AccLLM: Accelerating Long-Context LLM Inference Via Algorithm-Hardware Co-Design

Yanbiao Liang, Huihong Shi, Haikuo Shao, and Zhongfeng Wang, Fellow, IEEE

Abstract—Recently, large language models (LLMs) have achieved huge success in the natural language processing (NLP) field, driving a growing demand to extend their deployment from the cloud to edge devices. However, deploying LLMs on resource-constrained edge devices poses significant challenges, including (1) intensive computations and huge model sizes, (2) great memory and bandwidth demands introduced by the autoregressive generation process, and (3) limited scalability for handling long sequences. To address these challenges, we propose AccLLM, a comprehensive acceleration framework that enables efficient and fast long-context LLM inference through algorithm and hardware co-design. At the algorithmic level, we integrate (1) pruning, (2) Λ -shaped attention, and (3) an innovative W2A8KV4 (2-bit weights, 8-bit activations, and 4bit KV cache) quantization scheme, thus effectively reducing memory and bandwidth requirements while facilitating LLMs' long-sequence generation. At the hardware level, we design a dedicated FPGA-based accelerator with a reconfigurable computing engine to effectively and flexibly accommodate diverse operations arising from our compression algorithm, thereby fully translating the algorithmic innovations into tangible hardware efficiency. We validate AccLLM on the Xilinx Alveo U280 FPGA, demonstrating a $\uparrow 4.07 \times$ energy efficiency and a $\uparrow 2.98 \times$ throughput compared to the state-of-the-art work FlightLLM.

Index Terms—Large language models, quantization, pruning, compression, acceleration, algorithm-hardware co-design.

I. INTRODUCTION

Large language models [1]–[4] (LLMs) have revolutionized natural language processing (NLP) with their outstanding capabilities, enabling a wide range of applications [5], including code generation [6], document summarization [7], chatbots [2], and question answering [8]. This impressive potential has driven growing interest in extending LLMs' deploying beyond traditional cloud-based platforms to edge devices, such as smart vehicles, robots, and embedded systems [9]–[11]. However, mainstream works have merely focused on optimizing and accelerating LLMs on GPUs [12], [13] with powerful resource capacity, making them unsuitable for resource-constrained edge scenarios [14], [15].

To facilitate the widespread deployment of LLMs on edge devices, significant efforts [10], [16] have focused on developing hardware accelerators tailored for LLMs, with a particular

This work was supported by the National Key R&D Program of China under Grant 2022YFB4400600.

Yanbiao Liang and Huihong Shi contributed equally to this work. Yanbiao Liang, Huihong Shi, and Haikuo Shao are with the School of Electronic Science and Engineering, Nanjing University, Nanjing, China (e-mail: {ybliang, shihh, hkshao}@smail.nju.edu.cn).

Zhongfeng Wang is with the School of Electronic Science and Engineering, Nanjing University, and the School of Integrated Circuits, Sun Yat-sen University (email: zfwang@nju.edu.cn).

Correspondence should be addressed to Zhongfeng Wang.

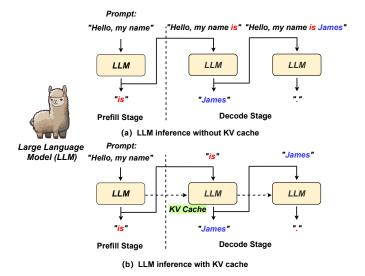


Fig. 1. The inference pipelines of LLMs (a) without KV cache and (b) with KV cache.

emphasis on FPGAs due to their efficiency and reconfigurability. However, these approaches still face substantial design challenges. First, LLMs impose substantial computations and extremely large model sizes. For example, a widely adopted LLM, Llama-2-7B [4], consists of 7 billion parameters, requiring approximately 14GB of memory in FP16 format and about 12 TFLOPS of computation to perform a single inference with 512 input tokens and 512 output tokens. To tackle these issues, model compression techniques such as pruning [17], [18] and quantization [19]–[21] have been proposed. For example, SpareGPT [17] uses pruning to enhance LLMs' efficiency at the architectural level by removing approximately 50% of unimportant components. In contrast, AWQ [21] proposes activation-aware weight quantization to quantize weights of LLMs to lower-bit integers, thus enhancing hardware efficiency at the operator level. Despite the promising results of these compression techniques, compressed LLMs remain too large for execution on edge devices with extremely constrained hardware resources, underscoring the need for more aggressive and effective model compression strategies.

<u>Second</u>, the decode stage of LLMs exhibits substantial memory overhead and bandwidth demands. As shown in Fig. 1 (a), LLMs operate through two distinct stages [22]: the prefill and decode stages, each characterized by unique operation types and computational patterns. Specifically, in the prefill stage, LLMs process all tokens in the input prompt in parallel to generate the first output token. This stage is dominated by matrix-matrix (MM) multiplications, making it

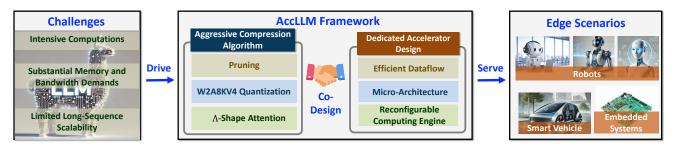


Fig. 2. Driven by the (1) intensive computations and large model sizes of LLMs, (2) substantial memory overhead and bandwidth demands introduced by the autoregressive generation process, and (3) limited scalability for handling long sequences, we propose a comprehensive LLM acceleration framework dubbed **AccLLM**, which incorporates (1) an aggressive compression algorithm and (2) a dedicated accelerator, thus facilitating extensive real-world applications of LLMs on edge devices.

highly computationally intensive [23]. In contrast, the decode stage predicts output tokens sequentially, using both the prefill context and previously generated tokens. The autoregressive process in the decode stage involves repeated computations for each newly generated token, introducing redundancy. To mitigate this, as depicted in Fig. 1 (b), caching mechanisms (i.e., Key-Value (KV) cache [24]) are proposed to store previous KV states and avoid redundant computations. However, this comes at the cost of increased memory overhead [25], highlighting the need for efficient cache compression methods, particularly given the limited memory resources available on edge devices. Moreover, the use of KV caches makes the decode stage primarily dependent on vector-matrix (VM) multiplications, rendering it memory-bound [26]. Unfortunately, existing works are mainly dedicated to accelerating the prefill stage and cannot be effectively adapted to support the decode stage of LLMs, resulting in under-utilization of computing engines due to substantial memory and bandwidth demands associated with the decode stage [27]-[29].

Third, LLMs are generally required to handle long sequences but face significant challenges in terms of both performance and memory requirements. For example, chatbots need to process and maintain coherence over extended interactions, such as day-long conversations [2]. However, extending LLMs beyond their pre-trained sequence lengths often results in performance degradation [30]. Additionally, the memory requirements for the KV cache during the autoregressive decode stage increase with sequence length [25], further exacerbating the challenge of handling long sequences efficiently. A promising solution to enhance LLMs' long-sequence scalability is Λshaped attention [25], [31], which combines global attention (targeting important initial tokens) with window attention (focusing on the most recent tokens). Therefore, effectively accelerating Λ -shaped attention and integrating it with other compression and optimization techniques is crucial for LLM acceleration but remains a less-explored area.

To tackle the challenges mentioned above and facilitate the deployment of LLMs on edge devices, we offer the following contributions:

- We propose AccLLM (as shown in Fig. 2), a comprehensive LLM acceleration framework that leverages algorithm-hardware co-design to enable efficient and fast LLM inference on FPGAs.
- At the algorithm level, we introduce an aggressive compression algorithm, which integrates (1) pruning, (2) an

- innovative W2A8KV4 quantization that quantizes LLMs' weights to 2-bit, activations to 8-bit, and KV cache to 4-bit, and (3) Λ-shaped attention, thus effectively enhancing computational efficiency, reducing memory and bandwidth requirements, while enabling long-sequence generation capabilities for LLMs.
- At the hardware level, we develop a dedicated FPGA accelerator featuring a reconfigurable computing engine to translate our algorithmic innovations into real hardware efficiency. Specifically, our accelerator is designed to accommodate: (1) both dense and sparse operations resulting from pruning, (2) diverse bit-widths (2/4/8-bit) introduced by our W2A8KV4 quantization, and (3) MM multiplications in the prefill stage and VM multiplications in the decode stage inherent in LLMs, as well as (4) the Λ-shaped attention integrated into our compression algorithm.
- We conduct extensive experiments and ablation studies to validate the effectiveness of our AccLLM framework.
 Particularly, compared to the SOTA work FlightLLM [16] on Xilinx Alveo U280 FPGA, we achieve an ↑4.07× energy efficiency with ↑2.98× throughput.

