2025 Design, Automation & Test in Europe Conference (DATE 2025)

FineQ: Software-Hardware Co-Design for Low-Bit Fine-Grained Mixed-Precision Quantization of LLMs

Xilong Xie[1], Liang Wang[1]*, Limin Xiao[1]*, Meng Han[1], Lin Sun[2], Shuai Zheng[1], Xiangrong Xu[1]
[1] School of Computer Science and Engineering, Beihang University, Beijing 100191, China
{xxl1399, lwang20, xiaolm, hanm, zhengshuai, xxr0930}@buaa.edu.cn
[2] Jiangsu Shuguang Optoelectric Co., Ltd
m13773555855@163.com

***Abstract*—Large language models (LLMs) have significantly advanced the natural language processing paradigm but impose substantial demands on memory and computational resources. Quantization is one of the most effective ways to reduce memory consumption of LLMs. However, advanced single-precision quantization methods experience significant accuracy degradation when quantizing to ultra-low bits. Existing mixed-precision quantization methods are quantized by groups with coarse granularity. Employing high precision for group data leads to substantial memory overhead, whereas low precision severely impacts model accuracy. To address this issue, we propose FineQ, software-hardware co-design for low-bit fine-grained mixed-precision quantization of LLMs. First, FineQ partitions the weights into finer-grained clusters and considers the distribution of outliers within these clusters, thus achieving a balance between model accuracy and memory overhead. Then, we propose an outlier protection mechanism within clusters that uses 3 bits to represent outliers and introduce an encoding scheme for index and data concatenation to enable aligned memory access. Finally, we introduce an accelerator utilizing temporal coding that effectively supports the quantization algorithm while simplifying the multipliers in the systolic array. FineQ achieves higher model accuracy compared to the SOTA mixed-precision quantization algorithm at a close average bit-width. Meanwhile, the accelerator achieves up to 1.79× energy efficiency and reduces the area of the systolic array by 61.2%.**

***Index Terms*—Large Language Models, Fine-Grained Mixed-Precision Quantization, Accelerator.**

I. Introduction

Large language models (LLMs) based on transformers [1] have demonstrated remarkable performance in various natural language processing tasks through their enormous model size. However, LLMs, as one of the most significant computational workloads today, are challenging to deploy on edge devices due to their extensive memory and computational resources required [2]. For instance, deploying the LLaMA-2-70B in FP16 requires at least 140 GB of memory [3], markedly exceeding the 80 GB capacity of an A100 GPU. Weight quantization [4]–[11] is one of the most effective ways to reduce the memory consumption of LLMs while preserving performance, achieved by storing parameters in a low-precision representation.

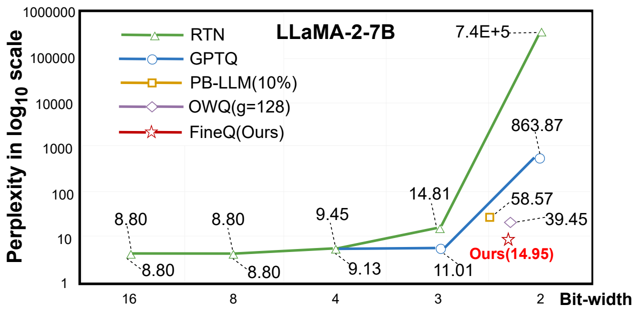


Fig. 1. Perplexity of LLaMA-2-7B on C4 under differnet bit-widths. Lower perplexity means better model accuracy. Round-to-nearest (RTN), GPTQ, PB-LLM (10% weight of FP16), and OWQ(g=128) suffer from accuracy loss at ultra-low bits. FineQ demonstrates better performance than these methods.

To enable the future deployment of LLMs on edge devices [12], more aggressive quantization and compression techniques will be essential. Existing quantization methods are still inadequate for this purpose, as reducing precision to ultra-low bits($<$ 3 bits) leads to a significant degradation in model accuracy. Currently, weight quantization techniques primarily fall into single-precision quantization and mixed-precision quantization. Several single-precision quantization methods, such as RTN [11], GPTQ [4], have successfully quantized the FP16 LLMs into 4-bit or 3-bit formats while maintaining acceptable performance. Regrettably, as shown in Figure 1, advanced single-precision quantization methods for LLMs experience notable performance degradation under ultra-low bit quantization. This observation underscores the challenges that existing methods encounter when quantizing models to ultra-low bits.

The substantial loss of model accuracy in single-precision quantization is mainly due to the inadequate handling of weight outliers [5]. Outliers are identified by a small set of values with exceptionally high magnitudes [13]. Single-precision quantization methods indiscriminately clip both outliers and normal values, leading to notable drops in model accuracy [6]. To maintain model accuracy, recent works apply mixed-precision quantization methods. For instance, OWQ [8] and LLM-MQ [9] quantize the majority of normal weight values to 2 bits while representing outliers in FP16 format. PB-LLM [10] binarizes a subset of weights in LLMs while preserving outliers with high precision. Though existing mixed-precision quantization algorithms preserve model accuracy to some ex-

* Liang Wang and Limin Xiao are corresponding authors of this paper.

