QSPEC: Speculative Decoding with Complementary Quantization Schemes

Juntao Zhao, Wenhao Lu, Sheng Wang, Lingpeng Kong, and Chuan Wu

{juntaozh, whlu, u3009618}@connect.hku.hk, {lpk, cwu}@cs.hku.hk

The University of Hong Kong

Abstract

Quantization is widely adopted to accelerate inference and reduce memory consumption in large language models (LLMs). While activation-weight joint quantization enables efficient low-precision decoding, it suffers from substantial performance degradation on multi-step reasoning tasks. We propose QSPEC, a novel quantization paradigm that decouples efficiency from quality by integrating two complementary schemes via speculative decoding: low-precision joint quantization for fast drafting and high-precision weight-only quantization for accurate verification. QSPEC reuses both weights and KV cache across stages, enabling near-zero-cost switching without retraining or auxiliary models. Compared to high-precision baselines, QSPEC achieves up to $1.64\times$ speedup without quality degradation, and outperforms state-of-the-art speculative decoding methods by up to $1.55\times$ in batched settings. Furthermore, QSPEC supports plug-and-play deployment and generalizes well across model scales, quantization methods, and workloads. These properties make QSPEC a practical and scalable solution for high-fidelity quantized LLM serving under memory-constrained scenarios. Our code is available at `https://github.com/hku-netexplo-lab/QSpec`.

1 Introduction

Large language models (LLMs) have demonstrated remarkable abilities across various domains, including mathematics, coding, and planning (Shao et al., 2024; Guo et al., 2024a; Huang et al., 2024). Nonetheless, their immense scales pose substantial challenges for deployment due to high memory and computational demands, especially in resource-limited scenarios (*e.g.*, inference on edge devices). Quantization has been an effective compression technique to facilitate LLM inference with limited resources (Lin et al., 2024a; Ashkboos et al., 2024; Zhao et al., 2024b; Lin et al., 2024b). By converting high-precision values (*e.g.*, FP16) into their

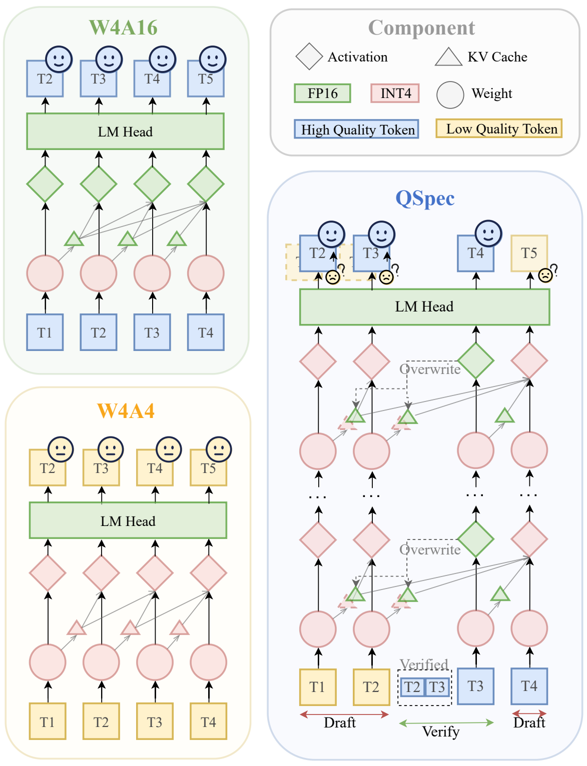


Figure 1: Diagrams of different 4-bit quantization schemes. **W4A16:** uses 4-bit weight and 16-bit activation for inference. **W4A4:** further adopts 4-bit activation to utilize low-precision W4A4 kernels. **QSPEC:** accelerates W4A16 by drafting tokens with W4A4 and verifying them with W4A16, and applies KV cache overwriting for consistent memory consumption.

lower-precision counterparts (*e.g.*, INT4), quantization effectively lowers memory and computational requirements, allowing for larger serving batches and model sizes. Furthermore, the reduced memory footprint boosts token generation throughput by accelerating the typically memory-bound autoregressive decoding process (Zhao et al., 2024a).

Based on the quantized objects, recent quantization algorithms can be broadly classified into two categories: weight-only and WXAX: (1) Weight-only quantization, represented by W4A16 (Lin et al., 2024a), quantizes model weights to low precision (*e.g.*, 4-bit) for storage, and then dequantizes them to a higher precision (*i.e.*, FP16) during inference; (2) WXAX methods, such as W4A4 (Ashkboos et al., 2024; Zhao et al., 2024b) and W8A8 (Xiao et al., 2023), simultaneously quantize both weights and activations, and leverage low-precision hardware support for faster execution without dequantizing them to higher precision. Nevertheless, WXAX schemes generally suffer model performance degradation due to more low-precision activations used (as verified in Sec. 2). This poses a tough trade-off between efficacy and efficiency, raising the question:

"Is there a quantization solution that boosts efficiency while avoiding efficacy degradation?".

Considering the comparable performance claims on recent W4A4 methods (Zhao et al., 2024b; Ashkboos et al., 2024), we first contend that their conclusions are biased due to limited evaluation tasks, and W4A4 still experiences significant performance drops when compared to their higher-precision activation counterparts. Specifically, while W4A4 schemes such as Atom (Zhao et al., 2024b) and QuaRot (Ashkboos et al., 2024) perform well on general tasks, such as PIQA (Bisk et al., 2020), Winogrande (Sakaguchi et al., 2019) and ARC (Clark et al., 2018), they demonstrate notable performance declines in multi-step reasoning, particularly on mathematical and coding benchmarks (Xiong et al., 2024; Guo et al., 2024b) (shown in Table 1). This raises concerns about the comprehensiveness of evaluation and emphasizes the necessity of incorporating multi-step reasoning tasks into quantization assessment.

To answer the above question, we draw inspiration from speculative decoding (Leviathan et al., 2023; Chen et al., 2023), which combines rapid drafting of a small model with high-fidelity verification from a larger model to boost throughput without sacrificing output quality. We propose a novel paradigm called QSPEC, which employs complementary mixed-precision quantization execution to deliver **flawless acceleration** for high precision quantization. (*i.e.*, **efficiency improvements** that preserve the fidelity and memory overhead of high-precision quantization).

Our key insight is that a single weight-quantized model can losslessly toggle between two parallel activation modes: a fast, low-precision mode for drafting (*e.g.*, W4A4) and a high-precision mode for verification (*e.g.*, W4A16). As validated in Sec. 2.2, the token generation with different modes is highly similar. This allows us to adopt a lightweight 'draft-verify' scheme, as shown in Figure 1, where tokens drafted with W4A4 are selectively accepted by W4A16 with negligible switching costs. Unlike traditional speculative decoding that requires a draft model, QSPEC shares weights and KV cache across activation modes, achieving zero extra memory overhead.

We evaluate QSPEC on a range of model sizes, quantization methods, and batch sizes. Compared to W4A16, QSPEC offers up to $1.64\times$ higher token generation throughput while preserving fidelity. **It also effectively compensates for up to 51.11% quality loss observed in W4A4 on challenging multi-step reasoning tasks such as MATH (Hendrycks et al., 2021).** Furthermore, QSPEC requires no training and can be directly integrated into existing inference pipelines.

Our main contributions are summarized as follows:

- We demonstrate that multi-step reasoning tasks are more sensitive to quantization-induced quality degradation than standard benchmarks, and advocate their inclusion for more comprehensive evaluation.
- We validate and instantiate the feasibility of switching between two quantization schemes of a shared weight-quantized model, as well as their high token-level similarities, illuminating future development of quantization schemes.
- We introduce QSPEC, the first quantization paradigm that decouples efficiency from quality by combining complementary quantization schemes through speculative decoding with shared weights and KV cache.
- Experiments across various models and tasks reveal that QSPEC achieves up to $1.64\times$ acceleration without quality loss, making it well-suited for high-fidelity deployment in memory-constrained scenarios.

2 Motivation

2.1 Compromised Performance of Activation Quantization

State-of-the-art (SOTA) activation-weight joint quantization methods, like Atom (Zhao et al., 2024b) and QuaRot (Ashkboos et al., 2024), achieve notable speed-ups with negligible performance loss compared to weight-only ones. However, we argue that this conclusion is skewed by

Table 1: Performance of Atom-based quantization schemes with different weight and activation precision across diverse tasks. "Acc", "PPL" and "EM" stand for accuracy, perplexity, and exact match, respectively, with arrows indicating their positive trends. "W16A16" refers to standard FP16 inference, where both weights and activations are represented in FP16 precision.

