Fine-grained Structured Sparse Computing for FPGA-based AI Inference

Chen Zhang*, Shijie Cao‡, Guohao Dai*, Chenbo Geng*, Zhuliang Yao‡, Wencong Xiao‡, Yunxin Liu§, Ming Wu‡, Lintao Zhang‡, Guangyu Sun†, Zhigang Ji*, Runsheng Wang†, Ru Huang†
* Shanghai Jiao Tong University, † Peking University, ‡ Microsoft Research, § Tsinghua University,
chenzhang.sjtu@sjtu.edu.cn

Abstract—With the explosive growth in the number of parameters in deep neural networks (DNNs), sparsity-centric algorithm and hardware designs have become critical for low-latency AI serving systems. However, the inherent randomness in pruning methods often leads to fragmented data access and irregular computation patterns in sparse matrices, resulting in significantly reduced hardware efficiency. Addressing the balance between the 'randomness' required to maintain model accuracy and the 'regularity' needed for efficient hardware design is crucial for realizing effective sparse computing in AI. This paper proposes a Fine-grained Structured Sparsity (FSS) paradigm. The pruned sparse matrices in this paradigm exhibit characteristics of 'local randomness' and 'global regularity'. This dual-feature design allows AI accelerator hardware based on the FSS paradigm to maintain both high model accuracy and efficient hardware design. We implemented this novel accelerator on the Xilinx Alveo U280 and validated our concept across three different AI models, including CNN, RNN, and LLM, demonstrating performance that significantly outperforms prior methods.

Index Terms—FPGA Accelerator, Sparse Computing, Low-latency AI Inference, Convolutional Neural Networks, Recurrent Neural Networks, Large Language Models

I. Introduction

Artificial intelligence (AI) has recently revolutionized many fields of computer science, including computer vision and natural language processing, etc. [1]. Its tremendous success is primarily attributed to deep neural network models with extensive parameters and multiple layers, such as Convolutional Neural Networks (CNNs) [2], Recurrent Neural Networks (RNNs) [3], and Transformer-based large language models (LLMs) [4]. However, the significant parameter size and computational demands of deep neural network models place immense pressure on computing systems and processors. For example, GPT-3 [5] comprises numerous transformer blocks with 175 billion parameters, necessitating at least 16 NVIDIA A10 GPUs to operate under a batch size of 1. Sparsity-centric optimizations have emerged as a critical approach to reducing data volume and computational load [6]–[11].

Extensive research has demonstrated the pervasive presence of sparsity in various types of neural network models. Han et al. [12], [13] discovered that eliminating significant redundancy in CNN models—by pruning over 90% of the weight parameters—can maintain model accuracy with minimal degradation. Similar findings have been observed in different RNN models, where sparsity ranges from 50% to 97% [8], [14]. Additionally, several studies have explored sparsity in large language models (LLMs), revealing sparsity levels of at least 50% [11], [15]–[17].

While the existence of such sparsity suggests considerable potential for computational acceleration, where 80% sparsity could theoretically lead to a 5x speedup, the practical realization of these benefits is challenging. The irregular spatial distribution of non-zero data in sparse matrices results in highly irregular memory access patterns and computations, leading to suboptimal acceleration on conventional hardware. To address these challenges, numerous works have proposed specialized sparse accelerators [8], [18], [19]. These accelerators are designed with specialized hardware architectures to mitigate the difficulties posed by irregular computation patterns on general-purpose processors. Although these approaches have achieved some acceleration, they require complex circuit designs and substantial area overheads. Furthermore, they have not yet fully exploited the potential of sparsity.

On the other side, further works [20], [21] suggest using coarser-grained weight pruning methods to induce more structured sparsity patterns for better hardware acceleration. Coarse-grained pruning methods prune weights in the granularity of larger and more regular blocks. From the hardware perspective, blocks of non-zero weights can enable contiguous memory accesses and better utilize parallel computation resources. Unfortunately, it becomes challenging to maintain the same model accuracy when block sparsity is applied. Block sparsity constrains the locality of the non-zero weights, and important weights could be mistakenly pruned, resulting in model accuracy loss. Furthermore, the block size (i.e., pruning granularity) is application-sensitive, making it another hyper-parameter to tune. Existing work often needs to search a range of block sizes to find a trade-off between model accuracy and hardware efficiency [15], [20], [22], [23].

To address these issues, this work presents Fine-grained Structured Sparsity (FSS), a novel sparsity pattern for pruning deep neural networks. FSS divides each weight matrix row into regularly sized sub-matrices and applies fine-grained pruning to each sub-matrix independently to achieve consistent sparsity across all sub-matrices. FSS preserves the unstructured distribution of non-zero weights within each sub-matrix, thus maintaining higher model accuracy compared to block sparsity. Addition-

0000–0000/00$00.00 © 2021 IEEE

ally, the regularity of the sub-blocks simplifies hardware design, making it cost-effective. Experimental results in Section VI demonstrate that FSS achieves nearly the same model accuracy as unstructured sparsity and significantly outperforms block sparsity when pruning weights at the same sparsity level.

Furthermore, we demonstrate that FSS effectively addresses challenges in LLMs. Previous research [24], [25] has shown that, unlike CNNs and RNNs, the importance of weights in LLMs is often influenced by the values of activations, particularly those of large outliers. As a result, despite the weights in LLMs exhibiting a random distribution similar to CNNs and RNNs, the spatial distribution of important weights often clusters when considering the impact of activations. This clustering makes it difficult for conventional FSS methods to achieve optimal results. To address this issue, we propose a channel-wise re-distribution strategy, namely CR-FSS, which redistributes clustered important weights to align with the distribution characteristics that FSS excels at. CR-FSS designs adaptive re-distribution strategies to match the data distribution characteristics of different layers in LLMs.

Importantly, FSS is also well-suited for FPGA-based specialized hardware acceleration. We design an FPGA accelerator to leverage the benefits of FSS and eliminate the computational overheads associated with unstructured sparsity within FSS sub-matrix. Specifically: 1) our accelerator utilizes the intrinsic bank-balanced property of FSS to achieve high parallelism in SpMxV with guaranteed load balance; 2) it supports concurrent random access requests to vector elements without conflicts in SpMxV by adopting banked scratchpad memory to buffer vectors; 3) considering the unique data distribution characteristics of LLMs, we propose a channel re-distribution hardware unit that efficiently reorders data during transmission; and 4) to avoid the decoding overheads of FSS and CR-FSS, we introduce a novel sparse encoding format for FSS matrices that is decoding-free in our FPGA accelerator, called BBS, and a bitmap-based encoding for the channel re-distribution unit to adaptively reorder channels on the fly, called CRB. Notably, the FSS accelerator is highly efficient even for inference with a batch size of 1, by exploiting fine-grained parallelism from a single sample, which is challenging for unstructured sparsity.

Overall, this paper makes the following contributions:

- We propose Fine-grained Structured Sparsity, a novel sparsity pattern that can both maintain model accuracy and enable an efficient FPGA accelerator implementation. With FSSand CR-FSSextension, our work can efficiently and effectively leverage sparsity in a broad range of AI models, including CNN, RNN and LLM.
- We design a general FPGA-based accelerator for FSS that eliminates in-efficiencies caused by the irregular computation and memory access patterns, and achieves good efficiency for ubiquitous AI model inference at a batch size of 1.
- Implemented on Alveo U250 and U280 FPGA, the FSS accelerator achieves $1.5\times - 3.8\times$ speedup on ubiquitous AI models, including CNN, RNN, and LLM on high-end GPGPU with sparse tensor cores with negligible loss of model accuracy.

II. Background and Motivation

A. Sparsity in Deep Neural Networks

The redundancy in neural networks has been well recognized since the 1990s by Le-Cun et al. [26]. In recent years, fine-grained weight pruning approaches have removed over 90% of weight parameters in popular CV and NLP models [8], [13], [14], significantly reducing model size for deployment and inference services. Iterative pruning, first introduced by Han et al. [12], prunes individual weights below a monotonically increasing threshold value and then retrains the remaining weights iteratively. This method has been shown to retain model accuracy across a wide range of popular neural network models, including CNNs [19], [27], RNNs [28], [29], and LLMs [11], [15]–[17], [30].

Further research into LLMs [16], [24] has demonstrated that, although the distribution of weight values in LLMs is similar to that in typical AI models such as CNNs and RNNs, approximately 1% of activations in LLMs, known as outliers, have a significant impact on weight pruning results. These outliers have significantly larger values, often several times greater, or even one or two orders of magnitude higher, than other values in the matrix. Pruning weight values without considering these large-value activations can have a highly negative impact on model accuracy. Consequently, many works have proposed activation-aware pruning methods [17], [25], which achieve better results compared to direct pruning. Additionally, to efficiently utilize hardware's computation power, some studies [15], [31] have applied structured pruning to LLMs. However, direct pruning can still significantly reduce model accuracy.

B. Sparsity-centric Optimizations in AI Accelerator

Sparsity reduces computation and memory footprint, providing significant acceleration opportunities. However, the straightforward redundancy-oriented pruning introduces irregularity in models. To address these issues, researchers have proposed a series of solutions, which can be categorized into two main approaches.

