ATE-GCN: An FPGA-based Graph Convolutional Network Accelerator with Asymmetrical Ternary Quantization

Ruiqi Chen[1,+], Jiayu Liu[2,+], Shidi Tang[3], Yang Liu[4], Yanxiang Zhu[5], Ming Ling[3,*], Bruno da Silva[1]
[1]ETRO, Vrije Universiteit Brussel, [2]University College London, [3]Southeast University, [4] Fudan University, [5] VeriMake Innovation Lab
*Corresponding author: trio@seu.edu.cn

Abstract*—Ternary quantization can effectively simplify matrix multiplication, which is the primary computational operation in neural network models. It has shown success in FPGA-based accelerator designs for emerging models such as GAT and Transformer. However, existing ternary quantization methods can lead to substantial accuracy loss under certain weight distribution patterns, such as GCN. Furthermore, current FPGA-based ternary weight designs often focus on reducing resource consumption while neglecting full utilization of FPGA DSP blocks, limiting maximum performance. To address these challenges, we propose ATE-GCN, an FPGA-based asymmetrical ternary quantization GCN accelerator using a software-hardware co-optimization approach. First, we adopt an asymmetrical quantization strategy with specific interval divisions tailored to the bimodal distribution of GCN weights, reducing accuracy loss. Second, we design a unified processing element (PE) array on FPGA to support various matrix computation forms, optimizing FPGA resource usage while leveraging the benefits of cascade design and ternary quantization, significantly boosting performance. Finally, we implement the ATE-GCN prototype on the VCU118 FPGA board. The results show that ATE-GCN maintains an accuracy loss below 2%. Additionally, ATE-GCN achieves average performance improvements of 224.13× and 11.1×, with up to 898.82× and 69.9× energy consumption saving compared to CPU and GPU, respectively. Moreover, compared to state-of-the-art FPGA-based GCN accelerators, ATE-GCN improves DSP efficiency by 63% with an average latency reduction of 11%.

I. Introduction

Ternary quantization [1] reduces the bitwidth of weights to {-1, 0, 1} and can effectively simplify the matrix multiplication, which is the primary operation in neural network (NN) models. Consequently, this strategy has been applied to emerging NN models, such as Graph Attention Networks (GAT) [2] and Transformers [3], and achieved high accuracy. However, significant accuracy loss occurs when directly deploying ternary quantization on Graph Convolutional Networks (GCNs) [4] which are effective for graph-structured data and deliver top accuracy in node classification tasks [5]. This is mainly due to previous ternary quantization employing symmetric interval division, assuming the weights are generated the uniform or normal distribution. While the asymmetric ternary quantization strategy [6] has been proposed, it follows previously empirical interval divisions and also results in significant accuracy loss. To the best of our knowledge, ternary quantization deployment in GCNs remains unexplored.

Hardware (FPGA- or ASIC-based) NN inference accelerators benefit from ternary quantization, allowing them to outperform GPU implementations [7]. As NN models evolve, FPGA is an ideal candidate for target platform due to its flexibility and modular design, allowing quick adaptation to varying computational requirements [8]. The DSP-free design [9], [10] significantly enhances area efficiency for low-bit quantization. Additionally, matrix multiplication is simplified into addition operations, efficiently executed with combinational logic [11]. Furthermore, encoding weights in a specific format supports efficient matrix addition operations within simple counter-modules [2], enabling a DSP-light design. Although these designs avoid the inefficient overuse of DSP resources, they actually waste the high-performance fixed on-board computing resources, thereby limiting the design performance. Additionally, for low-precision data designs, previous research uses DSP-Packing [12]–[14] to boost DSP throughput and enhance system performance. However, the operating frequency of DSP is strictly limited by correction units and other peripheral circuits. These optimizations fail to fully exploit performance potential of device.

In view of this, we take a deeper look at multiple stages of GCN accelerator design with ternary quantization and highlight the following challenges: (1) **Ternary quantization on GCN leads to accuracy loss**. Directly applying the ternary quantization strategy can result in unacceptable accuracy loss (over 20%) as the weight distribution in GCNs does not align with previous empirical assumptions, which follow the bimodal distribution. (2) **On-chip resources remain underutilized**. In FPGA-based accelerator design, computational resources are fixed, and maximizing their use is a complex challenge. (3) **Toward peak computational performance**. For low-precision designs, both DSP-light and DSP-packing approaches are affected by auxiliary logic and data paths, limiting DSP's ability to achieve higher operational frequencies. To address these challenges, we contribute the following solutions:

- We propose specific interval divisions tailored to bimodal distributions with the asymmetric ternary quantization strategy. It is the first effectively deploy ternary weight quantization for GCNs with minimal accuracy loss.
- We propose ATE-GCN, which leverages the inherent resources of FPGAs while incorporating the benefits of ternary quantization. Specifically, we design a unified PE array that efficiently handles various matrix computation modes through optimized data paths, including Ternary mode and Sparse Matrix Multiplication (SpMM) mode.
- We introduce a cascaded design to reduce wiring in the PE array, increasing its operating frequency. Additionally, we propose a LUT-based tightly coupled design to support various precision computation modes, further boosting system throughput.