This work was supported in part by the National Science and Technology Major Project under Grant No. 2022ZD0117602; in part by the National Natural Science Foundation of China under Grants 62272026 and 62104014; in part by the Fundamental Research Funds for the Central Universities; in part by State Key Laboratory of Complex & Critical Software Environment under Grant No. CCSE-2024ZX-10.

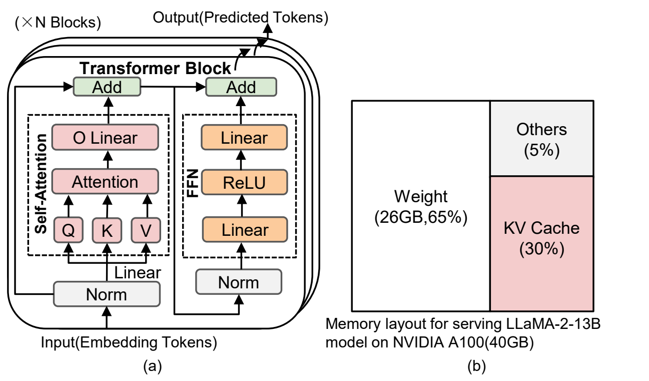


Fig. 2. (a) The transformer block architecture. (b) Memory layout for serving a 13B-parameter LLM on the NVIDIA A100 (40GB) [14].

tent, they still demonstrate shortcomings in the handling of outliers. In existing mixed-precision quantization methods [8]–[10], [15], weights are quantized in coarse-grained groups, with certain groups containing weights with large range. Therefore, these methods fail to adequately protect outliers and struggle to achieve a trade-off between memory overhead and model accuracy. Moreover, these methods treat outliers as sparse matrices, necessitating additional sparse formats and complex control logic to handle irregular computations and memory access, leading to lower computational efficiency.

In this work, we introduce FineQ, a low-bit fine-grained mixed-precision quantization method of LLMs with software-hardware co-design. We aim to achieve a balance between model accuracy and memory overhead by adopting a fine-grained outlier protection strategy. Specifically, we propose a series of innovative mechanisms and architectures, including a fine-grained mixed-precision quantization algorithm, an intra-cluster outlier protection mechanism, and a temporal coding-based hardware accelerator. The main contributions of this paper are summarized as follows.

1) We propose a fine-grained mixed-precision quantization algorithm that does not require retraining. The algorithm partitions model weights into smaller clusters and considers the distribution of outliers within each small cluster, enabling quantization to lower bit-width while maintaining model accuracy.

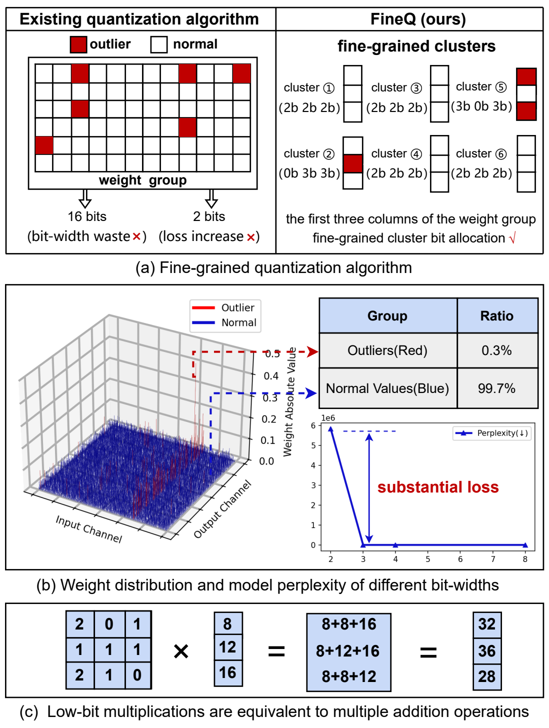
2) We introduce an intra-cluster outlier protection mechanism. By observing the distribution of weight outliers and the effects of various quantization bit-widths on model accuracy, we quantize the majority of normal values to 2 bits, apply 3-bit protection for outliers, and use an innovative memory alignment scheme that integrates encoding information with the data.

3) We present an efficient hardware accelerator based on temporal coding to support the quantization algorithm. Furthermore, temporal coding allows for efficient bitstream generation from weights [16], significantly simplifying the multiplier design in the systolic array. The accelerator achieves up to 1.79× energy efficiency and reduces the systolic array area by 61.2%.