The first approach involves hardware-friendly algorithms and software optimizations. The primary idea is to impose more regular sparsity patterns, making sparse matrices more amenable to hardware computation with features like continuous memory access, regular data structures, and organized computation patterns. These pruning patterns fully consider the model structure, such as filter and channel level sparsity for CNNs [32], [33], gate and cell state sparsity for RNNs [34], low-rank approximation [35], and block sparsity [10], [23]. As pointed out by Mao et al. [36] and Zhu [37], coarse-grained sparsity

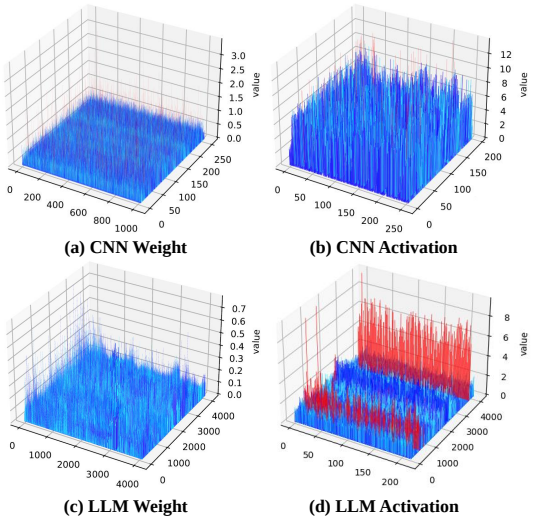


Fig. 1. Data Distribution of Two Representative AI Models, where (a)(b) stands for CNN (ResNet-56) and (c)(d) stands for LLM (OPT-6.7B).

benefits computation-intensive accelerators (e.g., GPUs) but results in a notable accuracy penalty compared to fine-grained approaches.

The second approach employs custom hardware to accelerate sparse matrix computation. These designs create specialized hardware structures to handle fragmented random data accesses and irregular computations on various hardware platforms, including ASICs [38]–[41], GPUs [42]–[45], and FPGAs [8], [9], [46], [47]. These specialized hardware solutions typically adopt a software and hardware co-design approach, enabling efficient parallelism and effectively skipping zeros. They also incorporate special logic in the data path to resolve irregular data movements. The key challenge lies in managing sparse computations' irregularities and imbalances with minimal circuit design complexity and hardware overhead. In our works presented at FPGA 2019 [6] and AAAI 2019 [7], we introduced a bank-balanced pruning method that facilitates a fine-grained structured sparsity pattern that balances the randomness required for algorithmic accuracy with the regularity needed for efficient hardware implementation. In 2020, NVIDIA's Ampere series GPUs introduced the Sparse Tensor Core [48] with a fixed 50% sparsity fine-grained structured computation method, known as N:M (2:4 or 4:8) sparse, which doubled the peak GPU performance.

C. Weight Distribution Analysis

Although our previous work demonstrated that FSS pruning achieves favorable sparsity outcomes in CNNs and RNNs, further experimentation has shown that the unique data distribution characteristics of LLMs make it difficult for a direct FSS pruning approach to maintain model accuracy. To efficiently support the three mainstream types of AI models—CNNs, RNNs, and LLMs—within a unified sparse computing framework, we conducted an in-depth analysis of the numerical distribution of LLM data, and yield two key observations.

Observation 1: For CNNs and RNNs, the important elements in their activation and weight matrices exhibit a random spatial distribution without outliers. Fig.1(a)(b) illustrates the three-dimensional visualization of the activation and weight matrices, where the height at each position represents the absolute value of that element. As shown in this figure, the distribution in both matrices is quite similar, with 100% of the weights falling within the range of $3 \times \sigma$, and no exceptionally large values. Additionally, we observe that the larger values (those in the top 50% by magnitude) are randomly distributed in space without any specific pattern. This distribution poses significant challenges for traditional coarse-grained structured sparsity methods, as they are more likely to mistakenly prune important weights, leading to reduced model accuracy, or they need to reduce the pruning ratio to maintain the desired model accuracy.

Observation 2: For LLMs, the spatial distribution of elements with large and small values in the weight matrix is quite similar to that observed in CNNs and RNNs. However, the distribution of activations differs significantly. Fig.1 shows the three-dimensional visualization of the activation and weight matrices, where the height at each position represents the absolute value of that element. We can observe that some elements have exceptionally high values, with some exceeding $10 \times \sigma$. Due to their large values, these important outliers contain crucial information. Therefore, even weights with relatively small values cannot be pruned if they are involved in computations with these outliers. We have observed a certain 'clustering effect' among the outliers: on one hand, the spatial locations of outliers tend to be consistent across different tokens, and we refer to a dimension with a majority of outliers as a channel. On the other hand, the spatial distribution of these outlier channels is random but often clustered together. This clustering of outlier channels presents challenges for traditional fine-grained structured pruning methods, as their design principle is to apply fine-grained pruning within sub-matrices locally. If important weights are concentrated in a local region, it becomes easier to inadvertently prune some crucial weights, thereby affecting model performance. Currently, there is no effective method to address this issue. Similar phenomenon has also been noted in previous studies [24], [49], [50], which exploit it to improve quantization; however, our focus is on addressing pruning challenges.

III. Framework Overview

Based on the analysis in Observation 1, we understand that preserving model accuracy after pruning requires retaining the randomly distributed important weights as much as possible. Therefore, the key to enabling AI hardware to fully benefit from the computational and storage advantages of sparsity lies in resolving the conflict between the "randomness" required by algorithms and the "regularity" needed for efficient hardware. This balance is

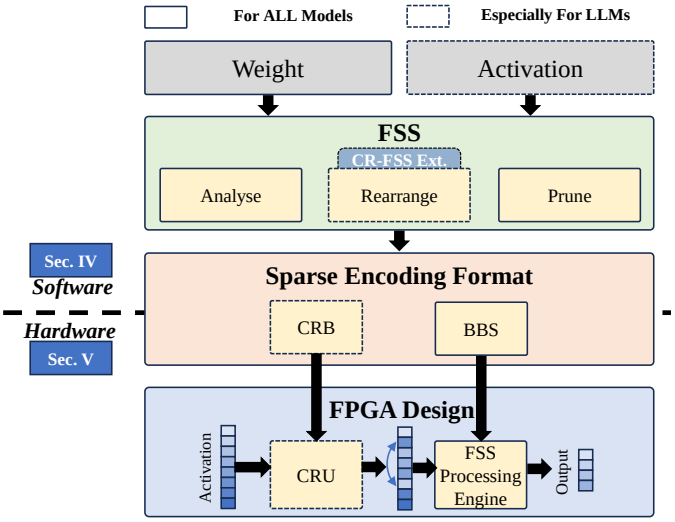


Fig. 2. Framework Overview

essential to achieve both high model accuracy and effective hardware acceleration. In this paper, we propose a new sparsity paradigm called Fine-grained Structured Sparsity (FSS). This approach retains the necessary algorithmic randomness locally within the weight matrix while maintaining global regularity. Our work demonstrates that this local randomness can effectively approximate the characteristics of original fine-grained pruning algorithms, while the global regularity can significantly reduce hardware design complexity, resulting in efficient acceleration at lower costs. To address the issues identified in Observation 2, we further propose the CR-FSS method, enabling large language models (LLMs) with specific sparse patterns to also benefit from the acceleration provided by FSS.

Building on Fine-grained Structured Sparsity, we developed a software-hardware co-designed sparse computing framework for AI inference. As illustrated in Fig.2, this framework comprises both software and hardware components. The software component automatically analyzes the importance of each value in the AI model's weight matrix, pruning unimportant weights while retaining crucial ones according to the Fine-grained Structured Sparsity paradigm, thereby generating a new FSS sparse matrix. For LLMs, our framework analyzes the activation and weight matrices, rearranging the elements in these matrices according to the CR-FSS method, an extension of FSS for LLMs. The compatible FSS and CR-FSS pruning schemes provide a unified sparse computing method that enables efficient acceleration of CNN, RNN, and LLM models using FPGA hardware accelerators.

In the hardware component, we first define a novel unified sparse encoding format suitable for FSS and CR-FSS hardware computations, called bank-balance sparse encoding (BBS). Based on BBS, we propose an efficient FPGA sparse computing accelerator that addresses the randomness in FSS with lower hardware complexity, maximizing parallelism and fully exploiting the potential of sparsity in AI models. To handle the dynamic characteristics of outliers in LLM models, we incorporate a Channel Re-distribution Unit (CRU) in the data input path, allowing the hardware to dynamically adapt to different outlier distributions across various LLM layers and models.

The remainder of this paper is organized as follows. Section IV introduces the Fine-grained Structured Sparsity paradigm and the pruning algorithm based on this paradigm, followed by a detailed analysis of its effects on different neural network models. Section V discusses the efficient sparse encoding format BBS and the design of its hardware accelerator.

IV. Fine-grained Structured Sparsity

A. Fine-grained Structured Sparsity (FSS)

For matrices represented in FSS, each matrix row is split into multiple equal-sized banks (i.e., sub-rows), and each bank has the same number of non-zero values. Fig.3 illustrates FSS with an example and compares it with unstructured sparsity and block sparsity. In this example, three sparse matrices with different sparsity patterns are all pruned from the dense example weight matrix in Fig.3(a) with a sparsity ratio of 50%. The straightforward pruning globally sorts the weights and prunes the smallest 50% of weights, leading to an unstructured sparse matrix (Fig.3(b)); Coarse-grained pruning induces a block sparse matrix (Fig.3(c)) by setting the block size to 2x2 and the block representative with the block average; Our bank-balanced pruning induces a bank-balanced sparse matrix (Fig.1(d)) by splitting each matrix row into 2 equal-sized banks and applying fine-grained pruning inside each bank independently.

We design this FSS sparsity pattern with consideration of both hardware efficiency and model accuracy. In general, partitioning weight matrix into multiple sub-matrices is mandatory for parallel computing. In FSS , each matrix row is split into multiple banks with the same size and same sparsity. This bank-balanced partitioning enables an efficient SpMxV design to exploit both inter-row parallelism and intra-row parallelism (i.e., inter-bank parallelism) with guaranteed load balance and no vector access conflicts. The detailed SpMxV design for FSS will be described in Section V-B. In addition, since FSS applies fine-grained pruning within each bank independently, the relatively large weights which contribute more to model accuracy in each bank can be preserved.

Another potential design for a sparsity pattern would be to split weight matrices into 2-D blocks like block sparsity and apply fine-grained pruning within each 2-D block. Larger weights within each block can be preserved as well in this scheme. However, after pruning, each 2-D block is still an unstructured sparse matrix. It is still challenging to design an efficient hardware accelerator architecture due to the irregularity of sparse sub-matrices. For example, parallelizing SpMxV across 2-D blocks leads to concurrent irregular vector accesses.

