht4Pn0swG.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.607. URL https://aclanthology.org/2024.acl-long.607/.

Yifan Zhang and Math AI Team. American invitational mathematics examination (aime) 2025, 2025. URL https://huggingface.co/datasets/math-ai/aime25.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2o: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Juntao Zhao, Wenhao Lu, Sheng Wang, Lingpeng Kong, and Chuan Wu. QSpec: Speculative decoding with complementary quantization schemes. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 4779–4795, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.240. URL https://aclanthology.org/2025.emnlp-main.240/.

Yilong Zhao, Jiaming Tang, Kan Zhu, Zihao Ye, Chi-Chih Chang, Chaofan Lin, Jongseok Park, Guangxuan Xiao, Mohamed Abdelfattah, Mingyu Gao, Baris Kasikci, Song Han, and Ion Stoica. Accelerating large-scale reasoning model inference with sparse self-speculative decoding. In *International Conference on Machine Learning and Systems*, 2026.

Yushu Zhao, Yubin Qin, Yang Wang, Xiaolong Yang, Huiming Han, Shaojun Wei, Yang Hu, and Shouyi Yin. From Quarter to All: Accelerating Speculative LLM Decoding via Floating-Point Exponent Remapping and Parameter Sharing. *arXiv preprint arXiv:2510.18525*, 2025b. URL https://arxiv.org/abs/2510.18525.

A HYPERPARAMETER ANALYSIS

We analyze how five hyperparameters affect the acceptance length and decode throughput of QSpar on RULER 64K with a batch size of 1. The reference configuration uses $\tau = 2$, $K_{\text{rerank}} = 8$, $K_{\text{reducer}} = 2$, $N_{\text{draft}} = 10$, and $M = 2$. Each result is normalized independently to the corresponding reference configuration for each model and sparse attention method. Each table entry reports normalized acceptance length followed by relative decode throughput (*Acceptance Length / Throughput*). The reported measurements are the medians of three runs, and bold rows denote the reference configuration.

A.1 K_{rerank} : RERANKER CANDIDATE-SET SIZE

Table 1: Effect of reranker candidate-set size K_{rerank}. Acceptance length remains close to the reference across $K_{\text{rerank}} = 4$–64. Each cell reports normalized acceptance length followed by relative decode throughput; the bold row is the reference for each model–method pair.

	Qwen3-8B			Llama3.1-8B		
K_{rerank}	Vegas	MagicDec Stream	MagicDec Quest	Vegas	MagicDec Stream	MagicDec Quest
4	0.99/1.01	1.01/0.97	1.00/1.04	1.01/1.02	1.00/1.00	1.00/1.00
8	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**
16	0.99/1.02	1.00/0.99	1.00/1.05	1.00/0.98	1.00/1.01	1.00/0.99
32	0.99/0.99	1.00/0.98	1.00/1.03	1.00/0.97	1.00/1.04	1.00/1.01
64	0.99/1.03	1.00/0.98	1.00/1.04	1.01/0.99	1.00/1.02	1.00/1.00

Table 1 varies the number of candidates rescored by the Reranker. Acceptance remains close to the reference across $K_{\text{rerank}} = 4$–64, indicating that the BF16 target token is typically contained within a small candidate set. Because reranking accounts for only a small fraction of the total execution time, increasing K_{rerank} causes only modest throughput changes across models and sparse attention methods.

A.2 τ : REDUCER MARGIN THRESHOLD

Table 2: Effect of reducer margin threshold τ. At $\tau = 1.5$, throughput improves across all model–method pairs while acceptance remains near or above the reference. Each cell reports normalized acceptance length followed by relative decode throughput; the bold row is the reference for each model–method pair.