The remainder of this paper is organized as follows: Sec. III reviews related works and Sec. III introduces preliminaries; Then, Sec. IV and Sec. V elaborate the algorithm and dedicated accelerator in AccLLM, respectively; Furthermore, Sec. VI present extensive experiments and ablation studies, consistently validating AccLLM's effectiveness; Finally, Sec. VII summarizes this paper.

II. RELATED WORKS

A. Pruning for LLMs

Pruning boosts model compactness at the architectural level by removing redundant parameters, ranging from individual weights (unstructured pruning [32]–[34]) to entire channels or layers (structured pruning [35]–[37]). Although unstructured pruning can achieve significant compression ratios, the resulting irregular sparsity is not conducive to hardware implementation [38]. In contrast, structured pruning is more compatible with hardware acceleration but often results in model accuracy degradation and limited sparsity [39]. To balance model accuracy and hardware efficiency, N:M semi-structured pruning [17], [18], where N out of every M elements are pruned, is commonly adopted in prevalent LLMs [16], [40]. For example,

SparseGPT [17] effectively prunes GPT-family models [3] to achieve 2:4 and 4:8 sparsity in a one-shot manner without any retraining. Moreover, Wanda [18] leverages the product of weights and input activations to achieve 2:4 semi-structured pruning, demonstrating improved perplexity in certain cases, such as Llama-2-13B [4].

B. Quantization for LLMs

Quantization is a pivotal compression technique that converts floating-point values into discrete integers, thus enhancing LLMs' efficiency at the operator level. It is typically categorized into two approaches: quantization-aware training (OAT) and post-training quantization (PTQ). QAT [41]-[43] typically achieves higher quantization accuracy by finetuning the entire model using full training data, leading to substantial computational costs. In contrast, PTQ [20], [44] relies on only a small dataset for calibration, making it a more feasible solution for LLM quantization. For example, GPTQ [44] introduces a one-shot weight quantization method using approximate second-order information, enabling the fast quantization of weights within GPT/OPT models to 3 or 4-bit with negligible accuracy degradation. To facilitate both weight and activation quantization, SmoothQuant [20] employs equivalent mathematical transformations to shift the quantization complexity from activations to weights, allowing both to be quantized to 8-bit. More recently, QServe [19] integrates innovative progressive group quantization, smooth attention, and activation-aware channel reordering to achieve more aggressive quantization in a W4A8KV4 configuration (4-bit weights, 8-bit activations, and 4-bit KV cache).

While these approaches demonstrate promising results in compressing LLMs while preserving performance, the residual computational and memory demands remain impractical for deployment on resource-constrained edge devices. This highlights the need for more aggressive model compression methods that combine orthogonal techniques, such as quantization and pruning, to produce even more compact LLMs.

C. LLM Accelerators

The remarkable performance of LLMs has driven efforts [9], [10], [16], [45] to deploy them in edge scenarios. One approach to achieve this goal involves integrating multiple edge devices into a unified system to enhance computational capacity and enable fast LLM acceleration. For instance, DFX [45] combines multiple FPGAs into a single large-scale accelerator, enabling low-latency, high-throughput inference for GPT-2 [46]. Another approach is to compress LLMs first and then design specialized accelerators tailored for the compact models. For example, LlamaF [9] uses quantization to compress both activations and weights of TinyLlama [47] into 8-bit formats and accelerates the resulting quantized MV multiplications with a fully pipelined accelerator. Moreover, FlightLLM [16] integrates quantization and pruning to compress LLMs and develops a dedicated accelerator with two key innovations: (1) a configurable sparse DSP chain optimized for diverse sparsity patterns to enhance computational efficiency, and

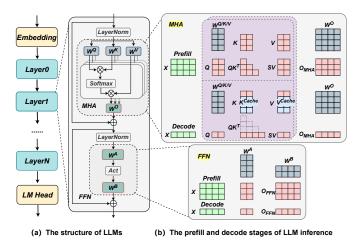


Fig. 3. (a) The structure of LLMs. (b) Illustrating the key computations during the prefill and decode stages of LLM inference.

(2) an always-on-chip decode scheme that reduces memory bandwidth requirements through low-precision quantization.

Despite the effectiveness of these accelerators, the substantial bandwidth demands of LLMs persistently limit achievable throughput, leading to under-utilized computing resources and underscoring the need for more aggressive compression algorithm. Additionally, these accelerators often lack adequate hardware support to address the scalability challenges associated with LLMs' long-sequence processing, impeding their deployment in real-world applications.

III. CHALLENGES AND MOTIVATIONS

In this section, we first outline the structure of LLMs and then explore three key challenges in LLM inference that motivate our proposed compression and acceleration framework.

A. Structure of LLMs

Fig. 3 (a) illustrates the architecture of LLMs, which consists of a sequence of Transformer decoder layers, each comprising a multi-head attention (MHA) module and a feed-forward network (FFN) module. In the **prefill stage**, the input prompt is first embedded into $X \in \mathbb{R}^{l \times d}$, where l represents the number of tokens and d denotes the embedding dimension. This embedded matrix then serves as the input to Transformer layers. As depicted in Fig. 3 (b) (top), within each layer, the MHA projects the input X into the query (Q_i) , key (K_i) , and value (V_i) matrices for the ith attention head. This is achieved through three linear transformations using their respective weight matrices W_i^Q , W_i^K , and W_i^V :

$$Q_i = X \cdot W_i^Q, \ K_i = X \cdot W_i^K, \ V_i = X \cdot W_i^V. \tag{1}$$

Next, matrix Q_i is first multiplied with K_i^{\top} to compute the attention score S_i , which is then multiplied with V_i to produce the attention output A_i for the i^{th} head. This process is expressed as follows, where d_k denotes the hidden dimension:

$$S_i = \operatorname{Softmax}(S_i') = \operatorname{Softmax}(\frac{Q_i K_i^{\top}}{\sqrt{d_k}}), \ A_i = S_i V_i.$$
 (2)

TABLE I
THE COMPUTATIONAL COMPLEXITY OF LINEAR OPERATIONS
ACROSS DIFFERENT STAGES IN LLMS

Linear Operation	Formula	Prefill Stage	Decode Stage	
Q/K/V/O _{MHA}	XW^Q, XW^K, XW^V, AW^O	$4ld^2$	$4d^2$	
Attention	QK^{\top}, SV	l(l+1)d	2(l+1)d	
FFN	$\sigma(XW^A)W^B$	$2ldd_{\mathrm{FFN}}$	$2dd_{\mathrm{FFN}}$	

Notes: l represents the input sequence length, d denotes the input feature dimension, and $d_{\rm FFN}$ is the FFN hidden dimension.

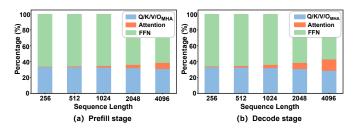


Fig. 4. The computational breakdown of Llama-2-7B [4] inference during (a) the prefill stage and (b) the decode stage across different sequence lengths.

Finally, the attention output from all h heads are concatenated and processed through the output linear projection layer W^O to produce the final result of the MHA module O_{MHA} :

$$O_{\text{MHA}} = \text{concat}(A_1, ..., A_d) \cdot W^O. \tag{3}$$

As shown in Fig. 3 (b) (down), in the **FFN** module, which consists of two linear layers $(W^A \text{ and } W^B)$ separated by a non-linear activation function $(\sigma(\cdot))$, the computation for a given input $X \in \mathbb{R}^{l \times d}$ can be expressed as:

$$O_{\text{FFN}} = \sigma(X \cdot W^A) \cdot W^B. \tag{4}$$

where $O_{\rm FFN}$ is output of the FFN module.