II. Background and Motivation

A. Transformer-Based Large Language Models

The transformer model serves as the backbone architecture for existing LLMs [1], enabling efficient processing for tasks



Group	Ratio
Outliers(Red)	0.3%
Normal Values(Blue)	99.7%

$$\begin{bmatrix} 2 & 0 & 1 \\ 1 & 1 & 1 \\ 2 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 8 \\ 12 \\ 16 \end{bmatrix} = \begin{bmatrix} 8+8+16 \\ 8+12+16 \\ 8+8+12 \end{bmatrix} = \begin{bmatrix} 32 \\ 36 \\ 28 \end{bmatrix}$$

Fig. 3. Three observations about hardware-software co-design of mixed-precision quantization.

such as natural language understanding and generation [17], [18]. As illustrated in Figure 2(a), each transformer block contains self-attention and feed-forward network (FFN). The self-attention mechanism quantifies the degree of association among different positions within a single sequence and it consists of QKV generation, attention, and a linear layer. The FFN consists of two linear layers and an activation function.

In recent years, the scale of large language models has continued to grow [19]. Figure 2(b) illustrates the memory distribution for a 13B-parameter LLM on an NVIDIA A100 GPU. Approximately 65% of the memory is allocated for the model weights [14]. To facilitate the deployment of LLMs on memory-constrained edge devices, weight quantization emerges as one of the most effective approaches, as it can substantially reduce the memory overhead of large language models. However, weight outliers significantly impact model accuracy, making it challenging to quantize the model to ultra-low bits while maintaining model accuracy. Thus, effectively protecting weight outliers is crucial [15], [20].

B. Related Works

Previous works [4], [7]–[10] on weight quantization, including single-precision and mixed-precision methods, aim to minimize memory overhead without affecting the accuracy of LLMs. Several single-precision quantization methods such as GPTQ [4] and AWQ [7] quantize the weight matrix in single-precision. Among them, GPTQ introduces an innovative one-shot weight quantization technique that leverages approximate second-order information. AWQ protects the salient weights by observing the distribution of activation values. Regrettably, these methods suffer from a significant precision loss when quantizing to ultra-low bits, due to outlier issues. In addition, some mixed-precision quantization methods are proposed to maintain model accuracy. For instance, OWQ [8] and LLM-MQ [9] store weight outliers in FP16 Format separately from normal values. PB-LLM [10] binarizes a portion of the model's weights while preserving some outliers. However, these methods still suffer from accuracy loss due to inter-group data variability. Additionally, methods like Olive [21] and Tender [22] employ hardware-software co-design for outlier-aware quantization. They quantize both activations and weights to 4 bits and focus on outliers in activations rather than weights.

C. Observation and Key Idea

Through extensive exploration of existing implementations, we have three crucial observations that indicate significant optimization potential for hardware-software co-design of mixed-precision quantization, as shown in Figure 3.

Observation I: Mixed-precision quantization methods through coarse-grained groups struggle to strike a balance between model accuracy and memory overhead. Existing mixed-precision quantization algorithms operate at the group level, with each group typically consisting of more than 128 weight values, and the distribution of outliers within the group is random. As illustrated in Figure 3(a), a coarse-grained group may contain only a small number of outliers. Using a higher bit-width to store the entire group increases memory overhead, while a lower bit-width may fail to capture certain outliers, significantly impacting model accuracy.

Key idea: We propose a fine-grained intra-cluster quantization algorithm that splits weight data into smaller clusters, each containing only three weight values. Furthermore, we allocate bit-widths based on the data within each cluster, balancing both model accuracy and memory overhead.

Observation II: Employing excessively high precision for weight outliers is unnecessary. Figure 3(b) visualizes the input weight distribution of a representative linear layer in the LLaMA-2-7B model [3] and the model accuracy achieved with uniform quantization under different bit-widths [23]. The blue color represents normal values in the weights while the red color indicates outliers. Through observation, we found that over 99% of the weights have very similar values. This indicates that only sparse outliers require additional processing, while the majority of weights can be quantized to lower bits. The distribution of outliers is not random but instead tends to concentrate within specific channels. Moreover, reducing the quantization bit-width from 3 bits to 2 bits significantly affects model accuracy. In contrast, increasing the bit-width from 3 bits to 16 bits has a limited impact on accuracy.

Key idea: We apply quantization across weight channels and allocate 3 bits for outliers, further reducing the memory overhead without significantly impacting model accuracy.

Observation III: Low-bit quantization can significantly simplify the design of hardware multipliers. Low-bit multiplication operations can be transformed into multiple accumulation operations. For instance, an 8×2 multiplication can be represented as an $8 + 8$ addition operation. Similarly, as illustrated in Figure 3(c), low-bit matrix multiplication can be decomposed into multiple parallel accumulation operations without affecting the computational results.