	Qwen3-8B			Llama3.1-8B		
τ	Vegas	MagicDec Stream	MagicDec Quest	Vegas	MagicDec Stream	MagicDec Quest
1.0	0.99/1.05	1.12/1.13	0.96/0.96	1.02/1.01	0.99/1.03	1.07/1.09
1.5	0.99/1.04	1.00/1.02	1.00/1.05	1.01/1.05	1.00/1.05	1.06/1.06
2.0	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**

Table 2 varies the reducer margin threshold τ. A larger τ makes more prediction positions eligible for candidate retention, up to the global budget M. This broader retention can increase the chance of preserving the BF16 target token, but may also reduce throughput by increasing candidate-handling and verification overhead. Consistent with this trade-off, lowering τ from 2.0 to 1.5 keeps acceptance close to or above the reference across all model–method pairs while increasing throughput by 1.7%–6.3%. Reducing the threshold further to 1.0 produces greater variation: it substantially improves both metrics for Qwen3-8B with MagicDec-Stream and Llama3.1-8B with MagicDec-Quest, but reduces both metrics for Qwen3-8B with MagicDec-Quest. These configuration-dependent results suggest that selecting τ separately for each model and sparse attention method could further optimize the trade-off between acceptance and throughput.

A.3 M : REDUCER CANDIDATE-POSITION BUDGET

Table 3: Effect of the global candidate-position budget M. Setting $M = 0$ removes additional candidates and lowers both acceptance length and throughput; larger budgets have model- and method-dependent effects. Each cell reports normalized acceptance length followed by relative decode throughput; the bold row is the reference for each model–method pair.

	Qwen3-8B			Llama3.1-8B		
M	Vegas	MagicDec Stream	MagicDec Quest	Vegas	MagicDec Stream	MagicDec Quest
0	0.85/0.81	0.52/0.54	0.87/0.88	0.95/0.80	0.71/0.73	0.97/0.83
1	0.95/0.98	0.90/0.91	0.97/0.99	0.98/0.84	0.73/0.72	1.01/0.96
2	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**
4	1.01/1.02	1.00/0.98	0.99/1.00	1.02/0.95	1.14/1.25	1.07/1.11

Table 3 varies the global candidate-position budget M, which limits the number of uncertain positions that retain an additional candidate. With $K_{\text{reducer}} = 2$, M also equals the maximum number of additional candidate leaves. Reducing M below 2 generally decreases both acceptance and throughput, with the largest degradation observed for MagicDec-Stream. In particular, $M = 0$ disables candidate retention by the Reducer and is therefore equivalent to a reranking-only configuration, resulting in substantial performance losses for MagicDec-Stream. The effect of increasing M beyond 2 depends on whether the resulting acceptance gain offsets the additional candidate-processing overhead. For Llama3.1-8B with MagicDec-Quest, increasing M to 4 improves normalized acceptance from 1.00 to 1.07 and throughput from 1.00 to 1.11, indicating that the acceptance gain outweighs the additional candidate overhead. In contrast, for Llama3.1-8B with Vegas, acceptance increases only to 1.02 while throughput decreases to 0.95, indicating that the additional overhead outweighs the limited acceptance gain.

A.4 K_{reducer} : REDUCER CANDIDATE-SET SIZE

Table 4: Effect of reducer candidate-set size K_{reducer}. With $K_{\text{reducer}} = 1$, removing alternative candidates reduces both metrics, whereas $K_{\text{reducer}} = 4$ preserves or improves them relative to the reference. Each cell reports normalized acceptance length followed by relative decode throughput; the bold row is the reference for each model–method pair.

	Qwen3-8B			Llama3.1-8B		
K_{reducer}	Vegas	MagicDec Stream	MagicDec Quest	Vegas	MagicDec Stream	MagicDec Quest
1	0.85/0.83	0.52/0.54	0.87/0.84	0.97/0.76	0.71/0.69	0.97/0.84
2	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**
4	1.04/1.08	1.27/1.16	1.00/1.03	1.04/1.00	1.34/1.30	1.06/1.03

Table 4 varies the number of candidates retained at each selected uncertain position. Setting $K_{\text{reducer}} = 1$ removes all alternative candidate branches and substantially reduces both acceptance and throughput, particularly for MagicDec-Stream. Increasing K_{reducer} from 2 to 4 preserves or improves both metrics across all evaluated model–method pairs, with the largest gains observed for MagicDec-Stream. This trend is consistent with Figure 2(c), which shows that expanding the candidate set increases the probability of retaining the BF16 target token at flipped positions. However, the cumulative target-token recall in Figure 2(c) largely saturates within the top-ranked candidates. Further increasing K_{reducer} beyond 4 is therefore expected to provide diminishing acceptance gains while increasing branching and verification overhead. Once the per-position candidate set reaches this range, allocating the candidate budget across more uncertain positions may be more effective than retaining additional low-ranked candidates at the same position.