In summary, since the prefill stage processes the input prompt in parallel, the computations involved in both MHA (input and output linear projections in Eq. (1) and Eq. (3), respectively, and attention computations in Eq. (2)) as well as FFN (linear projections in Eq. (4)) are MM multiplications.

In contrast, during the **decode stage**, the use of the KV cache eliminates repeated computations, allowing only one input token to be processed at a time. As a result, the computations in both MHA and FFN (Eqs. (1)–(4)) are reduced to <u>VM</u> multiplications.

B. Challenges and Motivations

1) Dominant Linear Layers: As outlined in Sec. III-A, LLMs primarily consist of three types of operations: $Q/K/V/O_{\rm MHA}$ projections within MHAs defined in Eq. (1) and Eq. (3), attention computations in MHAs formulated in Eq. (2), and linear projections within FFNs described in Eq. (4). Their computational complexities are summarized in Table I. Using Llama-2-7B [4] as an example, we analyze the computational breakdown during the prefill and decode stages for sequence lengths ranging from 256 to 4096. As shown in Fig. 4, the $Q/K/V/O_{\rm MHA}$ and FFN linear layers collectively account for over 90% of the computation. This dominance persists across varying sequence lengths and processing stages, underscoring the critical need to optimize linear layers for efficient deployment of LLMs on resource-constrained edge devices.

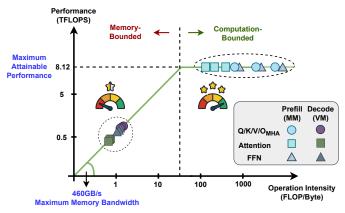


Fig. 5. Roofline analysis on the Xilinx Alveo U280 FPGA for the three primary types of linear operations during Llama-2-7B [4] inference.

- 2) Substantial Memory and Data Access Overhead Due to KV Cache in Attention Computations: As depicted in Fig. 4, while attention map computations account for only a small fraction (< 10%) of the total computations, they rely on the KV cache mechanism to store the K and V states of previous tokens, resulting in significant memory overhead, especially when handling long sequences. For example, the memory requirements of KV cache for a 7k-token context with Llama-2-7B could reach up to $3.5 \mathrm{GB}^1$, which causes severe burdens for on-chip data buffering as well as off-chip data accessing, highlighting the need for optimization in attention computations.
- 3) Memory-Bounded Decode Stage: In a single request, the prefill stage is executed only once, whereas the decoding stage is repeated for every output token in the response. Consequently, the decode stage often dominates the overall inference time, primarily due to the repeated loading of massive model parameters [36], [48]. To further investigate the computation characteristics during LLMs' inference, we conduct a roofline analysis of Llama-2-7B [4] on the Xilinx Alveo U280 FPGA platform. As illustrated in Fig. 5, we evaluate Q/K/V/O_{MHA}, Attention, and FFN linear operations across the prefill and decode stages for three typical sequence lengths (512, 1024, 2048), where the operation intensity \mathcal{I} [49] is defined as the ratio of arithmetic operations to memory accesses:

$$\mathcal{I} = \frac{\text{# of Operations}}{\text{# of Memory Accesses}}.$$
 (5)

High \mathcal{I} indicates greater opportunities for data reuse, making performance limited by computational resources rather than bandwidth. Conversely, low \mathcal{I} implies limited data reuse, leading to high bandwidth requirements and memory-bound performance [50]. As shown in Fig. 5, in the prefill stage, where multiple input tokens are processed in parallel, the resulting MM multiplications facilitate data reuse and achieve peak performance. In contrast, during the decode stage, the autoregressive generation process introduces intensive VM multiplications with limited data reuse. This leads to underutilization of computational resources and memory-bound per-

 $^{^1}$ Memory requirements of KV cache = 2 * num of layers * sequence length * num of heads * head dimensions * bit-width of FP16 = 2 * 32 * 7k * 32 * 128 * 16b = 3.5 GB

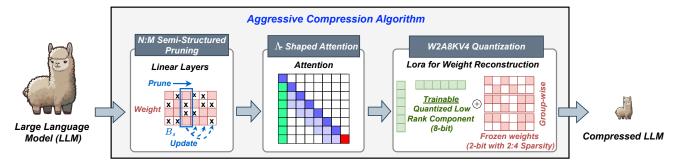


Fig. 6. The overview of our proposed aggressive compression algorithm, which includes (1) N:M semi-structured pruning to reduce the computational complexity of linear layers, (2) Λ -shaped attention to facilitate both KV cache and LLMs' long-sequence generation, and (3) an innovative W2A8KV4 quantization technique to boost overall throughput.

TABLE II
PERPLEXITY (\$\psi\$) OF LLAMA-2-7B [4] ON THE WIKITEXT-103 DATASET

[51] WITH VARYING SEQUENCE LENGTHS

Model	Pruning	χ Λ-Attention	Quantization	Model Size (GB)	3K	4K	5K	6K	7K
	Х	Х	Х	12.1	6.506	7.455	12.491	30.275	62.200
	1	X	X	6.60	13.775	16.309	27.966	65.122	116.967
Llama-2-7B	X	1	X	12.1	6.494	7.353	8.476	8.963	9.840
	X	×	1	1.66	5.830	6.374	11.807	32.477	88.048
	1	✓	X	6.60	13.903	16.140	18.785	20.284	22.643
	/	✓	1	1.53	8.038	8.524	9.316	9.512	9.869

formance, leading to an approximate 90% drop in performance compared to the peak. In summary, the latency of LLM inference is predominantly determined by the decode stage, where performance is primarily limited by bandwidth. Thus, minimizing bandwidth requirements during the decode stage is essential for accelerating LLM inference.

IV. AGGRESSIVE COMPRESSION ALGORITHM

In this section, we introduce an aggressive compression algorithm that is developed to effectively minimize computational and memory overhead while enhancing the longsequence generation performance of LLMs.

A. Overview

As illustrated in Fig. 6, our aggressive compression algorithm combines three key techniques: (1) 2:4 semi-structured pruning to reduce the *computational* complexity of costdominant *linear* layers, (2) Λ -shaped attention to simplify the *attention* mechanism, thereby reducing the KV cache storage burden and improving the scalability of LLMs for long-sequence generation, and (3) an innovative W2A8KV4 quantization technique that boosts the *throughput* of the memory-bounded decode stage while further reducing memory and data access overheads associated with the KV cache.

1) 2:4 Semi-Structured Pruning for Dominant Linear Layers: As discussed in Sec. III-B1, the linear layers dominate the computational workload across different sequence lengths and stages in LLM inference. To mitigate the substantial computational demands, we apply hardware-friendly 2:4 semi-structured pruning [17] to the weights of linear layers. As depicted in Fig. 6 (left), pruning is performed column-wise with a block size B_s . For 2:4 semi-structured pruning, we set the block size $B_s = 4$ and remove the 2 least significant weights from each block based on the importance metric S:

$$S_{ij} = \left[|W|^2 / \operatorname{diag}\left(H^{-1}\right) \right]_{ij}, H = X^{\mathsf{T}} X + \lambda I, \quad (6)$$

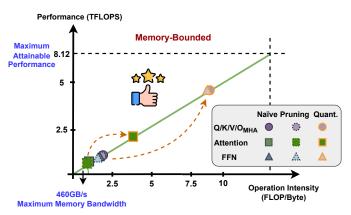
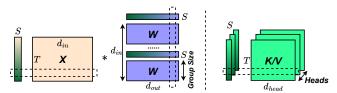


Fig. 7. Roofline analysis on the Xilinx Alveo U280 FPGA for the three primary types of linear operations during the decode stage of Llama-2-7B [4] inference with different compression methods.

where i and j represent the row and column indices of the weight matrix, respectively. X, W, and λ denote the inputs, weights, and the Hessian dampening factor, respectively.

After pruning each column, the weights in the remaining columns are updated to compensate for the pruning error based on the optimal brain surgeon (OBS) approach [52].

