






# EQ-ViT: Algorithm-Hardware Co-Design for End-to-End Acceleration of Real-Time Vision Transformer Inference on Versal ACAP Architecture

Peiyan Dong\*, Jinming Zhuang\* 

Zhuoping Yang , Shixin Ji , Yanyu Li, Dongkuan Xu, Heng Huang 

Jingtong Hu , *Senior Member, IEEE*, Alex K. Jones , *Fellow, IEEE*, Yiyu Shi , *Senior Member, IEEE*,

Yanzhi Wang, *Senior Member, IEEE*, and Peipei Zhou , *Senior Member, IEEE*

**Abstract**— While Vision Transformers (ViTs) have shown consistent progress in computer vision, deploying them for real-time decision-making scenarios ( $< 1$  ms) is challenging. Current computing platforms like CPUs, GPUs, or FPGA-based solutions struggle to meet this deterministic low-latency real-time requirement, even with quantized ViT models. Some approaches use pruning or sparsity to reduce model size and latency, but this often results in accuracy loss. To address the aforementioned constraints, in this work, we propose EQ-ViT, an end-to-end acceleration framework with novel algorithm and architecture co-design features to enable real-time ViT acceleration on AMD Versal Adaptive Compute Acceleration Platform (ACAP). The contributions are four-fold. **First**, we perform in-depth kernel-level performance profiling & analysis and explain the bottlenecks for existing acceleration solutions on GPU, FPGA, and ACAP. **Second**, on the hardware level, we introduce a new spatial and heterogeneous accelerator architecture, EQ-ViT architecture. This architecture leverages the heterogeneous features of ACAP, where both FPGA and artificial intelligence engines (AIEs) coexist on the same system-on-chip (SoC). **Third**, On the algorithm level, we create a comprehensive quantization-aware training strategy, EQ-ViT algorithm. This strategy concurrently quantizes both weights and activations into 8-bit integers, aiming to improve accuracy rather than compromise it during quantization. Notably, the method also quantizes nonlinear functions for efficient hardware implementation. **Fourth**, we design EQ-ViT automation framework to implement the EQ-ViT architecture for four different ViT applications on the AMD Versal ACAP VCK190 board, achieving accuracy improvement with 2.4%, and average speedups of 315.0x, 3.39x, 3.38x, 14.92x, 59.5x, 13.1x over computing solutions of Intel Xeon 8375C vCPU, Nvidia A10G, A100, Jetson AGX Orin GPUs, and AMD ZCU102, U250 FPGAs. The energy efficiency gains are 62.2x, 15.33x, 12.82x, 13.31x, 13.5x, 21.9x.

This work is supported in part by NSF awards #2213701, #2217003, #2133267, #2122320, #2324864, #2324937, #2328972 and NIH award #R01EB033387.

\* Peiyan Dong and Jinming Zhuang are co-first authors and have equal contributions.

Peiyan Dong, Yanyu Li, and Yanzhi Wang are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, USA (e-mail: dong.pe@northeastern.edu; li.yanyu@northeastern.edu; yanz.wang@northeastern.edu).

Jinming Zhuang, Zhuoping Yang, Shixin Ji, and Peipei Zhou are with the School of Engineering, Brown University, 345 Brook Street, Providence, RI 02912, USA (e-mail: jinming\_zhuang@brown.edu; zhuoping\_yang@brown.edu; shixin\_ji@brown.edu; peipei\_zhou@brown.edu).

Dongkuan Xu is with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695, USA (e-mail: dxu27@ncsu.edu).

Heng Huang is with the Department of Computer Science, University of Maryland, College Park, MD 20742, USA (e-mail: heng@umd.edu).

Jingtong Hu is with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261 USA (e-mail: jthu@pitt.edu).

Alex Jones is with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244, USA (e-mail: akj@syrr.edu).

Yiyu Shi is with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: yshi4@nd.edu).

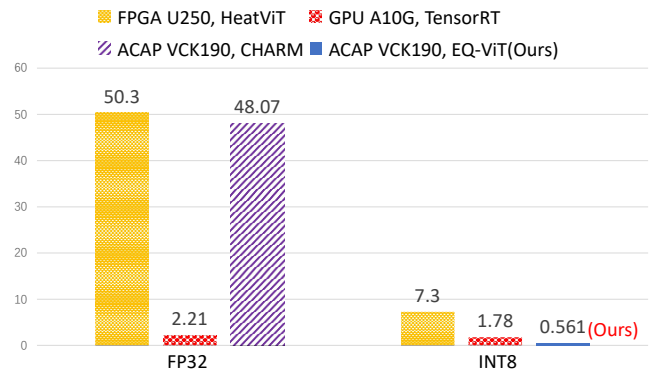


Fig. 1: E2E latency comparison for DeiT-T (FP32, INT8, batch size = 6) by using HeatViT on U250 FPGA, TensorRT on A10G GPU, CHARM on Versal ACAP VCK190 and EQ-ViT (ours) on ACAP VCK190.

**Index Terms**—design for space exploration, embedded systems, FPGA, hardware/software co-design, high-level synthesis, modeling, performance optimization, reconfigurable logic

## I. INTRODUCTION

Vision Transformers (ViTs) [1]–[3] have shown remarkable versatility in a broad range of application domains, including computer vision (e.g., image classification [1], [3], object detection [4], [5], image processing [6], and video understanding [7]), and in complex scenarios that involve multimodal data. Many networks [1], [8]–[10] use ViTs as the backbone [8], [9] and show superior transferability to various downstream tasks with minor fine-tuning.

**Low-latency real-time application scenarios.** Adopting ViT inference as a key chain for low-latency real-time decision-making usually requires stringent latency requirements. For example, in autonomous driving scenarios with a 120 km/h speed, 1 ms latency corresponds to 3 cm between a vehicle and a static object or 6 cm between two moving vehicles [11]. In such a life-critical system, deterministic low latency ( $< 1$  ms) is the first-class design citizen. European Organization for Nuclear Research (CERN) collaborates with autonomous driving software company Zenseact to apply CERN’s decision-making algorithm acceleration on FPGA at microsecond level to help avoid accidents in self-driving cars [12]. Such latency ( $< 1$  ms) is required in broader scenarios, including edge and cloud applications. **On the edge**, for example, radio access networks (RAN) [13] support interactive streaming media [14], augmented reality/virtual reality (AR/VR) [15], [16], robot systems control [17], online error detection in the manufacturing industry [18], and industrial IoT 4.0 [19]. RAN stack operates in low-latency at a transmission time interval of 1 ms or less (based on 5G standards). Thus,

TABLE I: Hardware specification comparisons on peak performance for data types FP32 and INT8, on-chip memory size, off-chip bandwidth (BW), TDP among AMD FPGA U250, Nvidia GPU A10G, Nvidia GPU Jetson AGX Orin, and AMD Versal ACAP VCK190.

Hardware Spec.	Tech. Node	FP32	INT8	Off-chip BW	On-chip Mem.	Off-chip Mem.	TDP
AMD FPGA U250	16nm	1.2T	6.95T	77GB/s	53MB	16GB	225W
Nvidia GPU A10G	8nm	35T	140T	600GB/s	14MB	24GB	300W
Nvidia GPU Jetson Orin	8nm	5.3T	85T	204GB/s	6MB	64GB	60W
AMD ACAP VCK190	7nm	6.4T	102T	25GB/s	33MB	8GB	<180W

it has to make control decisions at each millisecond [13]. In AR/VR, the latency requirement is <1 ms as the visual reaction time for human expected events is only around 1 ms [20]. **In the cloud**, to guarantee the quality of service, deep learning-based inference for cloud services in Microsoft Bing Search [21], Microsoft Azure Cloud [22], [23], and Google Cloud [24]–[26], all have a single-digit millisecond latency budget to process. Powered by next-generation cellular networks with 5G or 6G standard [13], optical interconnection network [27], and optical chiplet [28], [29] technology, the latency requirement will be more stringent. **Acceleration solutions that meet certain end-to-end (E2E) inference latency requirements and optimize the overall system energy efficiency, i.e., performance per watt, are desired.**

However, existing works fail to fulfill such stringent low-latency requirements, hindering the ViT deployment in low-latency application scenarios. We measure the E2E low batch inference latency for the representative ViT model DeiT-T [2] using the state-of-the-art (SOTA) acceleration frameworks on the FPGA and GPU, including HeatViT [30] on AMD U250 FPGA, and TensorRT [31] on Nvidia A10G GPU. As shown in Figure 1, in terms of E2E inference latency under single-precision floating-point (FP32) precision, U250 FPGA takes 50.3 ms, which far exceeds the low-latency real-time requirement, e.g., <1 ms, while A10G GPU takes 2.21 ms. We can achieve a lower inference latency by quantization [32] and deploying the 8-bit integer (INT8) inference on U250 FPGA and A10G GPU. Then the inference latency reduces to 7.3 ms on U250 FPGA and 1.78 ms on A10G GPU.