A.5 N_{draft} : DRAFT LENGTH

Table 5: Effect of draft length N_{draft}. Acceptance length generally increases with longer drafts, whereas throughput varies non-monotonically across model–method pairs. Each cell reports normalized acceptance length followed by relative decode throughput; the bold row is the reference for each model–method pair.

	Qwen3-8B			Llama3.1-8B		
N_{draft}	Vegas	MagicDec Stream	MagicDec Quest	Vegas	MagicDec Stream	MagicDec Quest
4	0.49/0.88	0.62/1.10	0.46/0.95	0.49/0.82	0.65/1.33	0.50/1.04
6	0.68/0.98	0.79/1.09	0.65/1.07	0.67/0.95	0.88/1.41	0.69/1.09
8	0.85/1.07	0.93/1.14	0.85/1.06	0.85/0.99	0.93/1.07	0.87/1.06
10	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**	**1.00/1.00**
12	1.14/1.02	1.03/0.93	1.15/0.99	1.11/0.87	0.78/0.67	1.14/0.95
16	1.53/1.14	1.39/0.92	1.50/1.00	1.44/0.94	1.28/0.92	1.46/0.89

Table 5 evaluates QSpar across different numbers of tokens generated along the main draft chain. Acceptance generally increases with draft length because each speculative iteration can accept a longer token sequence. Across $N_{\text{draft}} = 6$–10, throughput remains within 5% of the reference or improves across all evaluated model–method pairs, indicating that QSpar does not depend on a narrowly tuned draft length.

B CANDIDATE-TREE CONSTRUCTION AND VERIFICATION FOR QSPAR

Algorithm 1 describes the complete candidate-tree construction and verification procedure used by QSpar under greedy decoding. At each draft position t, the quantized sparse drafter produces a candidate set $\mathcal{C}_t$ and a hidden state $h_t^{drafter}$. The candidates are reranked by applying the corresponding BF16 LM-head rows to the hidden state produced by the quantized sparse drafter. Let $\widetilde{\mathcal{C}}_t = (\widetilde{c}_{t,1}, \widetilde{c}_{t,2}, \ldots)$ denote the resulting reranked candidate list, and let $\delta_t^{\text{rerank}} = z_{t,1}^{\text{rerank}} - z_{t,2}^{\text{rerank}}$ denote its top-1–top-2 margin.

The reranked top-1 candidates form the main draft chain. After the complete main chain is generated, the Reducer selects up to M positions with the smallest δ_t^{rerank} among those satisfying $\delta_t^{\text{rerank}} < \tau$. For the default configuration, $M = \lfloor 0.2 N_{drafter} \rfloor$; the algorithm treats M as an input to support the budget ablation. For each selected position $t \in \mathcal{U}_M$, the reranked second candidate is added as a leaf child of v_{t-1} and hence as a sibling of the corresponding main-chain node. No descendants are generated from these alternative leaves within the same speculative iteration.

The resulting tree is verified in a single pass using branch-aware attention masks. KV states for the common prefix are shared, while divergent suffixes maintain branch-specific states. Let ℓ_v and KV_v denote the verifier logits and the verifier KV state associated with node v, respectively. Under greedy decoding, a candidate child is accepted if and only if its token matches $\arg\max_w \ell_v(w)$. If the main-chain candidate is rejected but its alternative leaf matches the verifier prediction, the alternative leaf is selected and traversal terminates at that leaf. If no candidate matches the verifier prediction, the verifier's $\arg\max$ token is emitted as a replacement token.