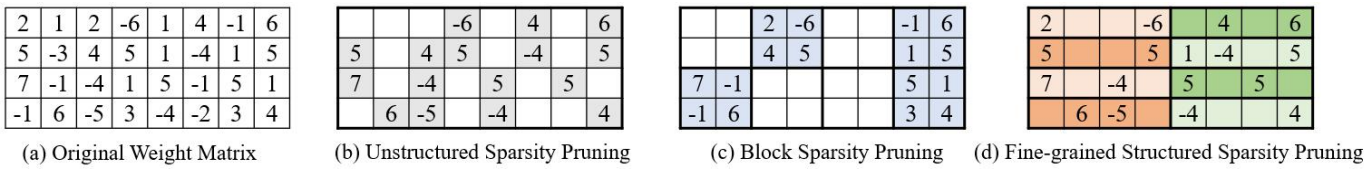


Fig. 3. Pruning (a) original weight matrix with (b) unstructured sparsity, (c) block sparsity, (d) fine-grained structured sparsity.

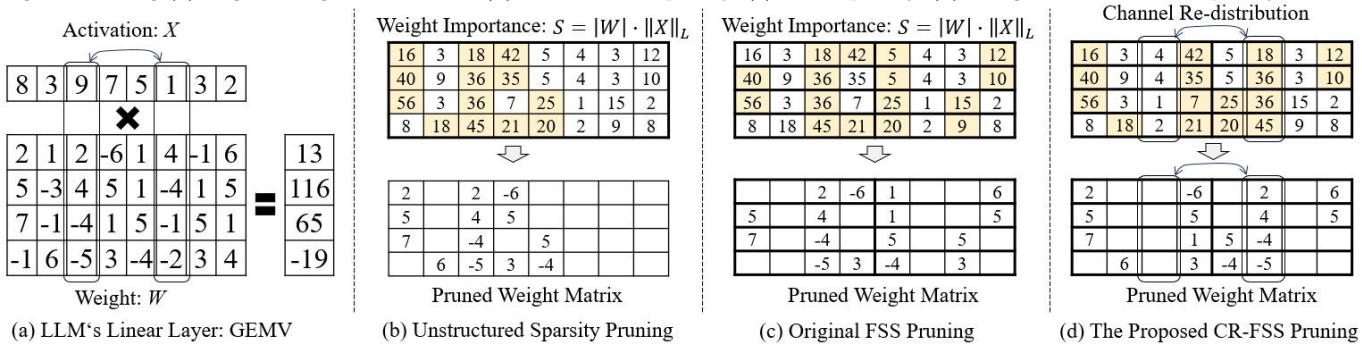


Fig. 4. Illustration of Channel Re-distributed Fine-grained Structured Sparsity

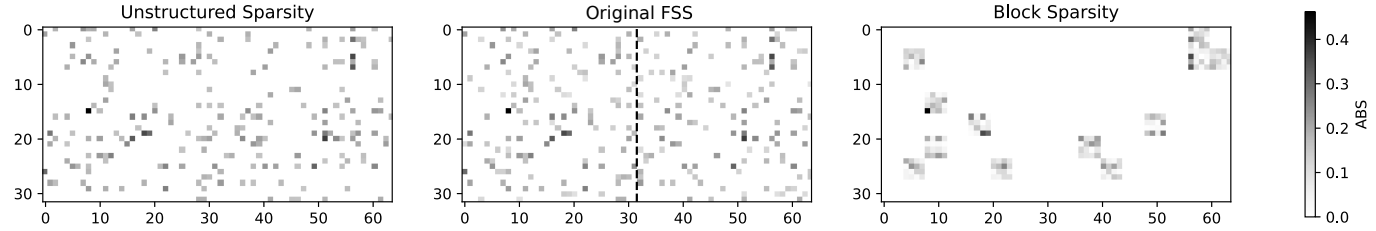


Fig. 5. Weight map visualization for RNN with (a) unstructured sparsity, (b) FSS, and (c) block sparsity (sparsity ratio = 90%).

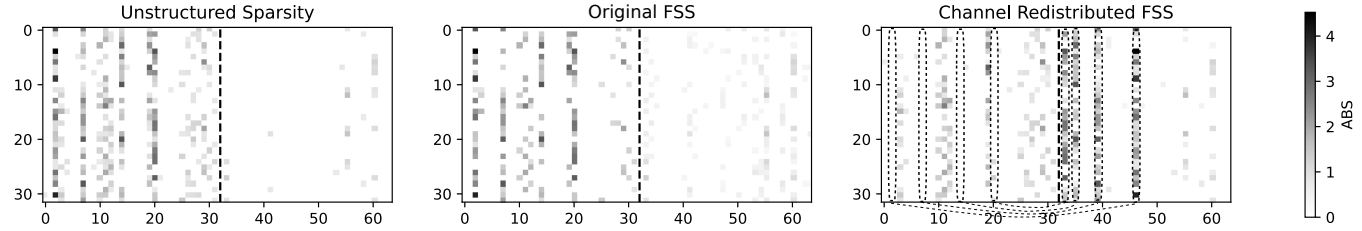


Fig. 6. Weight map visualization for LLM with (a) unstructured sparsity, (b) FSS, and (c) CR-FSS. (sparsity ratio = 50%)

B. Channel Re-distributed FSS (CR-FSS)

As revealed in Observation 2, the outlier values in activations have a significant impact on the importance of weights in large language models (LLMs). Despite the random distribution of outliers, they often exhibit clustering effects, meaning that outliers frequently appear in close positions. This leads to important weights also being located nearby. Therefore, although the distribution of weight values in LLMs is similar to that in CNNs and RNNs, directly applying Fine-grained Structured Sparsity (FSS) pruning may not achieve the desired minimal impact on model accuracy. Fig.4 illustrates a 32x64 region of the weight matrix from the OPT-6.7B model. Each pixel represents a weight element, where darker colors indicate more important weights and lighter colors denote less important ones. As shown in Fig.4(a), when applying unstructured sparsity pruning, as depicted in Fig.3(b), we observe a "channel clustering" effect. In this phenomenon, many important weights are concentrated in specific local regions (on the left), while other regions (on the right) contain relatively few important weights. If we apply the FSS method directly, as in Fig.3(d), it leads to the pruning of many critical weights (darker colors) in the left region, while less important weights (lighter colors) in the right region are unnecessarily preserved, as shown in Fig.4(b).

To address this issue, we propose a method called Channel Re-distribution FSS (CR-FSS). This method considers the influence of activations on weight importance during Fine-grained Structured Pruning. First, we obtain a weight importance matrix by calculating the product of the norms of weights and activations, as shown in Fig. 4(b). This matrix helps us examine the spatial distribution of important weights. If a significant number of important weights are clustered in a local region, direct FSS pruning may result in the removal of many important weights, adversely affecting model accuracy, as shown in

Fig. 4(c).

CR-FSS addresses this issue by redistributing the weights in the channel dimension, swapping channels with a high concentration of important weights with those having fewer (or no) important weights. This redistribution ensures a more balanced distribution of important weights across FSS sub-matrix. Pruning the weight matrix based on this redistributed structure avoids the excessive pruning of clustered important weights, thereby preserving as many important weights as possible. Fig. 4(d) demonstrates an example of channel redistribution. We sum the weight importance within each channel to obtain the channel importance and then reorder the channels based on their importance to achieve a uniform spatial distribution of important weights across the tensor. It is important to note that since GEMV computation ultimately sums up the results of vector multiplications, the channel redistribution only changes the order of computation without affecting the correctness of the final result, as illustrated in Fig. 4(a).

C. Weight Pruning Algorithm for FSS/CR-FSS

The FSS and CR-FSS methods can be effectively applied to various CNN, RNN, and LLM models, as well as different pruning algorithms. FSS requires the weight parameter matrix to be evenly partitioned into equally sized sub-matrices, which can take any regular shape such as vectors, squares, or rectangles. Within each sub-matrix, weights are pruned based on their importance, with weights of lower importance being removed. For CNNs and RNNs, weight importance is determined by the absolute value of the weights. For LLMs, weight importance is determined by the absolute value of the product of the weight and its corresponding activation. Other methods for defining weight importance are compatible with FSS and CR-FSS, offering researchers flexibility. However, a key requirement of FSS/CR-FSS is that all sub-matrices must maintain the same sparsity level, i.e., the number of non-zero elements must be consistent across all sub-matrices. Furthermore, FSS/CR-FSS can be integrated with various pruning algorithms, including iterative fine-tuning and post-training or one-shot pruning methods.

In this work, we apply the FSS pruning method iteratively to pre-trained CNNs and RNNs similar to previous work [8], and the one-shot pruning method to LLMs similar to previous work in [25]. Algorithm 1 illustrates the detailed pruning process for FSS and CR-FSS. Before applying the FSS pruning algorithm, this algorithm calculates the importance value of each channel and redistributes the weight matrix based on channel importance. The parameter for selecting the top H/L channels to swap is determined through experimental results and is set to a multiple (specifically 10× in our configuration) of the number of outliers. We iteratively increase the pruning percentage from 0% to the target sparsity, with the rate of increase with each pruning iteration. During the pruning process, if model accuracy drops significantly and cannot be recovered through fine-tuning, we revert the changes made in that iteration and terminate the pruning procedure.