Based on the requirements of deterministic E2E inference latency and the initial profiling results of existing solutions, several research questions arise: (a) *What are the limitations of the existing acceleration platforms in satisfying the low-latency demands?* (b) *With quantization optimization, do we have a better computing solution to achieve lower latency than FPGAs and GPUs?*<sup>1</sup> (c) *If so, how to achieve that?* (d) *Can we also improve the accuracy with integer quantization?*

Our answer to the second question is “Yes”. We propose EQ-ViT architecture and our implemented EQ-ViT design on AMD Versal ACAP VCK190 achieves a latency as low as 0.56

<sup>1</sup>Note that <1 ms latency requirement in the example discussion is for illustration purposes. The latency requirements differ across various application scenarios. We desire a solution that achieves lower latency than GPUs and FPGAs under the same throughput requirement or achieves higher throughput (or energy efficiency) than GPUs and FPGAs under the same latency requirement. In this paper, we discuss such a solution EQ-ViT.

ms, which has 3.2x latency improvement over A10G GPU and 13.1x over U250 FPGA. However, achieving latency as low as 0.56 ms on Heterogeneous Versal ACAP SoC involves a lot of design efforts. To ease the programming efforts, we propose EQ-ViT design automation framework to perform design space exploration and automatic code generation to facilitate the implementation. In addition, we propose EQ-ViT algorithm to improve the inference accuracy after the INT8 quantization and EQ-ViT algorithm-hardware co-design to meet the hardware constraints without hurting the algorithm accuracy. Our contributions are summarized below:

- **Detailed Profiling and Bottleneck Analysis:** To understand the performance constraints of existing solutions, we perform in-depth kernel-level performance profiling of ViTs on FPGA, GPU, and ACAP in Section II. Based on the bottlenecks for existing solutions, we propose our solution principles.
- **EQ-ViT Accelerator & Mapping:** We propose a novel spatial and heterogeneous accelerator template and programming mapping solution to take advantage of the ACAP heterogeneous features: the co-existence of FPGA and AIE vector cores on the same system-on-chip in Section IV. Our accelerator architecture features multiple spatial accelerators to improve the AIE core utilization and fine-grained pipeline to overlap the execution time of the accelerators that run on the FPGA and AIEs of the ACAP.
- **EQ-ViT Algorithm and Algorithm-Hardware Co-Design:** On the algorithm level, we develop a full quantization-aware training strategy, EQ-ViT algorithm, to quantize both weights and activations into 8-bit integers in Section V. This method improves accuracy on all four different ViT models. More importantly, our proposed EQ-ViT algorithm-hardware co-design quantizes the nonlinear functions with algorithm optimization and realizes efficient hardware implementation for Softmax and GeLU.
- **EQ-ViT Automation and System Implementation:** We design EQ-ViT automation framework to implement the EQ-ViT architecture for four different ViT models on the AMD Versal ACAP VCK190 board. Experiments in Section VI show EQ-ViT achieves accuracy improvement with 2.4% and average speedups of up to 315.0x, 3.39x, 3.38x, 14.93, 59.5x, 13.1x over computing solutions of Intel Xeon 8375C vCPU, A10G, A100, Jetson AGX Orin GPUs, and AMD ZCU102, U250 FPGAs.
- **EQ-ViT Generality Discussion:** We discuss how EQ-ViT mapping framework can be applied to other architecture, e.g., FPGA and GPU, to improve performance in Section VII. We further discuss the microarchitecture insights, i.e., what role reconfigurability plays in future heterogeneous architecture.

## II. BOTTLENECK ANALYSIS AND PROPOSED SOLUTION

In this section, we first explain the performance bottlenecks of the current solutions on FPGA, GPU, and ACAP. Then we discuss our proposed design principles.

First, **FPGAs are mainly constrained by the limited computation resources.** Table I indicates that AMD FPGA

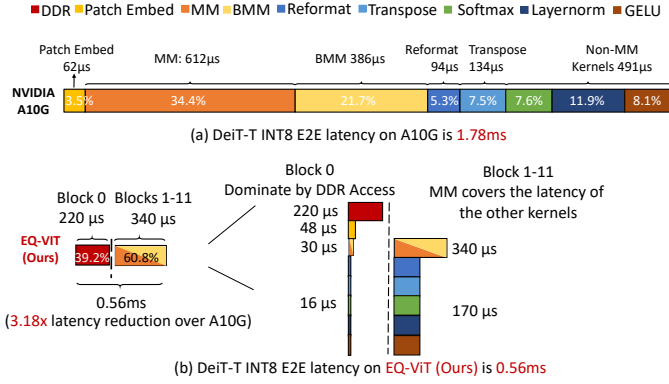


Fig. 2: E2E inference latency comparison of using TensorRT on NVIDIA A10G GPU and using EQ-ViT (ours) on AMD Versal VCK190 ACAP for the representative ViT model DeiT-T with INT8 precision when batch size = 6.

U250 (Ultrascale+, 16nm fabrication) has the lowest peak performance among the three hardware platforms, at 1.2 TFLOPS for FP32 and 6.95 TOPS for INT8 under 250 MHz. When transitioning from FP32 to INT8, the E2E latency decreases from 50.3 ms to 7.3 ms. However, both cases are computation-bound and latency can not be further reduced because of limited computation resources from DSP/LUT in FPGA.

GPUs have abundant computation cores, e.g., NVIDIA introduces Tensor Cores since Volta architecture. Table I reveals that GPU A10G (Ampere architecture, 8nm fabrication) boasts the highest peak performance at 35 TFLOPS for FP32 and 140 TOPS for INT8. Tools like TensorRT simplify inference streamline through methods such as quantization. However, despite the powerful hardware, Figure 1 shows that the E2E latency on GPU A10G is 2.21ms for FP32 and 1.78ms for INT8. This results in a modest 1.24x E2E improvement, significantly smaller than the theoretical peak computation performance improvement from FP32 to INT8 (4x, calculated as  $140T/35T$ ). To understand the performance bottleneck, we utilize NVIDIA Nsight System [33] and depict the kernel-level time breakdown for INT8 in Figure 2. We identify the following performance constraints for using TensorRT on the GPU: **1** Low Tensor Cores utilization for INT8 MM kernels. Although MM kernels constitute 34.4% of the total runtime, their effective throughput is 23 TOPS, representing only 16% utilization of the peak INT8 computation performance for GPU A10G. **2** TensorRT adopts an implicit quantization policy, which leads to BMM computing in FP32, not in INT8. Quantization enables MM and batch-MM (BMM) to compute in INT8 for higher throughput. However, according to the NVIDIA Nsight Compute kernel-level profiling report, BMM kernels compute in FP32. This is related to the implicit quantization strategy applied by TensorRT [34], which will quantize one kernel only when this kernel runs faster in INT8. Otherwise, TensorRT will assign a higher precision to this kernel, FP32, by default. Despite having only 1/6 of the total operations of MM kernels, BMM kernels contribute to 21.7% of the total runtime. We calculate their effective throughput as 6.3 TFLOPS, which is 18% of the peak FP32 computation performance for A10G. **3** The

data type conversion between FP32 and INT8 consumes non-negligible GPU cycles. MM kernels are processed in INT8 mode using NVIDIA Tensor Cores, while other kernels use FP32 mode with NVIDIA CUDA Cores. Data type conversions between FP32 and INT8, known as Reformat, are introduced. This operation is significant, accounting for 5.3% of the E2E latency. **4** The nonlinear kernels take significant GPU cycles. Non-MM kernels, such as Softmax, GeLU, and LayerNorm, collectively contribute 27.6% of the total, despite their operations being only 1.5% of MM kernels. This is due to these kernels involving special functions, such as exponent functions, division, and square root.

AMD Versal ACAP is a heterogeneous SoC, featuring ARM CPUs, FPGA, and AIE vector cores. The AIEs support several data types, including FP32, INT16, and INT8 [35]. ACAP integrates the aspects of both domains, that is, FPGA for reconfigurability and AIEs for abundant computation cores. We deployed the DeiT-T model FP32 version on the VCK190 board using CHARM [36], a SOTA deep learning inference framework on ACAP architecture. Figure 1 illustrates that CHARM has an E2E latency of 48.07 ms, which is 27x slower than using TensorRT on GPU A10G under FP32. This performance lag is mainly due to the significant load/store of the feature data from/to off-chip memory, caused by the FP32 model’s size exceeding the VCK190 on-chip storage capacity of 33MB. Quantizing the model into INT8 allows it to fit on-chip. However, without careful design, ACAP acceleration may face similar limitations (from **1** to **4**) as A10G, and potentially worse due to VCK190’s limited 4.2% off-chip BW compared to A10G. This leads to the following question: *How can we optimize latency for INT8 ViT on ACAP, given its high computation intensity but constrained off-chip BW?*

**Proposed Design Principles.** We propose EQ-ViT to optimize latency for INT8 ViT, which circumvents all constraints from **1** to **4** typically encountered in GPU. The key idea of EQ-ViT is to design multiple heterogeneous MM accelerators on AIEs, design other non-MM kernels on FPGA, and overlap the execution of kernels running on AIEs and FPGA. Figure 2(b) demonstrates the kernel runtime overlapping in EQ-ViT. However, new challenges appear. **First**, we need to enable explicit INT8 computation for BMMs and achieve high computation utilization for both MMs and BMMs. The computation and communication requirements of MMs and BMMs are different. Overlapping these two types of kernels can improve both computation and communication utilization. **Second**, we need to design efficient accelerators for nonlinear kernels (Softmax, GeLU, and LayerNorm). **Third**, we need to leverage the flexible on-chip memory architecture provided by FPGA on ACAP to enable data forwarding in adjacent kernels and further reduce off-chip memory access. **Fourth**, we need to carefully overlap the execution time and optimize workload partitioning and resource partitioning jointly, for utilization optimization, high throughput, and low latency. **Fifth**, we need analytical models to optimize the E2E latency under computation resource and communication bandwidth constraints. **Sixth**, we need to keep the accuracy after quantization and, if possible, enhance it.

TABLE II: Architecture and algorithm features of EQ-ViT and comparisons with prior works.