If all candidate tokens along the selected path are accepted, including the case where the selected path ends at an accepted alternative leaf, the verifier emits one additional bonus token from the logits at the terminal node. Only the KV states along the selected accepted path are committed for the next iteration; if rejection occurs, this path ends at the last accepted candidate before the replacement token. States belonging to discarded branches are released. The bonus or replacement token is appended to the output sequence, and its KV state is materialized when the token is consumed in the subsequent iteration.

Here, $\mathrm{Child}(v, y)$ returns the child of v whose token is y, or $\varnothing$ if no such child exists. The path list $\mathcal{S}$ stores the root and the candidate nodes along the selected accepted path. The flag `all_accepted` remains true only if the traversal terminates after accepting every selected candidate token.

Algorithm 1 Candidate-tree construction and verification in QSpar

Require: Prefix $x_{1:m}$, quantized sparse drafter $drafter$, BF16 verifier $verifier$, margin threshold τ, budget M

Ensure: Emitted tokens $\mathcal{P}$ and committed KV states

$v_0 \leftarrow \mathrm{Root}(x_{1:m})$
for $t = 1, \ldots, N_{drafter}$ **do**
  $\mathcal{C}_t \leftarrow \mathrm{TopK}(drafter, v_{t-1})$
  $\widetilde{\mathcal{C}}_t \leftarrow Reranker(\mathcal{C}_t; h_t^{drafter})$
  $\delta_t^{\mathrm{rerank}} \leftarrow z_{t,1}^{\mathrm{rerank}} - z_{t,2}^{\mathrm{rerank}}$
  Add $\widetilde{c}_{t,1}$ as the next main-chain node v_t
end for
$\mathcal{U}_M \leftarrow Reducer\Big(\{\delta_t^{\mathrm{rerank}}\}_{t=1}^{N_{drafter}}, \tau, M\Big)$
for $t \in \mathcal{U}_M$ **do**
  Add $\widetilde{c}_{t,2}$ as a leaf child of v_{t-1} and sibling of v_t
end for
$\mathcal{T} \leftarrow$ the resulting candidate tree
$\{\ell_v, \mathrm{KV}_v\}_{v \in \mathcal{T}} \leftarrow verifier(\mathcal{T})$ ▷ One pass with branch-aware attention
$v \leftarrow v_0$, $\mathcal{P} \leftarrow [\,]$, $\mathcal{S} \leftarrow [v_0]$
`all_accepted` $\leftarrow$ true
while v has candidate children **do**
  $y_v \leftarrow \arg\max_w \ell_v(w)$
  $u \leftarrow \mathrm{Child}(v, y_v)$ ▷ Returns $\varnothing$ if no child has token y_v
  if $u = \varnothing$ **then**
    Append y_v to $\mathcal{P}$ as the replacement token ▷ KV materialized in the next iteration
    `all_accepted` $\leftarrow$ false
    break
  else
    Append $\mathrm{token}(u)$ to $\mathcal{P}$
    $v \leftarrow u$
    Append v to $\mathcal{S}$
    if v is an alternative leaf **then**
      break
    end if
  end if
end while
if `all_accepted` **then**
  $b \leftarrow \arg\max_w \ell_v(w)$
  Append b to $\mathcal{P}$ as the bonus token ▷ KV materialized in the next iteration
end if
Commit $\{\mathrm{KV}_v\}_{v \in \mathcal{S}}$
Discard $\{\mathrm{KV}_v\}_{v \in \mathcal{T} \setminus \mathcal{S}}$
return $\mathcal{P}$ and the committed KV states

C Per-Method Acceptance-Length Recovery

Figure 8 reports the per-method results underlying the averages in Figure 6. The overall trend remains consistent across the three sparse attention methods. Although the magnitude of recovery varies with the KV-selection policy, QSpar recovers much of the acceptance loss from naive quantized sparse drafting in most configurations and frequently approaches or exceeds the BF16 reference. Values above the reference are consistent with the overlap between quantization- and sparsity-induced errors discussed in Section 3.2. These results show that the acceptance recovery observed in Figure 6 is not driven by a single sparse attention method.