+: Both authors contributed equally to this work.

978-3-9826741-0-0/DATE25/© 2025 EDAA

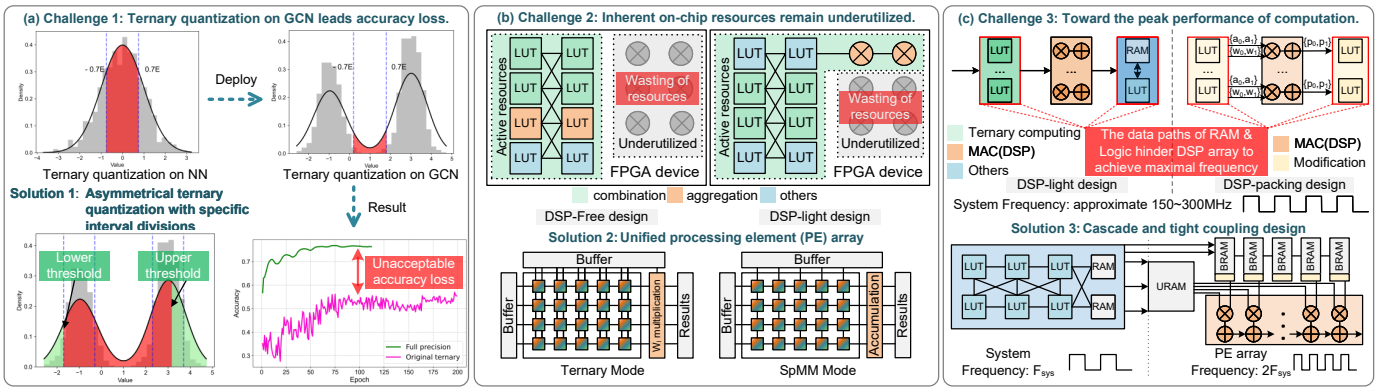

Fig. 1: Challenges in designing an FPGA-based GCN accelerator with ternary weights, spanning from software to hardware.

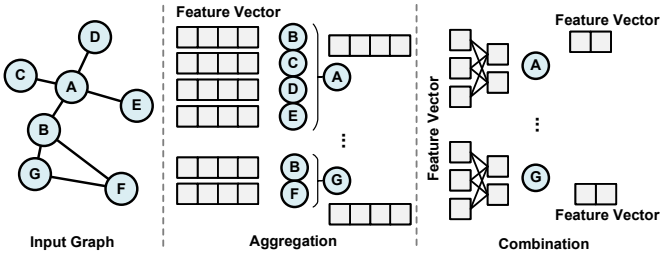

Fig. 2: The computing process of GCN.

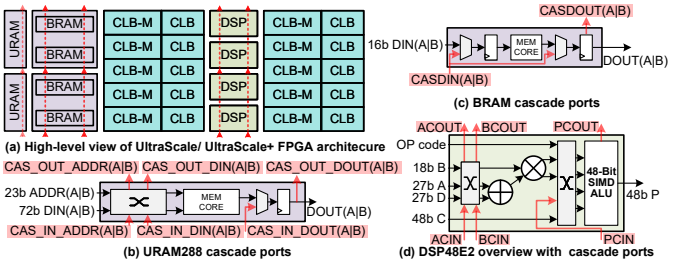

Fig. 3: The demonstration of UltraScale/ UltraScale+ FPGA inherent resources and casecade structures.

II. Motivations and Background

A. Motivations

1) Software: Ternary quantization has been widely applied to emerging NN models (such as GAT and Transformer). To extend such success to GCNs, we attempt to directly deploy the same strategy. Unfortunately, it leads to an unacceptable accuracy loss, exceeding 20%, as shown in Fig. 1(a). Further analysis shows that GCN exhibits the bimodal weight distribution. In contrast, the weights in models like GAT and Transformer follow a normal distribution [15], [16], aligning with previous empirical assumptions [1]. The asymmetric bimodal distribution of GCN weights makes it difficult for previous strategies and interval divisions to adapt effectively.

2) Hardware: Existing FPGA-based ternary quantization accelerator designs focus on resource savings, particularly in reducing DSP usage, such as DSP-free [9], [10] and DSP-light [2], [11] designs. While these approaches effectively reduce resource consumption, they may also lead to resource underutilization, as shown in Fig. 1(b). Since FPGA devices have fixed on-chip resources, it is crucial to maximize resource utilization and fully leverage the benefits of ternary quantization, especially in simplifying matrix multiplications. Meanwhile, for low-precision quantized data, although DSP-light designs reduce resource usage, they experience a drop in operating frequency due to the connection between the DSPs and the numerous LUT-based computation circuits. This hinders DSPs run at theoretical peak frequencies above 700 MHz [17]. Moreover, while DSP-Packing [12]–[14] designs increase accelerator throughput, the presence of LUT-based correction circuits force the entire system to run at the same frequency, limiting its overall performance. Therefore, achieving higher DSP operating frequencies is crucial for maximizing the system's peak performance.