Task	Metric	W16A16	Quantization	
			Atom (W4A16)	Atom (W4A4)
WikiText-2	PPL ↓	7.73	7.87 (+0.15%)	8.58 (+0.85%)
PIQA (10-shot)	EM ↑	78.6	77.5 (-1.40%)	75.6 (-3.81%)
MBPP (0-shot)	EM ↑	42.0	41.5 (-1.19%)	30.5 (-27.38%)
GSM8K (8-shot)	EM ↑	79.0	73.4 (-7.09%)	54.2 (-31.39%)

limited evaluation benchmarks, which fail to capture the negative impacts of activation quantization.

To substantiate this claim, we conduct experiments on Llama-3-8B-Instruct models (Dubey et al., 2024) quantized with W16A16, W4A16, and W4A4 across four benchmarks: PIQA (Bisk et al., 2020), WikiText-2 (Merity et al., 2016), GSM8K (Cobbe et al., 2021), and MBPP (Austin et al., 2021). While PIQA and WikiText-2 are commonly used in quantization evaluation, GSM8K and MBPP involve multi-step reasoning, which remains underexplored in the quantization context despite its great importance. Detailed descriptions of the benchmarks are provided in Appendix B.

As listed in Table 1, Atom-based quantization schemes show comparable performance to W16A16 across commonly adopted tasks such as on PIQA and WikiText-2, aligning with the claims in Zhao et al. (2024b). However, W4A4 suffers a nearly 30% average performance decline on complex reasoning tasks (*i.e.*, MBPP and GSM8K), whereas W4A16 only experiences about 4%. This indicates that activation quantization leads to several times more performance degradation on multi-step reasoning tasks, despite the improved efficiency. Besides, the performance trend observed on multi-step reasoning tasks shows a stronger correlation with quantization precision than perplexity does, validating their adequacy in evaluation.

In summary, activation quantization still incurs significant performance loss on more advanced multi-step reasoning tasks. This necessitates the inclusion of them in quantization evaluation for a more comprehensive assessment. Furthermore, this also underscores the demand for a quality-preserving yet efficient quantization paradigm.

2.2 High-Similarity Token Predictions

Despite the notable performance decline caused by activation quantization, we observe, more microscopically, high similarity in top-1 token predictions between quantization schemes with high and low precision activations. Specifically, we first employ Atom-based W4A16 greedy sampling to generate the golden token sequences for the GSM8K test set, obtaining the prediction probabilities for each top-1 answer token. Subsequently, we perform one Atom-based W4A4 forward pass (*i.e.*, prefill) on the concatenated input of each question and its corresponding golden answer to acquire the token probabilities as well. This allows us to assess the prediction discrepancy between W4A4 and W4A16. As illustrated in Figure 2, we observe that (1) the majority of token prediction probabilities of both W4A4 and W4A16 exceed 80%, and most of the tokens associated with high probabilities are accepted. (2) Compared to accepted tokens, the number of rejected ones is negligible, underscoring the high similarity between the two quantization methods.

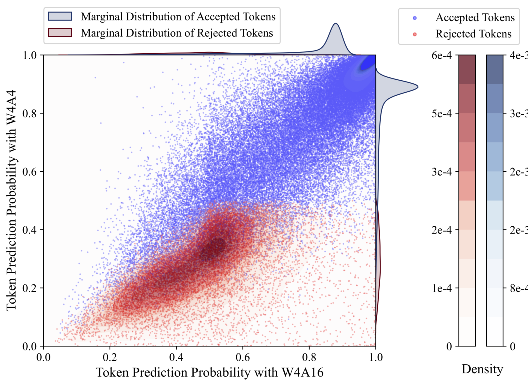


Figure 2: Scatter plot of token prediction probabilities for Atom-based W4A4 and W4A16 on GSM8K test set, along with their two-dimensional and marginal probability distributions. A striking similarity between the two quantization schemes is observed, laying the foundation of QSPEC.

Combined with the analysis in Sec. 2.1, this can be interpreted that a small set of salient token variations can trigger a snowball effect of errors, especially on multi-step reasoning tasks where the subsequent steps are closely conditioned on the previous ones, akin to findings in (Zhang et al., 2023), thus impairing the performance of the low-precision activation scheme. Prior studies indicate that low similarity leads to frequent token rejections, thereby diminishing the efficiency of speculative decoding (Leviathan et al., 2023). The high token-level similarity we observe implies that generating high-quality outputs may only require detecting and correcting a limited number of activa-

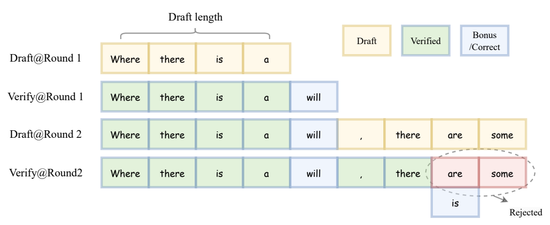


Figure 3: A mini-sample of QSpec, where yellow, red, and blue tokens represent W4A4 draft tokens, rejected tokens, and tokens generated directly by W4A16, respectively. While these green ones are draft tokens that have been verified and accepted by W4A16.

tion quantization-induced errors. This insight motivates our proposal of a quantization-aware speculative decoding framework that leverages token generation similarity.

3 Method

Targeting an efficient quantization scheme without sacrificing performance or increasing memory consumption, we propose a new quantization paradigm called **Spec**ulative decoding with complementary **Q**uantization execution (QSpec). As shown in Figure 1 and Figure 3, QSpec employs a draft-verify pipeline for next-token prediction with varying activation precisions and shared low-precision quantized weights, instead of a single quantization scheme. Only quantization schemes are switched, and no additional weights are incorporated in this process.

3.1 QSpec

Draft Phase. Current LLMs typically utilize an autoregressive process for next-token prediction. A new token is drawn from a probability distribution conditioned on all previously generated tokens. This process can be formulated as:

$$t_{i+1} \sim p_{i+1}(t) := \mathcal{M}(t_{i+1}|T_{\leq i}), \qquad (1)$$

where $\mathcal{M}$ denotes the model including the weight and activation configurations, while t_{i+1} and $T_{\leq i}$ represent the next predicted token and the preceding token sequence $(t_0, t_1, \ldots, t_i)$, respectively.

Compared with previous research (Leviathan et al., 2023; Chen et al., 2023), we employ a weight-shared quantization scheme with low-precision activations, rather than a standalone small-sized model, to speculate the next γ tokens $\hat{T}_{i+1:i+\gamma}$ and their associated distributions $\hat{p}_{i+1:i+\gamma}(t)$. In $\hat{T}_{i+1:i+\gamma}$, each token $\hat{t}_j$ is sampled from $\mathcal{M}_l(\hat{t}_j|T_{\leq i}, \hat{T}_{i+1:j-1})$, where $j \in [i+1, i+\gamma]$ and $\mathcal{M}_l$ represents our quantized model executed with low-precision activation. Thanks to the reduced activation precision, this scheme enables fast token generation.

Verify Phase. To compensate for the performance decline incurred by excessive quantization, we employ a high-precision weight-only quantization scheme to verify the proposed draft token sequence. This ensures that the final generation quality aligns with that of a high-precision activation quantization scheme. All drafted tokens are verified in parallel for higher efficiency.

Formally, the high-precision quantization scheme $\mathcal{M}_h$ receives as input the concatenation of $T_{\leq i}$ and $\hat{T}_{i+1:i+\gamma}$, producing high-quality prediction probabilities $p_{i+1:i+\gamma+1}(t)$ through a single forward pass. Following this, an acceptance policy $\mathcal{A}$, which will be detailed later, is applied to rectify each drafted token sequentially. Once a token $\hat{t}_{i+j}$ is rejected, all subsequent tokens are discarded, and token t_{i+j} is resampled according to the distribution $p_{i+j}(t)$. In the optimal scenario, all drafted tokens from the low-precision quantized model are accepted by the high-precision model. Subsequently, an additional token $t_{i+\gamma+1}$ is sampled from $p_{i+\gamma+1}(t)$. From this point, a new draft-verify cycle commences, persisting until the sequence is finalized.

Acceptance Policy. To maintain high reproducibility, both low-precision and high-precision activation quantization schemes employ greedy decoding. This means that one drafted tokens $\hat{t}_{i+j}$ is accepted as t_{i+j} only when the top-1 tokens from p_{i+j} and $\hat{p}_{i+j}$ coincide; otherwise, this token is rejected. Nonetheless, we claim that alternative strategies, as outlined in (Leviathan et al., 2023), can be directly applied to our method due to the similarities in the framework. Figure 3 illustrates a mini-sample of this cycle with the draft token length $\gamma = 4$. The model initially speculates four tokens using the W4A4 scheme. Subsequently, adhering to a predefined acceptance policy, it accepts all drafted tokens after verifying them through the W4A16 scheme. In the second loop, however, only the first two tokens are accepted. A new token "is" is directly derived from the prediction probability of the W4A16 scheme, and another draft-verify cycle will commence from the ninth token.