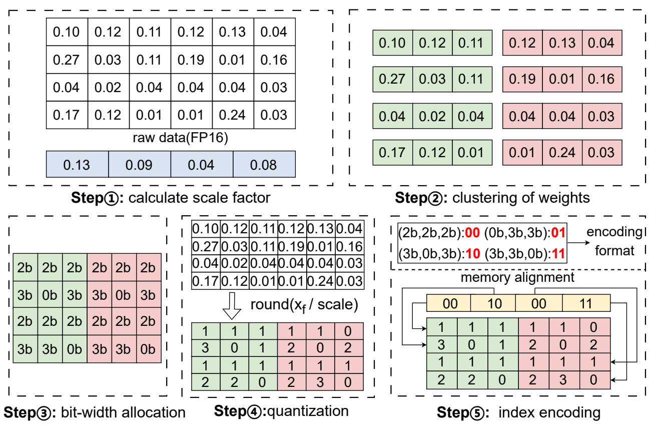


Fig. 4. An example of fine-grained intra-cluster quantization algorithm.

Key idea: We propose an efficient accelerator based on temporal coding, which encodes weight data into bitstreams. The accelerator significantly simplifies the multiplier design in the systolic array and improves hardware efficiency.

III. Algorithm Design

In this section, we introduce the fine-grained intra-cluster quantization algorithm, which partitions the weights into fine-grained clusters and separates them from other clusters to minimize quantization error. We propose an intra-cluster outlier protection mechanism that allocates 3 bits for outliers, effectively balancing memory overhead and model accuracy. Moreover, we provide a walking example of the algorithm.

Algorithm 1: Algorithm for fine-grained intra-cluster weight quantization.

```
Input: Weight Matrix $W$; Layer Number $L$;
Output: Quantization Weight Matrix $Q$
for each layer $l$ and each channel $c$ in the weight matrix $W_l$ do
    $W_{l,c} \leftarrow$ weight vector of channel $c$ in layer $l$
    $num\_clusters \leftarrow \lceil \text{length}(W_{l,c})/3 \rceil$
    Divide $W_{l,c}$ into $num\_clusters$ of size 3
    for each cluster $C$ in $W_{l,c}$ do
        $max\_val \leftarrow \max(C)$
        $min\_val \leftarrow \min(C)$
        if $max\_val > 4 \times min\_val$ then
            Encode the top two values with 3 bits
            Set remaining values to 0
        end
        else
            Encoding all values in $C$ with 2 bits
        end
        foreach $C_i$ in clusters do
            if $C_i$ has no neighbor then
                Assign default encoding to $C_i$
            else
                if neighbor $C_j$ has encoding then
                    $E(C_i) \leftarrow E(C_j)$
                else
                    $E(C_i), E(C_j) \leftarrow \arg\min_l Loss(C_i, C_j, l)$
                end
            end
        end
        $Q \leftarrow UniformQuantization(C)$
    end
end
```

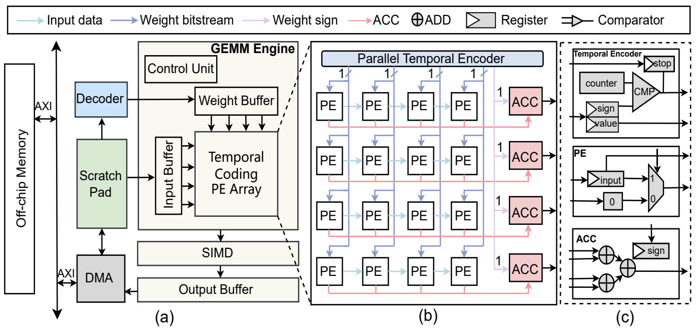


Fig. 5. (a) Overview of FineQ architecture. (b) Temporal coding PE array design. (c) Temporal Encoder, PE, and ACC design.

A. Fine-grained Intra-cluster Quantization Algorithm

To balance memory overhead and model accuracy, we propose a fine-grained intra-cluster quantization algorithm, as illustrated in steps ① and ② of Figure 4. First, the scaling factor for each channel is computed according to Equation 1, where b is the bit-width, s represents the scale factor, x_{max} is the maximum value, abs is the absolute value operation. Then, as shown in Figure 3(b), most outliers are concentrated in specific channels. We perform per-channel quantization of weights and divide every three weights within each channel into a fine-grained weight cluster. The bit-width allocation for each cluster is determined exclusively by the data within that cluster. Before bit-width allocation within clusters, outliers within each cluster are identified by comparing the maximum and minimum values. If the maximum exceeds four times the minimum, the outlier protection mechanism is applied; otherwise, all cluster values are quantized to 2 bits.

$$s = \frac{abs(x_{max})}{2^{b-1}-1}; \quad x_q = round\left(\frac{x_f}{s}\right) \tag{1}$$

B. intra-cluster Outlier Protection Mechanism

The intra-cluster outlier protection mechanism is used to protect certain outliers in the model weights, thereby preserving model accuracy. Figure 4 ③ provides an example of bit-width allocation within a cluster. As shown in Figure 3(b), quantizing to 3 bits strikes a balance between memory overhead and model accuracy. Unlike existing methods that typically use 16 bits to protect outliers [8], [9], we represent outliers with 3 bits. When outliers are detected within a cluster, the two largest values are encoded with 3 bits, while the smallest value is sacrificed.