Prior Works	Computing Platform		Accelerator Features						Algorithm & Algorithm-Hardware Co-Design Feat.		
	Board Type	GOPS/(GB/s)	Multi Spatial Accelerators	Hardware Specialization	On-chip Forwarding	Fine-grained Pipeline	Explicit Quant.	Compute Util.	Activation-aware Quant.	Nonlinear Quant.	Accuracy Gain
TensorRT [31]	GPU	-	×	×	×	×	×	Low	-	-	-
Herald [37]	ASIC	-	✓	×	×	×	-	High	-	-	-
MAGMA [38]	ASIC	-	✓	✓	×	×	-	High	-	-	-
ViA [39]	FPGA U50	372/316=1.18	✓	✓	×	✓	×	High	-	-	-
CHARM [36]	ACAP VCK190	6400/25.6=250	✓	✓	×	✓	×	High	-	-	-
ViTCoD [40]	ASIC	256/76.8=3.3	×	✓	×	×	×	High	×	×	×
HeatViT [30]	FPGA ZCU102	1260/19.2=65.6	×	✓	×	×	✓	High	×	✓	×
Auto-ViT-Acc [41]	FPGA ZCU102	1260/19.2=65.6	×	✓	×	×	✓	High	×	×	×
SSR [42]	ACAP VCK190	102,400/25.6=4000	✓	✓	✓	✓	✓	High	×	×	×
<b>EQ-ViT (Ours)</b>	ACAP VCK190	102,400/25.6=4000	✓	✓	✓	✓	✓	High	✓	✓	✓

Note: [31], [36]–[39] are architecture and mapping frameworks. [30], [40], [41] and EQ-ViT (ours) are algorithm-hardware co-design frameworks.

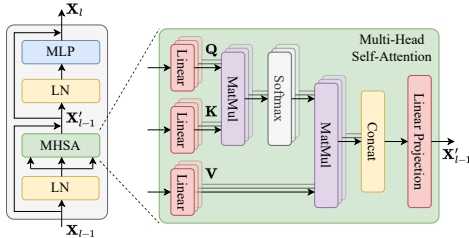


Fig. 3: Computation flow of one transformer encoder.

### III. BACKGROUND AND RELATED WORKS

In this section, we first discuss the background for Vision Transformer model architecture, and existing quantization methods for ViT in Section III-A. In Section III-B, we discuss prior works on hardware acceleration and mapping frameworks on ASICs, FPGAs, GPUs, and ACAP. We also discuss algorithm-hardware co-design frameworks. We summarize our proposed methodologies in hardware accelerator architecture and algorithm with the prior works in Table II.

#### A. Vision Transformer

Transformers were initially proposed to handle the learning of long sequences in NLP tasks. Great interest has surged following the work [43] that applies a transformer architecture for image classification without reliance on convolutional architectures (CNN). With more data, data enhancement techniques, or extended training epochs, ViTs can achieve significantly improved task accuracy [2]. Currently, ViTs excel over CNNs in terms of both speed and accuracy in various computer vision tasks, including image classification [15], object detection [44], and real-time object detection [45] (Better than SOTA YOLOs).

**ViT Architectures.** The input image is first divided and arranged into a sequence of patches (or tokens). This sequence is then passed through an  $L$ -layer Transformer encoder [46]. Each Transformer layer/block consists of two main components (Figure 3): a multi-head self-attention (MSA) module and a multi-layer perceptron (FFN) module. For instance, the DeiT-T model is composed of  $L = 12$  Transformer blocks, where the typical input image resolution is  $224 \times 224$  with a patch size of  $16 \times 16$ . Consequently, this results in a sequence of  $n = 196$  tokens, each token being embedded with  $64 \times 3$  dimensions and utilizing  $h = 3$  heads, and  $dim = 64$  per head. **Quantization on Transformers.** Quantization is one of the most powerful ways to decrease neural networks' computa-

tional operations and memory footprint. Current quantization methods can be divided into two categories: quantization-aware training (QAT) [47] and post-training quantization (PTQ) [48]. NLP-oriented Transformers mainly employ PTQ for two reasons [49]–[51]: • QAT needs open dataset. If the dataset is not publicly available, users have to use PTQ. • QAT requires significant computational resources to support the training of large model sizes (usually over 350M), to which academics usually have limited access. However, the compact model size of ViT and the presence of public datasets make it a suitable candidate for QAT, thereby sidestepping the notable accuracy decrease that is often associated with PTQ. [52] proposes a QAT method for ViTs with information-rectified. However, this work does not quantize nonlinear operations, which causes more hardware overhead because of data conversion between different data types (dequantizing and requantizing), and etc. Moreover, several existing works [30], [53]–[55] utilize **model pruning or sparsity** to reduce the computation operations in ViTs. However, these techniques often lead to unavoidable accuracy drops. *In EQ-ViT, we aim to implement a fully quantized ViT through the QAT algorithm and to improve the accuracy.*

#### B. Transformer Accelerators on Hardware

Hardware acceleration for neural networks spans various platforms like ASICs, GPUs, FPGAs, and ACAPs, as shown in Table II. ACAP stands out with its high theoretical INT8 performance but faces a challenge with its relatively low off-chip bandwidth. This requires more design efforts due to the high computation-to-communication (CTC) ratio on ACAP. Nevertheless, EQ-ViT incorporates all the listed accelerator and algorithm-hardware co-design features, achieving the highest computation utilization and the lowest latency for ViT compared to existing works.

**Hardware Acceleration and Mapping Framework.** TensorRT [31] provides a general quantization solution on GPUs. However, TensorRT adopts an implicit quantization policy and faces low INT8 tensor core utilization due to its sequential execution model, i.e., calling each kernel one after another. Herald [37] introduces a heterogeneous system with simultaneous spatial accelerators (accs), allowing for optimization exploration as different accs may have varied CTC ratios. While Herald integrates well-designed accs, EQ-ViT goes a step further by supporting accs hardware specialization and jointly optimizing accs scheduling and designing. MAGMA [38]

proposes an automatic framework for multi-tenancy heterogeneous architectures but suffers from significant latency due to off-chip communication. This is not ideal for scenarios that are sensitive to time. In contrast, EQ-ViT customizes on-chip forwarding among any two adjacent accs to optimize off-chip access. ViA [39] applies a well-customized spatial solution on U50 FPGA, supporting at most two spatial accs, while EQ-ViT explores more accs. FLAT [56] applies a tensor fusion mechanic and a tiling method to reduce communication in attention-based models. CHARM proposes an open-source framework that composes multiple specialized accelerators, but it only supports FP32 data type and falls short of meeting real-time requirements on ACAP. *EQ-ViT features a spatial architecture with customized accs. The fine-grained pipeline structure and on-chip data forwarding achieve deterministic low latency.*

**Algorithm-Hardware Co-Design Acceleration for ViT.** ViT architecture works [30], [40], [41] also consider algorithm adaption, e.g., sparsity, to speed up model inference. ViTCoD [40] efficiently prunes and polarizes attention maps to create denser or sparser fixed patterns, reducing attention computations. HeatViT [30] employs image-adaptive token pruning and 8-bit quantization to eliminate model redundancy, resulting in improved on-device throughput. Auto-ViT-Acc [41] utilizes network search to tune the quantization choices for the best latency under the frame-per-second (FPS) performance constraints. SSR [42] provides a framework that explores the latency throughput tradeoff for the transformer-based applications. While enabling the hardware accelerator features, there is a lack of discussion about the algorithm design and algorithm-hardware co-design features. However, these works have two main limitations. (i) In [40] and [41], the nonlinear operators in ViT models are computed in FP32, leading to significant hardware overhead. HeatViT [30] uses polynomial approximations for GeLU and Softmax, quantizing them into INT8. However, this approach consumes a significant amount of FF/LUT resources due to the exponent ‘e’ in Softmax. EQ-ViT (ours) employs ‘2’ as the exponent, resulting in lower FF/LUT resource usage. (ii) Task accuracy degrades. ViTCoD applies uniform pruning pattern to compress the attention matrix, leading to accuracy drops of 0.5%~1%. HeatViT and Auto-ViT-Acc fail to consider the inherent data distribution within ViTs, resulting in inconsistencies between the quantization strategy and the data distribution. In contrast, *EQ-ViT introduces a hardware-efficient nonlinear quantization and achieves better task accuracy than full-precision models through activation-aware quantization.*

#### IV. EQ-ViT FRAMEWORK & ARCHITECTURE

In this section, we first illustrate the proposed framework and the EQ-ViT heterogeneous accelerator. We then elaborate on the detailed mapping methodology.

##### A. EQ-ViT Framework Overview

Our EQ-ViT provides the optimization for algorithm/hardware co-design. In Fig 4, our framework takes the latency and accuracy requirement and the hardware information from the user. These combined constraints will decide the final

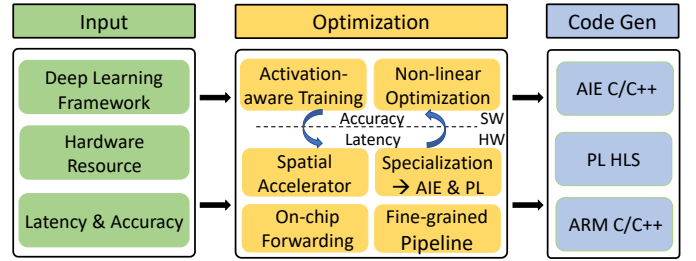


Fig. 4: EQ-ViT software/hardware co-design framework.

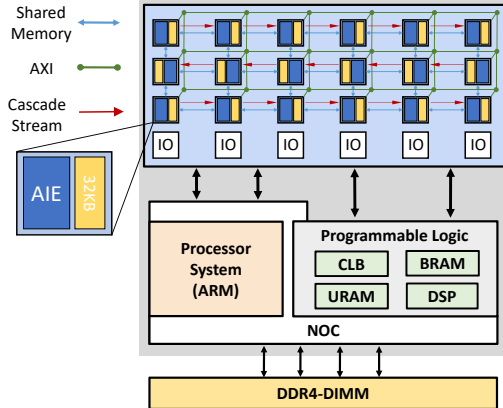


Fig. 5: Versal ACAP architecture overview.

quantization strategy by the activation-aware training and mapping strategy through Eq. (1)-(7) in Section IV-E. Given an application, our EQ-ViT will conduct activation-aware training and provide accuracy under 32bits, 16bits, 8bits, and 4bits for both activations and weights. Then according to the accuracy constraint and the hardware information, EQ-ViT will pick a quantization strategy that meets the accuracy requirement while best fitting the vector processors(AIEs). For instance, Versal VEK280 provides peak performance under 8bits x 4bits mode whereas VCK190 provides peak performance under 8bits x 8bits mode. Then we use Eq. (1)-(7) to optimize the throughput under the latency constraint and quantization strategy. If model quantization is insufficient to target a single board, our work can be used in concert with partitioning approaches to map larger models onto multiple devices [57]. Our EQ-ViT framework also includes a Python-based code generation toolflow. Based on the generated mapping strategy, it can instantiate the code template to generate the design source files including ARM CPU host code, FPGA high-level synthesis code, and AIE intrinsic C/C++ code which can be directly compiled and deployed on Versal ACAP.