KV Cache Overwriting. A key advantage of QSpec lies in its shared-weight architecture,

Table 2: Comparison of individual quantization schemes, regular speculative decoding, and QSPEC across memory, computation, and generation aspects.

Method	Memory		Computation		Generation	
	Draft Weight	Draft KV	W4A4 Kernel	Draft-Verify	High Acceptance Rate	High Fidelity
W4A16	✗	✗	✗	✗	-	✓
W4A4	✗	✗	✓	✗	-	✗
Speculative Decoding	✓	✓	?	✓	?	✓
QSpec (no-overwrite)	✗ (1x)	✓ (1.25x)	✓	✓	✗ (0.8x)	✓
QSPEC	✗ (1x)	✗ (1x)	✓	✓	✓ (1x)	✓

which naturally aligns the behavior of low- and high-precision activations. This allows the high-precision verification stage to produce activation patterns and KV cache values that serve as high-fidelity substitutes for those generated during the low-precision draft stage. Leveraging this alignment, QSPEC overwrites the lower-quality KV caches from W4A4 with accurate A16 caches from W4A16 for accepted tokens, enabling subsequent decoding to benefit from higher-quality context. This design not only boosts token acceptance rates but also sets QSPEC apart from prior speculative decoding methods, which use separate models and cannot reuse KV representations. By sharing weights and reusing KV caches within a single model, QSPEC eliminates the need for dual cache maintenance, reducing memory usage without sacrificing accuracy.

As shown in Table 2, we compare QSPEC against individual quantization configurations (*i.e.*, W4A4 and W4A16) and speculative decoding in terms of memory, computation, and generation. QSPEC provides four key benefits: **(1) Memory-Efficient.** By sharing weights and overwriting KV caches, QSPEC incurs memory costs on par with standalone high-precision activation quantization without any memory overhead caused by speculative decoding. **(2) No Efficiency–Efficacy Trade-off.** QSPEC leverages speculative decoding to boost efficiency while preserving output quality, avoiding the compromises in conventional quantization. **(3) Plug-and-Play Compatibility.** QSPEC requires only an acceptance policy and a cache-overwriting step, requiring no additional training or classifiers. Hence, it can be quickly integrated into existing quantization models. **(4) High Acceptance Rate.** Common weights and KV cache overwriting ensure consistent predictions, leading to a high rate of token acceptance.

3.2 Advantages of High Acceptance Rate

A key advantage of QSPEC lies in its superior acceptance rate. While state-of-the-art speculative decoding approaches such as EAGLE (Li et al., 2024b) and Medusa (Cai et al., 2024) rely on distilled draft models and tree-structured drafting to balance speed and accuracy, QSPEC achieves high performance without such compromises. We analyze how QSPEC demonstrates superior performance compared to these methods in quantized settings below.

Expected average accepted token number. Let $p_a(t)$ be the probability of accepting a token t. The average accepted token number is given by Equation 2, where k controls the tree's branching factor (width) . Larger k indicates larger expected token acceptance. When $k = 1$, this method reduces to standard speculative decoding.

$$H(k) := \sum_{l}^{\gamma} \sum_{i=1}^{k^l} \prod_{j \in Path(r,t_i)} p_a(a_j). \tag{2}$$

Cost Analysis. Denote by $C_d(\cdot)$ and $C_p(\cdot)$ the computation cost for draft and verify, The first argument indicates the number of sequences in a draft tree, and the second argument is the prefix sequence length. For a prefix with length s, we have *per-valid-token* cost v by dividing the total cost by the number of accepted tokens in Equation:

$$v = \frac{\overbrace{C_d(1;s) + \cdots + C_d\left(k^{\gamma-1}; s+\gamma-1\right)}^{\text{Drafting cost}} + \overbrace{C_p\left(k^{\gamma-1}; s+\gamma\right)}^{\text{Verify cost}}}{\underbrace{H(k)}_{\text{Accepted tokens}}} \sim \frac{C(k)}{H(k)}. \tag{3}$$

Batch serving efficiency of QSPEC. The efficacy of speculative decoding relies on the assumption that verification cost C_p approximates the target model's decoding cost C_d, leveraging underutilized resources for parallel verification.(Chen et al., 2023; Leviathan et al., 2023; Yan et al., 2025) This holds in single-request, memory-bound scenarios. However, tree-structured drafting, with verification cost $C_p(k^{\gamma-1}; s+\gamma)$, scales poorly as $k^{\gamma-1}$ far exceeds non-tree sequence length γ. Therefore, in batched serving, tree-based C_p enters compute-bound territory easily, significantly surpassing C_d and incurring high memory overhead, invalidating the assumption. QSPEC, using W4A4 quantization, avoids tree structures, maintaining low C_p and memory usage and balancing the loads between drafting and verification stages, QSPEC delays the onset of this inefficiency. In Section 4, QSPEC demonstrates excellent speedup even at a batch size of 32, also outperforming EAGLE (Li et al., 2024b) in comparison experiments.

4 Experiments

Our evaluation answers three key questions:

Q1: Does QSPEC preserve the quality of high-precision weight-only quantization? (Sec. 4.2)

Q2: Does QSPEC speed up high-precision weight-only quantization methods, and surpass speculative decoding methods in quantized scenarios? (Sec.4.3)

Q3: How do various factors (*e.g.*, quantization method, sequence length) influence the acceptance rate and acceleration performance of QSPEC? (Sec. 4.4)

4.1 General Setup

Benchmarks. We assess QSPEC with two primary criteria: (1) generation fidelity and (2) end-to-end serving speedup. For fidelity evaluation, we adopt not only traditional tasks, including PIQA (500, 10-shot) (Bisk et al., 2020), WinoGrande (500, 5-shot) (Sakaguchi et al., 2019), and WikiText2 (Merity et al., 2016), but also challenging multi-step reasoning tasks such as GSM8K (All, 8-shot) (Cobbe et al., 2021), MATH (All, 4-shot) (Hendrycks et al., 2021), MBPP (200, 0-shot) (Austin et al., 2021), and HumanEval (All, 0-shot) (Chen et al., 2021). To measure the acceleration, we use all the above reasoning tasks and two additional chatbot datasets, namely ShareGPT (RyokoAI, 2021) and LMsys-1K (Zheng et al., 2023a). Due to space constraints, we present results for GSM8K, HumanEval, and LMsys-1K in the main text, with remaining results detailed in the Appendix A. Following the setup of Atom (Zhao et al., 2024b), we randomly sampled the dataset for the request prompts to reduce the workload. Due to memory limitations, we vary the batch size from 8 to 32 and serve all requests in a first-come, first-served (FCFS) manner. Once any request is finished, we refill the batch, adhering to the continuous batching approach of ORCA (Yu et al., 2022). All experiments use greedy sampling for token generation.

Models. To assess the effectiveness and scalability of our approach, we conduct experiments using multiple models from the Llama family (Touvron et al., 2023; Dubey et al., 2024) with varying scales and capacities: Llama3.2-3b, Llama2-7b, Llama3-8b-instruct, and Llama2-13b.

Implementations. All experiments are performed on a node equipped with four NVIDIA L20 GPUs (48GB HBM each) running CUDA 12.5. To demonstrate the versatility of QSPEC, we implement two SOTA 4-bit quantization methods, namely Atom (Zhao et al., 2024b) and QuaRot (Ashkboos et al., 2024). For W4A16 configurations, we incorporate AWQ-style (Lin et al., 2024a) weight dequantization logic for runtime inference. We select Atom to showcase the acceleration of QSPEC. We use these Group-wise quantization schemes with a group size of 128. We configure the default draft token length γ to 3. The implementation of QSPEC follows Atom's setup, incorporating flashinfer (Ye et al., 2025).

Baselines. We evaluate QSPEC against baseline quantization configurations: W4A4, W4A16, and W16A16. We also include EAGLE (Li et al., 2024b), which employs a pruned draft model with tree-structured speculative decoding, as a baseline to compare with regular speculative decoding. In our quantized experiments, we utilize FP16 precision for the EAGLE draft model and EAGLE-Quant (W4A16) for the target model. This choice was necessitated by two factors: (1) the official EAGLE quantization implementation (fast-gpt) lacks efficient batching support, and (2) applying GPTQ quantization to the EAGLE draft model resulted in substantial degradation of the acceptance rate.