Following bit-width allocation, as illustrated in Figure 4 step ④, uniform quantization is performed according to Equation 1, where $round$ is an approximation operation, x_f and x_q are a floating-point value and the quantized one. The bit-width allocation method effectively considers the distribution of outlier points within the cluster. The quantized data is converted from float to int, with outliers being effectively protected.

To distinguish the encoding scheme for each cluster, we use 2-bit encoding to specify the bit-width allocation: '00' indicates all values are 2 bits, '01' means the first value is zero with the remaining two encoded in 3 bits, '10' sets the second value to zero, and '11' sets the third value to zero. Additionally, we introduce a compression strategy that requires adjacent clusters to use the same encoding. If two adjacent clusters use different encoding schemes, we can apply fine-tuning to select the encoding. By iterating through four encoding schemes to select the one that minimizes the average error relative to the original data. As shown in Figure 3(b), most weights are encoded in 2 bits, so fine-tuning is rarely needed. To ensure aligned memory access, we employ a specialized encoding scheme for both indices and data. As shown in Figure 4 step ⑤. Specifically, four index values are encoded within a single byte to distinguish the encoding formats of the subsequent eight clusters. This approach reduces the overhead of storing sparse outlier indices compared to existing mixed-precision methods and facilitates more efficient memory access. Moreover, the entire quantization algorithm is performed offline and the pseudocode is shown in Algorithm 1.

IV. Architecture Design

While the FineQ algorithm can be implemented in software, its full potential is realized through a custom accelerator design. In this section, we propose an efficient accelerator based on temporal coding. Our hardware design effectively supports fine-grained mixed-precision quantization algorithm through an efficient decoder unit that supports various cluster encoding schemes. Furthermore, we design a temporal coding PE array, which simplifies multipliers compared to conventional systolic array. Finally, we present a computational example.

A. Architecture Overview

Figure 5(a) presents the overview of FineQ architecture, it consists of a decoder unit, a vector processing unit, on-chip buffers for input/weight/output data, a direct memory access (DMA) unit, a temporal coding systolic array with input-stationary dataflow [24] and a control unit. The architecture features a six-stage pipeline: **(1) Off-chip Memory Access:** Weights and input data are read from off-chip memory and transferred to on-chip buffers via DMA. **(2) Decode:** The weight data is loaded into the buffer after passing through the decoder. **(3) Input Preloading:** The control unit sends control signals to the systolic array, while the input data is loaded into its registers. **(4) Matrix Multiplication:** Matrix multiplication is executed within the temporal coding systolic array, and the partial sums are then forwarded to the vector unit. **(5) Vector Processing:** Activation functions are executed by the vector unit. **(6) Write-back to Off-chip Memory:** Finally, the results are written back to the off-chip memory unit via DMA.

B. Decoder Design

The design of the weight decoder unit is shown in Figure 6. First, the input is divided into index and weight data, where the index is used to distinguish the quantization format of the cluster. Since the decoded data may be either 3 bits or 2 bits, we ensure all decoded data is 3 bits by padding with zeros. Next, the data is passed through multiple selection units. These units use the index information to output the decoded data

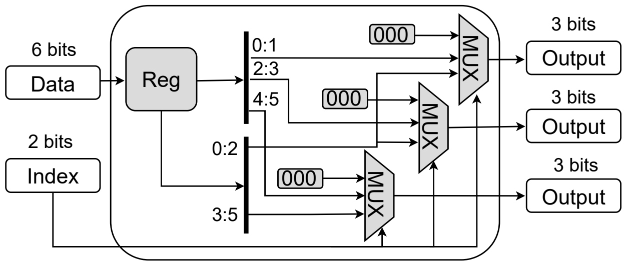


Fig. 6. FineQ decoder design

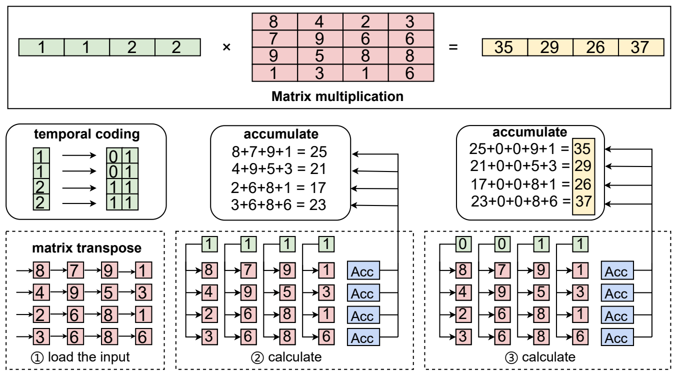


Fig. 7. An example of matrix multiplication with temporal coding PE array.

and determine whether to perform zero-padding. Finally, the process decodes a cluster of data into three 3-bit weights.