##### B. Versal Heterogeneous Platform

For AMD Versal shown in Figure 5, the heterogeneous SoC is composed of the vector processor array, i.e., AIE, the programmable logic (PL), and the CPU. The data can be transferred among CPU, PL, and AIE through the Network-on-Chip (NoC). The AIE array contains  $8 * 50 = 400$  very long instruction words (VLIW) processors (AIEs). AIE runs at 1GHz frequency and supports up to two loads, two moves, one vector, one scalar, and one store instruction in each clock cycle. Each AIE has a 32KB local memory that can be shared with the other three adjacent AIEs. There are 2Kb vector registers and 3Kb accumulation registers in the AIE

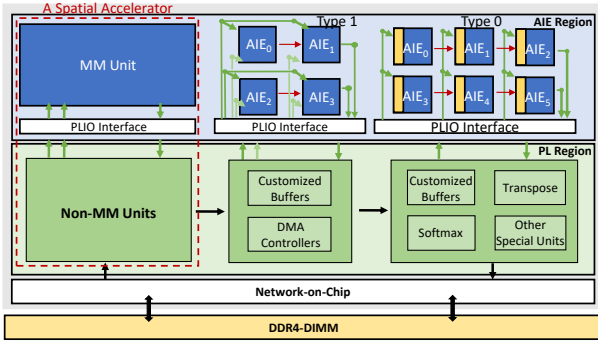


Fig. 6: Proposed EQ-ViT architecture overview.

for data reuse. In terms of computation, an AIE is capable of processing 128 INT8 MACs in one cycle. In terms of data movement between AIEs, the bulk of data can be shared via the local memory (256 bits/cycle bandwidth) or the AXIS network (32 bits/cycle bandwidth). Data transmission between AIE and PL is through the 39 programmable logic I/O (PLIO) tiles on the bottom of the AIE array.

### C. EQ-ViT Heterogeneous Accelerator Overview

Figure 6 shows the overall EQ-ViT architecture on ACAP. It is composed of multiple spatial accelerators with MM units allocated to the AIE region and non-MM units allocated to the PL region. The MM and non-MM units are connected through the PLIO interface. We design specialized MM units for the computation-intensive kernels, e.g. MM, BMM, and Conv, by exploring 3D parallelism on the AIE array. By leveraging the flexibility of the PL region, we implement non-MM units for transpose, Softmax, LayerNorm, and GeLu. Based on these building blocks, our proposed EQ-ViT architecture has the following hardware characteristics: (1) We apply **spatial architecture** that multiple accelerators compute different kernels with high AIE utilization at the same time instead of using one unified accelerator and launching it sequentially. (2) To reduce the expensive off-chip memory access, we explore the **on-chip data forwarding** between different spatial accelerators. (3) We propose a **fine-grained pipeline** structure within each spatial accelerator to further overlap the execution of nonlinear and element-wise kernels with MMs to reduce latency. The details will be elaborated in Section IV-D.

### D. Hardware Design Methodology

**High Utilization Matrix Multiply Design on Single AIE and AIE Array.** When designing the MM/BMM kernels under INT8 data type, efficient communication between PL SRAM, AIE local memory, and registers is important to saturate the abundant computation resource. We optimize MM/BMM kernels from two levels, the single AIE and AIE array levels.

In the single AIE level, based on the byte-level flexibility of AIE, we write efficient AIE intrinsic instructions to make full use of the 2Kb vector register to sustain the 128 MACs/cycle throughput with two 256 bits/cycle load instructions. The 128 MACs can be constructed as a 16x8 MAC array where the second dimension is the reduction dimension. Under the constraints of 2Kb vector register as well as the 256 bits/cycle load bandwidth, we customize the 128 MACs into an 8\*8\*2 3D-SIMD instruction. Based on our atomic 8\*8\*2 MAC operation,

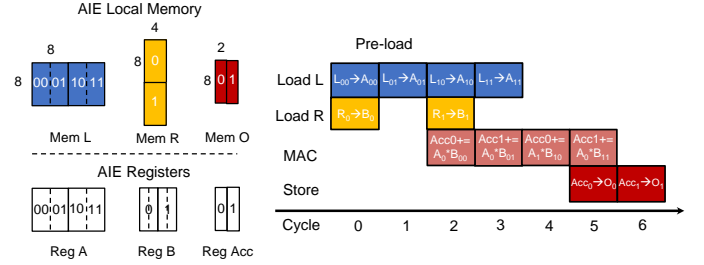


Fig. 7: Efficient single AIE design.

the execution pipeline of a MM with size 8\*16\*4 is shown in Figure 7. In order to achieve back-to-back issued MAC instructions, we allocate 8\*8 and 8\*4 8bits vector registers and use the double buffer technique to hide the latency of loading from local memory to the vector registers. After two cycles of pre-loading the data into AIE registers for LHS and RHS operands, the MAC operations can be issued without idle cycles. Based on this scheduling, it can also handle the MM with a larger size at the expense of only two preload cycles.

When scaling out to the AIE array, the shape variance of the multiple layers within a transformer block often leads to hardware underutilization [36], [37], [58]. Thus for each layer within a transformer block, we design a customized MM unit that perfectly matches the shape of the layer. The number of AIEs utilized in each MM unit are proportional to the total number of operations within the layer. We propose two kinds of MM units as shown in Figure 6. For AIEs of Type 0 that take both the activation and weights as their operands, we efficiently allocate the AIE local memory to make sure the weight of all the blocks fit and loaded during initialization without further excessive loads. Thus it saves the PLIO of sending the right-hand-side (RHS) operands (weights). For the kernels that the weights can't fit in the AIE local memory or the two operands are both activations (attention batch dot), we map them to AIE design of Type 1.

**Element-wise and Nonlinear Kernel Design.** Element-wise kernels and nonlinear kernels including Transpose, VectorAdd, Reformat, Softmax, LayerNorm, and GeLU account for less than 2% of the total operations. However they collectively contribute 40% of the total execution time as shown in Fig 2. To overlap the latency of these operations with the MM operations, we apply a similar line-buffer methodology proposed in SSR [42] to enable a fine-grained pipeline. Beyond the proposed method, we further apply quantization to the nonlinear kernels introduced in Section V-C, significantly reducing the number of resources used in the PL.

### E. Hardware Design Optimization

To meet the latency constraints while optimizing the throughput in the system, we mathematically formulate a mixed-integer-programming (MIP) [59] optimization problem to guide the design space exploration and determine the hardware resource partitioning and configuration for each spatial accelerator. We denote the number of accelerators and batches as  $Acc$  and  $B$ . These are hyperparameters. The ViT graph is denoted as  $G$  and the start execution time of each node included in the graph is referred to as  $T_n$ .  $D_{n,m}$  refers to a binary dependency matrix of the nodes in the graph where

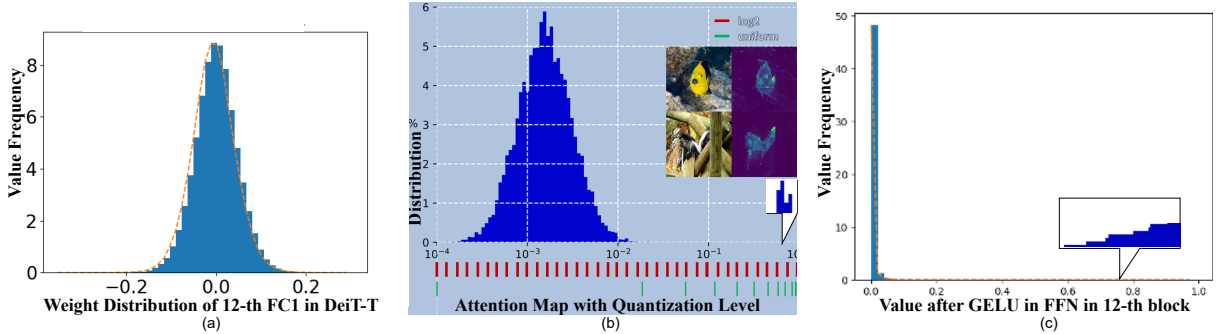


Fig. 8: Data distribution in DeiT-T. (a) A representative normal distribution of the weight of the 12<sup>th</sup> FC1 layer. (b) & (c) Long-tail distribution for attention map and activation distribution after GeLU.