Algorithm 1 Pruning Algorithm for FSS/CR-FSS

Input:
The matrix to be pruned, $M \in \mathbf{R}^{m \times n}$;
The calibration activation data, $X \in \mathbf{R}^{batchsize \times n}$ (Only For CR-FSS) ;
The number of banks per row, $BankNum$;
The expected sparsity, $Sparsity$;
Output:
The pruned matrix, M_p;

```
if FSS== True then
    Score = M (Score ∈ R^{m×n})
else if CR-FSS== True then
    for each N_i ∈ M.cols do
        Score_[:,i] = N_i × ||X_i||_2 (Score ∈ R^{m×n})
        S_i = sum(Score_[:,i])
    end for
    Pick Top H channel with highest S_i
    Pick Top L channel with lowest S_i
    for i ∈ H, j ∈ L do
        if i, j is not in the same sub-matrix then
            Swap channel i and j for M and Score
        end if
    end for
end if
for each M_i ∈ M.rows do
    Divide the row M_i into BankNum blocks;
    for each bank ∈ M_i do
        Sort the elements in bank according to elements' Score;
        Calculate the bank internal threshold T in line with Sparsity;
        for each element ∈ bank do
            prune element if element's Score < T;
        end for
    end for
end for
return the pruned matrix, M_p;
```

TABLE I
Percentages of the largest weights that are preserved in various sparsity patterns, sp = 90%(CNN/RNN), 50%(LLM)

sp=90%	Weight Matrix	Unstr. Sp.	Block Sparse	FSS
RNN	W_{ix}	100%	42.76%	91.30%
	W_{cx}	100%	24.24%	84.45%
CNN	Layer1	100%	32.4%	64.16%
	Layer2	100%	29.17%	70.355%
sp=50%	Weight Matrix	Unstr. Sp.	FSS	CR-FSS
LLM	Linear	100%	85.45%	88.15%
	FFN1	100%	86.54%	92.79%
	FFN2	100%	82.54%	91.75%

D. Analysis of Our Pruning Method

Intuitively, a pruning method should remove only smaller weights and preserve larger weights that con-

tribute more to model accuracy. However, structured pruning approaches often mistakenly prune important weights, leading to accuracy degradation. In this section, we analyze and demonstrate the efficiency and effectiveness of the proposed FSS and CR-FSS approaches using real cases.

To verify the pruning effectiveness of FSS/CR-FSS and compare them with unstructured sparsity and block sparsity, we analyze and visualize the weight matrices after applying the respective pruning methods on a real LSTM [51], CNN [52] and LLM [53]. In this analysis, the sparsity ratios are all set to 90%, the sub-matrix size for FSS is 32, and the block size for block sparsity is 4×4.

Figure 5 visualizes three types of pruning methods for a 32×64 sub-matrix randomly selected from the entire 1500×1500 W_{ix}. Grey grids indicate non-zero parameters, with the grey level representing the magnitude of the absolute value. For the second matrix represented by FSS, each row has two sub-matrices (left and right sides of the dashed line). Each bank has 3 non-zero weights. We can observe that the weight map of FSS is very similar to the weight map of unstructured sparsity, whereas the weight map of block sparsity differs significantly due to the locality constraint.

Figure 6 visualizes three types of pruning methods with a 32×64 sub-matrix selected from a FFN layer's weight matrix of size 4096×16384. By comparing Fig. 6(b) and (c), we observe that the CR-FSS method preserves some crucial channels in the weight map that would typically be pruned under the standard FSS method. This is due to the standard FSS's limitation where important weights, unfortunately located in a sub-area with more significant weights, are pruned. The CR-FSS method, however, adjusts its pruning pattern based on the contextual relevance of weights within the matrix, ensuring that essential connections are maintained for better model performance.

Table I shows the percentage of the largest weights preserved in various sparsity patterns. We present the results for W_{ix} and W_{cx}, with similar results observed for other weight matrices. Unstructured sparsity, achieved through fine-grained pruning, naturally preserves 100% of the largest weights by globally pruning weights with the smallest magnitudes. FSS preserves more than 80% of the largest weights by fine-grained pruning within each bank, while block sparsity preserves less than half (or even a quarter) of the largest weights. For LLM, CR-FSS further improves the ratio of the preserved important weights by 3% to 9%. Since prior researches have revealed that outliers, comprising only 0.1% to 1% of the total weights, have significantly impact on model accuracy [49], [50], the additional preservation of 3% to 9% of important weights by CR-FSS compared to FSS is crucial for enhancing model performance. Our experimental results further demonstrate that FSS outperforms block sparsity in CNN and RNN models, and CR-FSS surpasses FSS in LLM models in both achievable sparsity and accuracy, as detailed in Section VI.C and Fig 11.

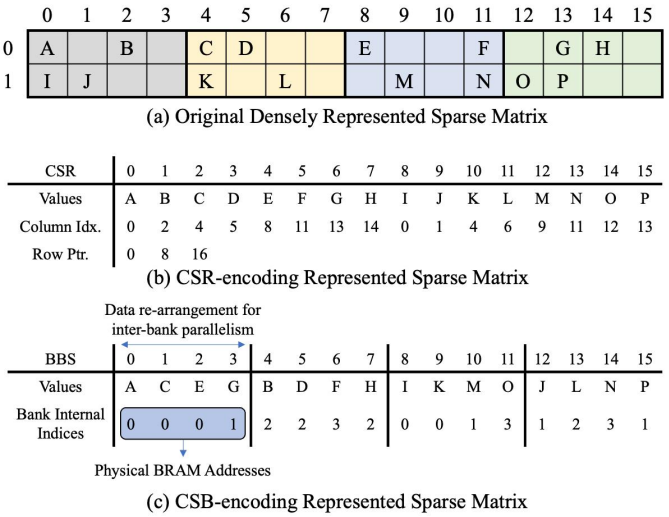


Fig. 7. Compressed Sparse Banks (CSB) Encoding Format

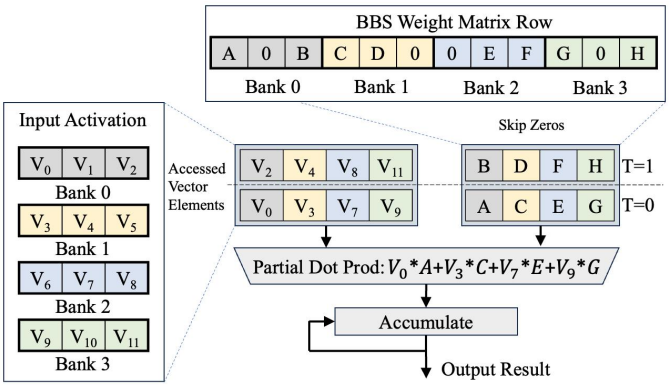


Fig. 8. The SpMxV Processing Engine for FSS and CR-FSS

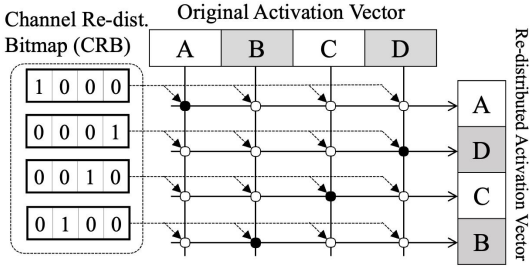


Fig. 9. Channel Re-dist. Bitmap (CRB) and Channel Re-dist. Unit (CRU)

V. Accelerator Design

A. Compressed Sparse Banks (CSB) Encoding Format

Various sparse matrix formats have been proposed to reduce the memory footprint of sparse matrices. However, existing formats introduce decoding overheads when performing sparse matrix multiplications. For FPGA implementation, decoding sparse formats consumes hardware resources and incurs latency. In order to liminate decoding overheads, we introduce a sparse matrix format called Compressed Sparse Banks (CSB) that is specifically designed for FSS.

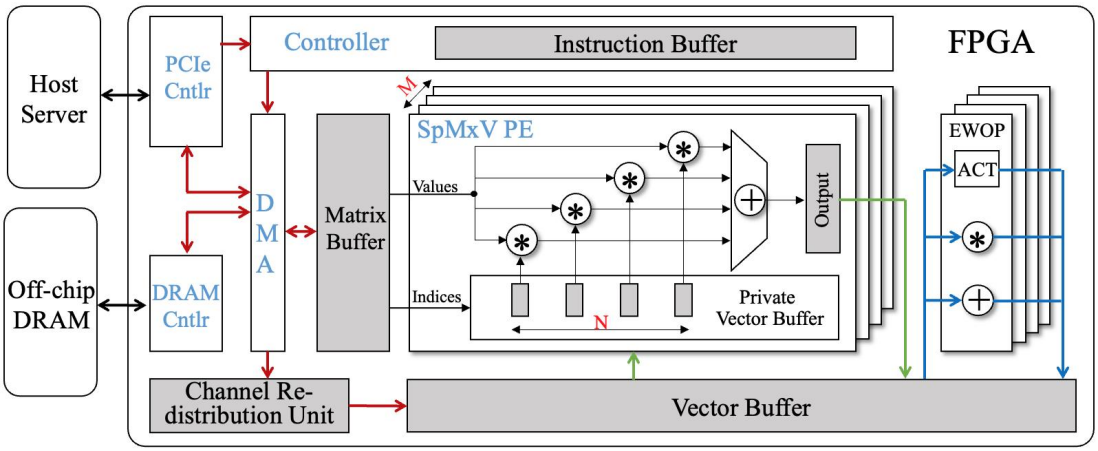


Fig. 10. Accelerator Architecture

Compressed Sparse Row (CSR) is a commonly used sparse matrix format [54]. We use CSR as a representative encoding of existing formats for explanation and comparison. Fig. 7(a) shows an FSS sparse matrix represented in dense format. Fig. 7(b) shows its corresponding CSR encoding. CSR incurs two types of overheads for SpMxV operation. First, CSR format encodes all nonzero elements in a row-major order. Thus, rearranging the non-zero elements are inevitable in order to exploit inter-bank parallelism in SpMxV. Second, CSR format stores column indices and row pointers to track the location of each non-zero value. Thus, calculating memory addresses is required to fetch vector elements. Other encoding formats, such as CSC and COO have similar limitations [54].