C. Temporal Coding PE Array Design

Based on the idea that low-bit multiplication can be transformed into multiple accumulations, we propose a temporal coding PE array. It consists of parallel temporal encoder, processing elements (PEs), and accumulation units(ACCs). Temporal coding [16] is a lossless encoding method where the number of ones in the bitstreams represents the value. As shown in Figure 7, the value 2 is encoded as '11' and the value 1 as '01'. Through temporal coding, low-bit data is converted into fixed-length bitstreams. Figure 5(b) shows the hardware architecture of the temporal encoder. The temporal encoder records the input data value, compares it with the counter, and outputs the result through a comparator. Additionally, the parallel temporal encoder can receive termination signals from the control module to stop bitstream generation. Then, the generated bitstreams are broadcast to each column of processing elements (PEs), and due to the 1-bit transmission width, the overhead associated with broadcasting remains minimal. Next, the PE stores the activation data. When weight bitstreams arrive, the selector determines whether to output the activation data or zero. In the accumulation unit, the sign bits of the weights and the outputs from each PE row are processed and accumulated using an adder tree structure. Overall, this design simplifies the MAC units of traditional systolic arrays and improves hardware efficiency.

Figure 7 also provides an example of matrix multiplication performed using temporal coding PE array. First, the control unit issues the control flow signal, and the input data is loaded into the PE units. Second, the bitstreams generated by the temporal encoder are broadcasted to the PE array. In step ②, the partial results are output by the accumulation unit, and in step ③, these results are reused and further accumulated until the computation is complete. It can be observed that the final computation results match those of the matrix multiplication.

V. Evaluation

In this section, we evaluate FineQ in terms of model accuracy, area, and energy efficiency.

A. Evaluation Setup

Quantization Setup. We choose LLaMA family(3B-13B) [3] to evaluate our method. We use perplexity as the evaluation metric, which is a widely used one for LLMs. The perplexity of LLMs is evaluated on the WikiText2 [25] and C4 datasets [26]. Lower perplexity means better model accuracy.

Quantization Baseline. We compare FineQ with five existing weight quantization methods: (1) Uniform [23] uses symmetric uniform quantization; (2) RTN [11] rounds all weights to the nearest quantized value on a fully uniform, asymmetric per-row grid; (3) GPTQ [4] leverages approximate second-order information; (4) PB-LLM [10] binarizes a subset of the weights; (5) OWQ [8] employs mixed precision including FP16 and INT2. RTN, Uniform, and GPTQ are single-precision methods, thus the bit width is set to 2. PB-LLM protects 10% of the weight values, resulting in an average bit-width of 2.7 bits. OWQ employs a group size of 128, achieving an average bit-width of 2.25 bits. For FineQ, we reduce the weight bit-width to 2.33 bits.

Accelerator Baseline. We compare the area and energy efficiency of the FineQ accelerator with the systolic array using MAC Units. They have the same on-chip buffer size and use input-stationary dataflow [24]. The models used as workloads are from the quantization baseline.

Architecture Implementation. We implement the temporal coding PE array and decoder using Verilog HDL and verify the design through RTL simulations. The temporal PE array consists of 4096 PEs and 64 Decoders. The accelerator is synthesized by Synopsys Design Compiler [27] in a 45 nm process technology [28]. We also develop a cycle-level simulator to estimate the overall performance.

B. Accuracy Evaluation

Performance on LLMs. We first evaluate the perplexity of the FineQ quantization algorithm, with an average bit-width quantized to 2.33 bits. As shown in Table I, FineQ outperforms existing methods on various models. Uniform, RTN, and GPTQ utilize single-precision quantization, and the results indicate that quantizing to 2 bits in these methods leads to significant loss due to the lack of protection from outliers. PB-LLM and OWQ employ mixed-precision quantization and are quantized by coarse-grained groups, making it difficult to effectively protect weight outliers under ultra-low bits. Furthermore, PB-LLM binarizes a substantial portion of the weights, employing a more aggressive strategy that results in a decrease in model accuracy. FineQ protects outliers through fine-grained quantization, thereby effectively maintaining model accuracy.

TABLE I
PERPLEXITY RESULTS OF QUANTIZED MODEL ON WIKITEXT2 AND C4 DATASETS.