$D_{n,m} = 1$  means node  $m$  depends on  $n$ .  $E_{n,a}$  and  $A_{n,a}$  are integer and binary matrix variables representing the execution time and allocation map of each node on every accelerator. Eq. (2) limits the finish time of every node in batch 1 as the latency of the first batch should meet a certain budget, e.g., *Budget* as 1ms. The goal is to maximize the overall throughput calculated as Eq. (1) and (3). Eq. (4) and (5) guarantee each node will be mapped to only one accelerator and each time one hardware accelerator will only execute one logic node in the graph. The execution order should follow the dependency map (Eq. (6)). The sum of hardware utilization should meet the hardware constraints (Eq. (7)).

$$\text{maximize } B/Lat_{all} \quad (1)$$

$$\text{s.t. } T_n + E_{n,a} \times A_{n,a} \leq \text{Budget} \quad \forall n \in (G_1) \quad (2)$$

$$Lat_{all} = T_n + E_{n,a} \times A_{n,a} \quad \forall n \in (G) \quad (3)$$

$$\sum_{a=1}^{Acc} A_{n,a} = 1 \quad \forall n \in G \quad (4)$$

$$T_m \geq T_n + E_{n,a} \times A_{n,a} \text{ or } T_n \geq T_m + E_{m,a} \times A_{m,a} \quad (5)$$

$$\forall (n, m) \in G, \forall a \in Acc, D_{n,m} = 0, A_{m,a} = A_{n,a}$$

$$T_m \geq T_n + E_{n,a} \times A_{n,a} \quad \forall (n, m) \in G, D_{n,m} = 1 \quad (6)$$

$$\sum_{U \in \{RAM, AIE, PLIO, DSP\}} U_a \leq HW_{\{RAM, AIE, PLIO, DSP\}} \quad (7)$$

$$\forall a \in Acc$$

## V. EQ-ViT ALGORITHM

In this section, to optimize the task accuracy of the quantized models, we first probe into a comprehensive analysis of the data distribution (weight & activation) of ViTs and arrive at several discoveries. Then we develop activation-aware QAT to quantize ViTs and improve accuracy. Furthermore, as we find that Softmax, GeLU take too much resource (REG, LUT, DSP, etc.) and can not fit on board for the whole system implementation, we propose INT-Softmax<sub>2<sup>n</sup></sub> and I-GeLU<sub>Imp</sub> to reduce hardware resources.

### A. Discovery of Data Distribution within ViTs

**Weight:** Data follows a standard normal distribution (Figure 8 (a)). **Activation:** Two key features impact the quantization strategy, *long-tail distribution* and *channel-wise outliers*.

**Long-tail distribution.** • *Attention map.* The attention map is the feature map of Softmax output. To preserve the informative message of the Softmax, we plot attention maps in the Real and Log domain (Figure 8 (b)), which reveals a long-tail distribution. Two cases are put here to show the attention map value. Compared to the uniform quantization (with 8-bit),

which assigns only one bin to such a large number of values, the log2 method has more resolution (24 bins) to cover this data range. This indicates that the low-bit log2 method plays an ideal quantization choice. • *Activation after GeLU.* The GeLU [60] function in FFN causes a truncation effect that concentrates the resulting values around zero, showcasing a clear one-sided stacking pattern (Figure 8 (c)).

**Channel-wise outliers.** • *Large inter-channel variations in the residual link addition.* As shown in Figure 9 (b), the channel-wise ranges in ViTs exhibit more significant fluctuations than in ResNets. As the channels with outliers require larger scales than others, using common quantization methods like layer-wise quantization with the same parameters for all channels would result in an unacceptable quantization error. • *Systematic and fixed outliers.* Although outliers appear in every sequence, they are concentrated in *fixed* channel dimensions of the residual link addition, as shown in Figure 9 (a) (We test 1024 images). Thanks to the *fixed* pattern, we predict and pre-load locations onboard, enabling efficient computation with matrix multiplication.

### B. Activation-aware QAT

Based on the data distribution within ViTs, we propose two novel quantization methods, *long-tail-oriented quantization* and *outlier-predictable QAT*. Assuming the bit-width is  $b$ , the quantizer  $Q(X|b)$  can be formulated by mapping a floating-point number  $X \in R$  to the nearest quantization bin. Among various quantizers, uniform [61] and log2 [62] are typically used. Apart from the special data distribution highlighted in Section V-A, we apply layer-wise uniform quantization on weights and activations.

#### 1) Long-tail-oriented Quantization

• **Log2Q on Attention map.** Based on V-A, we apply Log2Q on the attention map to preserve the informative content as:

$$\text{Attn}_Q = \text{Log2Q}(\text{Attn}|b) = \text{clip}(\lfloor -\log_2(\text{Attn}) \rfloor, 0, 2^b - 1), \quad (8)$$

For the activation after GeLU, we clip the value to  $[-10, 10]$  with the uniform quantization.

2) **Outlier-predictable QAT.** Significant outliers existing in the residual link addition are *channel-wise* and *fixed*. Therefore, we propose the *outlier-predictable training* that obtains the precise channel indices of outliers in the addition of residual links and regularizes scales of outliers with different *power-of-two coefficients (PTC)* in channel-wise.

• **Power-of-Two Coefficients on the Residual Link Quantization.** Given the input activation (token)  $X \in B \times L \times C$  ( $B$ :

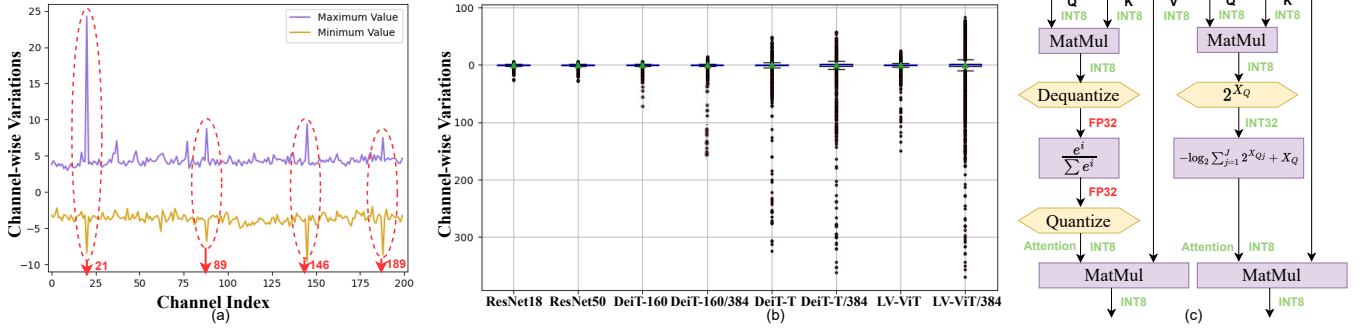


Fig. 9: (a) Channel-wise minimum and maximum values of the second residual link addition in the 9<sup>th</sup> block of DeiT-T. (b) Channel-wise ranges of the last residual link addition in representative models. (c) Comparison of common INT-Softmax [63] and INT-Softmax<sub>2<sup>n</sup></sub> in quantized MSA inference.

batch size,  $L$ : token/sequence length,  $C$ : channel dimension of one token), and the PTC  $r \in \mathbb{N}^C$ , then the quantized activation  $X_Q$  can be formulated as:

$$X_Q = Q(X|b) = \text{clip}(\lfloor \frac{X}{2^{r_s}} \rfloor + z, 0, 2^b - 1), \quad (9)$$

$$s = \frac{\max(X) - \min(X)}{2^R(2^b - 1)}, \quad z = \text{clip}(\lfloor -\frac{\min(X)}{\max(X)} \rfloor, 0, 2^b - 1) \quad (10)$$

where the outlier channel index is  $i$ , PTC is  $r \in [2,3,4]$ ,  $s$  is scaling factor and  $z$  is zero-point.

• **Outlier-predictable Training.** It includes three stages as Algorithm 1: 1) Initialize the PTC with the full-precision model estimated by  $3\sigma$  method [64]. 2) Search for the channel index  $i$  and the PTC  $r$  with the  $l_2$  regularization. 3) Fix the index  $i$  and  $r$  obtained in stage 2 and fine-tune the model.

#### Algorithm 1: Outlier-predictable training

```

1 . Given the full-precision ViT Model, the test subdataset  $D$ ,
   the number of blocks  $L$ , Epochs for searching, Epochf for
   fine-tuning, and quantized low-bit  $b$ ;
// Stage1: Initialize the PTC  $r$  with the outlier
// estimated from the full-precision Model.
2 foreach  $l \in [0, 1, \dots, L - 1]$  do
3    $i_o, r_o = \text{Check\_Outliers}_{3\sigma}(\text{Model}_l, D)$ ;
4   Quantize Model $l$  by Eq.(9) with  $i_o, r_o, b$  into QModel $l$ ;
5 end
// Stage2: Search for the channel index  $i$  of outliers
// and determine the PTC  $r$  by the  $l_2$  regularization.
6  $r = r_o, i = i_o$ ;
7 foreach  $eps \in [0, 1, \dots, \text{Epoch}_s - 1]$  do
8   // These three operations are gradient-free.
9    $i, r = \text{Check\_Outliers}_{3\sigma}(\text{QModel}_l)$ ;
10   $r_i = \text{argmin}_{r_i \in \{1, 2, \dots, R\}} \|X_i - \lfloor \frac{X_i}{2^{r_i s}} \rfloor \cdot 2^{r_i s}\|_2$ ;
11  // Following Eq.(11), Eq.(8), Eq.(12) and Eq. (9).
12  taskloss, quantizationloss = Quantize(QModel,  $b$ );
13  SGDBackward(taskloss, quantizationloss);
14 end
// Stage3: Finetune the  $b$ -bit quantization.
15 Fix  $r$  and  $i$  and quantize QModel with fine-tune Epochf;
16 Finalize the quantized ViT Model.

```

#### C. Nonlinear Operations Quantization

1) **INT-Softmax<sub>2<sup>n</sup></sub>.** We replace natural constant  $e$  inside Softmax with the power of 2 [65] with integer inputs.  $i$  represents the  $i$ th token:

$$\text{INT-Softmax}_{2^n}(X) = \frac{\exp(X_i)}{\sum_{l=1}^L \exp(X_l)} \rightarrow \frac{2^{X_i}}{\sum_{l=1}^L 2^{X_l}}, \quad (11)$$

TABLE III: Model structures of four different ViT models.