B. Background

1) Computing process of GCN: GCN [4] follows a neighborhood aggregation process, where the feature vector for each vertex is calculated by recursively aggregating and transforming the representation vectors of its neighbors. Fig. 2 shows the main computing process of the GCN. In this paper, we specifically foucs on undirected graphs and the inference process. The general computation for the k-th later of a GCN is formulated as: $h^{(k+1)} = \sigma\left(\tilde{A}h^{(k)}W^{(k)}\right)$ where σ is an activation function, typically ReLU. $\tilde{A}$ is the normalized adjacency matrix, calculated as $(\tilde{A} = D_u^{-\frac{1}{2}}AD_v^{-\frac{1}{2}})$, where D is the degree matrix of the graph. Additionally, the computing process of GCN can be divided into Aggregation phase and Combination phase as described in Eq. 1.

$$\begin{cases} h_v'^k = \sum_{u\in N(v)\cup\{v\}} \frac{h_u^{(k)}}{\sqrt{D_u \cdot D_v}}, \\ h_v^{k+1} = \sigma(W^k \otimes h_v'^k) \end{cases} \quad \begin{cases} h_u'^k = W^k \otimes h_u^k \\ h_v^{k+1} = \sigma(\sum_{u\in N(v)\cup\{v\}} \frac{h_u'^k}{\sqrt{D_u \cdot D_v}}) \end{cases} \tag{1}\tag{2}$$

Since the Aggregation phase involves sum operations, the computation order can be rearranged without affecting the correctness, as shown in Eq. 2. Additionally, previous studies have demonstrated that that this rearrangement can effectively reduce the number of operations [18].

2) FPGA Structure: The state-of-the-art (SOTA) FPGAs from AMD (Xilinx) primarily consists configurable logic blocks (CLBs), URAMs, BRAMs, and DSPs, as shown in Fig. 3. A slice in the CLB of AMD's UltraScale/UltraScale+ FPGAs contains four 6-input LUTs, a 8-bit carry chain, eight flip-flops and multiplexers. The CLB-M is an enhanced version of CLB and can be configured to function as memory. URAM is a 288kb high-density SRAM block in FPGAs. While port aspect ratios are fixed, it has dedicated cascade ports for data

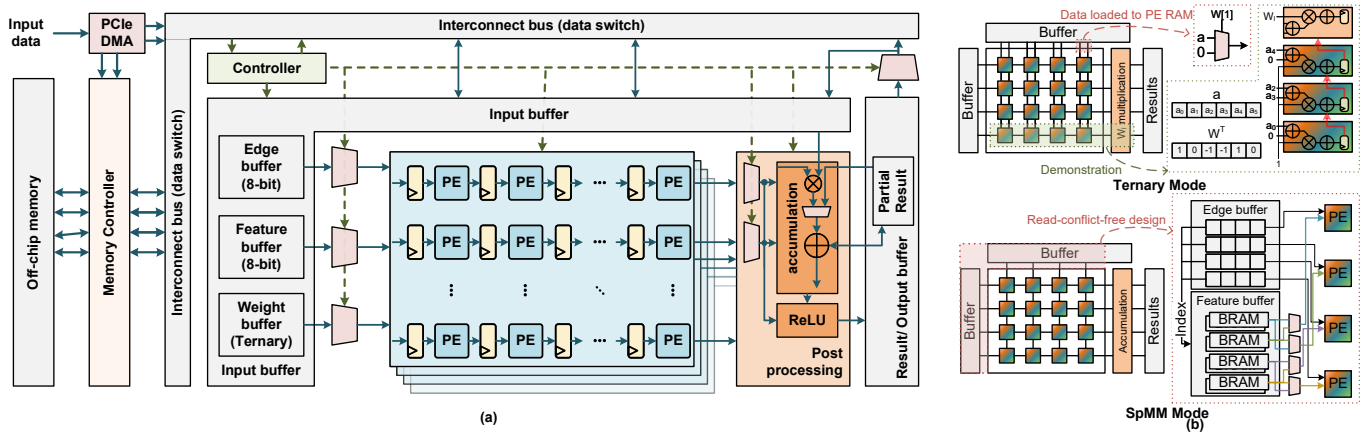

Fig. 4: (a) ATE-GCN architecture. (b) Different computation modes of PE array.

and addresses. Moreover, the read and write cascading ports are separated. BRAM has a 16-bit data width (basic block) but can be configured for different port aspect ratios and operation modes. BRAM also includes cascading ports and supports dynamically cascadable connections for its two data ports in the same direction as the DSP cascades. The DSP primarily consists of a 27-bit pre-adder, a 27×18 multiplier, and a SIMD accumulator. The OP port allows for dynamic switching between different operational modes. DSP blocks can be cascaded through the adder and input ports, enabling larger or more complex operations across multiple DSP blocks.