Model(PPL)			**LLaMA-2-3B**		**LLaMA-2-7B**		**LLaMA-2-13B**	
Method	Avg. Bits	Seq.	Wiki	C4	Wiki	C4	Wiki	C4
FP16	16	2048	7.35	9.58	6.61	8.81	5.97	8.19
RTN	2	2048	1.6E+5	1.6E+5	4.3E+4	7.4E+5	6.3E+4	6.0E+4
Uniform	2	2048	6.3E+6	6.5E+6	5.8E+6	5.8E+6	2.6E+5	2.1E+5
GPTQ	2	2048	1675.56	5090.50	256.17	863.87	248.59	506.32
PB-LLM 10%	2.7	2048	60.38	123.04	28.59	58.57	131.54	208.34
OWQ(g128)	2.25	2048	34.51	75.78	22.95	39.45	15.19	26.03
FineQ(Ours)	2.33	2048	**13.69**	**19.04**	**10.94**	**14.95**	**13.16**	**18.55**

TABLE II
PERPLEXITY ACROSS DIFFERENT SEQUENCE LENGTHS.

SeqLen.(PPL)		**32**		**256**		**1024**	
Method	Avg. Bits	Wiki	C4	Wiki	C4	Wiki	C4
FP16	16	39.19	22.14	10.90	11.21	7.35	9.19
RTN	2	4.2E+4	3.5E+4	5.3E+4	5.5E+4	5.0E+4	6.8E+4
Uniform	2	4.3E+6	5.3E+6	5.0E+6	5.4E+6	5.4E+6	5.3E+6
GPTQ	2	2.0E+5	1.7E+5	1.5E+5	1432.38	2.3E+5	1289.9
PB-LLM 10%	2.7	286.13	271.18	52.60	73.19	32.41	58.97
OWQ(g128)	2.25	5.4E+4	6.3E+4	71.58	81.01	29.53	44.74
FineQ(Ours)	2.33	**64.47**	**26.68**	**20.89**	**18.46**	**12.52**	**15.77**

Sequence Length Sensitivity. Table II presents a comparison of FineQ and previous methods on the LLaMA-2-7B model across three different sequence lengths. The results indicate that, while FineQ exhibits a slight increase in perplexity as sequence length decreases, it consistently outperforms prior methods in model accuracy. Additionally, FineQ exhibits greater robustness to variations in sequence length, demonstrating lower sensitivity to changes compared to other approaches.

C. *Performance and Area Evaluation*

Area and Power. We compare the accelerator area breakdown in Table III. The systolic array primarily consists of MAC units and accumulators. At the 400 MHz clock frequency and under the input- stationary dataflow configuration, our temporal coding PE array reduces the area of the systolic array by 61.2% and achieves a 62.9% reduction in power consumption. Furthermore, as illustrated in Figure 8, we present the power consumption distribution for each module in the FineQ PE array, where the Acc unit constitutes 71.8%, the PE Array 25.9%, and the Temporal Encoder accounts for only 2.3%. The simplification of multipliers has significantly reduced the overhead of the PE array.

TABLE III
THE AREA AND POWER BREAKDOWN OF ACCELERATOR CORE MODULES

Architecture	Setup	Area	Power
Systolic Array	64×64 PEs	0.954 mm^2	88.793 mw
FineQ Decoder	64	0.008 mm^2	0.187 mw
FineQ PE Array	64×64 PEs	0.370 mm^2	32.891 mw

Energy Efficiency. Figure 9 illustrates the normalized energy efficiency of the accelerator at different sequence lengths under the same on-chip buffer size. Compared to the baseline design,

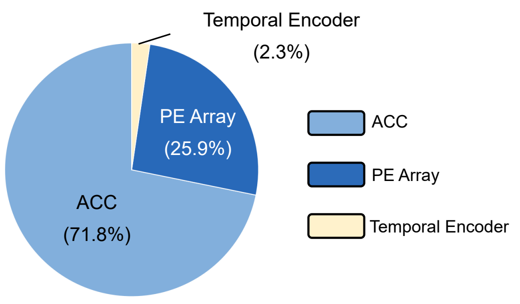


Fig. 8. Breakdown of power in FineQ PE array.

the accelerator exhibits higher energy efficiency, attributed to the simplified design of most PEs and the reduction in the complexity of multipliers. Overall, the accelerator achieves up to 1.79× average energy efficiency on various models.

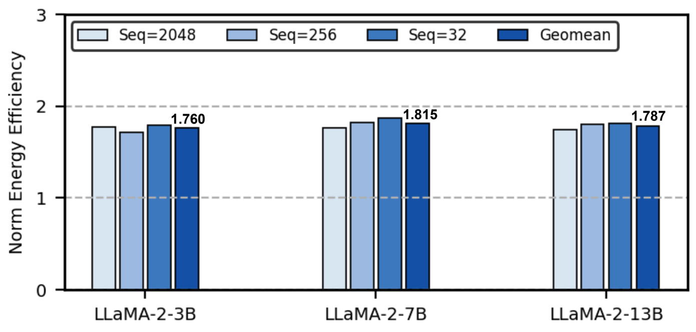


Fig. 9. Normalized energy efficiency over baseline accelerator.