Model	#Head	Embed. Dim	Depth	Precision	Model (MB)	MACs (G)
DeiT-T	3	192	12	INT8	5.6	1.3
DeiT-160	4	160	12	INT8	4	0.9
DeiT-256	4	256	12	INT8	7.4	2.1
LV-ViT-T	4	240	12	INT8	6.75	1.6

• **Log2Q with INT-Softmax<sub>2<sup>n</sup></sub>.** Similar to [66], we utilize Log2Q on the attention map. We then integrate the power of 2 inside Softmax and the operation can be modified as:

$$\text{Attn}_Q = \text{Log2Q}(\text{Attn}|b) = \text{clip}(\lfloor -\log_2 \sum_{l=1}^L 2^{\hat{X}_l + \hat{X}_i} \rfloor, 0, 2^b - 1), \quad (12)$$

The exponent function is a crucial component of Softmax, but its nonlinearity makes it expensive to implement on hardware. Combined with Log2 quantization, Softmax function can be executed with only addition computation and removes division thus can be implemented by LUTs on FPGA instead of AIEs. The difference between normal MSA and our method is shown in Figure 9 (c). Normal MSA needs to be dequantized and requantized around full-precision Softmax. The floating-point exponential calculation of INT-Softmax<sub>2<sup>n</sup></sub> is replaced with BitShift and addition and keeps integer-only data type.

2) **I-GeLU<sub>Imp</sub>.** We adapt I-GeLU [67] to a combination with linear kernels and lookup table under INT8 mode, since  $1+L(x)$  is an odd function within the range (0,2):

$$\text{I-GeLU}_{\text{Imp}} = \begin{cases} 0 & \text{if } -(2^8 - 1) \leq x \leq -3 \\ \{0, 0, 0, 0, 1\} & \text{if } x \in \{-2, 1, 0, 1, 2\} \\ x & \text{if } 3 \leq x \leq 2^8 - 1 \end{cases}, \quad (13)$$

The original data value in  $-2 \leq x \leq 2$  is  $\{-0.0257, -0.1152, 0, 0.5919, 1.3885\}$ . For implementation, we pre-load the requantized integer value directly on-board as Equation (13).

## VI. EXPERIMENTS

### A. Experiment Settings

**Application and Training Framework Setup.** Our experiments are conducted on ImageNet-1k [68], Cifar-100, Cifar-10 [69] datasets in PyTorch 3.8. We use two representative vision transformers, DeiT [2] (with three model variants), and LV-ViT [70], in Table III. The baseline models with FP32 are obtained from the TorchVision. The outlier-predictable training follows Q-ViT with distribution-guided distillation (DGD) techniques [52], and the training process is executed on 4 NVIDIA V100 GPUs. We set stage 1 to 70 epochs, and stage 2 to 30 epochs.

**Hardware Setup.** We evaluate EQ-ViT the on AMD ACAP VCK190. We compare EQ-ViT with other SOTA implementations on CPU, FPGA, and GPU. For each model, we iterate



TABLE IV: Comparison of EQ-ViT and works on CPU, GPU, FPGA, ACAP in latency &amp; energy efficiency on four models.

Model	# of Batch	Metric	PyTorch Xeon8375 10nm	TensorRT A10G 8nm	TensorRT A100 7nm	TensorRT Orin 8nm	HeatViT ZCU102 16nm	HeatViT U250 16nm	SSR VCK190 7nm	EQ-ViT (ours) VCK190 7nm	EQ-ViT (ours) VEK280 (est.) 7nm
DeiT-T	6	Latency (ms)	167.68	1.78	1.84	7.97	32.72	7.3	0.54	<b>0.56</b>	<b>0.33</b>
		FPS (image/sec.)	36	3371	3260	753	183	822	11111	<b>10695</b>	<b>18010</b>
		Energy.Eff (FPS/W)	3.8	15.8	18.6	17.7	19.4	10.2	213.7	<b>224.7</b>	<b>427.8</b>
DeiT-T-160	6	Latency (ms)	129.01	1.78	1.73	7.92	29.75	6.34	0.50	<b>0.46</b>	<b>0.28</b>
		FPS (image/sec.)	47	3371	3468	758	202	946	11976	<b>13187</b>	<b>21702</b>
		Energy.Eff (FPS/W)	4.9	16.9	20.0	19.0	21.9	12.2	206.8	<b>280</b>	<b>503.5</b>
DeiT-T-256	6	Latency (ms)	294.61	2.07	2.09	10.44	39.33	9.13	0.98	<b>0.89</b>	<b>0.53</b>
		FPS (image/sec.)	20	2899	2871	575	153	657	6122	<b>6726</b>	<b>11393</b>
		Energy.Eff (FPS/W)	2.2	12.5	15.0	13.2	14.7	8.5	102.9	<b>142.8</b>	<b>269.3</b>
LV-ViT-T	6	Latency (ms)	213	2.55	2.54	10.1	43.21	9.36	0.85	<b>0.61</b>	<b>0.37</b>
		FPS (image/sec.)	28	2353	2362	594	139	639	7059	<b>9836</b>	<b>16017</b>
		Energy.Eff (FPS/W)	3	10.6	12.9	13.5	13.5	7.8	115.3	<b>202.8</b>	<b>359.9</b>

the inference for over 60s and perform this measurement 10 times to calculate the average inference latency. On CPU, we measure the inference latency on an m6i.large instance from Amazon AWS using pytorch 2.0.1. The instance has two Intel Xeon 8375C vCPU cores running at 2.9 GHz and thermal design power (TDP) is 300W. On GPUs, we measure the performance of TensorRT [31] on A10G (8nm), A100(7nm), and Jetson AGX Orin (8nm). We first use onnx 1.14.0 to compile the PyTorch model into onnx format, then use TensorRT 8.6 and its Python interface to compile the onnx model into the TensorRT engine. To perform the INT8 inference, we enable the `tensorrt.BuilderFlag.INT8` flag in compilation. The power consumption of the GPUs is measured via `nvidia-smi` [71]. For the CPU and GPU experiments, the PyTorch models are from the Meta Research [72].

On FPGA, we compare EQ-ViT with HeatViT [30] on AMD Zynq ZCU102 and AMD Alveo U250. We compare EQ-ViT with SSR [42] on the same device VCK190. We measure the power of VCK190 using AMD Board Evaluation and Management [73]. To be noted, EQ-ViT provides algorithm & algorithm/hardware co-design to explore different quantization strategies, e.g. activations 8bits, weights 4bits (A8W4), without accuracy loss. We add the new estimated results (est.) in Table IV when using the A8W4 quantization on AMD Versal VEK280 which provides 4x 8bits x 4bits MAC operations/cycle/AIE over VCK190 with 8bits x 8bits precision. Our estimation shows that EQ-ViT further reduces the latency by 1.67x using VEK280 over VCK190. This gain can not be achieved without the algorithm & algorithm/hardware co-design, demonstrating the key new contribution of EQ-ViT.

### B. ViT Inference Perf. & Energy Eff. Analysis

**①Performance and energy efficiency comparison among CPU, GPU, FPGA, and ACAP.** We apply our EQ-ViT framework to four different ViT applications under INT8 quantization mode and evaluate the on-board implementation on AMD Versal VCK190. We compare EQ-ViT with six works on CPU, GPUs, and FPGAs regarding latency and energy efficiency on four models in Table IV. Here we report the performance when setting the latency budget as 1 ms. EQ-ViT DSE finds the optimal throughput design under this latency constraint when the batch size is set to 6. The achieved latencies are 0.56ms, 0.46ms, 0.89ms, and 0.61ms for the four applications. In contrast, the solutions on other platforms have larger latency and do not meet the latency constraint under the same batch size. For all four applications, the average latency

TABLE V: Latency comparison between on-board measurements and MIP modeling estimations for four ViT models.

Model	# of AIE	Estimation	On-board	Error Rate
DeiT-T	394	0.58 (ms)	0.56 (ms)	4%
DeiT-160	396	0.48 (ms)	0.46 (ms)	5%
DeiT-256	399	0.92 (ms)	0.89 (ms)	3%
LV-ViT-T	398	0.59 (ms)	0.61 (ms)	-3%

TABLE VI: Resource utilization of Softmax and GeLU before vs. after EQ-ViT algorithm changes for hardware efficient implementation on VCK190.

Operations	Softmax [36]	INT-Softmax(ours)	GeLU [36]	INT-GeLU(ours)
REG	62415 (4.17x)	14962	22238 (137x)	162
LUTLogic	94739 (14.48x)	6545	14222 (142x)	100
LUTMem	37668 (18834x)	2	1392 (-)	0
RAM	147 (9.19x)	16	1 (-)	0
DSP	196 (7.00x)	28	128 (-)	0

gains are 315.0x, 3.39x, 3.38x, 14.93x, 59.5x, 13.1x, and the gains of energy efficiency are 62.2x, 15.33x, 12.82x, 13.31x, 13.5x, 21.9x when comparing to Intel Xeon 8375C vCPU, A10G, A100 GPUs, and AMD ZCU102, U250 FPGAs. We list the latency improvement from the four features (4.2x, 3.4x, 2.3x, 2.7x) in the last two columns in Table VIII, together achieving 89x latency reduction from 50ms using FP32 model with CHARM to 0.56ms using the INT8 model with EQ-ViT on VCK190. We also applies the int8 GEMM solution proposed by [35]. For DeiT-T with batch equals 6, it achieves 12.1 ms latency as it only implement a monolithic accelerator and requires the weights and activation to be accessed from the off-chip memory. By applying the on-chip data forwarding, fine-grained pipeline and multiple spatial accelerators, EQ-ViT achieves 21.6x performance improvement. We further discuss the in-depth performance analysis in Section VII.

### ②Analytical model VS. EQ-ViT on-board implementation.

We evaluate the latency of the four ViT models on AMD Versal VCK190 and compare them with the proposed MIP modeling. Guided by the MIP, all four cases utilize over 98.5% AIE. The error rate in percentage refers to the difference between the estimated latency by MIP and our real on-board implementation. On average, the MIP modeling achieves a high prediction accuracy and has less than 4% error rate.