The proposed CSB format takes advantage of the balanced property of FSS and eliminates the need for decoding. Fig. 7(c) shows the CSB representation of the corresponding matrix. The CSB encoding uses two arrays to represent a bank-balanced sparse matrix. In the first array (i.e., values), all non-zero values are first arranged in row-major order. Inside each row, the first non-zero elements in each banks (e.g., $\langle A, C, E, G\rangle$) are listed first, then the second elements, and so on. The purpose of this data rearrangement is to explicitly expose inter-bank parallelism, thus every successive N elements in CSB can be directly fetched and computed upon in parallel. The second array (i.e., indices) lists the bank internal indexes of non-zero values which are column indices modulo bank size K. When each of the N vector banks is stored in a separate BRAM block on FPGA, the bank internal indices can be directly regarded as physical addresses to fetch the N corresponding vector elements in the BRAM blocks.

B. SpMxV Processing Engine for FSS

The core computation in RNNs and LLMs is sparse matrix-vector multiplication (SpMxV), while in CNNs, it is matrix-matrix multiplication, which can also be decomposed into SpMxV operations. In our SpMxV design, the computation consists of multiple dot product operations, each corresponding to a row in the sparse matrix and the dense vector. The standard practice of using multiple processing elements (PEs) to parallelize dot products across matrix rows can reduce computation time. However, the irregular memory access patterns of unstructured sparse matrices limit the potential for further parallelism within a dot product.

In addition to inter-row parallelism, FSS enables an efficient SpMxV design to exploit intra-row parallelism (i.e. inter-bank parallelism) through the bank-balance partitioning. Fig. 8 illustrates how to exploit inter-bank parallelism in computing a dot product of two vectors (i.e., a BBS matrix row and the dense vector). The multiplications for the non-zero elements inside each bank are performed serially, while the multiplications in different banks are performed in parallel. In this example, the sparse matrix row is divided into 4 banks, as is shown in different colors. The size of each bank is 3 and the sparsity is 1/3. The multiplied dense vector is divided into 4 banks accordingly. Our design computes the dot product of two vectors by accumulating dot products of subvectors whose sizes are all the number of banks (N). Each bank of the sparse matrix row provides one non-zero element to form one sub-vector (e.g., $\langle A, C, E, G\rangle$), while dense vector elements are fetched based on the indices of non-zero values to form another sub-vector (e.g., $\langle v_0, v_3, v_7, v_9\rangle$). For computing a dot product of sub-vectors, N pair-wise multiplications are executed in parallel. Multiple dot products of sub-vectors are calculated in sequential and accumulated to obtain the dot product of complete vectors.

C. Channel Re-distribution Unit

The Channel Re-distribution Unit (CRU) is designed to perform dynamic re-distribution of activation channels in the input data path. Since the weight matrix is read-only during inference, its channel re-distribution can be pre-arranged offline. However, activations cannot be pre-arranged because each layer's computation generates new activations, which then serve as inputs for the subsequent layer. The re-distribution pattern for each layer's activations is distinct, necessitating dynamic adjustment of

their positions before they are fed into the FSS Processing Engine.

The design goal of the CRU is to parallelize the re-distribution of input vector data without impeding the data transfer process to the computation units. To achieve this, we employ a cross-bar switch design, utilizing one-hot bitmap encoding to control the corresponding switches. As illustrated in Fig. 7, this method allows the re-distribution to be completed as the data flows through the unit, without introducing additional clock cycles.

D. Overall Architecture Design

Accelerator Integration: Fig. 10 shows the overall architecture of the FSS accelerator, which includes a sparse matrix-vector multiplication unit (SpMxV Unit), an element-wise vector operation unit (EWOP Unit), a channel re-distribution unit (CRU), a direct memory access module (DMA) for load/store operations, on-chip memories for matrices and vectors (Matrix Memory and Vector Memory), and a central controller. Before hardware acceleration, the host server uses the FSS pruning method to prune weight matrices and represents sparse matrices in the proposed Compressed Sparse Banks (CSB) format, as well as the Channel Re-distribution Bitmap (CRB). A lightweight compiler then generates instructions for the hardware accelerator to execute the computation of AI models. The controller receives and stores these instructions from the host server in the instruction buffer and dispatches them to the corresponding modules for execution.

At the center of Fig. 10, the detailed architecture of a processing element (PE) is shown. Each PE contains a private vector buffer (PVB) to buffer the dense vector being multiplied, as vector elements are randomly accessed multiple times for all matrix rows in SpMxV. The PE computes the dot product of two vectors by accumulating dot products of sub-vectors through the following steps: (1) The PE reads N matrix row elements from the matrix memory and N vector elements from the private vector buffer based on the sparse indices. (2) N multipliers operate simultaneously to obtain N scalar products. (3) An N-input adder tree sums the N scalar products to calculate the partial dot product. (4) An additional accumulator is used to obtain the complete dot product. (5) The dot product result is written back to the global vector memory. The PE is fully pipelined, allowing one operation to be processed per clock cycle. With M PEs and N multipliers per PE, this PE array achieves $M \times N$ parallelism for a single SpMxV operation.

EWOP Unit: The EWOP unit performs various element-wise operations on vectors based on the instruction opcode. Vector addition and multiplication generate one result vector by reading two source vectors. Activation functions only read one source vector and apply nonlinear functions to it to generate one result vector. The EWOP unit contains M operators operating in parallel for each kind of operations to reduce latency.

Controller: In the computation flow of AI models, some SpMxV and EWOP operations can be executed simultaneously. The software compiler analyzes the computation flow graph and identifies dependencies within the instructions. The controller then parallelizes these instructions based on the dependencies indicated by the software compiler. When either the SpMxV unit or the EWOP unit becomes idle (indicating that an instruction has finished execution), the controller checks if the next instruction depends on the instruction currently being executed by the other unit. If no dependency exists, the controller dispatches the next instruction to the idle unit, enabling simultaneous operation of the SpMxV unit and the EWOP unit.

E. Comparison to other SpMxV Accelerators

Recently, numerous sparse matrix-vector (SpMxV) hardware designs have emerged [55], [56]. These accelerators typically target matrix densities in the range of E^{-3} to E^{-9}, corresponding to sparsities of 99.9% to 99.9999999%, primarily focusing on tasks such as scientific computing. In contrast, the sparsity in AI workloads typically ranges from 50% to 90% and rarely exceeds 99% [12], [57]. This significant disparity in density (or sparsity) levels, spanning over two orders of magnitude or even more, leads to vastly different data patterns, which in turn result in notable distinctions in the design principles for optimized accelerators in each domain.

One key difference lies in their sparse encoding/decoding schemes. For matrices with extremely high sparsity, the non-zero elements are distributed very sparsely, which implies that only one in a thousand or even a million elements is non-zero. In such cases, a list-based encoding approach is often employed, where each non-zero element is recorded along with two additional pieces of information, such as its column and row indices, to track its position. For high-dimensional sparse tensors, even more indices are required, resulting in a complex encoding structure. However, this encoding method becomes inefficient for matrices with sparsity levels between 50% and 90% in AI matrix computations. For example, in a matrix with 50% sparsity, the ratio of zero to non-zero elements is 1:1, meaning that compressing a single zero element would require more index codes if we were to use a similar sparse encoding format as for extremely sparse matrices. This not only imposes significant storage and memory access overhead but also increases the complexity of encoding and decoding computations.

As a result, accelerators designed for AI matrix computations typically employ an array-based encoding scheme. In this approach, non-zero elements and their positional codes are organized in a way that aligns closely with the matrix's inherent structure. For example, the CSB encoding method proposed in this paper (Fig.7) arranges non-zero elements and their corresponding indices into tiles of the same size, using local encoding within each block. This significantly reduces the complexity of encoding. Additionally, AI workloads require consideration not

only of computational efficiency but also of how the sparse pattern impacts model accuracy, a factor that must be carefully addressed in the design of sparse AI accelerators.

In summary, the differences in sparsity levels and data distribution patterns lead to distinct approaches in the design of sparse encoding format and corresponding SpMxV accelerators for AI workloads compared to those for scientific computing.

VI. Experiments

A. Evaluation Method

In this paper, we are focused on evaluating the efficiency and effectiveness of the FSS and CR-FSS methods. We demonstrate the advantages of the FSS-based weight pruning method over the block sparse method in terms of model accuracy and sparsity of the pruned weights. On LLMs, we further demonstrate the advantage of the CR-FSS method over the original FSS method. The efficacy of the FSS method across CNNs, LSTMs, and LLMs is evidenced through extensive weight pruning experiments, highlighting its flexibility and applicability. For attention layers, which are weight-free and thus cannot benefit from FSS/CR-FSS optimizations, we adopt the block sparse attention scheme as detailed in prior work [58]. To ensure a fair comparison, we follow the configuration established by previous state-of-the-art research [47], which places the attention matrix and KV caches within UltraRAM. For quantization, methods akin to SmoothQuant [24] and FlightLLM [47] are employed, quantizing both weights and activations to enhance computational efficiency.

On the hardware part, we compare the performances on two hardware platforms: GPUs and FPGAs. For GPUs, our baseline comparison involves the use of Sparse Tensor Cores in the Ampere architecture, which employs a sparse method similar to the approach proposed in our previous study [6]. However, thanks to the flexible reconfigurability of FPGA, our accelerator approach can accommodate higher sparsity levels compared to the GPU's Sparse Tensor Cores, which are limited to fixed sparsity ratios such as 2:4. In the context of FPGAs, while some subsequent studies have employed N:M schemes, our work primarily focuses on comparing the implementations of CR-FSS and FSS on FPGAs.

B. Experiment Setup

AI Models and Data sets: 1) CNN Models: We utilize the ResNet-56 model configuration as described in [52]. This model was tested on the ImageNet dataset [59], which comprises over 14 million images. 2) RNN Models: We adopt the LSTM model from [51], configured with 1500 hidden units. This model was evaluated using the PTB dataset [60], commonly used in Natural Language Processing (NLP) research. The PTB dataset contains 929k training words, 73k validation words, and 82k test words, with a vocabulary of 10k words. 3) LLM Models: We selected one state-of-the-art large language model OPT-6.7B [53]. The accuracy and performance evaluation was performed using the WikiText-103 and WikiText-2 datasets [61].