III. ATE-GCN

A. Training GCN with Asymmetrical Ternary Quantization

To simplify matrix multiplication and further reduce memory consumption, we apply ternary values (2-bit) to weights and 8-bit to activations. A typical ternary quantization strategy quantizes the weights into $\{-W_l, 0, +W_l\}$, as shown in Eq. 3, where Δ_l serves as the threshold parameter, while $\tilde{w}_{li}$ and w_{li}^t represent the floating-point weights and ternary weights, respectively. However, as illustrated in Fig. 1(a), GCN weights do not follow a normal distribution, making the symmetric threshold ineffective. Therefore, we adopt an asymmetric threshold, similar to the [6], as shown in Eq. 4.

$$w_{li}^t = \begin{cases} +W_l & \tilde{w}_{li} > \Delta_l, \\ 0 & |\tilde{w}_{li}| \leqslant \Delta_l, \\ -W_l & \tilde{w}_{li} < -\Delta_l. \end{cases} \quad (3)$$

$$w_{li}^t = \begin{cases} +W_l & \tilde{w}_{li} > \Delta_l^p, \\ 0 & \Delta_l^n \leqslant \tilde{w}_{li} \leqslant \Delta_l^p, \\ -W_l & \tilde{w}_{li} < \Delta_l^n. \end{cases} \quad (4)$$

Therefore, our loss function denotes as:

$$\begin{aligned} J(W_l) &= \left\| \tilde{w}_l - w_l^t \right\|_2^2 \\ &= (\tilde{w}_l)^2 \cdot \left(|I_{\Delta_l^p}| + |I_{\Delta_l^n}| \right) \\ &\quad - 2 \cdot \tilde{w}_{li} \cdot \left(\textstyle\sum_{i \in I_{\Delta_l^p}} \tilde{w}_{li} + \sum_{i \in I_{\Delta_l^n}} \tilde{w}_{li} \right) + C. \end{aligned} \quad (5)$$

Where $I_{\Delta_l^p} = \{i \mid \tilde{w}_{li} > \Delta_l^p\}$, $I_{\Delta_l^n} = \{i \mid \tilde{w}_{li} < \Delta_l^n\}$. Δ_l^p and Δ_l^n are independent together. $C = \sum_{i=1}^{n} |\tilde{w}_{li}|^2$ is an independent constant. It is evident that solving for both Δ_l^p and Δ_l^n simultaneously leads to a large solution space. Therefore, to accelerate the training process, previous methods typically rely on empirical interval divisions to directly obtain the threshold values, as follows,

$$\Delta_l^{p*} \approx 0.7 \cdot \frac{1}{|I^p|} \cdot \textstyle\sum_{i \in I^p} |\tilde{w}_{li}|, \ \Delta_l^{n*} \approx 0.7 \cdot \frac{1}{|I^n|} \cdot \sum_{i \in I^n} |\tilde{w}_{li}|. \quad (6)$$

The premise of the empirical interval divisions assumes that W_l values are generated from either a uniform or normal distribution. However, as shown in the upper-right corner of Fig. 1(a), the weight distribution of the GCN exhibits a bimodal pattern. Therefore, we assume it to be a combination of two normal distributions, denoted as N_p and N_n. We divide the upper and lower bounds into $\pm 0.7E_p$ and $\pm 0.7E_n$, respectively, as indicated by the blue dashed lines in the lower-left corner of Fig. 1(a).

Therefore, we can get ternary weights from the full precision weights. Moreover, to ensure a stable training process, we introduce a normalization factor, which is typically set to the scale of the weight matrix. Finally, we employ quantization-aware training [19] to train GCN model. That simplifies matrix multiplication and further reduces memory consumption.

B. Architecture Overview

The overall architecture of the ATE-GCN accelerator is shown in Fig. 4. It includes a PCIe DMA, an interconnect bus for data switching, a memory controller, a controller, input buffers, unified PE arrays, a post-processing module, and intermediate result/output buffers. In the figure, the blue arrows represent data paths, while the green arrows represent control signal paths.

Data is transferred from the PC to the FPGA accelerator via PCIe, including graph information and weights. Benefiting from ternary quantization, users are able to pre-load weight parameters into the on-chip memory using the *.coe* file. The off-chip storage can be either high-bandwidth memory (HBM) or multi-channel DRAM. In this work, we use the VCU-118 FPGA board for development, and therefore, we select dual-channel DDR with an 80-bit width per channel to meet the bandwidth requirements. The data is transferred through the interconnect bus to buffers composed of different types of on-chip memory, including the edge buffer, feature buffer, and weight buffer. Each buffer is reused by the PE array. The controller generates control signals to manage data flow into the rows and columns of the PE array. It also manages the different computation mode switching in the PE array, including Ternary mode and SpMM mode. The post-processing

TABLE I: Structure and data density of GCN on three typical datasets

Datasets		Cora (CR)	Citeseer (CS)	Pubmed (PB)
Structure	Nodes	2,708	3,327	19,717
	Edges	10,556	9,104	88,648
	Features	1,433	3,703	500
	Classes	7	6	3
Data density	A	0.18%	0.11%	0.028%
	h	1.27%, 78.0%	0.85%, 89.1%	10.0%, 77.6%
	W	100%, 100%	100%, 100%	100%, 100%

module handles W_l multiplication, reduction and the activation function (ReLU). After processing, the data is stored in the output/result buffer, where a control signal determines whether it is written back to the DDR, loaded into the input buffer for subsequent computation, or temporarily stored as partial results for future calculations.

