

From Quarter to All: Accelerating Speculative LLM Decoding via Floating-Point Exponent Remapping and Parameter Sharing

Yushu Zhao*, Yubin Qin*, Yang Wang[✉], Xiaolong Yang, Huiming Han, Shaojun Wei, Yang Hu, Shouyi Yin[✉]

BNRist, Tsinghua University

Email: wangyang_imec@mail.tsinghua.edu.cn, yinsy@tsinghua.edu.cn

Abstract—Large language models (LLMs) achieve impressive performance across diverse tasks but exhibit high inference latency due to their large parameter sizes. While quantization reduces model size, it often leads to performance degradation compared to the full model. Speculative decoding remains lossless but typically incurs extra overheads. We propose SPEQ, an algorithm–hardware co-designed speculative decoding method that uses part of the full-model weight bits to form a quantized draft model, thereby eliminating additional training or storage overhead. A reconfigurable processing element array enables efficient execution of both the draft and verification passes. Experimental results across 15 LLMs and tasks demonstrate that SPEQ achieves speedups of 2.07×, 1.53×, and 1.45× compared over FP16, Olive, and Tender, respectively.

Index Terms—large language model, speculative decoding, model quantization, algorithm–hardware co-design.

I. INTRODUCTION

Large language models (LLMs) have demonstrated exceptional performance across a wide range of downstream tasks and real-world applications, including knowledge tasks, reasoning tasks, and conversational tasks [16]. The escalating scale of LLMs, which now often exceed hundreds of billions of parameters [4], [17], introduces significant weight-loading latency during inference, as shown in Fig. 2(a). This latency acts as a primary bottleneck, degrading overall inference speed and posing a critical obstacle for real-time applications that require rapid responses.

Model compression techniques have been proposed to address the substantial memory requirements of LLMs, with quantization standing out as one of the most effective methods. By reducing the numerical precision of model weights from FP16 to lower bit-width formats, quantization significantly shrinks model size. Propelled by recent algorithmic innovations [6], [7], [9], [13], [19], the performance of quantized LLMs on GPUs and specialized hardware accelerators has seen marked improvement. Nevertheless, two critical challenges limit the widespread adoption of quantization. **(1) Quantization introduces cumulative errors in long-sequence generation**, degrading performance compared to the full-precision counterpart. As a case in point, 4-bit AWQ [13] incurs a significant 5.4% accuracy loss on the MATH task [8], [12]. **(2) Quantization modifies the model’s output distribution**, creating a potential misalignment that can compromise its effectiveness and safety in real-world applications [5].

To accelerate LLM decoding in a lossless manner, speculative decoding has been introduced as an effective strategy [1], [11], [15], [18], [21]. The methodology leverages a fast but less accurate draft model to autoregressively generate a sequence of future tokens. These draft tokens are subsequently verified in a single, parallel forward pass by the original target model, which allows for multiple tokens to be decoded per verification step, thereby speeding up inference without altering the final output (Fig. 1). While speculative decoding has garnered significant attention, its practical application is hindered

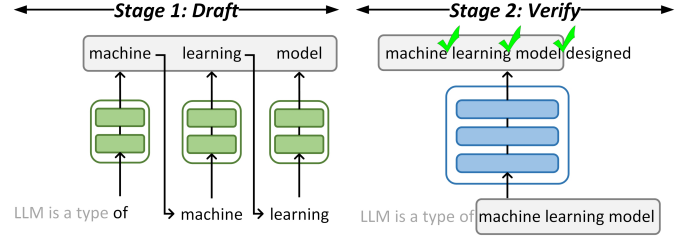


Fig. 1. Illustration of the speculative decoding process: a draft model generates multiple draft tokens, and a target model verifies the draft tokens in parallel.

by two persistent challenges: **(1) the difficulty of creating a high-quality draft model** and **(2) the suboptimal hardware utilization due to poor co-optimization of the draft and target models**. Current approaches to speculative decoding typically utilize a trained small auxiliary model or extra heads as the draft [1], [11], [15], which incurs considerable training burdens and requires additional memory for storage. Self-speculative decoding variants address the training cost by deriving a sparse draft model from the target model itself. However, the quality of this draft is often suboptimal [18], [21]. Critically, current research predominantly concentrates on GPU-centric software optimizations, failing to explore the co-optimization of the draft-verification pipeline on specialized hardware.

In this paper, we introduce SPEQ, an algorithm–hardware co-design for lossless LLM decoding acceleration, as depicted in Fig. 2(d). SPEQ utilizes a self-speculative decoding framework where the draft model is a quantized version of the target model, created via a **novel bit-sharing quantization algorithm**. This method converts standard FP16/BF16 models into a Bit-Sharing Floating Point (BSFP) format, from which a 4-bit draft model is extracted. This approach eliminates both training requirements and additional memory overhead. Furthermore, we co-design a **reconfigurable PE array** optimized to handle the forward passes for both the 4-bit quantized drafting and full-precision verification, ensuring high hardware utilization for the two passes. The key advantages of SPEQ are improved draft token quality, zero memory overhead, no additional training overhead, and hardware co-optimization. Our evaluations on 15 LLMs and challenging generation tasks show that SPEQ achieves an average draft token accept rate of 0.976. SPEQ accelerator achieves 2.07×, 1.53×, and 1.45× speedup compared over FP16 baseline, Olive [7], and Tender [9], respectively.