2) Λ -Shaped Attention for KV Cache: After resolving the dominant computational complexity of linear layers, we now focus on optimizing attention mechanisms. As outlined in Sec. III-B2, although attention computations constitute only a small portion of the total computations, the involved KV cache mechanism imposes substantial memory and data access overhead. To address this, we adopt Λ -shaped attention [25], which is inspired by the "attention sink" phenomenon, highlighting that retaining the KV of initial tokens with strong attention scores can largely recover the performance of windowed attention. As shown in Fig. 6 (middle), by combining these initial tokens with high attention scores with the most recent tokens (window attention), this approach effectively reduces the storage burden of the KV cache while enhancing the scalability of LLMs for long-sequence generation. For instance, when processing a 7K-token context in Llama-2-7B, Λ -shaped attention reduces the memory requirements of the KV cache from 3.5GB to 1G, achieving a 71.4% reduction. Moreover, this method enables Llama-2-7B [4] to handle longer sequences while maintaining promising perplexity results, as shown in Table II.



(a) Per-token quantization for activation + group-wise quantization for weight (b) Per-token quantization for K/V

Fig. 8. The W2A8KV4 quantization: (a) per-token quantization for activation + group-wise quantization for weight and (b) per-token quantization for K/V.

3) W2A8KV4 Quantization for Memory-Bound Decode Stage: As discussed in Sec. III-B1, the latency of LLM inference is predominantly constrained by bandwidth during the decode stage, due to the involved intensive memorybounded VM multiplications. While the 2:4 semi-structured pruning described in Sec. IV-A1 effectively reduces computational demands, it fails to alleviate bandwidth limitations or resolve the issue of low computation utilization, leading to limited hardware performance (TFLOPS). As demonstrated in Fig. 7, where we perform a roofline analysis for the three primary types of linear operations during the decode stage of Llama-2-7B [4] inference, performance (TFLOPS) remains largely unchanged even after pruning. To address this challenge, we provide an innovative W2A8KV4 quantization method that compresses LLM weights to 2-bit, activations to 8-bit, and KV cache to 4-bit. This approach offers two key benefits. First, as shown in Fig. 7, it significantly enhance the throughput of the memory-bound decode stage by reducing the precision/bit-width of operations and thus increasing computational intensity. Second, it further minimizes memory and data access overheads associated with the KV cache. A detailed explanation of this approach is provided in the following section.

B. W2A8KV4 Quantization

1) 2-bit Weights: To reduce the bandwidth requirements of dominant linear layers and enhance throughput during the memory-bounded decoding stage, we quantize the LLM weights to 2-bit.

First, to preserve performance in this low-bit configuration, we employ group-wise quantization [21], [53], [54]. As shown in Fig. 8 (a) (right), this method divides the weights W^i within the $i^{\rm th}$ output channel into multiple groups, with all weights $W^{i,j}$ in the $j^{\rm th}$ group sharing a scaling factor $S^{i,j}$ and zero point $Z^{i,j}$:

$$W_O^{i,j} = \text{clip}(\lfloor W^{i,j}/S^{i,j} \rceil + Z^{i,j}, 0, 2^b - 1), \text{ where}$$
 (7)

$$S^{i,j} = \frac{W_{\text{max}}^{i,j} - W_{\text{min}}^{i,j}}{2^b - 1}, \ Z^{i,j} = \text{clip}\left(\left[-\frac{W_{\text{min}}^{i,j}}{S^{i,j}}\right], 0, 2^b - 1\right).$$
(8)

Here, W_Q represents the quantized weights, and b is the quantization precision, set to b=2 in this case. To further enhance quantization performance, we integrate the learning weight clipping (LWC) method [53]–[55] into our group-wise quantization approach. It introduces two learnable parameters, λ and η , to facilitate quantization as follows:

$$X_{\text{max}} = \sigma(\lambda) \max(X), \ X_{\text{min}} = \sigma(\eta) \min(X),$$
 (9)

TABLE III

PERPLEXITY (\downarrow) OF LLAMA-2-7B [4] ON THE WIKITEXT-103 DATASET [51] WITH A SEQUENCE LENGTH OF 3K

Model	Pruning	W2	PEFT	WikiText-2
	Х	Х	Х	6.506
I lama 2 7D [4]	1	X	X	13.775
Llama-2-7B [4]	1	1	X	16.695
	✓	✓	✓	7.408

where σ represents the sigmoid function.

Despite these efforts, achieving 2-bit weight quantization in LLMs remains a significant challenge [21], [44], and this difficulty is further exacerbated when combined with 2:4 semi-structured pruning, as demonstrated in the third row of Table III. To mitigate the resulting performance degradation, inspired by the success of parameter-efficient fine-tuning (PEFT) [54], [56], [57], we further adopt LoRA fine-tuning to facilitate our pruning and quantization.

Specifically, we introduce a small set of learnable low-rank weights $A \in \mathbb{R}^{d_1 \times r}$ and $B \in \mathbb{R}^{d_2 \times r}$ $(r \leq d_2, d_1)$ on top of the pruned and quantized weight matrix W_Q to approximate the original weights $W \in \mathbb{R}^{d_1 \times d_2}$:

$$W \approx \widetilde{W}_Q + AB^T. \tag{10}$$

To stabilize fine-tuning, we initialize matrices \widetilde{W}_Q , A, and B following [54]:

$$\underset{\lambda,\eta,A,B}{\operatorname{argmin}} \| \mathcal{F}(X,W) - \mathcal{F}(X,\widetilde{W}_Q,A,B) \|, \tag{11}$$

where X represents the input of block \mathcal{F} . This initialization ensures that the outputs of the pruned and quantized blocks closely align with those of the original blocks at the start of fine-tuning. During the subsequent fine-tuning process, \widetilde{W}_Q is kept frozen in its 2-bit representation to minimize fine-tuning cost, while A and B remain trainable to mitigate the performance degradation caused by pruning and quantization.

It is worth noting that previous works [54], [57] typically retain the low-rank component A and B in floating-point precision after fine-tuning. Although their size is relatively small, this approach necessitates the use of floating-point computing units. To eliminate this overhead, we quantize the well-trained A and B to 8-bit before deployment, thus enabling their efficient processing in integers.

Consequently, the original weights W are approximated using \widetilde{W}_Q and quantized low-rank components A_Q and B_Q . The output of linear layers with input X can be expressed as:

$$XW \approx X(\widetilde{W}_Q + A_Q B_Q^T) = X\widetilde{W}_Q + X A_Q B_Q^T, \quad (12)$$

This method has two primary advantages: (1) As shown in Table III, this PEFT approach effectively mitigates the performance degradation caused by both quantization and pruning, successfully maintaining model performance even under aggressive compression settings; (2) This promising performance comes at the cost of only a slight memory overhead. For example, in the case of Llama-2-7B, LoRA weights account for only 3% of the original model's weights.

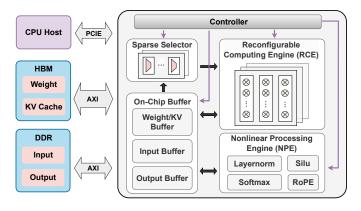


Fig. 9. The micro-architecture of our accelerator that integrates off-chip HBM/DDR interfaces, on-chip buffers, a sparse selector for sparsity support, a Reconfigurable Computing Engine (RCE) handling MM/VM multiplications, and a Nonlinear Processing Engine (NPE) for nonlinear operations.

2) 8-bit Activations and 4-bit KV Cache: To enable integer computations, we further employ 8-bit per-token quantization to LLM activations. As depicted in Fig. 8 (a) (left), it assigns each activation token an individual scaling factor to enhance performance. Notably, to further reduce memory requirements of KV cache beyond the Λ -shaped attention introduced in Sec. IV-A2, we apply 4-bit per-token quantization for keys and values. Specifically, for each token, K/V elements of each head share a common scaling factor, as illustrated in Fig. 8 (b).

V. FPGA-BASED RECONFIGURABLE ACCELERATOR

In this section, we first introduce the overall hardware architecture in Sec. V-A, followed by a detailed explanation of the reconfigurable computing engine in Sec. V-B. Finally, we illustrate the optimized dataflow in Sec. V-C, which facilitates both inter- and intra-layer pipelines.