C. Unified PE Array

1) Different computation modes: As shown in Fig. 4(b), the unified PE array operates in different modes, including Ternary mode and SpMM mode. The distinction between these modes is derived from our analysis of the data structures in a two-layer GCN computation on three typical datasets, as presented in Table I. First, we select the computation order as defined in Eq. 2 to reduce data operation. For the computation of $W^k \otimes h_u^k$, since W is the result of ternary quantization, we design the Ternary mode to compute these phases. For the subsequent computation, as the adjacency matrix (A) is sparse and remains unchanged during the process, we implemented the SpMM mode for efficient computation.

Ternary Mode: As shown in Fig. 4(b), the DSPs in the PE array are cascaded through PCIN and PCOUT to perform accumulation. The multiplication with W is completed in the post-processing multiplier. Since W is quantized to $\{-W_l, 0, +W_l\}$, which we convert to $W_l\{-1, 0, 1\}$. In the PE array, the pre-adder and ALU units are employed to add neighboring elements and accumulate the results in a single computation step. To implement this operation, {-1, 0, 1} is encoded as {2'b11, 2'b00, 2'b10}, where the MSB indicates if the weight is zero and the LSB determines the sign. Therefore, the data loaded into the PE RAM first passes through a selector controlled by $W[1]$, which is used to manage operations with a weight of zero. The sign is determined using an XOR operation. It is important to note that the example in Fig. 4(b) is only an abstract representation of the inputs. In the actual implementation, we also leverage DSP-Packing [12] to further improve throughput in ternary mode. More detailed configurations will be discussed in Section III-C2.

SpMM Mode: We perform the computations by the inner product. Although this method requires repeated access to the right matrix, data density analysis reveals that the right matrix is dense and exhibits well continuous data access characteristics. We designed a read-conflict-free mechanism by leveraging an input buffer and a selector, similar to the approach in [20], shown in Fig. 4(b). The post-processing module completes the remaining accumulation operations. With support for both DSP-packing and a high-frequency cascade design, the PE array can operate at double the system clock frequency (over 500MHz).

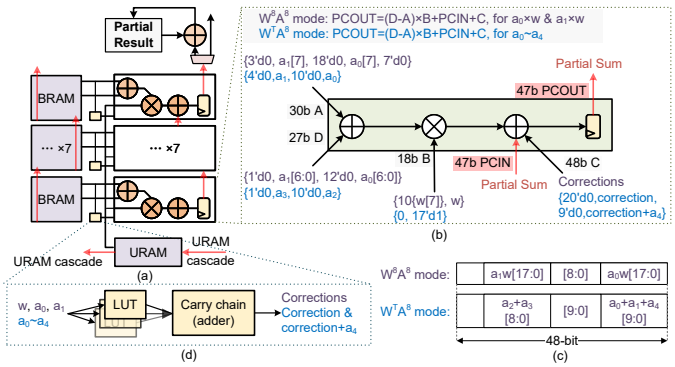

Fig. 5: (a) The microarchitecture of PE. (b) The detailed data path of a DSP under different operating modes. (c) The output data bit width and offset. (d) The CLB-based correction unit.

2) Microarchitecture design of PE: To achieve the PE operating frequency of twice other components, we adopt the cascade interconnect feature described in Sec II-B2. Moreover, to further boost throughput, we integrate DSP-Packing [12]. As shown in Fig 5(a), 9 DSPs are cascaded, corresponding to 9 BRAMs and 1 URAM, similar with [17]. Each BRAM provides 32-bit of input through its A and B ports, while the 72-bit output from the URAM is divided into 8-bit segments and distributed across 8 DSPs. This design is compatible with both ternary mode and SpMM mode. The difference between these two computation modes lies in the data width: W^8A^8 for SpMM and W^TA^8 for ternary mode. We configure DSP to perform both modes under $PCOUT = (D-A)B + PCIN + C$, avoiding the high fan-out problem caused by data packing on the A port input [21], as shown in Fig. 5(b). In W^8A^8 mode, a single DSP performs the computations for $a_0 \times w$ and $a_1 \times w$, with the C port used for the correction term input. In W^TA^8 mode, a single DSP performs the operations $a_0 + a_1 + a_4$ and $a_2 + a_3$,with the C port used for a_4 input. The packed partial sums are split and accumulated in post-processing.