4.2 Fidelity Evaluation

QSPEC effectively maintains the generation quality of W4A16, whereas W4A4 does not. As listed in Table 3, with the draft verification of W4A16, QSPEC exhibits only minimal performance fluctuations compared to W4A16. This negligible variation may stem from the non-deterministic algorithms of PyTorch (PyTorch Contributors, 2024) or occasional cases where two tokens have the same maximum prediction probability. In contrast, W4A4 experiences a substantial performance decline exceeding 10% across most tasks, with the reduction becoming more pronounced as task difficulty increases. For instance, compared to GSM8K and MBPP, the performance drop for W4A4 is much greater on the more challenging MATH and HumanEval tasks, showing declines of 51.11% and 38.73%, respectively. On the other hand, this also highlights the higher sensitivity of multi-step reasoning tasks to the negative effects of quantization compared to regular tasks, such as WikiText-2and WinoGrande. This observation fully aligns with our earlier analysis in Sec. 2, encouraging the incorporation of multi-step reasoning tasks into quantization evaluation.

Table 3: Performance of different quantization methods across multiple general and reasoning benchmarks: PIQA, WinoGrande, GSM8K, MATH, MBPP, and HumanEval. The quality degradation ratio is calculated by $\frac{W4A4}{W4A16} - 1$.

Method	Quantization	WikiText-2 PPL ↓	PIQA EM (%) ↑	WinoGrande EM (%) ↑	GSM8K EM (%) ↑	MATH EM (%) ↑	MBPP Pass@1 (%) ↑	HumanEval Pass@1 (%) ↑
Atom	W16A16	7.73	76.8	61.4	76.2	24.9	42.5	53.0
	W4A16	7.87	74.8	62.0	73.4	24.3	42.0	52.4
	QSPEC	7.87	75.0	62.0	73.4	24.3	40.5	52.4
	W4A4	8.6 (+9.58%)	65.8 (-12.03%)	56.2 (-9.35%)	54.7 (-25.47%)	15.5 (-36.21%)	33.0 (-21.43%)	31.7 (-39.50%)
QuaRot	W16A16	7.73	76.8	61.4	76.2	24.9	42.5	53.0
	W4A16	8.58	74.2	59.4	70.5	24.7	40.0	45.7
	QSPEC	8.58	74.4	59.2	71.0	24.7	40.5	47.6
	W4A4	10.2 (+19.24%)	62.6 (-15.63%)	53.8 (-9.43%)	42.0 (-40.43%)	12.3 (-51.11%)	28.5 (-28.75%)	28.0 (-38.73%)

Table 4: Comparison of token generation throughput across different model sizes, quantization configurations, and batch sizes for various datasets. All values are measured in token/s. "Avg." denotes the average speedup ratio for the corresponding row or column.

Model	Method	Batch	GSM8K	HumanEval	LMsys-1k
3B	W4A4	8	804.7	892.6	990.3
		16	1109.1	1289.8	1581.0
		32	1424.3	1488.2	2194.4
	W4A16	8	420.0	535.7	559.8
		16	578.5	804.4	925.8
		32	726.3	954.4	1336.4
	QSPEC	8	594.1 (1.41×)	723.6 (1.35×)	738.8 (1.32×)
		16	811.5 (1.40×)	1042.1 (1.30×)	1171.4 (1.27×)
		32	1030.4 (1.42×)	1248.5 (1.31×)	1576.0 (1.18×)
		Avg.	1.41×	1.32×	1.25×
7B	W4A4	8	349.5	471.2	419.4
		16	496.6	749.5	642.6
		32	620.0	1043.9	865.5
	W4A16	8	165.0	240.2	220.2
		16	231.8	407.3	358.0
		32	268.9	555.9	470.1
	QSPEC	8	253.7 (1.54×)	350.9 (1.46×)	310.3 (1.41×)
		16	359.8 (1.55×)	555.2 (1.36×)	473.1 (1.32×)
		32	441.8 (1.64×)	749.4 (1.35×)	628.4 (1.34×)
		Avg.	1.58×	1.39×	1.36×
13B	W4A4	8	194.7	261.5	228.2
		16	288.3	424.9	348.4
		32	369.8	665.4	508.8
	W4A16	8	94.8	140.0	127.9
		16	136.1	236.9	207.2
		32	207.5	365.5	287.4
	QSPEC	8	148.2 (1.56×)	201.2 (1.44×)	174.0 (1.36×)
		16	212.8 (1.56×)	323.3 (1.36×)	266.9 (1.29×)
		32	266.6 (1.28×)	483.0 (1.32×)	379.3 (1.32×)
		Avg.	1.47×	1.37×	1.32×

Table 5: Performance comparison of EAGLE-Quant, QSPEC, W4A16, and W4A4 on Llama-2-7b-chat-hf across different batch sizes and benchmarks. "OOM" indicates out-of-memory. Better cases for QSPEC or EAGLE is marked in gray. In the case of batch size=8, the speedup ratio of QSPEC compared to EAGLE is indicated in parentheses next to the data points.

Method	Batch Size	GSM8K (8-shot)	HumanEval (0-shot)	LMsys-1k
EAGLE	1	65.81	49.15	71.29
	8	140.16	136.86	167.57
	16	OOM	OOM	OOM
QSPEC	1	51.25	54.22	56.14
	8	208.95 (1.49×)	185.99 (1.36×)	260.48 (1.55×)
	16	292.82	255.11	463.35
W4A16	1	59.80	72.04	72.27
	8	146.34	163.54	213.66
	16	190.09	211.49	371.38
W4A4	1	64.79	73.47	71.55
	8	284.84	256.54	393.12
	16	401.77	330.71	713.67

4.3 Acceleration Evaluation

QSPEC exhibits a substantial efficiency boost compared to W4A16. In Table 4, we present the token generation throughput for both QSPEC and W4A16 across different model sizes, quantization configurations, and batch sizes on diverse datasets. On average, QSPEC achieves a throughput increase of 1.38× over W4A16 across all settings, with a peak improvement of 1.64×.

Larger models tend to yield better speedup ratios. We observe a consistent acceleration trend as the base model scales, demonstrating the promising scalability of our approach with larger models. While further validation is needed, resource constraints necessitate addressing this in future work.

QSPEC reduces latency through fast drafting and parallel verifying. As illustrated in Figure 4, we compute the per-valid-token latency by dividing the total latency by only the number of accepted tokens before averaging on all evaluation datasets. Notably, QSPEC achieves remarkable latency savings ranging from 26.5% to 30.6%. Besides, the per-token latency is further decomposed into two parts: draft and verify. Clearly, the primary gains of QSPEC arise from the rapid drafting capability and the reduced latency achieved through the parallel verification of multiple tokens.

QSPEC offers better memory and batching

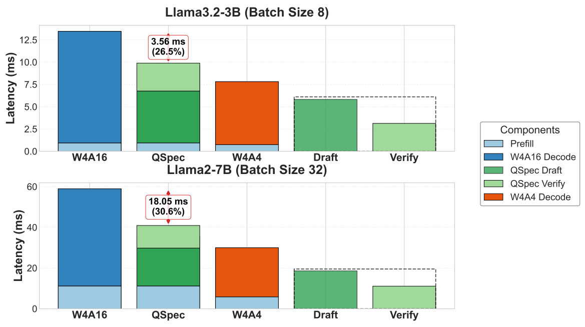


Figure 4: Per-valid-token latency decomposition for different methods. The latency of QSPEC is further decomposed into draft and verify categories for details.

efficiency than prior speculative decoding methods. Table 5 compares QSPEC and EAGLE on Llama-2-7b-chat-hf. EAGLE delivers optimal performance for single-sequence inputs (batch size=1). However, its efficiency degrades as the batch size grows (8 and 16). This observation aligns with our earlier analysis in Sec. 3.2: EAGLE's tree-structured drafting mechanism, designed to reconcile discrepancies between the draft and target models, introduces additional latency, and reduces the gains from higher acceptance rates in batched serving under quantization. Furthermore, the increasing key-value (KV) storage of EAGLE's draft model leads to out-of-memory (OOM) issues at batch size 16. In contrast, QSPEC demonstrates superior scalability and memory efficiency.

QSPEC excels in real-world deployment. We integrated QSPEC into vLLM(Kwon et al., 2023) to validate its performance in real-world serving scenarios. Despite a suboptimal implementation, our experiments demonstrate an average speedup of 1.24×, with effective acceleration even at a batch size of 32. Details are provided in Appendix A.4.

4.4 Ablation Study

Ablation on draft token length. To assess parameter sensitivity, we vary the draft token lengths γ, the only hyper-parameter of QSPEC, from 2 to 6 across all the benchmarks using Llama3.2-3b and Llama3-8b-instruct models. As depicted in Figure 5, an increase in γ leads to a gradual decline in the token acceptance rate, since all subsequent tokens are discarded once a token is rejected. Nevertheless, even at $\gamma = 6$, the token acceptance rate remains relatively high, approximately 74%, compared to $28 \sim 58\%$ in 160m-7b draft-target model pair under $\gamma = 5$ in conventional speculative decoding (Liu et al., 2024). Additionally, a

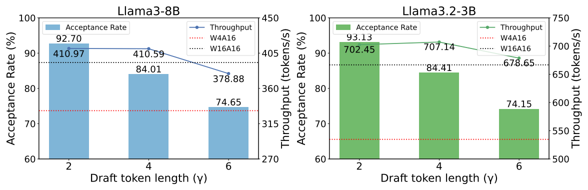


Figure 5: Acceptance rate and throughput of Llama3.2-3b (batch size 8) and Llama3-8b-instruct (batch size 16) with respect to the draft token length γ.

consistent improvement in throughput is observed compared to W4A16, indicating the robustness of QSPEC with respect to γ. More comprehensive comparison is provided in Appendix A.2.