A. Micro-Architecture

Our compressed LLMs involve four key computation types: (1) MM multiplications during the prefill stage and MV multiplications in the decode stage; (2) dense and sparse workloads from our 2:4 semi-structured pruning; (3) mixed-precision multiplications introduced by our W2A8KV4 quantization; and (4) Λ -shaped attention for efficient KV cache. To fully translate the algorithmic benefits into tangible hardware efficiency gains, we develop an FPGA-based reconfigurable accelerator, which incorporates: (1) a Reconfigurable Computing Engine (RCE) to effectively support both MM and MV processing; (2) a sparse selector to bypass zero elements and accelerate sparse workloads; (3) a flexible DSP packing strategy for mixed-precision support; and (4) an optimized dataflow to handle Λ -shaped attention.

As shown in Fig. 9, our accelerator also includes: (1) a controller to manage global control signals, (2) on-chip buffer and external memory to store data (inputs, weights/KV values, and outputs), and (3) a Nonlinear Processing Engine (NPE) to execute nonlinear operations (e.g. softmax, activation function, and layernorm). Specifically, the external memory comprises High Bandwidth Memory (HBM) to store large single-access data, such as weights and KV cache, and DDR

to store small single-access data, including input and output activations. Since nonlinear operations account for only a small part of total computations, the NPE processes them directly in floating-point to maintain accuracy.

Next, we will detail our RCE and its integration with the sparse selector and flexible DSP packing strategy to support the diverse computation workloads in compressed LLMs.

B. Reconfigurable Computing Engine (RCE)

As shown in Fig. 10, the RCE consists of T processing tiles, each comprising M PE blocks, and can be configured to compute both MM and VM multiplications. Furthermore, each PE block in RCE includes R precision-scalable multipliers to support mixed-precision multiplications. The RCE further incorporates a sparse selector composed of multiplexers to process weights with 2.4 semi-structured pruning.

1) MM and VM Multiplications: As shown in Fig. 10 (b), the RCE can be configured to operate in two distinct modes (MM and VM modes) for efficient execution of MM and VM multiplications, respectively.

MM Mode: To fully leverage the significant data reuse opportunities inherent in MM multiplication, we adopt an *input-output-parallel* dataflow. As illustrated in Fig. 10 (b) (left), *multipliers* within each PE block perform computations along input channels, providing a parallelism of R in input dimension. This enables partial sums to be directly accumulated across cycles within each block, thereby enhancing output reuse. Simultaneously, *blocks* within the same processing tile operate in parallel to handle different output channels of the weight, with input broadcast across blocks. This configuration achieves a parallelism of M in output dimensions and improves input reuse. Additionally, *different processing tiles* process inputs from separate tokens simultaneously, with weights broadcast across tiles, facilitating weight reuse and achieving parallelism of T in the token dimension.

VM mode: As the number of input tokens is reduced to 1, the weight reuse across different tokens in VM multiplication is no longer feasible. To maximize the available input and output reuse opportunity, we design a *output-parallel* dataflow for VM. As illustrated in Fig. 10 (b) (right), *multipliers* within each PE block concurrently process weights in the same input channel, offering a parallelism of R in input dimension. This also enables partial sums to be accumulated across cycles and thus enhances output reuse. *All PE blocks* within all processing tiles simultaneously process weights from different output channels, enabling parallelism of $M \times T$ in output dimension and facilitating input reuse.

2) Sparse and Dense Patterns: To effectively handle sparse patterns and reduce redundant computations between inputs and pruned weights, inputs are first processed by the sparse selector before entering the RCE. As illustrated in Fig. 10 (c), the sparse selector identifies and selects relevant inputs based on the sparse indices of the pruned weights while discarding irrelevant ones, thus significantly enhancing computational efficiency. The sparse selector can be disabled when supporting dense patterns.

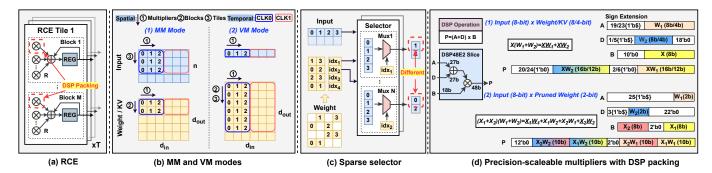


Fig. 10. (a) RCE architectural overview. (b) Reconfigurable operation modes (MM and MV modes) to support both MM and VM multiplications. (c) The incorporation of the sparse selector for 2:4 sparsity. (d) The proposed flexible DSP packing strategy to support mixed-precision multiplications via precision-scalable multipliers.

3) Mixed-Precision Multiplications: The W2A8KV4 quantization scheme described in Sec. IV-B introduces three types of mixed-precision multiplications: (1) 8-bit activations \times 8-bit LoRA weights, (2) 8-bit query $Q \times 4$ -bit key K and 8-bit attention score $S \times 4$ -bit key V in Eq. (2), and (3) 8-bit activations \times 2-bit weights. Meanwhile, as illustrated in Fig. 10 (b), the PE blocks within the same tile always operate in parallel to handle different output channels, allowing for effective input data reuse across blocks. To efficiently execute these mixed-precision computations while fully leveraging this input reuse, we propose an **flexible DSP packing strategy**. As shown in Fig. 10 (a), this strategy integrates two precision-scalable multipliers from adjacent blocks within the same processing tile into one DSP.

Specifically, as shown in Fig. 10 (d) (left), each DSP slice (DSP48E2) features a build-in 27×18 -bit multiplier, which performs the computation of $(A+D)\times B$, where A and D are 27-bit values, and B is a 18-bit value. To effectively handle (1) 8-bit activation \times 8-bit LoRA weights, as illustrated in Fig. 10 (d-1), we map two weights W_1 and W_2 to A and D, respectively, and activations X to B. This enables a single DSP to efficiently compute two multiplications via $X(W_1+W_2)=XW_1+XW_2$, thus greatly saving DSP consumption. Similarly, regarding (2)8-bit $Q/S\times 4$ -bit K/V, we treat the 4-bit K/V as W and the 8-bit Q/S as X, thus allowing for the packing of two multiplications using identical DSP data path routing.

For (3) 8-bit activations \times 2-bit weights, additional optimizations are required. Specifically, since 2-bit quantization for linear layer weights is always paired with pruning, the inputs are first processed by the sparse selector, which selects relevant inputs based on the pruned weights. However, as the pruned inputs typically vary between different weight rows (as indicated by the red dotted line in Fig. 10 (c)), the input reuse opportunities between adjacent PE blocks in RCE are eliminated. To overcome this limitation, we pack two inputs X_1 and X_2 into B, while mapping two weights W_1 and W_2 to A and D, respectively, as shown in Fig. 10 (d-2). This enables a single DSP to execute four multiplications by $(X_1+X_2)(W_1+W_2) = X_1W_1+X_1W_2+X_2W_1+X_2W_2$. The required results (i.e., X_1W_1 and X_2W_2) are then selectively extracted as the final output. As a result, the proposed flexible DSP packing strategy significantly enhances DSP utilization efficiency.

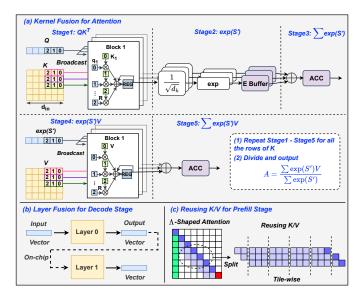


Fig. 11. Dataflow optimization with (a) kernel fusion for attention (in both prefill and decode stage), (b) layer fusion for decode stage, and (c) reusing K/V for prefill stage.

C. Dataflow Optimization

1) Kernel Fusion for Attention: The standard implementation of attention in Eq. (2) involves a sequential three-step computations - QK^T , Softmax(·), and SV - primarily due to the row-wise data dependency of the Softmax operation, which requires accumulating H datas in one row before performing the element-wise division:

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_{l=0}^{H} \exp(x_l)},$$
 (13)

where x_i represents an element within the row.

Since on-chip buffer is typically insufficient for storing all intermediate results in *prefill* stage, it leads to redundant off-chip data accesses. To address this issue and enhance performance, inspired by [13], [58], we fuse these computations into a single operation by reinterpreting Softmax operation:

$$A_{j,k} = \sum_{n=0}^{H} S_{j,n} V_{n,k} = \sum_{n=0}^{H} \frac{\exp(S'_{j,n})}{\sum_{l=0}^{H} \exp(S'_{j,l})} V_{n,k}$$
$$= \left(\frac{1}{\sum_{l=0}^{H} \exp(S'_{j,l})}\right) \left(\sum_{n=0}^{H} \exp(S'_{j,n}) V_{n,k}\right),$$
(14)

where $A_{j,k}$ is the element at j^{th} row and k^{th} column of the final attention output.