VI. CONCLUSION

In this paper, we propose FineQ, a software-hardware co-design for low-bit fine-grained mixed-precision quantization of LLMs. The key insight is to handle outliers within clusters at a fine granularity. We address the issue of coarse quantization granularity in LLMs by proposing a fine-grained mixed-precision quantization algorithm, effectively balancing model accuracy and memory overhead. FineQ achieves higher model accuracy compared to the SOTA mixed-precision quantization algorithm at a close average bit-width. Additionally, we introduce an accelerator based on temporal coding to support this algorithm, simplifying multipliers in the systolic array and achieving up to 1.79× energy efficiency.

References

[1] A. Vaswani, “Attention is all you need,” *Advances in Neural Information Processing Systems*, 2017.

[2] W. Luk, K. F. C. Yiu, R. Li, K. Mishchenko, S. I. Venieris, H. Fan *et al.*, “Hardware-aware parallel prompt decoding for memory-efficient acceleration of llm inference,” *arXiv preprint arXiv:2405.18628*, 2024.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.

[4] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, “Gptq: Accurate post-training quantization for generative pre-trained transformers,” *arXiv preprint arXiv:2210.17323*, 2022.

[5] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, “Zeroquant: Efficient and affordable post-training quantization for large-scale transformers,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 168–27 183, 2022.

[6] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, “Llm. int8 (): 8-bit matrix multiplication for transformers at scale. corr abs/2208.07339 (2022),” 2022.

[7] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, “Awq: Activation-aware weight quantization for on-device llm compression and acceleration,” *Proceedings of Machine Learning and Systems*, vol. 6, pp. 87–100, 2024.

[8] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park, “Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 12, 2024, pp. 13 355–13 364.

[9] S. Li, X. Ning, K. Hong, T. Liu, L. Wang, X. Li, K. Zhong, G. Dai, H. Yang, and Y. Wang, “Llm-mq: Mixed-precision quantization for efficient llm deployment,” in *The Efficient Natural Language and Speech Processing Workshop with NeurIPS*, vol. 9, 2023.

[10] Y. Shang, Z. Yuan, Q. Wu, and Z. Dong, “Pb-llm: Partially binarized large language models,” *arXiv preprint arXiv:2310.00034*, 2023.

[11] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, “Up or down? adaptive rounding for post-training quantization,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 7197–7206.

[12] S. Laskaridis, K. Katevas, L. Minto, and H. Haddadi, “Mobile and edge evaluation of large language models,” in *Workshop on Efficient Systems for Foundation Models II@ ICML2024*.

[13] X. Wei, Y. Zhang, Y. Li, X. Zhang, R. Gong, J. Guo, and X. Liu, “Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling,” *arXiv preprint arXiv:2304.09145*, 2023.

[14] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, “Efficient memory management for large language model serving with pagedattention,” in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[15] Z. Guan, H. Huang, Y. Su, H. Huang, N. Wong, and H. Yu, “Aptq: Attention-aware post-training mixed-precision quantization for large language models,” *arXiv preprint arXiv:2402.14866*, 2024.

[16] D. Wu, J. Li, Z. Pan, Y. Kim, and J. S. Miguel, “ubrain: A unary brain computer interface,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 468–481.

[17] K. Nassiri and M. Akhloufi, “Transformer models used for text-based question answering systems,” *Applied Intelligence*, vol. 53, no. 9, pp. 10 602–10 635, 2023.

[18] G. Keswani, W. Bisen, H. Padwad, Y. Wankhedkar, S. Pandey, and A. Soni, “Abstractive long text summarization using large language models,” *Int. J. Intell. Syst. Appl. Eng*, vol. 12, pp. 160–168, 2024.

[19] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, “A survey of large language models,” *arXiv preprint arXiv:2303.18223*, 2023.

[20] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer, “Squeezellm: Dense-and-sparse quantization,” *arXiv preprint arXiv:2306.07629*, 2023.

[21] 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 *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[22] J. Lee, W. Lee, and J. Sim, “Tender: Accelerating large language models via tensor decomposition and runtime requantization,” *arXiv preprint arXiv:2406.12930*, 2024.

[23] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, “Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 318–30 332, 2022.

[24] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, “Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

[25] S. Merity, C. Xiong, J. Bradbury, and R. Socher, “Pointer sentinel mixture models,” *arXiv preprint arXiv:1609.07843*, 2016.

[26] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, “Exploring the limits of transfer learning with a unified text-to-text transformer,” *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[27] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys®*. Springer Science & Business Media, 1997.

[28] C. H. Oliveira, M. T. Moreira, R. A. Guazzelli, and N. L. Calazans, “Ascend-freepdk45: An open source standard cell library for asynchronous design,” in *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2016, pp. 652–655.