Ablation on base quantization method. As shown in Appendix Table 9, QSPEC consistently achieves high acceptance rates across diverse quantization methods and datasets.

5 Related Work

This work builds on two lines of research: quantization and speculative decoding. Weight-only quantization (W4A16) offers better accuracy, while joint weight-activation methods (W4A4) enable faster inference but degrade performance on complex tasks. Speculative decoding improves efficiency by verifying drafted tokens, but existing approaches typically require retraining and are less effective under quantization. We provide detailed comparisons in Appendix D.

6 Conclusion

We begin by showing that multi-step reasoning tasks reveal performance degradation from activation quantization more clearly than current evaluation protocols, and encourage their incorporation for more comprehensive assessment. Leveraging high token-level similarities, we propose QSPEC, a novel quantization paradigm that seamlessly combines two complementary weight-shared quantization schemes through speculative decoding. Empirically, QSPEC delivers significant acceleration—up to 1.64×—without quality loss across diverse settings. With consistent memory usage and a plug-and-play design, QSPEC offers a practical and scalable solution for high-fidelity quantization, especially under memory constraints.

Acknowledgements

This work was supported in part by grants from Hong Kong RGC under the contracts 17204423, 17205824, C7004-22G (CRF), C5032-23G (CRF) and T43-513/23-N (TRS).

7 Limitations and Impacts

7.1 Impact Statement

This paper introduces QSPEC, a novel quantization paradigm that synergies complementary quantization schemes through speculative decoding to enhance computational efficiency while preserving model fidelity. The impact of QSPEC is twofold.

From an academic perspective, QSPEC establishes a new paradigm that decouples efficiency from quality preservation—a longstanding trade-off in prior quantization research. This is achieved by the complementary quantization schemes: a low-precision activation-weight joint quantization for fast token drafting, and a high-precision weight-only quantization for accurate verification, enabling the independent optimization for efficiency and quality. This illuminates the pursuit of extreme efficiency in quantization schemes (*e.g.*, W4A4) without the concern of performance degradation.

For industry applications, QSPEC provides a practical solution to accelerate inference without compromising output quality through efficient low-precision kernels. This prompts hardware vendors to reconsider their architectural support for low-precision execution, including specialized instruction set architectures (ISAs) and memory subsystems. Besides, the plug-and-play property of QSPEC further facilitates seamless integration into existing deployments of quantized models in memory-constraint scenarios (*e.g.*, edge devices).

Our research, focused on improving the computational efficiency of language model serving systems, is not anticipated to have direct negative social impact.

7.2 Limitation Discussion

The superior performance of QSPEC relies on the high acceptance rate, particularly in small to moderate batch-size scenarios where the throughput gap between low- and high-precision quantization becomes pronounced. In contrast, traditional tree-based speculative decoding methods falter in batch serving, as discussed in Sec. 4, making QSPEC's advantages most evident in these settings. However, QSPEC exhibits potential limitations in single-request scenarios, where other methods are preferentially optimized. With the popularity of LLMs and increasing batch serving, regular speculative decoding methods, including Medusa and EAGLE, degrade significantly (*e.g.*, Medusa's speedup drops below 1 at batch size of 16 in their Figure 22 (Cai et al., 2024)), whereas QSPEC excels.

To further address the limitations of QSPEC in single-request scenarios, future research will focus on leveraging its high acceptance rate and reducing the overhead of the draft stage. Specifically, we aim to develop adaptive mechanisms that dynamically adjust the draft model's sparsity, balancing latency and acceptance rate to achieve robust performance across both single-request and batch-size settings. Additionally, exploring hardware-aware optimizations, such as tailored low-precision kernels for resource-constrained devices, will enhance QSPEC's applicability in edge deployments. Furthermore, integrating QSPEC into popular repositories (*e.g.*, vLLM[1]) is also part of our future work. By unifying these advancements, QSPEC can evolve into a versatile quantization framework, delivering consistent acceleration and fidelity for diverse inference scenarios, from personal devices to large-scale serving systems.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *Preprint*, arXiv:2305.13245.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *Preprint*, arXiv:2404.00456.

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

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *Preprint*, arXiv:2401.10774.

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

[1]`https://docs.vllm.ai/en/latest/`

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, and 39 others. 2021. Evaluating large language models trained on code.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. 2024. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *Preprint*, arXiv:2402.12374.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

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. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Yuntian Deng, Wenting Zhao, Jack Hessel, Xiang Ren, Claire Cardie, and Yejin Choi. 2024. Wildvis: Open source visualizer for million-scale chat logs in the wild. *Preprint*, arXiv:2409.03753.

Shichen Dong, Wenfang Cheng, Jiayu Qin, and Wei Wang. 2024. Qaq: Quality adaptive quantization for llm kv cache. *ArXiv*, abs/2403.04643.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. *ArXiv*, abs/2404.16710.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024a. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Hongyi Guo, Zhihan Liu, Yufeng Zhang, and Zhaoran Wang. 2024b. Can large language models play games? a case study of a self-play approach. *arXiv preprint arXiv:2403.05632*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. *Preprint*, arXiv:2211.17192.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. Eagle-2: Faster inference of language models with dynamic draft trees. *Preprint*, arXiv:2406.16858.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. Eagle: Speculative sampling requires rethinking feature uncertainty. *Preprint*, arXiv:2401.15077.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. *Preprint*, arXiv:2503.01840.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024a. Awq: Activation-aware weight quantization for llm compression and acceleration. *Preprint*, arXiv:2306.00978.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. 2024b. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *Preprint*, arXiv:2405.04532.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2024. Online speculative decoding. *Preprint*, arXiv:2310.07177.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24. ACM.

PyTorch Contributors. 2024. Reproducibility. `https://pytorch.org/docs/stable/notes/randomness.html`. PyTorch Documentation, Accessed: January 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof qa benchmark. *Preprint*, arXiv:2311.12022.

RyokoAI. 2021. Sharegpt52k.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *Preprint*, arXiv:1907.10641.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023. How far can camels go? exploring the state of instruction tuning on open resources. *Preprint*, arXiv:2306.04751.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*.

Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, Chi Jin, Tong Zhang, and Tianqi Liu. 2024. Building math agents with multi-turn iterative preference learning. *Preprint*, arXiv:2409.02392.

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. 2025. Decoding speculative decoding. *Preprint*, arXiv:2402.01528.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. 2025. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Preprint*, arXiv:2501.01005.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA. USENIX Association.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A Smith. 2023. How language model hallucinations can snowball. *arXiv preprint arXiv:2305.13534*.

Juntao Zhao, Borui Wan, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024a. Llm-pq: Serving llm on heterogeneous clusters with phase-aware partition and adaptive quantization. *Preprint*, arXiv:2403.01136.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024b. Atom: Low-bit quantization for efficient and accurate llm serving. *Preprint*, arXiv:2310.19102.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric Xing, and 1 others. 2023a. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. *arXiv preprint arXiv:2309.11998*.

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. 2023b. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

A Full Experiment Result.

A.1 Acceleration Evaluation

Versus quantization configurations. Table 6 presents the comprehensive comparison of token generation throughput across multiple dimensions: model sizes, quantization configurations, and batch sizes, evaluated on various datasets. It is noteworthy that Llama2-7B shows higher speedup than Llama3-8B. This stems from the size difference primarily related to vocabulary, coupled with the introduction of Group-Query Attention (Ainslie et al., 2023), reducing the computation workload.

Versus speculative decoding. Table 7 presents a comprehensive comparison between our approach and the EAGLE method across multiple benchmarks.

A.2 Ablation Studies

Performance Trade-offs: QSPEC versus W4A16. We conduct a thorough analysis of the trade-offs between throughput and accuracy for our proposed framework against all baseline implementations. Figure 6 illustrates this comparison, plotting generation quality (accuracy) against computational efficiency (throughput). Our analysis reveals that while W4A4 suffers substantial performance degradation (18.5%-39.5% reduction) on multi-step reasoning benchmarks compared to W4A16, QSPEC achieves comparable accuracy to W4A16 while delivering significantly higher throughput. Although QSPEC's accuracy is marginally lower than W16A16 due to weight quantization-induced memory optimization, it successfully preserves the performance characteristics of W4A16 while offering superior computational efficiency.