Although DSP-packing can increase DSP throughput, it introduces errors during cascading and signed multiplication [12]. For cascading errors, we have reserved sufficient offset at the input to ensure that when 9 DSPs are cascaded for partial sum accumulation, their results do not interfere with each other, as shown in Fig. 5(c). For signed multiplication errors, a correction unit is required. However, previous error correction units have hindered the DSP from operating at higher frequencies. Therefore, we suggest utilizing Unsigned-Unsigned type multiplication, complemented by a correction unit, to replace Signed-Signed type multiplication. For a signed data X (is represented in bit-wise as $X = \{x_{[n-1]} \cdots x_{[1]} x_{[0]}\}$) is conversing into unsigned counterpart, denoted as X', that is described as follows:

$$\begin{aligned} X &= -x_{[n-1]} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_{[i]} \cdot 2^i \\ &= -x_{[n-1]} \cdot 2^n + \sum_{i=0}^{n-1} x_{[i]} \cdot 2^i = -x_{[n-1]} \cdot 2^n + X'. \end{aligned} \tag{7}$$

Therefore, when the inputs to the multiplier are a and w, the

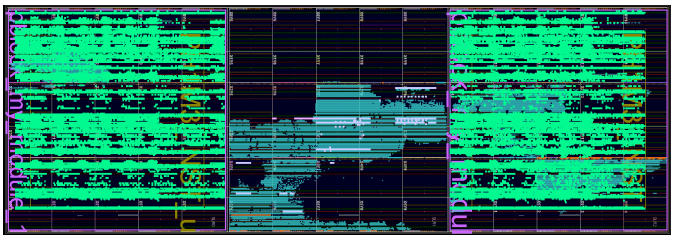

Fig. 6: Layout of ATE-GCN on the VCU118.

TABLE II: Resource utilization on VCU118 (UltraScale+ XCVU9P)

Resource	LUT	FF	BRAM	URAM	DSP
Used	276,612	581,802	871.5	432	3,894
Available	1,182,240	2,364,480	2,160	960	6,840
Utilization	23.4%	24.6%	40.35%	45.0%	56.93%

computation process of $a \times w$ can be transformed as follows:

$$\begin{aligned} a \times w &= (-a_{[n-1]} \cdot 2^n + a')(-w_{[n-1]} \cdot 2^n + w') \\ &= a'w' - w_{[n-1]} \cdot a' \cdot 2^n - a_{[n-1]} \cdot w' \cdot 2^n + a_{[n-1]} \cdot w_{[n-1]} \cdot 2^{2n}. \end{aligned} \tag{8}$$

Therefore, the corresponding correction term can simply be added through the C port of the DSP. The correction input is computed using the LUTs and carry chain within the CLB units adjacent to the DSP, as shown in Fig. 5(d).

IV. Experiments

A. Implementations

We benchmark the Asymmetrical Ternary Weight strategy with Ternary Weight strategy and Full precision on three typical datasets (Cora, Citeseer, and Pubmed, shown in Table I) with the software framework PyTorch Geometric (PYG) [22]. For a fair comparison, we keep the same network architecture (2 layers), learning rate, training epoch (maximum 200), training strategy (early stop), and optimizer.

We implement the ATE-GCN accelerator in Verilog HDL on an AMD Virtex UltraScale+ FPGA VCU118 Evaluation Kit. To meet the design requirements of twice PE array frequency, we generate explicit physical location mappings for DSP, URAM, BRAM, CLB, and CLB-M components to enforce strict timing constraints (1.5 ns). We measure the resulting frequency of the mapped design and interconnect utilization metrics to assess the extent of wiring reduction. Fig. 6 displays the layout of ATE-GCN, highlighting the PE arrays in green. The FPGA on the VCU118 is a multi-die device with multiple Super Logic Regions (SLR). Therefore, we choose to distribute the PE arrays across both sides (SLR0 and SLR2), with all data converging to SLR1 [23]. The PE arrays in our design are engineered to achieve a maximum frequency of 625MHz. For computational efficiency, we adjust the engine to 500MHz and set the peripheral circuits at 250MHz to balance power consumption and performance. The power consumption of FPGA core and hardware resource utilization (shown in Table II) are obtained after synthesis and implementation in Vivado 2022.2. What is more, we also measured the power consumption of the entire FPGA board with an electricity meter.

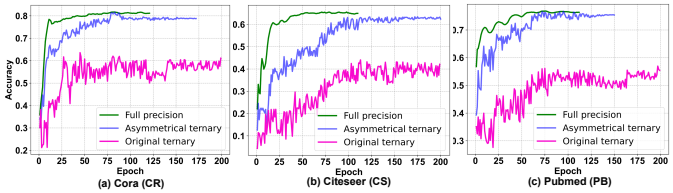

Fig. 7: Accuracy over training epochs CR, CS, and PB.