### ③Resource utilization before VS. after EQ-ViT hardware-efficient algorithm adaption for two non-MM kernels Softmax, GeLU.

We compare the hardware utilization of the optimized Softmax and GeLU implementation with the previous FP32 design reported in CHARM [36]. We normalize the number of processing units to 16, the same as the implementation in CHARM. The hardware utilization of our real implementation is illustrated

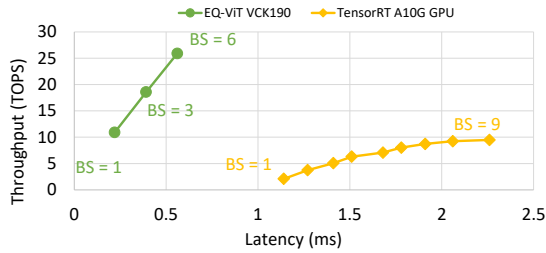


Fig. 10: Latency and throughput tradeoff comparison between EQ-ViT on VCK190 and TensorRT on A10G GPU.

TABLE VII: Comparison of the top-1 (%) accuracy with state-of-the-art methods on multiple datasets.

Model	FP32	PTQ					QAT		
		MinMax	EMA	Percentile	OMSE	FQ-ViT	LSQ	Q-ViT*	EQ-ViT
ImageNet Dataset									
DeiT-T	72.2	70.9	71.2	71.5	71.3	71.6	71.5	73.6	<b>74.5</b>
DeiT-160	68.1	67	67.6	67.8	67.9	68	67.9	70.1	<b>70.5</b>
DeiT-256	77.2	72.5	72.5	74	72.4	76.6	75.9	77.6	<b>78.2</b>
LV-ViT-T	79.1	75.4	75.4	76.9	75.3	77.4	78.7	80.1	<b>80.5</b>
Cifar-100 Dataset									
DeiT-T	85.6	85	85.1	85.3	85.1	85.4	85.3	86.2	<b>86.6</b>
DeiT-160	83.5	83	83.3	83.3	83.4	83.5	83.5	84.4	<b>84.4</b>
DeiT-256	87.1	85.8	85.9	86.5	85.7	87	86.9	88	<b>88.3</b>
LV-ViT-T	88.1	87.3	87.4	87.5	87.2	88.1	88.4	89.2	<b>89.5</b>
Cifar-10 Dataset									
DeiT-T	97.8	97.5	97.6	97.5	97.4	97.8	97.7	98.1	<b>98.3</b>
DeiT-160	96.3	96.1	96.2	96.3	96.1	96.4	96.5	96.9	<b>96.9</b>
DeiT-256	98.1	98	98	98.3	97.9	98.1	98	98.7	<b>98.9</b>
LV-ViT-T	98.7	98.6	98.6	98.7	98.5	98.8	98.6	99.2	<b>99.4</b>

Note: \* indicates our reproduced results with quantized nonlinear operations for a fair comparison; And all the models (except FP32) are quantized into 8-bit precision.

in Table VI. We normalize 1 URAM as 8 BRAM and report the total number of RAM used in both designs. For the Softmax layer, since we replace the resource-demanding operations, i.e. exponential and division, we saved the number of DSP and LUT by 7.0x and 14.48x respectively. Instead of using the double buffer technique applied in CHARM [36], by using the streaming pipelined architecture within this kernel, we save the LUTMem by 18834x and total RAM by 9.19x. For the GeLU kernel, with the LUT optimization, it no longer consumes LUTMem, RAM, and DSP and reduces REG and LUT by 137x and 142x. We show the overall implementation layout of DeiT-T in Figure 11 containing ten MM units and non-MM modules including AXI DMA, Transpose, and non-linear kernels.

**④ The effect of batch size on latency-throughput trade-off.** We can leverage the MIP-based analytical model to perform latency-throughput trade-off in EQ-ViT, e.g., find the designs that achieve the highest throughput under latency constraints. Figure 10 shows the latency-throughput Pareto fronts of EQ-ViT on VCK190 and TensorRT on A10G GPU. EQ-ViT achieves a better Pareto front than that of GPU.

**⑤ Can we leverage EQ-ViT when model sizes do not fit on-chip?** If a model can not fit on a single board, we can leverage EQ-ViT to explore how the model is most effectively partitioned onto multiple devices, which is our future work.

### C. Inference Accuracy Comparisons

We compare EQ-ViT accuracy with popular PTQ methods [61], [74], [75] and SOTA QAT methods [52], [76]. For the sake of fairness, we reproduced the results of Q-ViT with quantized GeLU and Softmax.

**Image Classification on Multiple Datasets.** ① ImageNet. Recent SOTA methods for PTQ suffer a significant drop in

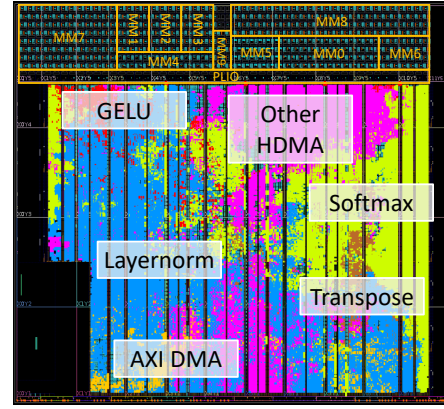


Fig. 11: EQ-ViT implementation layout on VCK190 with kernels highlighted in the FPGA and AIE portion of ACAP.

accuracy up to 3.8% (Table VII). In contrast, ours can enhance task accuracy up to 2.4% over the baseline by minimizing quantization errors and removing model redundancy. While the SOTA QAT method, Q-ViT, has made strides in correcting information distribution within ViT models, it still relies on floating-point computations for Softmax and GeLU, making it challenging for practical and efficient hardware deployment. In contrast, EQ-ViT leverages activation flow fitting and optimization to achieve an additional accuracy boost of 0.4%~0.9% over Q-ViT. Furthermore, EQ-ViT supports efficient implementation on ACAP. **② Cifar-100 & Cifar-10.** We extend results on Cifar datasets to showcase our validation. For Cifar-100 dataset, EQ-ViT can enhance accuracy up to 1.4% and achieve 0.3% ~ 0.4% higher accuracy than Q-ViT; For Cifar-10 dataset, EQ-ViT can enhance accuracy up to 0.8%, and reach 0.2% higher accuracy than Q-ViT. EQ-ViT models are equipped with quantized nonlinear operations. The full precision ViTs' accuracy is based on the original DeiT and Swin papers, which don't use DGD distillation. Q-ViT introduces DGD distillation to distill the knowledge from the larger-size ViT to the smaller-size one, which is integrated into our training setting. This is why EQ-ViT achieves better accuracy than the full-precision models. Notably, EQ-ViT also surpasses Q-ViT accuracy under the same training conditions.

## VII. GENERALITY DISCUSSION & MICROARCHITECTURE INSIGHTS

EQ-ViT performance improvements over prior solutions come from two folds: (1) Software aspect: EQ-ViT accelerator mapping & optimization techniques that fully leverage all the heterogeneous microarchitecture features on ACAP. For those, we explain how different optimization techniques included in EQ-ViT contribute to performance improvements and discuss whether and how those optimizations can be applied on other platforms including FPGA and GPU; (2) Hardware aspect: the heterogeneous microarchitecture features from ACAP that provide flexible mapping features to be applied on such architecture. Specifically, those EQ-ViT mapping features that can not be ported to FPGAs or GPUs reflect the corresponding architecture limitations on FPGAs or GPUs. **Quantization:** The performance gain from quantization comes from two parts: (1) the improved peak computation throughput, and (2) the reduced off-chip data access. Especially, if

TABLE VIII: Comparisons of FPGA, GPU, ACAP with SOTA framework implementations (Impl.) and EQ-ViT optimizations

Mapping features	FPGA+SOTA Impl. (HeatViT)	FPGA+EQ-ViT Optimizations	GPU+ SOTA Impl. (TensorRT)	GPU+EQ-ViT Optimizations	ACAP+SOTA Impl. (CHARM)	ACAP+EQ-ViT Optimization
Quantization	yes	yes	partial	partial ->yes	no	yes (4.2x)
On-Chip Forwarding	no	yes	no	arch limit	no	yes (3.4x)
Multi Spatial Accelerators	no	yes	no	arch limit	yes	yes (2.3x)
Fine-grained Pipelining	no	yes	no	arch limit	no	yes (2.7x)
Utilize AI-optimized PEs	no	arch limit	yes	yes	yes	yes
Estimated latency after EQ-ViT	7.3ms	3.9ms	1.8ms	1.05ms	50ms (1x)	0.561ms (89x)

the model size after quantization gets across a threshold and the weights can fit on-chip, there will be a huge improvement since all the intermediate data can be forwarded on-chip.

Accelerators on FPGA and ACAP can fully benefit from quantization, whereas GPU can not. Current GPU frameworks, e.g., TensorRT, can not fully cache intermediate data across different kernel function calls unless users explicitly rewrite multiple kernels into one kernel (fusion). Another GPU software limitation is the implicit quantized kernels. In our GPU profiling for quantized models, we observe that TensorRT generates a mixed precision model, where the BMM kernels are computed in FP32, not in IN8. In our analysis, if we can quantize the BMM, softmax, LayerNorm, and transpose kernels in GPU, the hypothetical latency of DeiT-T on A10G GPU can be reduced to 1.05 ms, which is 1.9x when compared to EQ-ViT latency.

**On-chip forwarding:** By applying on-chip forwarding, activations of the models can be kept inside the accelerator chip to reduce off-chip communication. This technique has been applied to the Versal ACAP and FPGA platforms. On ACAP, applying this technique gives 3x latency reduction.