Our contributions are summarized as follows:

- We analyze the bit-level representation of the FP16 weights in state-of-the-art LLMs and identify the bit-level underutilization of the exponent part.
- We propose bit-sharing quantization, which, to the best of our knowledge, is the first LLM quantization algorithm that enables

* Equal contribution.

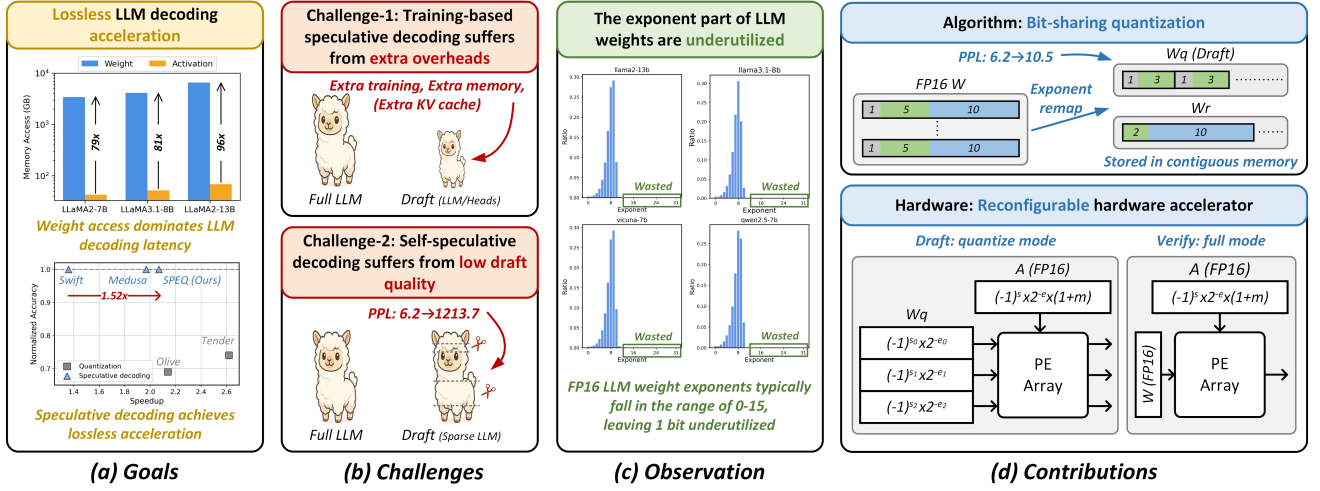


Fig. 2. Overview of SPEQ, an algorithm-hardware co-design approach for accelerating LLM speculative decoding.

bit-level sharing between the quantized model and the full FP16 model.

- We propose the SPEQ accelerator, a hardware design optimized for efficient LLM speculative decoding. It enables the efficient processing of both the quantized draft forward pass and the full verification forward pass.
- We demonstrate the effectiveness of SPEQ across 15 typical LLMs and generative tasks, achieving high acceptance rates and substantial speedups.

II. BACKGROUND

A. LLM Quantization

As depicted in Fig. 2(a), for models such as Llama3.1-8B, Llama2-13B, and Llama2-7B [17], under a representative scenario with a prefill length of 1024 tokens and a decode length of 1024 tokens, memory access for model weights constitutes 98.8% of the total memory operations during decoding. This underscores the importance of optimizing access of weight memory in the decode stage to mitigate overall latency.

Quantization [6], [7], [9], [13], [19] is a technique employed to reduce the memory footprint of model weights. It utilizes lower bit-width representations for weights, such as INT8 or INT4, to replace the standard FP16 format. Despite its effectiveness in model compression, several challenges limit the widespread real-world application of quantization. Firstly, while state-of-the-art 4-bit quantization methods achieve nearly lossless performance on discriminative tasks, **their efficacy on complex generative tasks remains unsatisfactory**. For instance, [12] demonstrated that applying 4-bit AWQ to Llama3.1-8B-instruct [4] results in a 5.4% accuracy degradation on the MATH dataset [8]. Furthermore, the altered output distribution of a quantized model can introduce **safety and alignment issues**. This discrepancy necessitates additional fine-tuning to ensure the model does not produce harmful or unintended content, which is a critical requirement for deployment in real-world applications [5].

Although quantization is a powerful method for compressing LLMs, these limitations hinder its broader adoption. Consequently, there is growing interest in exploring lossless methods for accelerating LLM decoding, such as speculative decoding, which aims to reduce latency without compromising model accuracy or safety.

B. Speculative Decoding

Speculative decoding [1], [11], [15], [18], [21] has emerged as a prominent lossless acceleration technique. This method employs a smaller, faster “draft” model to generate a sequence of candidate tokens, which are then validated in a single, parallel forward pass by the larger, original “target” model. The target model determines which tokens from the drafted sequence are correct and can be accepted, while discarding the rest, as illustrated in Fig. 1.