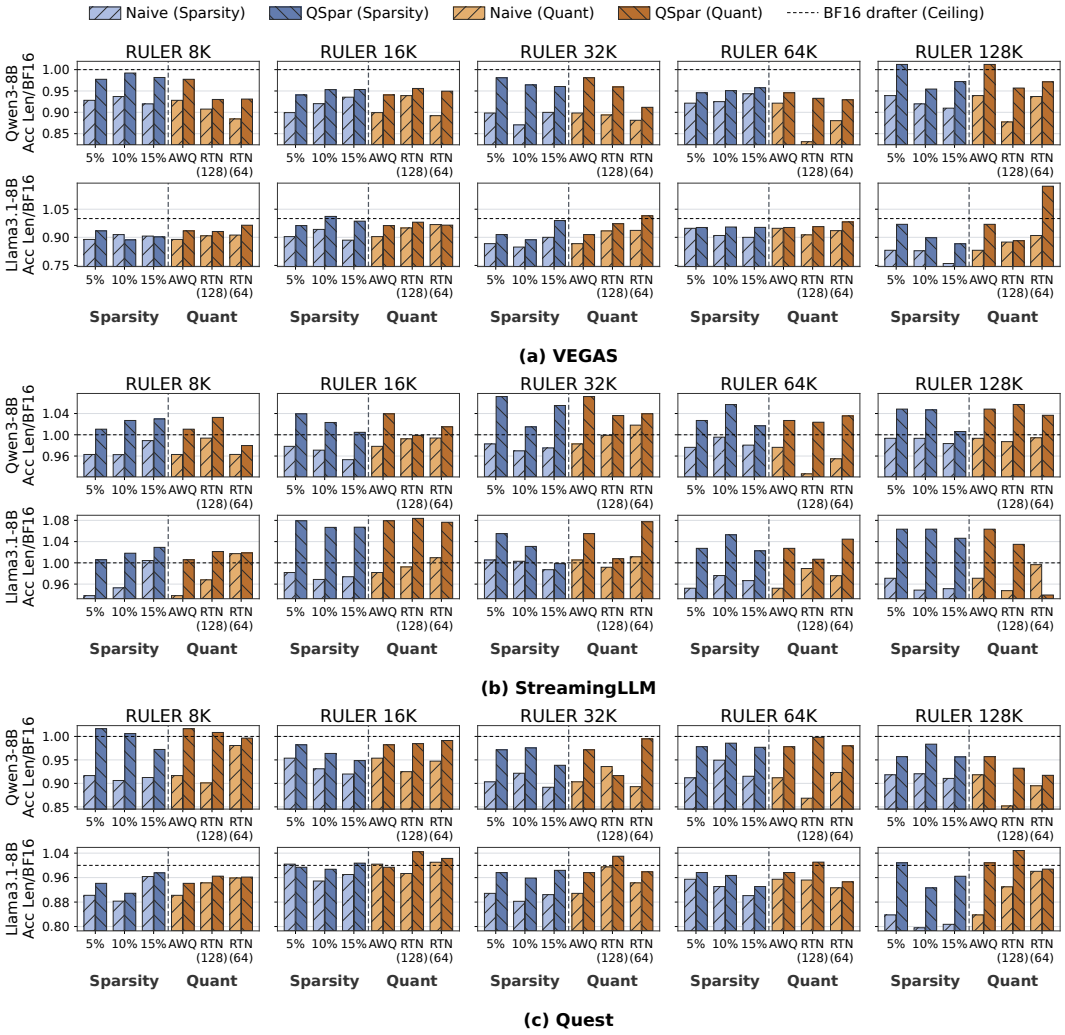

Figure 8: Per-method acceptance-length recovery across sparsity and quantization configurations for (a) Vegas, (b) MagicDec-Stream, and (c) MagicDec-Quest. Results are normalized to the corresponding BF16 drafter for Qwen3-8B and Llama3.1-8B on RULER (8K–128K). The sparsity sweep varies the sparsity ratio from 5% to 15% under AWQ, while the quantization sweep compares AWQ with RTN using group sizes of 128 and 64 at a fixed 5% ratio. Dashed lines denote the BF16 reference.