For all these models, we use a batch size of 1. We prune all layers with weight parameters, such as convolutional layers and linear layers, while keeping parameter-less layers in their original dense forms. This includes layers with varying degrees of sparsity and acceleration opportunities, such as attention layers [47] and normalization layers [62], which we leave for future research.

Hardware Platforms: 1) GPU: We use NVIDIA RTX A6000 GPGPUs with Sparse Tensor Core architectures as our GPU baseline. These GPUs offer approximately 300 TFLOPS of FP16 Tensor Core computation power and 768 GB/s GDDR6 external bandwidth. For block sparse operations, we use the optimized GPU implementation provided by OpenAI [63]. We leverage the cuSPARSELt [64] library, provided by the vendor, to utilize the Sparse Tensor Cores hardware as the baseline. 2) FPGA: Our FPGA implementations employ Xilinx Alveo U250 and U280 acceleration cards [65], [66]. The proposed accelerator configurations are detailed in Table II. The static region of each FPGA contains the deployment shell responsible for device bring-up and configuration via PCIe. We primarily exploit the dynamic region for accelerator implementation, where hardware resources available on these FPGAs are shown in Table III.

TABLE II
The Proposed Accelerator's Hardware Configurations

	U250	U280	A6000
Tech Node	16 nm	16 nm	8 nm
Frequency	225 MHz	225 MHz	1410 MHz
Wght.×Act.	4 × 8 bit	4 × 8 bit	8 × 8 bit
Computing Units	9216 DSPs	6912 DSPs	336 Tensor Cores
Mem. BW (GB/s)	URAM:2592 DDR: 77	URAM:1944 DDR: 460	DDR: 768

TABLE III
The Proposed Accelerator's Hardware Utilization

	Alveo U250	Alveo U280
LUT	795K/1,728K (46%)	704K/1,304K (54%)
FF	1140K/3,456K (33%)	913K/2,607K (35%)
BRAM	1684/2000 (84%)	1220/1490 (81.9%)
URAM	1152/1280 (90%)	864/960 (90%)
DSP	9216/11580 (80%)	6912/8490 (81.4%)

TABLE IV
Model accuracy/perplexity sensitivity to the block size in block sparsity and in FSS/CR-FSS. All sparsity is set to 50%.

	Block Sparse			FSS/CR-FSS			
block size	4x4	8x8	16x16	8	16	32	64
ResNet-50*	25%	20%	11.2%	91.9%	92%	92%	92%
LSTM	79.5	81.7	85.1	78.5	78.5	78.5	78.5
OPT-6.7B	16K	18K	21K	12.1	12.0	12.0	12.0

*Resnet-50 is higher the better, others are smaller the better

TABLE V
Trade-offs of FSS/CR-FSS Block Size.

Block Size	4	8	16	32	64
SP Granularity	25%	12.5%	6.25%	3.125%	1.5625%
Index Bitwidth	2	3	4	5	6

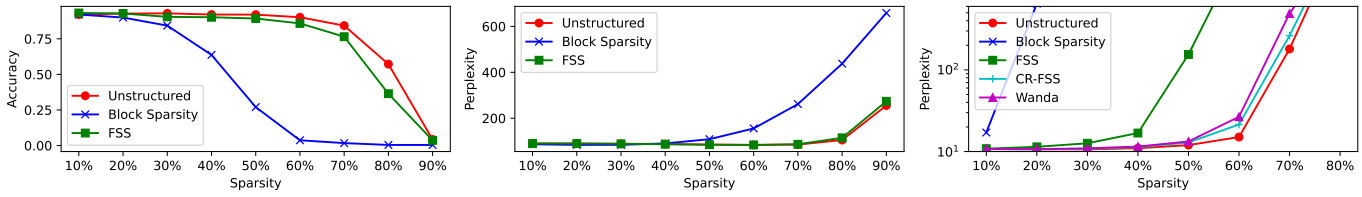
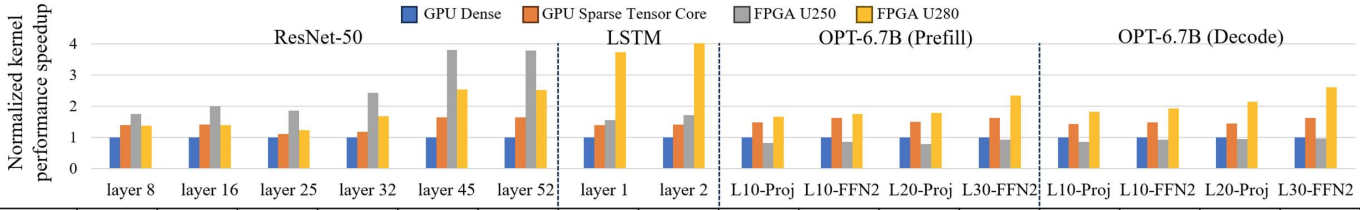


Fig. 11. Accuracy/Perplexity for Different Pruning Methods on Various Sparsity Ratio



Layer Config.	[64,3136, 256]	[128,1152, 784]	[256,2304, 196]	[256,2304, 196]	[2048,49, 512]	[2048,49, 512]	[6000, 1500]	[6000, 1500]	[128,4096, 4096]	[128,4096, 16384]	[128,4096, 4096]	[128,4096, 16384]	[1,4096, 4096]	[1,4096, 16384]	[1,4096, 4096]	[1,4096, 16384]
Sparsity	75%	81.25%	81.25%	87.50%	93.75%	93.75%	87.5%	93.75%	50%	62.5%	62.5%	68.75%	50%	62.5%	62.5%	68.75%

Fig. 12. Speedup of (CR-)FSS-based FPGA Acceleration over GPU Tensor Core and Sparse Tensor Core baseline. For ResNet-50, the layer config is [input channel, feature-map size, output channel]. For LSTM, the layer config is [input channel, output channel]. For Opt-6.7B, the layer config is [sequence length, token size, intermediate size].

C. Model Accuracy and Perplexity

We evaluate the model accuracy and perplexity with unstructured sparsity and FSS/CR-FSS applied respectively, as is depicted in Fig.11. Overall, the behaviour of FSS/CR-FSS is very close to unstructured sparsity. For CNN&RNN models, when sparsity is lower than 70%, the accuracy and perplexity of these methods can remain at a certain level. There is a sharp degradation when sparsity is higher than 80%. For LLM models, the perplexity grows with sparsity in both unstructured sparsity and FSS/CR-FSS.

Comparison to block sparse pruning. We further conduct an evaluation on the block sparsity with (4,4) block size. Fig.11 illustrates that the accuracy and perplexity deteriorate quickly when sparsity grows for block sparsity. For CNN model, the accuracy begins to decrease fast at 30% sparsity with block sparsity, while the turning point of FSS/CR-FSS is 70% sparsity. For LLM, the perplexity skyrockets even at 10% sparsity with block sparsity. This is because FSS/CR-FSS applies fine-grained pruning within each bank, while block sparsity is coarse-grained where important weights may be pruned. Moreover, the block sparsity does not take account for the activation's impact on the weight, while CR-FSS considers this point and swaps the channels of weight and activation matrices. Another advantage of FSS/CR-FSS is its stability in maintaining model accuracy. As shown in Table IV, for block sparse, the quality of all three models deteriorates significantly as the block size increases (e.g., a decline in ResNet accuracy, and an increase in perplexity for the other two models). In contrast, for FSS/CR-FSS, model quality remains highly stable, showing little dependence on block size.

Comparison to other N:M sparse pruning. We implement previous state-of-the-art N:M weight pruning methods Wanda [25] as it also prunes LLM weight matrices considering both activation and weight. Fig.11(c) demonstrates that our CR-FSS has a better performance than both original FSS and Wanda [25]. The original FSS method does not account for the impact of activation outliers and prunes weights solely based on their magnitudes, resulting in significantly lower accuracy compared to the other two methods. Specifically, the original FSS's accuracy at 40% sparsity is comparable to that of CR-FSS at 60% sparsity. Moreover, although both Wanda and CR-FSS consider activation's impact when pruning weight matrices, our CR-FSS swaps the channels to prevent mistakenly pruning of important weights in outlier-rich area, and hence leading to better model accuracy.

Trade-offs of FSS/CR-FSS's Block Size. Block size is the key to balance the sparsity performance gains and hardware cost. As illustrated in Table V, as the block size increases, the granularity of sparsity that can be represented becomes finer, but also requires a wider bitwidth for indexing. According to prior research [57] as well as the experimental results shown in Fig.11, the sparsity levels of the AI models mostly ranges from 50% to over 90%. Based on this observation, we determined a block size of 16. Reducing the block size to 8 would limit the supported granularity, failing to fully exploit the benefits of layers with sparsity exceeding 90%. Conversely, increasing the block size to 32 or beyond offers diminishing returns, as layers with sparsity levels above 95% are rare, and doing so would increase hardware cost due to the need for 5-bit indexing. It is worth noting that in NVIDIA's Ampere GPGPU, the Sparse Tensor Core operates in a fixed 2:4 sparsity mode, resulting in a fixed 2-bit index width. This mode only leverages 50% sparsity, and fails to fully exploit higher sparsity potentials, such as the 68.75% sparsity observed in the Opt-6.7B.