Following the analysis in [10], we conduct a theoretical study on the speedup achieved through speculative decoding. Suppose the draft length of the draft model is L , the time for drafting a single token is T_d , the accept rate is r , the time for verification is T_v . The time for the target model to auto-regressively generate a single token is T_{ar} . The average accept length L_a can be estimated as:

$$L_a = \frac{1 - r^{L+1}}{1 - r} \quad (1)$$

The speedup over standard autoregressive decoding is given by the following expression. We refer the readers to [10] for more details about the speculative decoding process.

$$speedup = \frac{L_a \times T_{ar}}{L \times T_d + T_v} \approx \frac{L_a}{L \times \frac{T_d}{T_{ar}} + 1} \quad (2)$$

Recent speculative decoding algorithms for large language models (LLMs) can be broadly classified into two main categories, as shown in Fig. 2(b). The first category employs either a smaller LLM or additional output heads as a draft model. For example, SpecInfer [15] trains compact LLMs through knowledge distillation, while Medusa [1] trains multiple “Medusa heads” that generate candidate future tokens based on the model’s final hidden states. However, the Medusa architecture presents significant drawbacks: these heads necessitate **substantial supplementary training** and introduce an **approximate 11% memory overhead** for a Vicuna-7B model instance. Such overheads limit the practicality of these methods in real-world scenarios where computational and storage resources are constrained.

The second category is self-speculative decoding. This approach eliminates the need for additional training and memory resources while improving the quality of the draft tokens. Techniques such as Draft&Verify [21] utilize offline Bayesian optimization to search for a fixed set of sparse layer combinations. Swift [18] employs dynamic

layer pruning to construct the draft model. However, **the pruned model often suffers from severe quality degradation**. For example, Fig. 2(b) shows that pruning half of the layers in Llama3.1-8B results in a perplexity increase exceeding 10^3 .

Despite the progress made in this area, current speculative decoding methods still exhibit significant limitations, including **the need for additional training and extra memory storage**, as well as the **low quality of the draft tokens**. To the best of our knowledge, no prior work has investigated the use of quantized models as draft models. This is primarily because the bit-level representation of commonly quantized models differs from that of the original FP16 models, which requires additional storage for the draft model.

III. SPEQ ALGORITHM

A. Bit-level Underutilization in LLM Weights

The weights of LLMs are typically stored in floating-point format, which represents a value as

$$(-1)^{sign} \times 2^{exponent-bias} \times (1 + mantissa) \quad (3)$$

In the widely used FP16 format, the exponent field consists of 5 bits, allowing for exponent values in the range $[0, 31]$. We analyze the exponent distribution of the weights in several state-of-the-art LLMs, including Llama2-13b [17], Llama3.1-8b [4], Qwen2.5-7b [20], and Vicuna-7b [22]. As shown in Fig. 2(c), the exponent values across nearly all linear weight tensors are confined to the range $[0, 15]$. Virtually no weights exhibit exponents in the range $[16, 31]$, suggesting that one exponent bit remains effectively unused. This underutilization arises from the training practices adopted in modern LLMs. Techniques such as weight decay and normalization are commonly employed to stabilize training and constrain the parameter values, which results in a limited dynamic range of the values and bit-level inefficiency.

While the above pattern is broadly consistent, certain rare cases exhibit exceptionally large values within specific tensors. For instance, the LLaMA2-13B model exhibits an outlier: a single value of 2.4062, located in the down_proj tensor of the 0th layer. This value is the sole instance across the entire model to exceed the typical range. To handle such rare outliers, we introduce a per-tensor scaling strategy that rescales the weights as needed. The procedure is described in Algorithm 1. This tensor-wise post-scaling operation incurs no performance degradation and negligible computational overhead.

B. Bit-sharing Quantization

In this section, we introduce the BSFP data type, which contains a 4-bit quantized value that shares the bit representation with the full FP16 model.

To create the bit-sharing property, the 4-bit quantized model needs to be in FP4 datatype. There are various FP4 representations, including E1M2, E2M1, and E3M0. We perform fine-grained group quantization with a group size of 128. Using E3M0 as an example, as shown in the middle part of Fig. 3, for each group, we first extract the middle 3 bits

Algorithm 1 Handling Rare Outliers

Input: FP16 weight W

$scale = 1.0$

if $max(abs(W)) > 2.0$ **then**

$scale = \frac{1.999}{W_{max}}$

$W = W * scale$

end if

$W_q, W_r = BSFPquantize(W)$

Output: $W_q, W_r, scale$

Original Binary (Value)	Naive Binary (Quantized Value)	Remap Binary (Quantized Value)
00000(0)	00000(0)	10010(2)
00001(1)	00001(0)	10011(2)
00010(2)	00010(2)	00010(2)
00011(3)	00011(2)	00011(2)
00100(4)	00100(4)	10110(6)
00101(5)	00101(4)	10111(6)
00110(6)	00110(6)	00110(6)
00111(7)	00111(6)	00111(6)
01000(8)	01000(8)	10000(8)
01001(9)	01001(8)	10001(9)
01010(10)	01010(10)	01010(10)
01011(11)	01011(10)	10101(11)
01100(12)	01100(12)	01100(12)
01101(13)	01101(12)	01101(12)
01110(14)	01110(14)	01110(14)
01111(15)	01111(14)	01111(14)