As shown in Fig. 11 (a), using a single row of Q as an example, the computations in Eq. (14) mainly consists of five stages. In Stage 1, QK^T is computed for several rows of K. The results are subsequently processed by the NPE in Stage 2 and 3 to obtain $\exp(S')$ and $\sum \exp(S')$, respectively. Further, the $\exp(S')$ will be multiplied by V in Stage 4 and accumulated to obtain $\sum \exp(S')V$ in Stage 5. These five stages are repeated until all rows of K are processed, generating the final $\sum \exp(S')V$ and $\sum \exp(S')$. Finally, the output is obtained by dividing $\sum \exp(S')V$ by $\sum \exp(S')$.

This rearrangement allows the multiplication of V to be executed before completing the division, thus enabling computation fusion and reducing data access costs. While the on-chip buffer is sufficient to store intermediate results (vector) during the decode stage, this kernel fusion technique also facilitates pipelining processing within attention computations. Therefore, we apply kernel fusion in both the prefill and decode stages to improve performance.

- 2) Layer Fusion for the Decode Stage: In the decode stage, the input and output activations are small vectors rather than large matrices, allowing them to be entirely stored within the on-chip buffer of the FPGA. To minimize off-chip memory access, we fuse computations of all layers in this stage by directly using the output of the current layer as the input to the subsequent layer, as illustrated in Fig. 11 (b).
- 3) Reusing K/V for the Prefill Stage: As illustrated in Fig. 11 (c), the Λ -shaped attention patterns between adjacent rows exhibit a one-token shift overlap, offering an opportunity to reuse K and V data during the prefill stage. As such, we vertically split the attention into several tiles based on the size of RCE in our accelerator and process them sequentially. Within each tile, multiple attention rows are computed in parallel while maintaining shared access to the KV data.

VI. EXPERIMENTS

A. Experimental Setup

1) Model, Datasets, Algorithm Setup, and Baselines: Model and Datasets: We evaluate our aggressive compression algorithm using the widely adopted LLM, Llama-2-7B [4], on the commonly used WikiText-103 and WikiText-2 [51] datasets and report their perplexity. Algorithm Setup: Our aggressive compression algorithm combines three key techniques: (1) 2:4 semi-structured pruning to reduce the computational complexity of cost-dominant linear layers, (2) Λ -shaped attention to simplify the attention mechanism, and (3) an innovative W2A8KV4 quantization technique that reduces memory and data access overheads. As for (1) 2:4 semi-structured pruning, we follow [17] to use 128 randomly sampled 2048-token segments from the first shard of the C4 dataset [59] as the calibration data. For (2) Λ -shaped attention mechanism, we set the KV cache size to 2048, consisting of 4 initial tokens and 2044 most recent tokens [25]. Regarding (3) W2A8KV4 quantization, we set the group size to 64 for groupwise quantization for weight. For the quantization initialization process in Eq. (11), we follow [54] and randomly sample 128

TABLE IV

KESOUKCI		ION OF OUR I	DEDICATED A	
Resources	BRAM	DSP	LUT	FF
Available	2016	9024	1304K	2607K
Used	513 (25.4%)	4497 (49.8%)	$420 {\rm K} (32.2\%)$	274K(10.5%)

TABLE V
PERFORMANCE OF LLAMA-2-7B [4] ON THE WIKITEXT-103 DATASET
[51] WITH VARYING SEQUENCE LENGTHS UNDER DIFFERENT
COMPRESSION ALGORITHMS

Method	Algorithm	Model Size		Pe	rplexity	y (\lambda)	
	Aigoritiiii	(GB)	3k	4k	5k	6k	7k
FP16	-	12.1	6.506	7.455	12.491	30.275	62.200
W8A8	SmoothQuant	6.03	6.778	7.743	13.090	32.478	66.430
W4A8KV4	QoQ	3.08	7.142	8.186	13.707	33.729	67.240
2:4 Pruning	SparseGPT	6.60	13.775	16.309	27.966	65.122	116.967
W2A8KV4 + 2:4 Pruning + Λ-Shaped Attention	Ours	1.53	8.038	8.524	9.316	9.512	9.869

TABLE VI
PERFORMANCE OF LLAMA-2-7B [4] ON THE WIKITEXT-2 DATASET [51]
WITH VARYING SEQUENCE LENGTHS UNDER DIFFERENT COMPRESSION
ALGORITHMS

Method	Algorithm	Model Size		Po	erplexity	y (↓)	
Method	Aigoriumi	(GB)	3k	4k	5k	6k	7k
FP16	-	12.1	18.49	7 20.608	30.619	63.461	114.484
W8A8	SmoothQuant	6.03	18.96	721.246	31.892	67.059	120.419
W4A8KV4	QoQ	3.08	41.220)44.845	62.171	113.396	180.845
2:4 Pruning	SparseGPT	6.60	54.510	667.892	102.321	194.244	317.622
W2A8KV4 + 2:4 Pruning + Λ-Shaped Attention	Ours	1.53	10.992	211.857	12.101	12.502	12.669

sentences from the training set of WikiText-103 and WikiText-2 [51], which serve as calibration datasets. Subsequently, we perform dataset-specific LoRA fine-tuning on their respective datasets. *Baselines*: We compare our compressed algorithm with four counterparts: (1) the half-precision (FP16) baseline, (2) the widely used LLM quantization work, SmoothQuant [20], (3) the SOTA W4A8KV4 LLM quantization framework, QoQ [19], and (4) the widely adopted LLM pruning method, SparseGPT [17], in terms of perplexity on varying sequence lengths and model size after compression.

2) Accelerator Setup and Baselines: Hardware Setup: The parallelism of the reconfigurable computing engine in our accelerator $(R \times M) \times T$ (see Fig. 10 (a)) is configured as $(32 \times 16) \times 16$. Our dedicated accelerator, AccLLM, is coded with Verilog and synthesized with the Vivado Design Suite. We evaluate its performance on the Xilinx Alveo U280 FPGA at a clock frequency of 225MHz. Table IV summarizes the U280 resource consumption of AccLLM. Additionally, we follow [60], [61] to develop a cycle-accurate simulator for our accelerator to provide fast and reliable performance estimations, which are validated against the RTL implementation to ensure correctness. Baselines: We compare AccLLM with (1) half-precision (FP16) Llama-2-7B [4] on NVIDIA A100 GPU, (2) mixed-quantized and sparse Llama-2-7B [4] on the SOTA FPGA accelerator, FlightLLM [16], and (3) W4A16 quantized and sparse ChatGLM2-6B [62] on its dedicated edge accelerator, EdgeLLM [10]. We compare with them in terms of throughput, power, and energy efficiency.

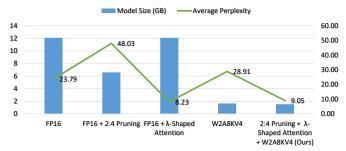


Fig. 12. The ablation studies on model size and average perplexity across 3k-7k lengths of Llama-2-7B [4] on the WikiText-103 dataset [51].