D. Accelerator Speedup

Kernel Speedup: Compared to GPU implementations, our approach achieves a higher acceleration ratio under the

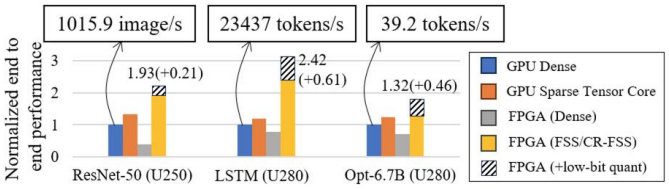


Fig. 13. End to end speedup. The performance data is based on the average inference time running on the dataset in SectionVI-B. We use U250 FPGA for CNN workloads and U280 FPGA for RNN and LLM workloads.

same algorithmic accuracy by utilizing greater sparsity, which leads to reduced memory access and computational demand. The sparse tensor cores of GPUs only support a fixed sparsity of 50%. Hence, we observed acceleration ratios ranging from 1.3x to 1.6x compared to the baseline of dense GPU computations. In CNN models, our FSS achieved sparsities between 60% and 90% across different layers, resulting in a performance increase of 2.1x to 3.8x over dense GPU matrix computations. The U250 FPGA, possessing more DSP resources, demonstrates greater advantages in the more computationally intense convolution operations compared to the U280 FPGA. For RNN models, FSS achieved an average sparsity of 80%, while block sparsity only reached 50%. In such cases, the FSS accelerator based on U280 FPGA exhibited acceleration ratios from 2.4x to 2.8x compared to dense GPU computations. However, due to external bandwidth limitations, the performance advantage of the U250 FPGA in RNN applications is not significant compared to the A6000 GPU with a bandwidth of 768GB/s.

For LLM models, since the block sparse pruning method results in significant accuracy loss at 10% sparsity, our focus shifted to comparing the FSS and CR-FSS schemes. CR-FSS maintains higher accuracy by preserving more critical weights. Although the perplexity turning points for FSS and CR-FSS occur at around 50% to 60% sparsity, CR-FSS allows some layers to exceed the average sparsity level, such as reaching 68.75% in layer L30. Relative to a 50% sparsity providing a theoretical 2x acceleration ratio, a 68.75% sparsity can offer up to a 3.17x theoretical speedup. Practical experiments have shown that CR-FSS on the U280 FPGA offers a performance improvement of 1.5x to 2.6x compared to dense GPU computations. However, due to the limited external bandwidth of the U250 FPGA and the lower sparsity in LLM models, its performance is even slower than the baseline.

End-to-end Performance: Fig.13 presents a detailed analysis of the end-to-end speedup and the performance breakdown of FPGAs. All performance metrics have been normalized to the baseline implementation using NV A6000's Dense Tensor Cores. For the naive FPGA implementation, we port Caffine [67] to U250 and U280 FPGAs as the baseline for dense models. Compared to GPU-accelerated dense models, the performance of the naive FPGA only ranges from 0.38x to 0.72x. This discrepancy primarily arises from GPUs' formidable computing capability up to 600 TFLOPS in Int8 precision and a DRAM read/write bandwidth of 768 GB/s, far surpassing that of FPGAs. By employing the sparsification strategies of FSS/CR-FSS, which significantly reduce the workload, we achieved acceleration effects ranging from 1.32x to 2.42x on FPGAs. Additionally, FPGAs benefit from lower precision quantization (e.g., 3-bit), an acceleration benefit GPUs struggle to utilize effectively. This acceleration is particularly pronounced in memory-bound models such as RNNs and LLMs, providing additional speedup ranging from 0.21x to 0.61x.

Comparison to other FPGA accelerators: Recent studies have leveraged FPGAs to accelerate LLMs, including initiatives like DFX [68], FACT [69], CTA [70], and LightLLM [47]. These efforts utilize a combination of model compression techniques to achieve significant acceleration, incorporating strategies such as lower precision quantization, sparse attention etc. These techniques are complementary to the FSS proposed in this paper and can be integrated to further enhance the performance of FPGA-based AI accelerators. Some works have adopted an N:M sparsity approach similar to that discussed in this paper [6], [7], such as FlightLLM [47]. This study primarily investigates the impact of the FSS method on model accuracy and its acceleration performance, as well as its applicability across different AI models including CNNs, RNNs, and LLMs. Furthermore, addressing the original FSS method's limitations in handling outliers within LLMs, this paper extends the method to CR-FSS. This approach solves the aforementioned challenges with minimal hardware overhead.

VII. Conclusion

This paper addresses the issue of low-latency AI model inference by proposing a sparse FPGA accelerator based on Fine-grained Structured Sparsity (FSS) and Channel Redistributed Fine-grained Structured Sparsity (CR-FSS). We evaluated the impact of our method on model accuracy across three different types of AI models: Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Large Language Models (LLMs). The computational efficiency and acceleration performance of the dedicated FPGA accelerators were validated on Alveo FPGA platforms U250 and U280. Compared to the high-end GPU NV A6000, which features Sparse Tensor Cores, our approach achieved acceleration ratios ranging from 1.5x to 3.8x.

References

[1] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," arXiv preprint arXiv:1803.01164, 2018.

[2] Y. LeCun, Y. Bengio et al., "Convolutional networks for images, speech, and time series," The handbook of brain theory and neural networks, vol. 3361, no. 10, p. 1995, 1995.

[3] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," IEEE transactions on neural networks and learning systems, vol. 28, no. 10, pp. 2222–2232, 2016.

[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," CoRR, vol. abs/1810.04805, 2018.
[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., "Language models are few-shot learners," Advances in neural information processing systems, vol. 33, pp. 1877–1901, 2020.
[6] S. Cao, C. Zhang, Z. Yao, W. Xiao, L. Nie, D. Zhan, Y. Liu, M. Wu, and L. Zhang, "Efficient and effective sparse lstm on fpga with bank-balanced sparsity," in Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, pp. 63–72.
[7] Z. Yao, S. Cao, W. Xiao, C. Zhang, and L. Nie, "Balanced sparsity for efficient dnn inference on gpu," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 5676–5683.
[8] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang et al., "Ese: Efficient speech recognition engine with sparse lstm on fpga," in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017, pp. 75–84.
[9] L. Lu and Y. Liang, "Spwa: an efficient sparse winograd convolutional neural networks accelerator on fpgas," in 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). IEEE, 2018, pp. 1–6.
[10] G. Varma, K. Kothapalli et al., "Dynamic block sparse reparameterization of convolutional neural networks," in Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, 2019, pp. 0–0.
[11] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in International Conference on Machine Learning. PMLR, 2023, pp. 10 323–10 337.
[12] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," Advances in neural information processing systems, vol. 28, 2015.
[13] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," ACM SIGARCH Computer Architecture News, 2016.
[14] J. Chorowski, R. J. Weiss, S. Bengio, and A. Van Den Oord, "Unsupervised speech representation learning using wavenet autoencoders," IEEE/ACM transactions on audio, speech, and language processing, vol. 27, no. 12, pp. 2041–2053, 2019.
[15] X. Ma, G. Fang, and X. Wang, "Llm-pruner: On the structural pruning of large language models," Advances in neural information processing systems, vol. 36, pp. 21 702–21 720, 2023.
[16] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re et al., "Deja vu: Contextual sparsity for efficient llms at inference time," in International Conference on Machine Learning. PMLR, 2023, pp. 22 137–22 176.
[17] H. Shao, B. Liu, and Y. Qian, "One-shot sensitivity-aware mixed sparsity pruning for large language models," in ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2024, pp. 11 296–11 300.
[18] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," ACM SIGARCH Computer Architecture News, 2017.
[19] X. Liu, J. Pool, S. Han, and W. J. Dally, "Efficient sparse-winograd convolutional neural networks," arXiv preprint arXiv:1802.06367, 2018.
[20] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," arXiv preprint arXiv:1705.08922, 2017.
[21] S. Narang, E. Undersander, and G. Diamos, "Block-sparse recurrent neural networks," arXiv preprint arXiv:1711.02782, 2017.
[22] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," in International Conference on Machine Learning. PMLR, 2018, pp. 2410–2419.
[23] N. Wen, R. Guo, B. He, Y. Fan, and D. Ma, "Block-sparse cnn: towards a fast and memory-efficient framework for convolutional neural networks," Applied Intelligence, vol. 51, pp. 441–452, 2021.
[24] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in International Conference on Machine Learning. PMLR, 2023, pp. 38 087–38 099.
[25] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," arXiv preprint arXiv:2306.11695, 2023.
[26] S. Solla, Y. Le Cun, and J. Denker, "Optimal brain damage," Advances in neural information processing systems, vol. 2, pp. 598–605, 1990.
[27] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, "Net-trim: Convex pruning of deep neural networks with performance guarantee," Advances in neural information processing systems, vol. 30, 2017.
[28] C. L. Giles and C. W. Omlin, "Pruning recurrent neural networks for improved generalization performance," IEEE transactions on neural networks, vol. 5, no. 5, pp. 848–851, 1994.
[29] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," Advances in neural information processing systems, vol. 30, 2017.
[30] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "Dota: detect and omit weak attentions for scalable transformer acceleration," in Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2022, pp. 14–26.
[31] M. Xia, T. Gao, Z. Zeng, and D. Chen, "Sheared llama: Accelerating language model pre-training via structured pruning," arXiv preprint arXiv:2310.06694, 2023.
[32] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, "Structured bayesian pruning via log-normal multiplicative noise," Advances in Neural Information Processing Systems, vol. 30, 2017.
[33] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," Advances in neural information processing systems, vol. 29, 2016.
[34] W. Wen, Y. He, S. Rajbhandari, M. Zhang, W. Wang, F. Liu, B. Hu, Y. Chen, and H. Li, "Learning intrinsic sparse structures within long short-term memory," arXiv preprint arXiv:1709.05027, 2017.
[35] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 806–814.
[36] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," arXiv preprint arXiv:1705.08922, 2017.
[37] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," arXiv preprint arXiv:1710.01878, 2017.
[38] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," ACM SIGARCH Computer Architecture News, vol. 45, no. 2, pp. 27–40, 2017.
[39] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," ACM SIGARCH Computer Architecture News, 2016.
[40] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in International Symposium on Microarchitecture (MICRO), 2018.
[41] G. Huang, Z. Wang, P.-A. Tsai, C. Zhang, Y. Ding, and Y. Xie, "Rm-stc: Row-merge dataflow inspired gpu sparse tensor core for energy-efficient sparse acceleration," in Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, 2023, pp. 338–352.
[42] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, "Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 359–371.
[43] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive compu-