Fig. 3. Illustration of remapping: the first row "Original" means the binary encoding and real value of FP16 exponent. "Naive" means using naive E3M0 quantization, the two neighbor values are rounded to the same quantized value. The bits in bold mean the binary representation of the quantized value. "Remap" means remapping the binary encoding to create unique representations for 9 and 11, while 0-3 and 4-7 are rounded to the same quantized value. The wasted bit is set to 1 if the bit representation of the remapped value is different from the original value.

of the exponent part of the FP16 values w_i to obtain the quantized values $Q(w_i)$ (the values in bold), then we compute a scale s to minimize the mean square error between the quantized values and the original values according to Equation 4. During inference, we multiply the scale s with the final output of the quantization group.

$$s = \frac{\sum_{i=0}^{127} w_i \times Q(w_i)}{\sum_{i=0}^{127} Q(w_i)^2} \quad (4)$$

We evaluate all possible types of FP4 representations on wikitext2 [14] with a sequence length of 2048, and from Table I, it is observed that using the E3M0 format for representing the quantized model results in the lowest perplexity for all three models, since the exponent part has a more significant impact than the mantissa part.

The key feature of BSFP is to utilize the wasted bit while better preserving the important values in the exponent. As shown in Table I, applying E3M0-based shared quantization without modification results in a significant increase in perplexity for the three models. This occurs because the nearest two exponent values are rounded to the same value, introducing substantial errors in the important large

TABLE I
PERPLEXITY RESULTS OF DIFFERENT QUANTIZATION METHODS FOR THE DRAFT MODEL WITH BIT-SHARING.

Method	Llama3.1-8b	Llama2-7b	Vicuna-7b
FP16	6.2	5.5	6.8
E1M2	3E+5	2E+4	3E+4
E2M1	7E+4	7E+3	1E+4
E3M0	251.8	153.9	103.0
Naive	251.8	153.9	103.0
+Remap	10.5	7.0	8.9

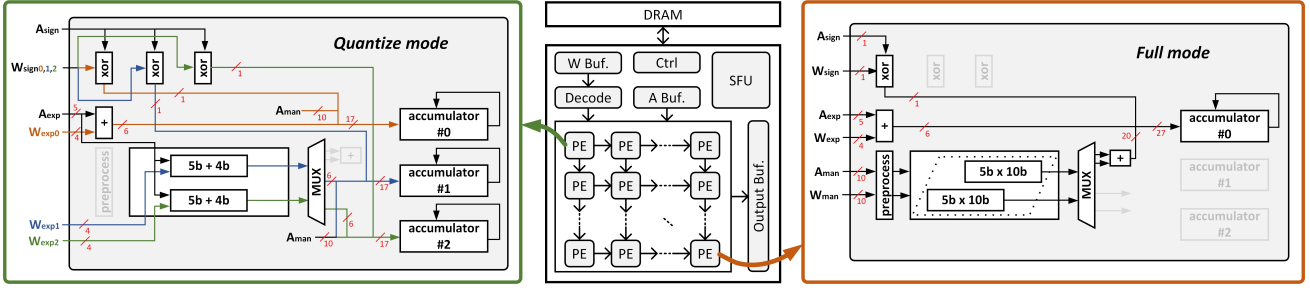


Fig. 4. Overview of SPEQ hardware accelerator. The reconfigurable PE is capable of processing both the quantized GEMM and the full GEMM with high hardware efficiency.

values.

To address this issue, we propose a novel remapping method, illustrated in Fig. 3. As shown in Fig. 2(c), values in the range of 8-11 are more critical than values in the range of 0-7 due to their larger magnitudes and quantities, which are essential for maintaining numerical precision. Therefore, instead of rounding 8 and 9 to 8, and 10 and 11 to 10, we preserve separate representations for 9 and 11 by assigning unique encodings, specifically 3'b000 and 3'b010. The original values represented by 3'b000 and 3'b010 are then rounded to 3'b001 and 3'b011, respectively. This remapping improves the precision of the quantized values but compromises the bit-sharing characteristic. To recover from this, we use the unused bit for correction. If a value's bit representation changes due to remapping, we set the unused bit to 1, signaling that this value needs to be looked up. The values in bold in Fig. 3 right represent the binary representation of the quantized values, which are stored in the quantized weight with the sign bit, denoted as W_q . The remaining values, including 2 exponent bits and 10 mantissa bits are concatenated and denoted as W_r . Using our remapping method gains a significant performance boost for LLMs, as illustrated in Table I, resulting in a significantly smaller perplexity increase than using naive E3M0 quantization.

C. Optimization for Speculative Decoding

We incorporate an early exit mechanism in the drafting process. If the predicted probability of the draft model is low, it suggests that the token is unlikely to pass verification, prompting us to stop drafting and avoid wasting resources on tokens that are likely to be rejected. We set a parameter γ for early stopping. If the sampled maximum draft logit $p_{draft}(x_{i+1}|x_1, \dots, x_i) < \gamma$, the drafting process is halted, and verification begins. Otherwise, drafting continues until the maximum draft length L is reached. Note that SPEQ incurs no extra KV cache overhead, as the quantized draft model and the full model share the same KV cache.