B. Advancement of Our Aggressive Compression Algorithm

Tables V and VI show the performance of Llama-2-7B on WikiText-103 and WikiText-2 datasets [51] under different LLM compression methods. We can draw the following conclusions: (1) Superior compression efficiency: Our aggressive compression algorithm achieves the smallest compressed model size (1.53 GB), which is only 12.6% of the FP16 baseline. When compared with SOTA quantization and pruning baselines, our approach can reduce the model size by $50.3\% \sim 76.8\%$. This remarkable efficiency is attributed to the progressive combination of the innovative W2A8KV4 quantization and 2:4 semi-structured pruning. (2) Better compression-accuracy trade-offs on short sequences: On the WikiText-103 dataset, our method achieves better tradeoffs between compression ratio and model performance when compared with SmoothQuant [20] and QoQ [19]. Specifically, we reduce the model size by 74.6% and 50.3% compared to SmoothQuant and QoQ, while incurring only a negligible increase in perplexity of 1.26 and 0.896, respectively, under the 3k sequence length. Furthermore, compared to the pruning baseline SparseGPT [17], our method not only achieves a 76.8% reduction in model size but also offers lower perplexity ($\downarrow 5.737$) under the 3k sequence length. On the WikiText-2 dataset, our approach consistently achieves the lowest model size and perplexity among all baselines, which further validates the effectiveness of our algorithm optimizations. (3) Exceptional performance on long sequences: On the WikiText-103 dataset, while our method incurs a slight perplexity increase over FP16 and quantization baselines for short sequences ($\leq 4k$ tokens), it surpasses all baselines on long sequences (\geq 5k tokens), highlighting the effectiveness of our adopted Λ -shaped attention technique. The benefits are even more pronounced on the WikiText-2 dataset, where our approach achieves a perplexity reduction of up to $101.815 \sim 304.953$ under the 7K token sequence, outperforming all baselines.

Algorithm Ablation Studies: We further validate the effectiveness of each component in our proposed aggressive compression algorithm by evaluating its impact on model size and average perplexity across sequence lengths ranging from 3k-7k. As illustrated in Fig. 12, we observe the following: (1) The 2:4 semi-structured pruning eliminates redundant parameters, shrinking the model size by 45.5% compared to the FP16 baseline, but this comes at the cost of a 24.24 increase in average perplexity. (2) Although not directly reducing the model size, Λ -shaped attention significantly enhances the

TABLE VII
COMPARISONS WITH SOTA TRANSFORMER ACCELERATORS

Accelerator	GPU	EdgeLLM [10]	FlightLLM [16]		Ours
Device	NVIDIA A100 GPU	Xilinx VCU128	Xilinx Versal VHK158	Xilinx Alveo U280	Xilinx Alveo U280
Frequency (MHz)	1410	125	225	225	225
Model	Llama-2-7B [4]	ChatGLM2-6B [62]	L	lama-2-7B [4]
DSP Used	-	5587	-	6345	4497
Throughput (Token/s)	45	75	92.5	55	164
Power (W)	220	50.77	155	45	33
Energy Efficiency (Token/J)	0.2	1.47	0.6	1.22	4.96

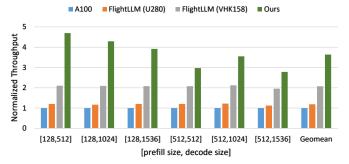


Fig. 13. Normalized throughput of AccLLM, FlightLLM, and A100 GPU. model's ability to handle long sequences, leading to a **15.56** decrease in average perplexity compared to the FP16 version. (3) The innovative W2A8KV4 quantization with LoRA fine-tuning achieves an impressive **86.3**% compression while maintaining comparable perplexity to the FP16 baseline. (4) The combination of these three techniques yields an **87.4**% overall reduction in model size while preserving performance.

C. Performance Evaluation of AccLLM Accelerator

The performance metrics of deploying LLMs on different hardware platforms are presented in Table VII. We can see that: (1) Compared to FP16 Llama-2-7B [4] executed on an A100 GPU, our algorithm and hardware co-optimized AccLLM achieves $\uparrow 3.64 \times$ throughput and $\uparrow 24.8 \times$ energy efficiency. (2) When compared with W4A16 quantized and sparse ChatGLM2-6B [62] on its dedicated edge accelerator, EdgeLLM [10], we offer $\uparrow 2.18 \times$ throughput and $\uparrow 3.37 \times$ energy efficiency. (3) Compared to the most competitive baseline, FlightLLM, which deploys mixed-quantized and sparse Llama-2-7B [4] on two FPGA platforms [16] (Xilinx Versal VHK158 and Alveo U280), we provide $\uparrow 1.77 \times$ and $\uparrow 2.98 \times$ throughput, along with $\uparrow 8.27 \times$ and $\uparrow 4.07 \times$ energy efficiency, respectively. Our performance advantages over FlightLLM stem from two key innovations: (1) the W2A8KV4 quantization scheme that alleviates the bandwidth bottleneck during the decode stage and (2) the flexible DSP packing strategy that maximizes DSP utilization, together leading to significant improvements to both throughput and energy efficiency.

We further compare the normalized throughput with the A100 GPU and FlightLLM when processing different input prefill and output decode token sizes. As shown in Fig. 13, we observe that: (1) Benefiting from both algorithmic and hardware optimizations, our method outperforms the A100 GPU and FlightLLM (U280 and VHK158) across various input and output token lengths, achieving a $1.77 \times \sim 3.64 \times 10^{-2}$

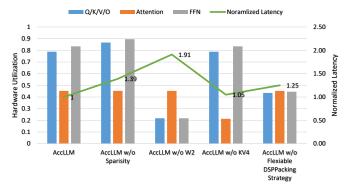


Fig. 14. The ablation studies on the end-to-end latency and hardware utilization of AccLLM evaluated on Llama-2-7B [4] containing Q/K/V/O, Attention, and FFN in decode stage.

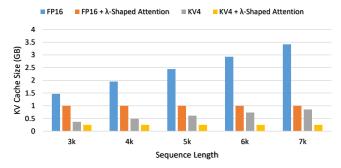


Fig. 15. The ablation studies on KV cache size of AccLLM.

improvement in geometric mean throughput. (2) Particularly, when the input and output token sizes are 128 and 512, our approach demonstrates much better performance. This performance gain stems from our accelerator's reconfigurability, which efficiently accommodates both prefill and decode stages.

Hardware Ablation Studies: We further conduct ablation studies on latency reduction and hardware utilization of different methods used in AccLLM. As shown in Fig. 14, we observe the following: (1) Although 2:4 semi-structured pruning provides limited improvements in hardware utilization, it effectively eliminates redundant parameters, thereby enhancing throughput and achieving $1.39 \times$ speedup. (2) The W2 quantization significantly reduces bandwidth requirements in linear layers and improves hardware utilization (approximately 4× compared to conventional W8 quantization), leading to $1.91 \times$ speedup. (3) The KV4 quantization alleviates bandwidth demands and enhances hardware utilization by $2\times$ during attention computation, resulting in a $1.05\times$ speedup, primarily since attention computations account for only a small fraction of total computations (as shown in Fig. 4). Despite the modest speedup, the KV4 reduces memory requirements of KV cache by 75% compared to the FP16 counterpart, which will be discussed in the following paragraph. (4) The flexible DSP packing strategy optimizes DSP utilization, achieving approximately 2× improvement in linear layers, contributing to a $1.28 \times$ overall speedup.

We also evaluate the memory footprint reductions of different attention optimizations related to the critical KV cache on varying sequence lengths. As demonstrated in Fig. 15: (1) The Λ -shaped attention effectively limits KV cache size to a fixed 1 GB (equivalent to 2+2044 selected tokens), regardless of the input sequence length. (2) KV4 quantization reduces

KV cache size by 75% compared to the FP16 baseline. (3) The combination of Λ -shaped attention and KV4 quantization achieves a remarkable reduction in KV cache size, limiting it to just 0.25 GB, highlighting the effectiveness of our approach in minimizing the memory footprint of KV cache.

VII. CONCLUSION

In this paper, we have proposed, developed, and validated AccLLM, a comprehensive algorithm-hardware codesign framework that enables efficient and fast inference for LLMs on the FPGA platform. Specifically, at the algorithmic level, we proposed an aggressive compression algorithm, which combines 2:4 semi-structured pruning, an innovative W2A8KV4 quantization scheme, and Λ -shaped attention, thus enhancing computational efficiency, reducing memory and bandwidth overhead, and enabling efficient long-sequence generation. At the hardware level, we design an FPGA-based dedicated accelerator that features a reconfigurable computing engine to fully unleash our algorithmic benefits and boost hardware efficiency. Extensive experimental results consistently demonstrate our effectiveness, achieving up to $\uparrow 4.07 \times$ energy efficiency and \$\frac{1}{2}.98\times\$ throughput compared to stateof-the-art LLM accelerators.