A.3 Datasets and Sampling

For the PIQA and Winogrande datasets, we randomly select 500 questions from each for performance evaluation. In contrast, we process the entire GSM8K and MATH datasets, as detailed in Atom Zhao et al. (2024b). When adapting QSPEC to the Quarot method, we sample 200 samples from the GSM8K dataset, while sampling 700 from the MATH dataset, ensuring balanced representation by selecting 100 questions from each of the seven distinct question types. Additionally, we sample 200 questions from the MBPP dataset, while processing the entire HumanEval dataset. This preserves a thorough assessment of the model's performance across various datasets. For acceleration evaluation, we maintain the random seed at 42, and sample 100 samples from the GSM8K, MATH, MBPP, HumanEval, ShareGPT, and LMSys datasets, respsectively. This consistent methodology guarantees that our evaluation remains reproducible and representative across these diverse datasets.

A.4 QSPEC on vLLM: Real-World Serving Evaluation

To validate QSPEC's effectiveness in real-world serving scenarios, we integrated it into vLLM[2](Kwon et al., 2023) by developing custom kernels and a tailored vLLM worker, enabling shared weights and KV rewriting. We conducted performance tests on Llama-3-8b-instruct model with five diverse test sets: Wild Chat(Deng et al., 2024), GSM8K(Cobbe et al., 2021), MBPP(Austin et al., 2021), MT-Bench(Zheng et al., 2023b), and GPQA-Diamond(Rein et al., 2023)—covering domains such as chat, mathematics, coding, and general knowledge. Note that vLLM's support for the speculative decoding paradigm remains suboptimal[3] due to its complex scheduling and memory management mechanisms. Our experiments were performed on an NVIDIA A100 GPU (40GB), with batch sizes ranging from 1 to 32, a draft length $\gamma = 3$, and a baseline of autoregressive decoding using W4A16 with same weights. Results are shown in Table 8. QSPEC achieves an average speedup of 1.24×, maintaining effective acceleration even at a batch size of 32—a challenging feat for prior work. Additionally, we report acceptance rates across these test sets, with QSPEC achieving an impressive 93%–95% acceptance rate. This aligns with our findings in the Section 2, suggesting that leveraging these high acceptance rates for further acceleration is a promising direction for future QSPEC research.

Our integration of QSPEC into vLLM also initially aimed to enable a fair comparison with EAGLE (Li et al., 2024b). Unfortunately, we could not reproduce EAGLE's performance on vLLM, as it exhibited significant performance degradation in batched scenarios, even losing speedup in the

[2]We implemented QSPEC on Commit 9a7c3a0 of vLLM from 21 Jan. 2025

[3]For more detailed explanation, please refer to https://docs.vllm.ai/en/stable/features/spec_decode.html.

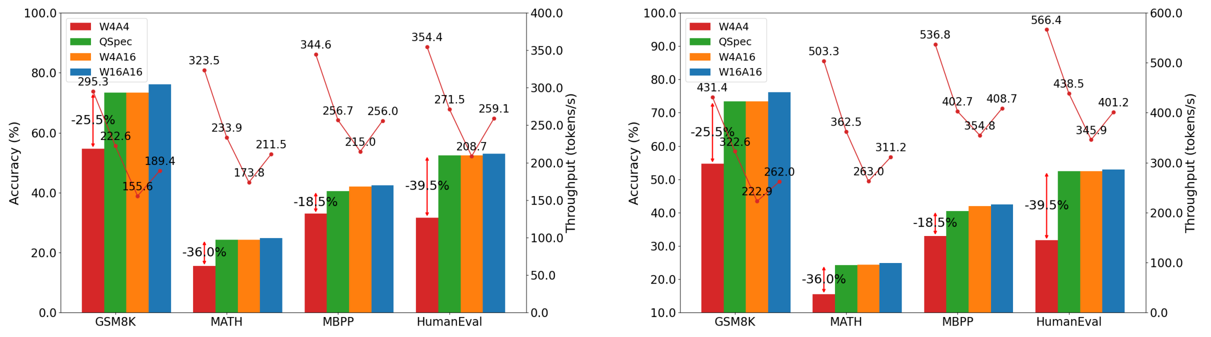
Table 6: Comparison of token generation throughput across different model sizes, quantization configurations, and batch sizes for various datasets. All values are measured in token/s. "Avg." denotes the average speedup ratio for the corresponding row or column.

Model	Method	Batch	GSM8K	MATH	MBPP	HumanEval	ShareGPT	LMsys-1k	Avg.
3B[1]	W16A16	8	511.1	588.7	756.6	647.2	785.7	711.2	–
		16	666.5	845.6	1171.0	948.3	1292.2	1126.4	–
		32	833.4	1081.5	1697.7	1111.6	1975.6	1553.3	–
	W4A4	8	804.7	921.2	1002.0	892.6	1091.6	990.3	–
		16	1109.1	1374.5	1548.0	1289.8	1763.5	1581.0	–
		32	1424.3	1899.3	2300.6	1488.2	2777.3	2194.4	–
	W4A16	8	420.0	476.7	604.5	535.7	610.4	559.8	–
		16	578.5	715.9	989.7	804.4	1080.2	925.8	–
		32	726.3	933.8	1536.7	954.4	1704.5	1336.4	–
	QSPEC	8	594.1 (1.41×)	648.2 (1.36×)	760.1 (1.26×)	723.6 (1.35×)	787.5 (1.29×)	738.8 (1.32×)	1.33×
		16	811.5 (1.40×)	936.0 (1.31×)	1157.8 (1.17×)	1042.1 (1.30×)	1294.5 (1.20×)	1171.4 (1.27×)	1.27×
		32	1030.4 (1.42×)	1240.2 (1.33×)	1617.4 (1.05×)	1248.5 (1.31×)	1969.6 (1.16×)	1576.0 (1.18×)	1.24×
		Avg.	1.41×	1.33×	1.16×	1.32×	1.21×	1.25 ×	1.28×
7B	W16A16	8	213.4	254.3	278.8	316.7	322.4	285.3	–
		16	290.3	362.1	447.7	505.1	541.3	441.6	–
		32	340.9	441.6	585.3	663.6	735.3	564.2	–
	W4A4	8	349.5	411.7	396.1	471.2	471.8	419.4	–
		16	496.6	612.2	614.3	749.5	760.9	642.6	–
		32	620.0	793.6	801.5	1043.9	1083.2	865.5	–
	W4A16	8	165.0	193.1	224.5	240.2	243.5	220.2	–
		16	231.8	286.5	384.4	407.3	435.9	358.0	–
		32	268.9	359.9	480.0	555.9	620.2	470.1	–
	QSPEC	8	253.7 (1.54×)	291.5 (1.51×)	298.3 (1.33×)	350.9 (1.46×)	345.7 (1.42×)	310.3 (1.41×)	1.44×
		16	359.8 (1.55×)	420.2 (1.47×)	466.7 (1.21×)	555.2 (1.36×)	557.8 (1.28×)	473.1 (1.32×)	1.37×
		32	441.8 (1.64×)	527.2 (1.46×)	575.3 (1.20×)	749.4 (1.35×)	770.0 (1.24×)	628.4 (1.34×)	1.39×
		Avg.	1.58×	1.48×	1.25×	1.39×	1.31×	1.36×	1.39×
8B	W16A16	8	189.4	211.5	256.0	259.1	290.7	265.8	–
		16	262.0	311.2	408.7	401.2	511.0	447.4	–
		32	303.8	390.8	566.3	522.6	820.0	649.8	–
	W4A4	8	295.3	323.5	344.6	354.4	395.9	366.8	–
		16	431.4	503.3	536.8	566.4	697.5	621.1	–
		32	532.8	688.5	755.7	763.7	1167.9	956.8	–
	W4A16	8	155.6	173.8	215.0	208.7	231.1	215.6	–
		16	222.9	263.0	354.8	345.9	422.8	369.4	–
		32	299.3	363.3	509.8	468.7	706.0	580.5	–
	QSPEC	8	222.6 (1.43×)	233.9 (1.35×)	256.7 (1.19×)	271.5 (1.30×)	285.0 (1.23×)	268.3 (1.24×)	1.29×
		16	322.6 (1.45×)	362.5 (1.38×)	402.7 (1.14×)	438.5 (1.27×)	507.5 (1.20×)	453.5 (1.23×)	1.28×
		32	400.2 (1.34×)	483.0 (1.33×)	578.1 (1.13×)	573.0 (1.22×)	798.8 (1.13×)	684.5 (1.18×)	1.27×
		Avg.	1.44×	1.36×	1.15×	1.26×	1.19×	1.22×	1.27 ×
13B[1]	W16A16	8	121.9	146.6	183.1	182.0	187.1	160.1	–
		16	169.6	211.2	304.4	291.0	311.0	243.0	–
		32	202.4	253.8	426.0	423.5	311.0	334.2	–
	W4A4	8	194.7	228.2	253.6	261.5	259.8	228.2	–
		16	288.3	349.2	415.3	424.9	431.5	348.4	–
		32	369.8	469.9	606.7	665.4	431.5	508.8	–
	W4A16	8	94.8	112.9	143.4	140.0	146.7	127.9	–
		16	136.1	171.9	250.8	236.9	255.9	207.2	–
		32	207.5	241.6	376.4	365.5	255.9	287.4	–
	QSPEC	8	148.2 (1.56×)	167.9 (1.49×)	193.6 (1.35×)	201.2 (1.44×)	194.5 (1.33×)	174.0 (1.36×)	1.42×
		16	212.8 (1.56×)	248.6 (1.45×)	316.8 (1.26×)	323.3 (1.36×)	327.4 (1.28×)	266.9 (1.29×)	1.29×
		32	266.6 (1.28×)	320.0 (1.32×)	451.5 (1.20×)	483.0 (1.32×)	327.4 (1.28×)	379.3 (1.32×)	1.32×
		Avg.	1.56×	1.47×	1.27×	1.37×	1.29×	1.32×	1.38×