IV. SPEQ HARDWARE ACCELERATOR

A. Overview

Fig. 4 presents the overall architecture of SPEQ accelerator. It is composed of DRAM, on-chip buffers, a control unit, a special function unit (SFU), and a reconfigurable PE array. The control unit manages data movement between DRAM and the on-chip buffer, and controls the processing of the PE array. The on-chip buffers contain a 512 KB W buffer, a 512 KB A buffer, and a 512 KB output buffer. The SFU processes the activation functions. The reconfigurable PE array contains 32×32 PE units, which are capable of handling quantized forward and full forward.

The SPEQ hardware accelerator is primarily designed for FP16 processing, meaning that during the full forward pass, both the activations and weights are represented in FP16. **This accelerator is also capable of supporting BF16 models.** For models represented in the BF16 format, we first round the exponent values that are smaller than 112 up to 112. Subsequently, a similar remapping process is applied to the exponent component. In this manner, the exponent part of the BF16 representation is successfully reduced to 5 bits with bit-sharing property. Furthermore, we pad the mantissa component with three zeros. This results in weights being represented in the same format as FP16 (S1E5M10), allowing us to leverage the quantized forward and full forward passes within the PE array.

B. Decoder for BSFP

In Section III-B, we introduced the remapping scheme in BSFP to enhance the performance of the draft model. This remapping results in a different bit representation compared to standard FP4 and FP16, necessitating the use of special decoders to encode the values for subsequent computation.

The decoder design for the exponent of the quantized value W_q is shown in Fig. 5(a). The 3-bit W_{q-exp} is input into the decoder unit. As illustrated in Fig. 3, the values that need to be looked up are 9 and 11, represented in binary as 3'b000 and 3'b010, respectively. We first extract the 0th and 2nd bits and pass them through a NOR gate. If the output is 0, indicating that no decoding is required, a zero is appended to the end of the value to produce the output. If the output is 1, indicating that the value corresponds to 9 or 11, the final binary representations are 4'b1001 or 4'b1011. We set the 0th, 1st, and 3rd bits to 1, 0, and 1, respectively, and use the 1st bit of W_{q-exp} as the 2nd bit of the output. In this way, we obtain the final output of 4 bits.

The decoder design for the exponent of the FP16 value is shown in Fig. 5(b). The 3-bit W_{q-exp} and 2-bit W_{r-exp} from W_r are used as inputs. If the 0th bit of W_{r-exp} is 0, meaning that the weight remains unchanged, the two parts are concatenated to obtain the final output. If the 0th bit of W_{r-exp} is 1, we use the 1st and 2nd bits of W_{q-exp} for the MUX unit, the MUX takes in 2 bits and outputs 3 bits as shown in Fig. 5. The output value from the MUX is then concatenated with the 0th bit of W_{r-exp} to obtain the final result.

C. Reconfigurable PE Array

The PE array in the SPEQ accelerator adopts a reconfigurable architecture, consisting of 8 tiles with 128 PEs each. During drafting, each PE operates in quantized mode, processing three 5-bit weights and one FP16 activation to generate three partial sums. During verification, it processes one FP16 weight and one FP16 activation to produce a single partial sum.

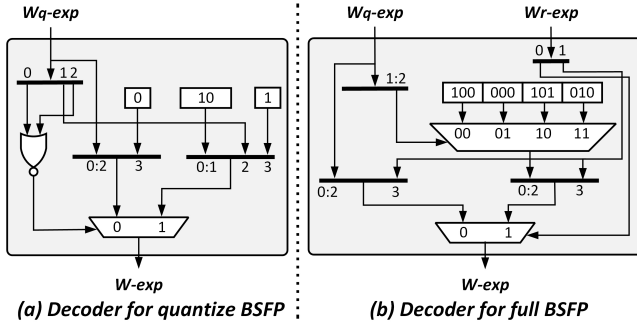


Fig. 5. Decoder design for BSFP.

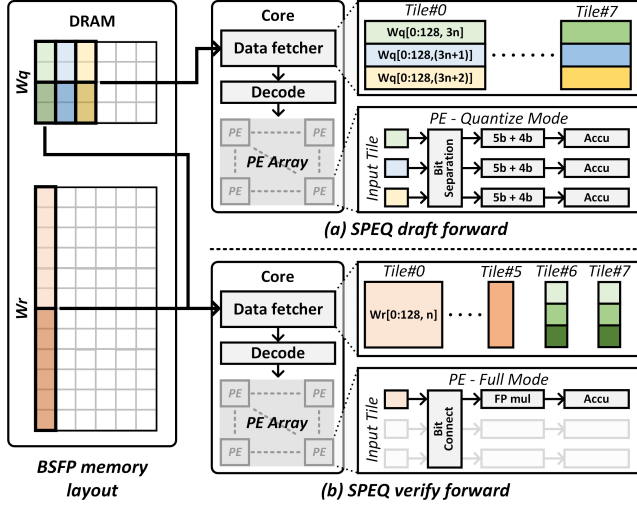


Fig. 6. Workflow of the quantize mode and full mode of SPEQ hardware.