TABLE III: Accuracy across different datasets at various precision

Precisions	Cora (CR)	Citeseer (CS)	Pubmed (PB)
Full precision	81.16%	64.95%	76.36%
Original Ternary	61.12%(-20.04%)	42.12%(-22.83%)	55.41%(-20.95%)
Asymmetrical Ternary	**78.79%(-2.37%)**	**62.42%(-2.53%)**	**75.38%(-0.98%)**

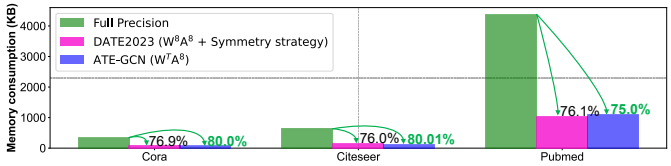

Fig. 8: The ternary weights effectively reduce the memory consumption. The percentages represent the memory savings compared to full precision.

TABLE IV: Performance and energy consumption comparison of CPU, GPU, and ATE-GCN (hidden channels=128)

Evaluation metrics		**CPU**	**GPU**	**ATE-GCN**
Latency	Cora	8.61ms	700.2μs	**46.04μs**
	Citeseer	9.98ms	1.13ms	**75.48μs**
	Pubmed	214.02ms	1.89ms	**606μs**
ATE-GCN speedup ratio (avg.)		224.13×	11.1×	—
Power	Core	47.2W	74.12W	**11.77W**
	Board	—	254W	**45.93W**
ATE-GCN improvement ratio of energy consumption (avg.)		898.82×	69.9×	—

B. Evaluation of Asymmetrical Ternary Quantization

The validation accuracy curves of different strategies across all training epochs on three typical datasets are illustrated in Fig. 7. It can be observed that the original ternary quantized model is underfitted. Although the asymmetric ternary quantization exhibits unstable training loss in the early stages, it converges as the number of epochs increases. Additionally, the asymmetric ternary quantization strategy results in an average accuracy loss of less than 2% across three datasets, as shown in Table III. In contrast, the original ternary quantization strategy averagely shows a much larger accuracy drop of -20%.

Ternary quantization not only simplifies multiplication operations in GCN but also reduces memory consumption. This is critical for FPGA-based accelerators, as it allows more data to be cached in on-chip memory. As shown in Fig 8, we compare the memory savings of ternary quantization with previous optimization strategies [24]. The sparse format is unified as COO, with the baseline being the memory usage of full-precision data, and the comparison W^8A^8 combined with symmetric matrix compression. It can be observed that ternary quantization results in an average memory savings of 78.36% and 8.64%, respectively.

C. Cross Platform Comparison

We compare performance and energy consumption on two baseline platforms: CPU (Intel i7-12700F) and GPU (Nvidia

TABLE V: Comparison with SOTA FPGA-based GCN accelerators on the Cora (CR), CiteSeer (CS), and PubMed (PB) datasets

Accelerator	FPGA Board	Frequency	DSP	Optimization	Latency (μs)			Norm. DSP		
					CR	CS	PB	CR	CS	PB
FP-GNN [8]	VCU128	225 MHz	8704	Unified processing module, Adaptive Graph Partition strategy	36.0	61.8	539	36.0	61.8	539
DATE2023 [24]	Stratix10 MX	220 MHz	3960	Symmetrical matrix, INT8 quantization	62.77	100.37	882	28.56	45.66	401.28
ATE-GCN	VCU118	**250 MHz (PE 500 MHz)**	3891	Ternary quantization, High frequency PE array	46.04	75.48	606	**20.58**	**33.74**	**270.91**

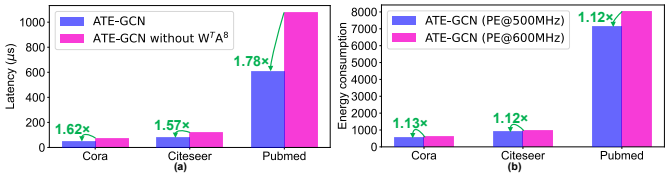

Fig. 9: The ablation Study. (a) Effectiveness of ternary-weights-based design. (b) Comparison of energy consumption at different frequencies.

RTX4090), as shown in Table IV. The power of the CPU core is collected from the Intel Power Gadget. The power of the GPU Core and the board (including the core) are measured by the GPU-Z. ATE-GCN achieves up to 224.13× and 11.1× improvement in end-to-end latency in CPU and GPU implementations, respectively. Due to the presence of many peripherals on the motherboard, it is difficult to obtain accurate CPU-board power consumption. As a result, we use core power consumption as the standard for evaluating energy consumption. Overall, ATE-GCN achieves lower energy consumption by 898.82× and 69.9× on average compared to CPU and GPU, respectively.

We further compare ATE-GCN with SOTA FPGA-based GCN accelerators, as shown in Table V. Due to the ternary quantization and high operating frequency PE array design, ATE-GCN uses the least amount of DSP resources while delivering better performance (1.11× on average). Moreover, we normalize by the number of DSPs, the major computation resources. ATE-GCN achieves average performance 1.86× and 1.41× faster than FP-GNN and DATE2023 [24], respectively.