REFERENCES

- [1] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," *ArXiv*, vol. abs/2302.13971, 2023.
- [2] O. J. Achiam et al., "Gpt-4 technical report," 2023.
- [3] S. Zhang et al., "Opt: Open pre-trained transformer language models," ArXiv, vol. abs/2205.01068, 2022.
- [4] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *ArXiv*, vol. abs/2307.09288, 2023.
- [5] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Barnes, and A. S. Mian, "A comprehensive overview of large language models," *ArXiv*, vol. abs/2307.06435, 2023.
- [6] M. Chen et al., "Evaluating large language models trained on code," ArXiv, vol. abs/2107.03374, 2021.
- [7] T. Zhang, F. Ladhak, E. Durmus, P. Liang, K. McKeown, and T. Hashimoto, "Benchmarking large language models for news summarization," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 39–57, 2023.
- [8] E. Kamalloo, N. Dziri, C. L. A. Clarke, and D. Rafiei, "Evaluating open-domain question answering in the era of large language models," *ArXiv*, vol. abs/2305.06984, 2023.
- [9] H. Xu, Y. Li, and S. Ji, "Llamaf: An efficient llama2 architecture accelerator on embedded fpgas," ArXiv, vol. abs/2409.11424, 2024.
- [10] M. Huang, A. Shen, K. Li, H. Peng, B. Li, and H. Yu, "Edgellm: A highly efficient cpu-fpga heterogeneous edge accelerator for large language models," ArXiv, vol. abs/2407.21325, 2024.
- [11] O. Friha, M. A. Ferrag, B. Kantarci, B. Cakmak, A. Ozgun, and N. Ghoualmi-Zine, "Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 5799–5856, 2024.
- [12] R. Y. Aminabadi et al., "Deepspeed- inference: Enabling efficient inference of transformer models at unprecedented scale," SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15, 2022.
- [13] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. R'e, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *ArXiv*, vol. abs/2205.14135, 2022.
- [14] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genç, G. Dinh, Q. Huang, K. Keutzer, M. W. Mahoney, Y. S. Shao, and A. Gholami, "Full stack optimization of transformer inference: a survey," *ArXiv*, vol. abs/2302.14017, 2023.
- [15] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Lev-skaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," *ArXiv*, vol. abs/2211.05102, 2022.

- [16] S. Zeng et al., "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2024.
- [17] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," ArXiv, vol. abs/2301.00774, 2023.
- [18] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," *ArXiv*, vol. abs/2306.11695, 2023.
- [19] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han, "Qserve: W4a8kv4 quantization and system co-design for efficient llm serving," *ArXiv*, vol. abs/2405.04532, 2024.
- [20] G. Xiao, J. Lin, M. Seznec, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," *ArXiv*, vol. abs/2211.10438, 2022.
- [21] 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.
- [22] A. Agrawal, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, and R. Ramjee, "Sarathi: Efficient Ilm inference by piggybacking decodes with chunked prefills," *ArXiv*, vol. abs/2308.16369, 2023.
- [23] S. Zhao, D. Israel, G. V. den Broeck, and A. Grover, "Prepacking: A simple method for fast prefilling and increased throughput in large language models," *ArXiv*, vol. abs/2404.09529, 2024.
- [24] S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, and J. Gao, "Model tells you what to discard: Adaptive kv cache compression for llms," *ArXiv*, vol. abs/2310.01801, 2023.
- [25] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," ArXiv, vol. abs/2309.17453, 2023.
- [26] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.
- [27] Y. Qin, Y. Wang, D. Deng, Z. Zhao, X. Yang, L. Liu, S. Wei, Y. Hu, and S. Yin, "Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction," *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023.
- [28] T. J. Ham, Y. Lee, S. H. Seo, S.-U. Kim, H. Choi, S. Jung, and J. W. Lee, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), pp. 692–705, 2021.
- [29] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, 2021.
- [30] O. Press, N. A. Smith, and M. Lewis, "Train short, test long: Attention with linear biases enables input length extrapolation," ArXiv, vol. abs/2108.12409, 2021.
- [31] C. Han, Q. Wang, H. Peng, W. Xiong, Y. Chen, H. Ji, and S. Wang, "Lm-infinite: Zero-shot extreme length generalization for large language models," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2024, pp. 3991–4008.
- [32] H. Shao, B. Liu, and Y. Qian, "One-shot sensitivity-aware mixed sparsity pruning for large language models," ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 11 296–11 300, 2023.
- [33] A. Syed, P. H. Guo, and V. Sundarapandiyan, "Prune and tune: Improving efficient pruning techniques for massive language models," in *Tiny Papers @ ICLR*, 2023.
- [34] P. Xu, W. Shao, M. Chen, S. Tang, K.-C. Zhang, P. Gao, F. An, Y. Qiao, and P. Luo, "Besa: Pruning large language models with blockwise parameter-efficient sparsity allocation," ArXiv, vol. abs/2402.16880, 2024
- [35] X. Ma, G. Fang, and X. Wang, "Llm-pruner: On the structural pruning of large language models," ArXiv, vol. abs/2305.11627, 2023.
- [36] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Ré, and B. Chen, "Deja vu: Contextual sparsity for efficient Ilms at inference time," in *International Conference on Machine Learning*, 2023.
- [37] Y. Li, Y. Yu, Q. Zhang, C. Liang, P. He, W. Chen, and T. Zhao, "Losparse: Structured compression of large language models based on low-rank and sparse approximation," in *International Conference on Machine Learning*, 2023.

- [38] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *ArXiv*, vol. abs/1611.06440, 2016.
- [39] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," 2017 IEEE International Conference on Computer Vision (ICCV), pp. 1398–1406, 2017.
- [40] A. K. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," *ArXiv*, vol. abs/2104.08378, 2021.
- [41] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi, "Metaicl: Learning to learn in context," ArXiv, vol. abs/2110.15943, 2021.
- [42] J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," *ArXiv*, vol. abs/2109.01652, 2021.
- [43] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. Raffel, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," ArXiv, vol. abs/2205.05638, 2022.
- [44] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *ArXiv*, vol. abs/2210.17323, 2022.
- [45] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 616–630, 2022.
- [46] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.
- [47] P. Zhang, G. Zeng, T. Wang, and W. Lu, "Tinyllama: An open-source small language model," ArXiv, vol. abs/2401.02385, 2024.
- [48] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, "Quest: Query-aware sparsity for efficient long-context llm inference," ArXiv, vol. abs/2406.10774, 2024.
- [49] S.-C. Kao, S. Subramanian, G. Agrawal, A. Yazdanbakhsh, and T. Krishna, "Flat: An optimized dataflow for mitigating attention bottlenecks," Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, 2021.
- [50] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, apr 2009.
- [51] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," ArXiv, vol. abs/1609.07843, 2016.
- [52] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," *IEEE International Conference on Neural Networks*, pp. 293–299 vol.1, 1993.
- [53] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. J. Qiao, and P. Luo, "Omniquant: Omnidirectionally calibrated quantization for large language models," *ArXiv*, vol. abs/2308.13137, 2023.
- [54] B. Liao and C. Monz, "Apiq: Finetuning of 2-bit quantized large language model," in Conference on Empirical Methods in Natural Language Processing, 2024.
- [55] Z. Liu, K.-T. Cheng, D. Huang, E. P. Xing, and Z. Shen, "Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation," 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4932–4942, 2021.
- [56] J. E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *ArXiv*, vol. abs/2106.09685, 2021.
- [57] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," ArXiv, vol. abs/2305.14314, 2023.
- [58] Z. Bai, P. Dangi, H. Li, and T. Mitra, "Swat: Scalable and efficient window attention-based transformers acceleration on fpgas," *ArXiv*, vol. abs/2405.17025, 2024.
- [59] C. Raffel, N. M. 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," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2019.
- [60] J. Dass, S. Wu, H. Shi, C. Li, Z. Ye, Z. Wang, and Y. Lin, "Vitality: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention," 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 415–428, 2022.
- [61] H. Shi, H. Shao, W. Mao, and Z. Wang, "Trio-vit: Post-training quantization and acceleration for softmax-free efficient vision transformer," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, pp. 1296–1307, 2024.

[62] T. G. A. Zeng *et al.*, "Chatglm: A family of large language models from glm-130b to glm-4 all tools," *ArXiv*, vol. abs/2406.12793, 2024.