Each PE includes four main components: (1) a sign processing unit that handles sign bits via XOR operations; (2) an adder unit that combines activation and weight exponents; (3) a multiplication unit employing a Wallace tree for mantissa operations; and (4) an accumulation unit that performs FP32 accumulation on the weighted products.

The workflow for quantized inference and full inference is shown in Fig. 6. In full mode, the processing of the PE array is identical to that of a standard FP16 PE array. However, the actual bit-width per element is 15 bits, as the unused exponent bit is ignored after decoding. This results in a total input bit-width of 16 (activation) + 15 (weight) = 31 bits. Within the multiplication unit, the 10-bit mantissa of the weight is divided into two 5-bit segments: the upper and lower halves. Each segment is multiplied with the 10-bit mantissa of the activation value using two parallel Wallace trees. The outputs of these two multiplications are then summed to produce a 20-bit product. Simultaneously, the exponents of the weight and activation are added using a 5-bit adder tree. Finally, the sign, exponent, and mantissa are concatenated and passed to the accumulation unit, where FP32 accumulation is performed.

In quantize mode, each PE tile receives 3×128 quantized weight elements as shown in Fig. 6. The weights of three different weight groups corresponding to the same activation are processed by a single PE. Each weight group undergoes its own MAC operation to obtain the final output. Afterward, the results are accumulated in a single PE tile.

TABLE II
AVERAGE DRAFT LENGTH AND ACCEPT RATE OF 5 LLMs ON 3 GENERATIVE TASKS.

Model	Humaneval		MT-bench		GSM8K		Mean
	L	r	L	r	L	r	
Vicuna-7b	8.02	0.968	8.40	0.964	7.48	0.977	0.970
Llama2-7b	6.05	0.981	4.47	0.986	6.38	0.987	0.985
Llama3.1-8b	5.10	0.975	5.69	0.979	5.31	0.967	0.974
Llama3.2-3b	5.61	0.953	6.05	0.978	4.83	0.964	0.965
Llama2-13b	5.80	0.986	6.61	0.992	6.57	0.991	0.990

The PE in quantize mode receives an activation and three quantized weights. The total input bit-width is 16 (activation) + 3×5 (weight) = 31 bits, the same as in full mode. Each of the three quantized weights consists of 1 bit for the sign and 4 bits for the exponent. For the first weight, the exponent is added to the activation exponent in the exponent adder of the PE. For the second and third weights, we utilize the components in the mantissa multiplier to perform the addition. The two adders in the two Wallace trees function as separate adders during this process, while the remaining components in the multiplier are masked out. The resulting added exponents of the activation and the three weights are concatenated with the mantissa part of the activation and its sign. These are then fed into three accumulation units to perform accumulation separately.

V. EVALUATION

A. Experiment Setup

LLM Benchmarks. For the evaluation, we utilize SOTA LLMs including Llama3.1-8b [4], Vicuna-7b [22], Llama2-7b [17], Llama3.2-3b [4], and Llama2-13b [17]. The evaluation is conducted across 3 generative tasks: MT-bench, GSM8K, and Humaneval.

- **GSM8K** [3] includes 8,000 high-quality grade-school mathematics problems designed to evaluate the mathematical reasoning capabilities of LLMs. We use 8-shot prompt for GSM8K evaluation.
- **Humaneval** [2] is a benchmark focused on assessing the code generation capabilities of LLMs.
- **MT-Bench** [22] is a benchmark designed to evaluate the multi-turn conversational abilities of LLMs.

Hardware Implementation. We complete the synthesis, placement, and routing of SPEQ accelerator based on 28nm CMOS technology. We obtain the working power based on VCS and Verdi simulation.

Hardware Accelerator Baselines. We compare the speedup and energy efficiency between SPEQ and existing quantization-based hardware accelerators, including Olive [7] and Tender [9]. Note that Olive and Tender result in substantial increases in perplexity and accuracy degradation. For the Llama2-7b model, **4-bit Olive leads to a perplexity increase of 38.7** ($5.5 \rightarrow 44.2$), while **4-bit Tender causes a perplexity increase of 31.0** ($5.5 \rightarrow 36.5$). The **severe performance degradation** associated with these methods limits their applicability, whereas our approach is **lossless**. We also include the results for 8-bit Olive and Tender with better accuracy performance.

B. LLM Benchmarks

We report the average draft length L , accept rate r , and speedup on the 3 benchmarks. Note that our method is a lossless acceleration method, the accuracy is the same as the full model. For all tasks, we set the generation length to 256. We use a maximum draft length of 16 and an early stopping threshold γ of 0.6. Further evaluation of hyperparameter settings is in Section V-E.

TABLE III
SPEEDUP OF 5 LLMs ON 3 GENERATIVE TASKS.

Model	HumanEval	MT-bench	GSM8K	Mean
Vicuna-7b	2.05×	2.03×	2.12×	2.07×
Llama2-7b	2.11×	2.04×	2.16×	2.10×
Llama3.1-8b	2.01×	2.08×	2.00×	2.03×
Llama3.2-3b	1.93×	2.09×	1.96×	2.00×
Llama2-13b	2.13×	2.21×	2.19×	2.18×

TABLE IV
AREA AND POWER BREAKDOWN OF SPEQ ACCELERATOR AT 500 MHZ.