W4A16 setting. Moreover, due to the poor performance and inconsistent compatibility of other speculative decoding methods on vLLM, our additional experiments here do not include comparisons with other methods. We also note that the EAGLE team released EAGLE-3(Li et al., 2025) in late March 2025, open-sourcing partial pretrained model weights in April and claiming competitive speedup on vLLM. However, as of now, the code to reproduce this claim on vLLM is unavailable, preventing us from evaluating EAGLE-3's performance on vLLM. Thus, we exclude comparisons with EAGLE-3, in line with the guidelines on handling contemporaneous work. This does not detract from the fact that both QSPEC and EAGLE-3 represent outstanding contributions developed con-

Table 7: Performance comparison of EAGLE-Quant, QSPEC, W4A16, and W4A4 on Llama-2-7b-chat-hf across different batch sizes. Results are reported for GSM8K (8-shot), MATH (4-shot), MBPP (0-shot), HumanEval (0-shot), ShareGPT, and LMsys-1k benchmarks. "OOM" indicates out-of-memory errors. Better case for QSPEC or EAGLE is marked in gray. In the case of batch size=8, the speedup ratio of QSPEC compared to EAGLE is indicated in parentheses next to the data points.

Method	Batch Size	GSM8K (8-shot)	MATH (4-shot)	MBPP (0-shot)	HumanEval (0-shot)	ShareGPT	LMsys-1k
EAGLE	1	65.81	70.11	68.53	49.15	79.60	71.29
	8	140.16	210.52	138.01	136.86	247.42	167.57
	16	OOM	OOM	OOM	OOM	OOM	OOM
QSPEC	1	51.25	48.36	56.87	54.22	56.77	56.14
	8	208.95 (1.49×)	249.97 (1.19×)	185.13 (1.34×)	185.99 (1.36×)	329.44 (1.33×)	260.48 (1.55×)
	16	292.82	356.93	269.87	255.11	562.07	463.35
W4A16	1	59.80	63.65	75.49	72.04	76.04	72.27
	8	146.34	185.52	180.12	163.54	250.06	213.66
	16	190.09	251.59	254.24	211.49	458.57	371.38
W4A4	1	64.79	66.32	74.72	73.47	73.09	71.55
	8	284.84	369.27	250.11	256.54	492.21	393.12
	16	401.77	540.82	357.62	330.71	895.77	713.67



Batch Size 8 Batch Size 16

Figure 6: Comparison of accuracy and efficiency among W16A16, W4A16, W4A4, and QSPEC across various datasets with batch sizes of 8 and 16 on Llama3-8b-instruct model. The bars and lines represent the accuracy and throughput of each method.

currently. As discussed in Section 7.2, QSPEC and EAGLE exemplify two distinct design paradigms: EAGLE follows a training-intensive route with an independently optimized draft model and a refined acceptance policy, while QSPEC adopts a training-free framework that leverages shared weights and KV cache in a unified structure. Each approach has its own strengths, and we look forward to future opportunities for dialogue and collaboration with the EAGLE team.

Table 8: Performance of QSPEC on vLLM across different batch sizes and test sets, with acceptance rates per test set respectively. All speedup values are reported with a × multiplier.

Test Set	Batch Size						Acceptance Rate (%)
	1	2	4	8	16	32	
Wild Chat	1.29×	1.33×	1.29×	1.26×	1.28×	1.13×	93.1
GSM8K	1.36×	1.33×	1.23×	1.24×	1.25×	1.01×	92.1
MBPP	1.34×	1.26×	1.16×	1.18×	1.18×	1.12×	95.5
MT-Bench	1.33×	1.24×	1.28×	1.30×	1.16×	1.15×	93.5
GPQA-Diamond	1.33×	1.31×	1.21×	1.18×	1.23×	1.10×	94.1

A.5 Artifact Documentation

We provide the official implementation of QSPEC in the supplementary materials. The codebase is fully documented and includes[4]:

- **Installation guide** covering dependency setup (CUDA 12.5, Python 3.10), environment recommendations (NVIDIA A100 or L20), and instructions for installing required third-party libraries and compiling QSPEC kernels.
- **Docker support** for reproducible deployment, including editable mount path and data path configurations, and build/run scripts.
- **Execution scripts** for reproducing our throughput and latency results using `demo.py`, which supports different model paths, speculative token lengths, and batch sizes.

[4]https://github.com/hku-netexplo-lab/QSpec

- **Pretrained models** hosted on Huggingface for QSPEC's Llama-3-8b-instruct model used in our vLLM experiments.
- **Notes and caveats** describing limitations of our current implementation (*e.g.*, not optimized for all GPU types, partial vLLM integration, cold start auto-tuning delay).
- **License and Intended Use:** We confirm that all third-party artifacts used in this work (*e.g.*, vLLM, Huggingface-hosted models) were accessed and used in accordance with their licenses (Apache License 2.0) and intended research purposes. Our implementation of QSPEC is released under the Apache License 2.0 and is explicitly intended for academic and non-commercial use. Users are instructed to obtain such resources directly from their original providers and to comply with the corresponding terms of use.

All artifacts are accompanied by a `README.md` file that details the usage and experimental instructions. The code is released under an anonymous GitHub repository to ensure reproducibility.

A.6 Understanding FP16 vs. W4A16 Performance in Main Results

While W4A16 quantization (*e.g.*, AWQ) is often expected to outperform FP16 in small-to-medium batch sizes due to its design for improved efficiency in weight-only quantization (Lin et al., 2024a), our main results consistently show FP16 surpassing W4A16 across various implementations, including vLLM, Atom's system, and even Hugging Face's official benchmarks[5]. This discrepancy may lead some readers to question the relative performance, as implementation-specific factors such as device characteristics, kernel optimizations, and system engineering significantly influence outcomes. We conducted a complementary experiment to elucidate this phenomenon, illustrated in Figure 7, comparing FP16 and W4A16 under different implementation settings.

In this experiment, Atom-FP16 and Atom-AWQ are derived from Atom's system implementation, following the end-to-end benchmark settings of our main experiments (Section 4), with FlashInfer integrated into a Punica-style serving system to support continuous batching (Yu et al., 2022). Conversely, Benchmark-FP16 and Benchmark-AWQ are sourced from the AutoAWQ repository, leveraging an optimized AWQ kernel and Flash-Attention, but employing a classic dummy benchmark method (directly invoking the model for the context length duration)[6]. Additionally, we include vLLM-FP16 and vLLM-AWQ, implemented in the vLLM framework (Commit 9a7c3a0, Appendix A.4), to provide a broader perspective.

Figure 7 reveals distinct performance trends across batch sizes of 8, 16, and 32. In Atom's implementation, FP16 consistently outperforms AWQ across all batch sizes, aligning with our main results. However, in the AutoAWQ dummy benchmark, AWQ exhibits superior throughput, reversing the trend. In vLLM, AWQ slightly outperforms FP16 at a batch size of 8, but FP16 surpasses AWQ at batch sizes of 16 and 32. These variations underscore the impact of system implementation and kernel optimization on relative performance. Since our main experiments strictly adhere to Atom's setup for fair and rigorous comparisons (Sec. 4), the observed speed disparity between FP16 and W4A16 due to implementation differences does not undermine the validity of our claims regarding QSPEC's performance.