tation time for residual networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1039–1048.
[44] Z. Yao, V. Gripon, and M. Rabbat, "A gpu-based associative memory using sparse neural networks," in International Conference on High Performance Computing & Simulation (HPCS), 2014, pp. 688–692.
[45] C. Guo, Y. Zhou, J. Leng, Y. Zhu, Z. Du, Q. Chen, C. Li, B. Yao, and M. Guo, "Balancing Efficiency and Flexibility for DNN Acceleration via Temporal GPU-Systolic Array Integration," in Proceedings of the Design Automation Conference, 2020.
[46] Z. Que, T. Nugent, S. Liu, L. Tian, X. Niu, Y. Zhu, and W. Luk, "Efficient weight reuse for large lstms," in 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), vol. 2160. IEEE, 2019, pp. 17–24.
[47] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang et al., "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," in Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2024, pp. 223–234.
[48] H. Abdelkhalik, Y. Arafa, N. Santhi, and A.-H. A. Badawy, "Demystifying the nvidia ampere architecture through microbenchmarking and instruction-level analysis," in 2022 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, 2022, pp. 1–8.
[49] 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.
[50] 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.
[51] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," arXiv preprint arXiv:1409.2329, 2014.
[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[53] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin et al., "Opt: Open pre-trained transformer language models," arXiv preprint arXiv:2205.01068, 2022.
[54] D. Langr and P. Tvrdik, "Evaluation criteria for sparse matrix storage formats," IEEE Transactions on parallel and distributed systems, vol. 27, no. 2, pp. 428–440, 2015.
[55] L. Song, Y. Chi, L. Guo, and J. Cong, "Serpens: A high bandwidth memory based accelerator for general-purpose sparse matrix-vector multiplication," in Proceedings of the 59th ACM/IEEE design automation conference, 2022, pp. 211–216.
[56] S. Li, D. Liu, and W. L. Liu, "Efficient fpga-based sparse matrix–vector multiplication with data reuse-aware compression," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 12, pp. 4606–4617, 2023.
[57] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "Extensor: An accelerator for sparse tensor algebra," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 319–333.
[58] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang et al., "Big bird: Transformers for longer sequences," Advances in neural information processing systems, vol. 33, pp. 17 283–17 297, 2020.
[59] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in 2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009, pp. 248–255.
[60] P. Wagner, N. Strodthoff, R. Bousseljot, W. Samek, and T. Schaeffter, "Ptb-xl, a large publicly available electrocardiography dataset (version 1.0.3)," PhysioNet, 2022, https://physionet.org/content/ptb-xl/1.0.3/.
[61] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," arXiv preprint arXiv:1609.07843, 2016.
[62] N. Zheng, B. Lin, Q. Zhang, L. Ma, Y. Yang, F. Yang, Y. Wang, M. Yang, and L. Zhou, "{SparTA}:{Deep-Learning} model sparsity via {Tensor-with-Sparsity-Attribute}," in 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 2022, pp. 213–232.
[63] S. Gray, A. Radford, and D. P. Kingma, "Gpu kernels for block-sparse weights," arXiv preprint arXiv:1711.09224, vol. 3, no. 2, p. 2, 2017.
[64] cusparselt: A high-performance cuda library for sparse matrix-matrix multiplication. [Online]. Available: https:https://docs.nvidia.com/cuda/cusparselt/
[65] Xilinx, "Alveo u200 and u250 data center accelerator cards data sheet," https://www.xilinx.com/support/documentation/data_sheets/ds962-u200-u250.pdf., 2020.
[66] A. Xilinx, "U280 data center accelerator card data sheet," DS963, p. v1, 2020.
[67] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 11, pp. 2072–2085, 2018.
[68] 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," in 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2022, pp. 616–630.
[69] 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," in Proceedings of the 50th Annual International Symposium on Computer Architecture, 2023, pp. 1–14.
[70] H. Wang, H. Xu, Y. Wang, and Y. Han, "Cta: Hardware-software co-design for compressed token attention mechanism," in 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2023, pp. 429–441.

VIII. Biography Section

(Member, IEEE) is a Tenure-Track Assistant Professor with Shanghai Jiao Tong University. Previously, he received the Ph.D. degree in EECS from Peking University in 2017 and then served as a senior researcher at Microsoft Research and a GPGPU architect at Alibaba. His research interests include computer architectures and heterogeneous computing for cloud & edge AI systems. He received FPGA 2015 Best Paper Nomination, TCAD 2019 Donald O. Pederson Best Paper Award, etc.

Shijie Cao is a senior researcher at the Systems Research Group in Microsoft Research Asia. He received his Ph.D. degree in Computer Science from Harbin Institute of Technology in 2021. His research interests lie at the intersection of computer system and deep learning, including domain-specific architectures, software-hardware co-design, deep learning compression and acceleration.

Guohao Dai is a Tenure-Track Associate Professor at Shanghai Jiao Tong University. He received his Bachelor's and Ph.D. degrees in Electronic Engineering from Tsinghua University in 2014 and 2019, respectively. He has published over 50 high-impact papers in top-tier journals and conferences in the fields of circuit design automation and computer architecture, with more than 1,000 citations on Google Scholar.

Chenbo Geng is studying for a Ph.D. at Shanghai Jiaotong University. He received his B.S. degree in Computer Science from Harbin Institute of Technology in 2024. His research interests include machine learning systems and software-hardware co-design.

Zhuliang Yao received the Ph.D. degree of Computer Science from Institute for Advanced Study (IASTU), Tsinghua University. He was a research intern at Microsoft Research Asia when this work was done. His research interests include computer vision and machine learning especially fundamental algorithm and structures in deep learning.

Wencong Xiao received the Ph.D. degree in CS from Beijing university, Beijing, China, in 2019. He is a staff engineer in PAI team of Alibaba Cloud. His research interests include machine learning systems and GPU cluster scheduling.

Yunxin Liu (Senior Member, IEEE) is a Guoqiang Professor at the Institute for AI Industry Research (AIR), Tsinghua University. He received his Ph.D., M.S. and B.S. degrees from Shanghai Jiao Tong University (SJTU), Tsinghua University, and University of Science and Technology of China (USTC), respectively. His research interests are mobile computing and edge computing. He received MobiSys 2021 Best Paper Award, SenSys 2018 Best Paper Runner-up Award, MobiCom 2015 Best Demo Award, and PhoneSense 2011 Best Paper Award.

Ming Wu received the Ph.D. degree from Institute of Computing Technology, Chinese Academy of Science in 2007. He is the CTO of Shanghai TreeGraph Blockchain Research Institute. Previously, he served as a senior researcher at Microsoft Research Asia. His research interests include distributed storage and transactional processing systems, large scale AI computation platforms, and high performance blockchain systems. He was in the program committee of OSDI, EuroSys, ASPLOS, HotDep, and MiddleWare.

Lintao Zhang (Fellow, IEEE) received his Bachelor's degree from Peking University and his Ph.D. from Princeton University. He is currently the Chief Scientist of BaseBit Technologies Inc. Before joining BaseBit, he was a researcher at Microsoft Research. Dr. Zhang has published over fifty papers in topics ranging from formal verification and SAT solving to cloud systems and AI. He has won many awards for his research, including Best Paper Awards in OSDI and DATE, Most Influential Paper Awards in DAC and ICCAD, a CAV Award, and an IEEE A. Richard Newton Technical Impact Award.

Guangyu Sun (Senior Member, IEEE) received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, in 2003 and 2006, respectively, and the Ph.D. degree in computer science from Pennsylvania State University, State College, PA, USA, in 2011. He is an Associate Professor with the Center for Energy-Efficient Computing and Applications, Peking University, Beijing. His research interests include computer architecture, acceleration system, and electronic design automation for modern applications. Dr. Sun is currently serving as an Associate Editor for ACM Journal on Emerging Technologies in Computing Systems.

Zhigang Ji (Senior Member, IEEE) received his B. Eng. degree in Electrical Engineering from Tsinghua University in 2003, the M. Eng degree in Microelectronics from Peking University in 2006 and the Ph.D. degree in Microelectronics from Liverpool John Moores University in 2010. In 2020, he joined Shanghai Jiaotong University, where he currently holds the position as Professor in Nanoelectronics and director of LEMON lab. He has authored or co-authored over 200 scientific papers, including IEDM and VLSI. His current research interests include nanoscale CMOS and non-CMOS devices, DTCO, and emerging technologies for applications such as hardware security and new-paradigm computing.

Runsheng Wang received the B.S. and Ph.D. (highest honors) degrees from Peking University in 2005 and 2010. He joined Peking University in 2010, where he is currently a Full Professor at the School of Integrated Circuits. He has authored/coauthored 1 book, 4 book chapters, and over 200 scientific papers, including more than 40 papers published in IEDM and VLSI-T. His current research interests include nanoscale CMOS devices, characterization and reliability, circuit and device interaction, and new-paradigm computing. He was awarded the IEEE EDS Early Career Award by the IEEE EDS, the NSFC Award for Excellent Young Scientists, the Natural Science Award (First Prize) by the Ministry of Education of China, etc.

Ru Huang (Fellow, IEEE) received the Ph.D. degree in microelectronics from Peking University, Beijing, China, in 1997. She is a Professor with Peking University. She is also serving as the President of Southeast University, Nanjing, China since 2022. She is an elected Academician of the Chinese Academy of Science, an elected member of TWAS Fellow. Her research interests include nano-scaled CMOS devices, ultra-low-power new devices, new device for neuromorphic computing, emerging memory technology, and device variability/reliability.