D. Ablation Study

We conducted ablation studies on three datasets to quantitatively evaluate the gains brought by the ternary quantization-based PE design. Fig. 9 compares the latency between using the W^8A^8 computation mode throughout the entire process and incorporating ternary quantization with the W^TA^8 mode. The data shows that introducing the W^TA^8 mode results in an average speedup of 1.67×. This improvement is due to the fact that a single DSP can process more operated data in W^TA^8 mode. Furthermore, we attempt to increase the PE frequency to 600 MHz, but this resulted in higher power consumption (15.73W), which increased energy consumption.

V. Conclusion

This paper presents ATE-GCN, which accelerates GCNs through an asymmetric ternary quantization strategy combined with a high-frequency unified PE array co-designed. To address the bimodal distribution of GCN weights, we propose an asymmetric interval division that significantly reduces the accuracy loss. Furthermore, we design a hardware accelerator architecture to leverage the advantages of ternary quantization. By efficiently utilizing DSP and related hardware resources, we implemente a unified PE array design that operates at a high frequency (over 500 MHz). ATE-GCN achieves a speedup of 224.13× and 11.1× over CPU and GPU implementations, respectively, and delivers 1.63× higher DSP efficiency compared to SOTA FPGA accelerators. To the best of our knowledge, this is the first ternary quantization deployment for GCN models along with the corresponding accelerator design.

References

[1] B. Liu *et al.*, "Ternary Weight Networks," in *ICASSP*, 2023.
[2] Z. He *et al.*, "FTW-GAT: An FPGA-Based Accelerator for Graph Attention Networks With Ternary Weights," *IEEE Trans. Circuits Syst. II-Express Briefs*, 2023.
[3] R.-J. Zhu *et al.*, "Scalable matmul-free language modeling," *arXiv*, 2024.
[4] T. N. Kipf *et al.*, "Semi-Supervised Classification with Graph Convolutional Networks," in *ICLR*, 2017.
[5] L. Zhang *et al.*, "A Feature-Importance-Aware and Robust Aggregator for GCN," in *CIKM*, 2020.
[6] J. Ding *et al.*, "Asymmetric Ternary Networks," in *ICTAI*, 2017.
[7] S.-E. Chang *et al.*, "Mix and Match: A Novel FPGA-Centric Deep Neural Network Quantization Framework," in *HPCA*, 2021.
[8] T. Tian *et al.*, "FP-GNN: Adaptive FPGA accelerator for Graph Neural Networks," *Futur. Gener. Comp. Syst.*, 2022.
[9] M. Véstias *et al.*, "Efficient Design of Low Bitwidth Convolutional Neural Networks on FPGA with Optimized Dot Product Units," *ACM T. Reconfigurable Technol. Syst.*, 2022.
[10] D. Gerlinghoff *et al.*, "Table-Lookup MAC: Scalable Processing of Quantised Neural Networks in FPGA Soft Logic," in *FPGA*, 2024.
[11] M. Molina *et al.*, "Power-Efficient Implementation of Ternary Neural Networks in Edge Devices," *IEEE Internet Things J.*, 2022.
[12] J. Sommer *et al.*, "DSP-Packing: Squeezing Low-precision Arithmetic into FPGA DSP Blocks," in *FPL*, 2022.
[13] Y. Bai *et al.*, "FET-OPU: A Flexible and Efficient FPGA-Based Overlay Processor for Transformer Networks," in *ICCAD*, 2023.
[14] J. Zhang *et al.*, "Uint-Packing: Multiply Your DNN Accelerator Performance via Unsigned Integer DSP Packing," in *DAC*, 2023.
[15] M. Li *et al.*, "A Statistical Characterization of Attentions in Graph Neural Networks," in *ICLR Workshop*, 2019.
[16] W. Huang *et al.*, "BiLLM: Pushing the Limit of Post-Training Quantization for LLMs," in *ICML*, 2024.
[17] A. Samajdar *et al.*, "Scaling the Cascades: Interconnect-Aware FPGA Implementation of Machine Learning Problems," in *FPL*, 2019.
[18] J. Li *et al.*, "GCNAX: A Flexible and Energy-efficient Accelerator for Graph Convolutional Neural Networks," in *HPCA*, 2021.
[19] I. Hubara *et al.*, "Accurate post training quantization with small calibration sets," in *ICML*, 2021.
[20] B. Liu *et al.*, "Towards High-Bandwidth-Utilization SpMV on FPGAs via Partial Vector Duplication," in *ASP-DAC*, 2023.
[21] J. Wang *et al.*, "High-Performance Mixed-Low-Precision CNN Inference Accelerator on FPGA," *IEEE Micro*, 2021.
[22] M. Fey *et al.*, "Fast Graph Representation Learning with PyTorch Geometric ," in *ICLR*, 2019.
[23] S. Kashani *et al.*, "A 475 MHz Manycore FPGA Accelerator for RTL Simulation," in *FPGA*, 2024.
[24] G. R. Nair *et al.*, "FPGA Acceleration of GCN in Light of the Symmetry of Graph Adjacency Matrix," in *DATE*, 2023.