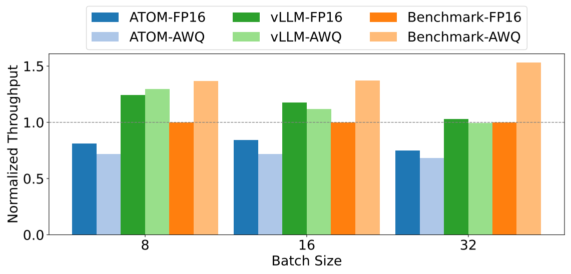


Figure 7: Normalized throughput of FP16 and AWQ implementations across batch sizes 8, 16, and 32. Atom-FP16 and Atom-AWQ use Atom's system with FlashInfer and continuous batching. Benchmark-FP16 and Benchmark-AWQ are from AutoAWQ with optimized AWQ kernels and Flash-Attention, using a dummy benchmark method. vLLM-FP16 and vLLM-AWQ are implemented in vLLM. The generation length is set to 512.

B Evaluation Datasets

We comprehensively evaluate the performance of QSPEC, covering language modeling, commonsense reasoning, mathematical reasoning, code and chatbot. Each dataset is briefly introduced below,

[5]huggingface.co/docs/transformers/main/quantization/awq

[6]github.com/casper-hansen/AutoAWQ

along with the associated sampling strategy for enhanced efficiency.

Language Modeling and Commonsense Reasoning.

- **WikiText-2** (Merity et al., 2016): A language modeling dataset comprising over 100 million tokens extracted from high-quality Wikipedia articles.
- **PIQA** (Bisk et al., 2020): A benchmark for physical commonsense reasoning, focusing on questions about everyday physical interactions.
- **Winogrande** (Sakaguchi et al., 2019): A dataset of 44,000 problems inspired by the Winograd Schema Challenge, designed to test commonsense reasoning with reduced linguistic biases.

Mathematical Reasoning

- **GSM8K** (Cobbe et al., 2021): A collection of 8,500 linguistically diverse grade school math problems that require multi-step reasoning using basic arithmetic operations.
- **MATH** (Hendrycks et al., 2021): A dataset of 12,500 challenging competition-level math problems, each accompanied by detailed step-by-step solutions.

Code

- **MBPP** (Austin et al., 2021): A benchmark of approximately 1,000 beginner-level Python programming problems, each with a task description, solution code, and automated test cases.
- **HumanEval** (Chen et al., 2021): An evaluation set of 164 original programming problems used to assess functional correctness in code synthesis from docstrings.

Chatbot

- **ShareGPT52K** (RyokoAI, 2021): This dataset comprises approximately 52,000 conversations collected via the ShareGPT API before it was discontinued. The dataset captures both user prompts and the corresponding responses from OpenAI's ChatGPT, providing insights into human-AI dialogue dynamics.
- **LMsys-chat-1M-1K** (Zheng et al., 2023a): Gathering one million authentic conversations with 25 leading large language models (LLMs), this dataset was sourced from over 210,000 unique IP addresses interacting with the Vicuna demo and Chatbot Arena websites.

C Datasets Sampling

To construct our test sets, we randomly sampled from the original datasets using *torch.sample* with a fixed seed of 42, and constructed them as prompt instructions following the Open-Instruct templates (Wang et al., 2023). The sample sizes for each dataset are as follows:

- **Fidelity Evaluation:**
  - **WikiText-2**: All samples
  - **PIQA**: 500 samples
  - **Winogrande**: 500 samples
  - **GSM8K**: 200 samples
  - **MATH**: 700 samples, balanced by selecting 100 questions from each of the seven distinct question types
  - **MBPP**: 200 samples
  - **HumanEval**: All samples
- **Acceleration Evaluation:** 100 samples from each dataset, maximum output length set to 200 tokens.

D Related Work

Quantization is a common technique for deploying LLMs on resource-limited scenarios. Broadly, recent quantization algorithms can be classified into two categories: weight-only W4A16 and weight-activation joint W4A4. Notably, AWQ (W4A16) (Lin et al., 2024a) redistributes the quantization burden by scaling salient weight channels to protect them from degradation. In contrast, W4A4 aggressively quantizes activations to leverage low-precision hardware for improved speed at the cost of model quality degradation. To address this challenge, Atom (Zhao et al., 2024b) proposes reordering outlier channels in the activation through offline profiling. Similarly, QuaRot (Ashkboos et al., 2024) employs Hadamard matrices to apply computational invariance on weights. Despite these advancements, our observations indicate that W4A4 methods still exhibit substantial degradation compared to weight-only quantization approaches across multi-step reasoning tasks. Works such as

W4A8 (Lin et al., 2024b) and adaptive quantization (Zhao et al., 2024a; Dong et al., 2024) seek to identify an optimal trade-off point. However, these methods struggle to fully preserve the generation quality associated with higher precision.

Speculative Decoding leverages a draft model to generate candidate tokens, which are then validated by a target model (Leviathan et al., 2023). Recent research has primarily focused on improving the acceptance rate and generation speed of candidate tokens. SpecInfer (Miao et al., 2024) introduces a boost-tuned small language model to generate candidate tokens in tree structures, enabling single-pass verification. In contrast, EAGLE (Li et al., 2024b) adopts an aggressive pruning strategy for the draft model's architecture, allowing penultimate layer feature prediction with minimal computational overhead. Self-speculative decoding, a subset of this technique, employs a single model for both draft generation and verification. LayerSkip (Elhoushi et al., 2024) introduces a training methodology for early exit with layer drop, subsequently verifying partially generated tokens through full model inference. Medusa (Cai et al., 2024) augments the original LLM with additional heads atop the final hidden state while relaxing the acceptance policy. However, these approaches inevitably require retraining of the original model, which can be computationally expensive and time-consuming. We further demonstrate their deficiency in batched serving under quantization scenario.

Tree-Structured Drafting. Tree-structured drafting (Chen et al., 2024; Miao et al., 2024; Li et al., 2024b,a) is a widely adopted technique to improve acceptance rates in speculative decoding. Instead of generating a single draft token at a time using the draft model $\mathcal{M}_d$, $\mathcal{M}_d$ select k (*i.e.*, top-k) tokens and infers γ times to form a draft tree with depth γ. The target model $\mathcal{M}$ then verifies the tokens using masked tree attention, with the exact verification strategy depending on the sampling method (Cai et al., 2024; Miao et al., 2024). Under greedy sampling, the highest-probability token at each position is selected, forming the longest branch with the common prefix as the accepted sequence. In this tree, the root node presents the past sequence, any token t_i has a path from the root node r to t_i, denoted by $Path(r, t_i)$, consisting of ancestors $a_1, \ldots, a_j$.

E Supplementary Figures and Tables

To improve the clarity of the main text and streamline presentation, we provide additional visualizations and ablation results related to QSpec in this appendix. These include supporting data referenced in the main body and additional experiments.

Table 9: Ablation study comparing acceptance rates (%) across base quantization methods using QSpec.

Quantization Method	**ShareGPT**	**MATH (4-shot)**	**MBPP (0-shot)**
Atom	83.8	89.4	88.6
QuaRot	81.6	88.9	85.4

Table 10: Ablation study comparing acceptance rates (%) across large reasoning model and difficult reasoning tasks using QSpec.

Model	**GPQA-Diamond**	**Super-GPQA**	**AIME**	**ARC**	**MMLU**
Llama3-8b -Instruct	94.1	96.5	96.1	92.6	92.4
	OpenBookQA	**RACE**	**SQuAD v2**	**TruthfulQA**	**HellaSwag**
	92.6	94.2	95.0	92.0	91.7
	HumanEval	**LAMBADA**	**Social IQa**	**Avg.**	
	87.5	89.6	93.8	**92.93**	
DeepSeek-R1-Distilled -QWen14B	96.2	96.0	97.9	90.4	90.8
	OpenBookQA	**RACE**	**SQuAD v2**	**TruthfulQA**	**HellaSwag**
	90.5	92.8	92.8	88.6	96.7
	HumanEval	**LAMBADA**	**Social IQa**	**Avg.**	
	96.7	94.3	91.7	**93.49**	

Table 11: Performance results for the reasoning model DeepSeek-R1-Distilled-QWen14B. We follow the settings of our main experiments and set the batch size (bs) to 16.

Dataset	**GSM8K**	**MATH**	**MBPP**	**HUMANEVAL**	**SHAREGPT**	**LMSYS**	**AVG.**
W4A16 (tokens/s)	139.38	203.52	200.84	216.54	194.31	194.22	191.47
QSpec (tokens/s)	171.56	282.17	278.74	292.75	259.48	246.07	255.13
Speedup	1.23	1.39	1.39	1.35	1.34	1.27	1.33

F AI Assistance Statement

We used AI tools (*e.g.*, ChatGPT) exclusively for language polishing and LaTeX formatting. No part of the core research, including ideation, experimental design, or analysis, was generated by AI tools. All scientific contributions are the sole work of the authors.