Module	Area	Power (quantize mode)	Power (full mode)
PE	39.4%	36.5%	40.0%
Decoder	3.5%	3.2%	3.1%
SRAM	35.1%	32.1%	30.2%
VPU	14.8%	15.3%	14.5%
Others	7.2%	12.9%	12.2%
Total	6.3mm²	508mW	559mW

Table II presents the average draft length and accept rate of SPEQ. SPEQ achieves an average accept rate of 0.977 for the 5 LLMs, indicating the high quality of the draft model and the high alignment between the quantized draft model and the target model.

Table III shows the speedup results of SPEQ compared with FP16 baseline. SPEQ achieves an average speedup of 2.08×

 compared with naive FP16 autoregressive decoding for the 5 LLMs.

C. Hardware Accelerator Performance

Area and Energy Evaluation. Table IV shows the power and area breakdown of SPEQ. It has a total area of 6.3 mm². The PE array takes up 39.4% area, the decoder only occupies 3.5% of the total area, which is a small overhead. In the quantize mode, the PE array takes up 36.5% energy. The decoder unit only occupies 3.2% power. In the full mode, the PE array takes up 40.0% energy. The decoder unit only occupies 3.1% power. When working at 500 MHz, the power of quantize mode is 508 mW, and the power of full mode is 559 mW. The similar power consumption of the quantize mode and full mode indicates high hardware utilization of SPEQ accelerator in both the two modes.

Comparison with Other LLM Quantization Accelerators. The comparison of decoding speedup is shown in Fig. 7. We use the mean speedup of the three tasks as the speedup of SPEQ. Methods exhibiting **severe performance degradation** (4-bit Olive and 4-bit Tender) are marked in gray. SPEQ achieves to an average speedup of 2.07×

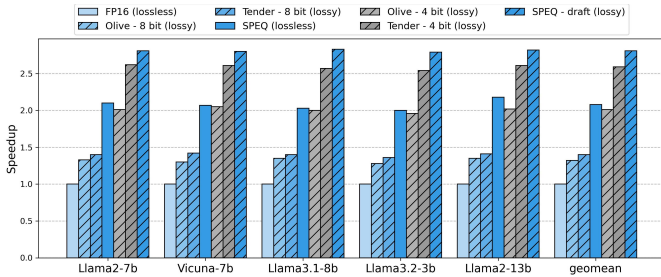


Fig. 7. Speedup of SPEQ compared with FP16 baseline, Olive, and Tender.

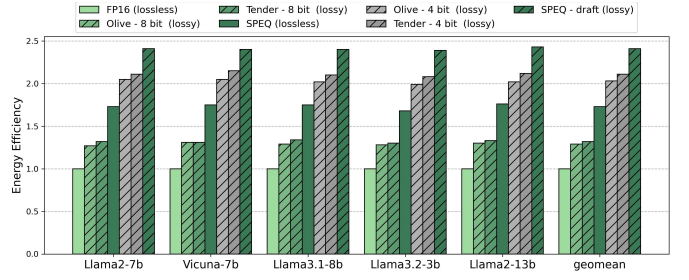


Fig. 8. Energy efficiency of SPEQ compared with FP16 baseline, Olive, and Tender.

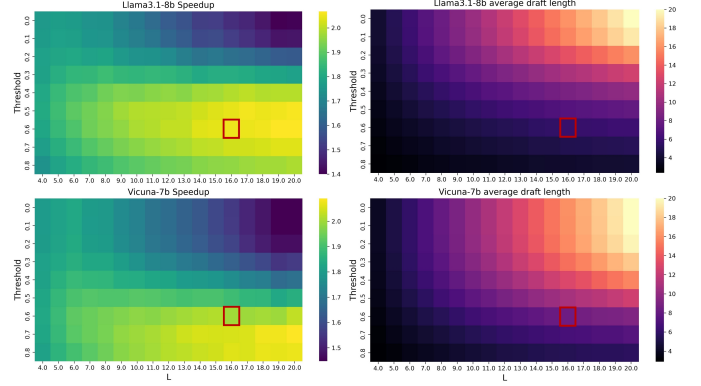


Fig. 9. Ablation study of the hyperparameter settings. The square stands for the default setting of $L = 16, \gamma = 0.6$.

bit Olive [7], and 8-bit Tender [9], respectively. Additionally, SPEQ demonstrates a similar speedup when compared with 4-bit Olive.

The superior performance of SPEQ derives from the efficient processing of the quantized forward, where only the exponents are added to perform multiplication between the quantized value and the activation. SPEQ achieves a high accept rate and long draft length, minimizing the latency overhead of the verification process.

The comparison of energy efficiency is shown in Fig. 8. SPEQ achieves 1.74×

 energy efficiency compared with the FP16 baseline, 1.35× and 1.32× compared with 8-bit Olive and Tender.

D. Comparison with Other Speculative Decoding Methods

We further compare SPEQ with other speculative decoding methods, including Swift [18] Medusa [1] for Vicuna-7b on MT-bench. As shown in Fig. 2(a), SPEQ achieves a speedup of 2.03×

surpassing Swift by 1.52× and **Medusa by 1.05×** without additional training and extra memory overhead.

E. Ablation Study on Hyperparameter Settings

We conduct an ablation study on the hyperparameter settings of the maximum draft length L and the early stopping threshold γ for Llama3.1-8b and Vicuna-7b on MT-bench. The value of L ranges from 4 to 20 with an interval of 1, and γ ranges from 0 to 0.8 with an interval of 0.1. The results are shown in Fig. 9. Our default setting of $L = 16, \gamma = 0.6$ yields significant speedup. The optimal choice of hyperparameters depends on the specific model and task. Although our default settings may not represent the absolute optimal configuration, the results in Table III demonstrate that they achieve performance comparable to the optimal settings.

VI. CONCLUSION

In this paper, we propose SPEQ, an algorithm-hardware co-design solution that accelerates LLM decoding without sacrificing accuracy. SPEQ employs a bit-sharing quantization algorithm with a reconfigurable PE array to support both quantized draft and full model verification, providing efficient computation and high hardware utilization. Experiments on 15 LLMs and generative tasks demonstrate that SPEQ outperforms prior lossy quantization accelerators, offering a promising solution for accelerating LLM inference in real-world applications.

REFERENCES

- [1] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, “Medusa: Simple LLM inference acceleration framework with multiple decoding heads,” in *ICML*. OpenReview.net, 2024.
- [2] M. Chen, J. Twarek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, M. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, “Evaluating large language models trained on code,” *CoRR*, vol. abs/2107.03374, 2021.
- [3] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Twarek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, “Training verifiers to solve math word problems,” *CoRR*, vol. abs/2110.14168, 2021.
- [4] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Rozière, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. M. Kloumann, I. Misra, I. Evtimov, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnston, J. Saxe, J. Jia, K. V. Alwala, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, and et al., “The llama 3 herd of models,” *CoRR*, vol. abs/2407.21783, 2024.
- [5] K. Egashira, M. Vero, R. Staab, J. He, and M. T. Vechev, “Exploiting LLM quantization,” in *NeurIPS*, 2024.
- [6] E. Frantar, S. Ashkboos, T. Hoefer, and D. Alistarh, “GPTQ: accurate post-training quantization for generative pre-trained transformers,” *CoRR*, vol. abs/2210.17323, 2022.
- [7] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, “Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization,” in *ISCA*. ACM, 2023, pp. 3:1–3:15.
- [8] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, “Measuring mathematical problem solving with the MATH dataset,” in *NeurIPS Datasets and Benchmarks*, 2021.
- [9] J. Lee, W. Lee, and J. Sim, “Tender: Accelerating large language models via tensor decomposition and runtime requantization,” in *ISCA*. IEEE, 2024, pp. 1048–1062.
- [10] Y. Leviathan, M. Kalman, and Y. Matias, “Fast inference from transformers via speculative decoding,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 19 274–19 286.
- [11] Y. Li, F. Wei, C. Zhang, and H. Zhang, “EAGLE: speculative sampling requires rethinking feature uncertainty,” in *ICML*. OpenReview.net, 2024.
- [12] Z. Li, Y. Su, R. Yang, Z. Xie, N. Wong, and H. Yang, “Quantization meets reasoning: Exploring LLM low-bit quantization degradation for mathematical reasoning,” *CoRR*, vol. abs/2501.03035, 2025.
- [13] J. Lin, J. Tang, H. Tang, S. Yang, W. Chen, W. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, “AWQ: activation-aware weight quantization for on-device LLM compression and acceleration,” in *MLSys*. mlsys.org, 2024.
- [14] S. Merity, C. Xiong, J. Bradbury, and R. Socher, “Pointer sentinel mixture models,” in *ICLR (Poster)*. OpenReview.net, 2017.
- [15] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R. Y. Y. Wong, A. Zhu, L. Yang, X. Shi, C. Shi, Z. Chen, D. Arfeen, R. Abhyankar, and Z. Jia, “Specinfer: Accelerating large language model serving with tree-based speculative inference and verification,” in *ASPLOS (3)*. ACM, 2024, pp. 932–949.
- [16] OpenAI, “GPT-4 technical report,” *CoRR*, vol. abs/2303.08774, 2023.
- [17] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton-Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, “Llama 2: Open foundation and fine-tuned chat models,” *CoRR*, vol. abs/2307.09288, 2023.
- [18] H. Xia, Y. Li, J. Zhang, C. Du, and W. Li, “SWIFT: on-the-fly self-speculative decoding for LLM inference acceleration,” in *ICLR*. OpenReview.net, 2025.
- [19] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, “Smoothquant: Accurate and efficient post-training quantization for large language models,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 38 087–38 099.
- [20] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, “Qwen2.5 technical report,” *CoRR*, vol. abs/2412.15115, 2024.
- [21] J. Zhang, J. Wang, H. Li, L. Shou, K. Chen, G. Chen, and S. Mehrotra, “Draft& verify: Lossless large language model acceleration via self-speculative decoding,” in *ACL (1)*. Association for Computational Linguistics, 2024, pp. 11 263–11 282.
- [22] L. Zheng, W. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, “Judging llm-as-a-judge with mt-bench and chatbot arena,” in *NeurIPS*, 2023.