For GPU, the on-chip forwarding is limited compared to FPGA or ACAP. The flexibility in PL logic in FPGA and ACAP allows multiple accelerators to communicate with each other with arbitrary data forwarding per the user’s control. In GPU, shared memory can be explicitly controlled by the user. However, one shared memory in one stream multi-processor (SM) can not directly forward the data to the other shared memory in another SM. It has to go through off-chip DDR or HBM. This is the microarchitecture limitation on GPU.<sup>2</sup>

**Multiple spatial accelerators:** On FPGA and ACAP platforms, compared with sequentially-called one unified accelerator, the spatially-called multiple accelerators can reach higher hardware utilization as each hardware accelerator has smaller hardware resources and can be specialized for the kernel.

In GPUs, horizontal fusion [78], [79] is motivated by similar reasons, i.e., using multiple kernels running at the same time instead of launching kernels sequentially. The key idea is to allocate different groups of SM working simultaneously whereas each SM group works on one type of kernel. However, such multiple spatial accelerators in GPU have less flexibility than in FPGA and ACAP. The partition in GPU is in the SM granularity, therefore, different hardware resources, i.e., computation processing elements (PE), and on-chip storage, across different accs have a fixed ratio. In FPGA and ACAP,

<sup>2</sup>On-chip forwarding between SMs can not be implemented on Nvidia GPUs before Ampere generation. However, as the successor of Ampere architecture, the Hopper architecture uses distributed shared memory (DSMEM) [77], enabling fast communication between shared memory and potentially providing more flexibility in on-chip forwarding among SMs on GPUs.

PL provides users with full flexibility to partition computation PE (DSPs, LUT, AIEs) and on-chip storage (BRAM, URAM) with arbitrary ratios across different accs.

**Fine-grained pipelining:** Applying the fine-grained pipelining enables execution overlap among accelerators, and leads to higher resource utilization and lower latency. Fine-grained pipelining can be easily implemented in FPGA and ACAP, on the contrary, it is not easily implemented on GPUs. We analyze the DeiT-T inference on A10G, if we can hack all BMM kernels to be computed in INT8, the latency reduces from 1.8ms to 1.05ms, however, this can not be further reduced. The 1.05ms latency includes MM kernels at 0.78ms and non-MM kernels at 0.27ms. Unlike ACAP, which allows full programmability and flexibility to allow AIE and FPGA within the ACAP SoC to run simultaneously, the current GPU programming model does not allow the simultaneous execution between GPU Tensor cores and GPU CUDA cores.

## VIII. SUMMARY AND CONCLUSION

We summarize our generality discussion in Table VIII. The FPGA platforms are highly flexible and can support most of the EQ-ViT optimization methods, but without the AI-optimized processing element like tensor cores or AI engine, the computing capability limits the performance of FPGAs. GPUs have the highest theoretical throughput and bandwidth, but the relatively fixed architecture limits their performance in latency-critical situations. The ACAP platform has both flexibility and AI-optimized PE, thus reaching the lowest latency with the optimization of EQ-ViT.

This implies interesting research questions, e.g., what other kinds of applications will let ACAP, a combination of FPGA and AI-optimized SoC, achieve the better of both worlds? Shall we introduce FPGA or reconfigurable architecture in broader GPU architecture to improve latency? If FPGA is too fine-grained, what is the least reconfigurability needed in future architecture to balance performance and adaptability? We leave these in our future work.

## REFERENCES

- [1] A. Dosovitskiy *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *ICLR*, 2021.
- [2] H. Touvron *et al.*, “Training data-efficient image transformers & distillation through attention,” in *ICML*, 2021, pp. 10 347–10 357.
- [3] Z. Liu *et al.*, “Swin transformer: Hierarchical vision transformer using shifted windows,” *ICCV*, 2021.
- [4] N. Carion *et al.*, “End-to-end object detection with transformers,” in *ECCV*. Springer, 2020, pp. 213–229.
- [5] X. Zhu *et al.*, “Deformable detr: Deformable transformers for end-to-end object detection,” in *ICLR*, 2020.
- [6] H. Chen *et al.*, “Pre-trained image processing transformer,” in *CVPR*, 2021, pp. 12 299–12 310.
- [7] L. Zhou *et al.*, “End-to-end dense video captioning with masked transformer,” in *CVPR*, 2018, pp. 8739–8748.
- [8] W. Wang *et al.*, “Pyramid vision transformer: A versatile backbone for dense prediction without convolutions,” in *ICCV*, 2021.

- [9] K. Han *et al.*, “Transformer in transformer,” in *NeurIPS*, 2021.
- [10] H. Cao *et al.*, “Swin-UNET: Unet-like pure transformer for medical image segmentation,” *arXiv preprint arXiv:2105.05537*, 2021.
- [11] M. Rafie, “Autonomous vehicles drive ai advances for edge computing,” <https://www.3dincites.com/2021/07/autonomous-vehicles-drive-ai-advances-for-edge-computing/>.
- [12] CERN, “Colliding particles not cars: CERN’s machine learning could help self-driving cars,” 2023, last accessed JANUARY 25, 2023. [Online]. Available: <https://home.cern/news/news/knowledge-sharing/colliding-particles-not-cars-cerns-machine-learning-could-help-self>
- [13] W.-H. Ko *et al.*, “Edgeric: Empowering realtime intelligent optimization and control in nextg networks,” *arXiv preprint arXiv:2304.11199*, 2023.
- [14] G. Feng *et al.*, “Risgraph: A real-time streaming system for evolving graphs to support sub-millisecond per-update analysis at millions ops/s,” in *SIGMOD ’21*. ACM, 2021.
- [15] Y. Li *et al.*, “Efficientformer: Vision transformers at mobilenet speed,” *Advances in Neural Information Processing Systems*, 2022.
- [16] —, “Rethinking vision transformers for mobilenet size and speed,” 2022.
- [17] Y. Nagamatsu *et al.*, “Basic implementation of fpga-gpu dual soc hybrid architecture for low-latency multi-dof robot motion control,” in *IROS*, 2020.
- [18] D. Gizopoulos *et al.*, “Architectures for online error detection and recovery in multicore processors,” in *DATE*, 2011, pp. 1–6.
- [19] R. Basir *et al.*, “Fog computing enabling industrial internet of things: State-of-the-art and research challenges,” *Sensors*, vol. 19, no. 21, 2019.
- [20] P. Koppermann *et al.*, “Low-latency x25519 hardware implementation: Breaking the 100 microseconds barrier,” *MICPRO*, 2017.
- [21] J. Soifer *et al.*, “Deep learning inference service at microsoft,” in *opML*. USENIX Association, May 2019.
- [22] A. Putnam *et al.*, “A reconfigurable fabric for accelerating large-scale datacenter services,” *IEEE Micro*, vol. 35, no. 3, pp. 10–22, 2015.
- [23] J. Fowers *et al.*, “A configurable cloud-scale dnn processor for real-time ai,” in *ISCA*, 2018, pp. 1–14.
- [24] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *ISCA*, 2017, pp. 1–12.
- [25] N. Jouppi *et al.*, “Motivation for and evaluation of the first tensor processing unit,” *IEEE Micro*, vol. 38, no. 3, pp. 10–19, 2018.
- [26] —, “Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings,” in *ISCA*, 2023.
- [27] F. G. de Magalhães *et al.*, “Optical interconnection networks: The need for low-latency controllers,” in *Photonic Interconnects for Computing Systems*, 2022, pp. 73–105.
- [28] M. Miscuglio *et al.*, “Photonic tensor cores for machine learning,” *Applied Physics Reviews*, vol. 7, no. 3, p. 031404, 07 2020.
- [29] F. P. Sunny *et al.*, “A survey on silicon photonics for deep learning,” *J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 4, jun 2021.
- [30] P. Dong *et al.*, “Heatvit: Hardware-efficient adaptive token pruning for vision transformers,” in *HPCA*. IEEE, 2023, pp. 442–455.
- [31] H. Vanholder, “Efficient inference with tensorrt,” in *GPU Tehnology Conference*, vol. 1, no. 2, 2016.
- [32] H. Wu *et al.*, “Integer quantization for deep learning inference: Principles and empirical evaluation,” *arXiv preprint arXiv:2004.09602*, 2020.
- [33] NVIDIA Developer, “Nvidia nsight systems.” [Online]. Available: <https://developer.nvidia.com/nsight-systems>
- [34] Nvidia, “Nvidia TensorRT Documentation .” [Online]. Available: <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#working-with-int8>
- [35] J. Zhuang *et al.*, “High Performance, Low Power Matrix Multiply Design on ACAP: from Architecture, Design Challenges and DSE Perspectives,” in *DAC*, 2023, pp. 1–6.
- [36] —, “CHARM: Composing Heterogeneous Accelerators for Matrix Multiply on Versal ACAP Architecture,” in *FPGA*, 2023, p. 153–164.
- [37] H. Kwon *et al.*, “Heterogeneous dataflow accelerators for multi-dnn workloads,” in *HPCA*. IEEE, 2021, pp. 71–83.
- [38] S.-C. Kao *et al.*, “Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores,” in *HPCA*. IEEE, 2022.
- [39] T. Wang *et al.*, “Via: A novel vision-transformer accelerator based on fpga,” *IEEE TCAD*, vol. 41, no. 11, pp. 4088–4099, 2022.
- [40] H. You *et al.*, “Vitecod: Vision transformer acceleration via dedicated algorithm and accelerator co-design,” in *HPCA*. IEEE, 2023.
- [41] Z. Lit *et al.*, “Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization,” in *FPL*. IEEE, 2022, pp. 109–116.
- [42] J. Zhuang *et al.*, “SSR: Spatial Sequential Hybrid Architecture for Latency Throughput Tradeoff in Transformer Acceleration,” in *FPGA*, 2024, p. 55–66.
- [43] A. Dosovitskiy *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *ICLR*, 2021.
- [44] Z. Zong *et al.*, “Detrs with collaborative hybrid assignments training,” in *ICCVW*, 2023, pp. 6748–6758.
- [45] W. Lv *et al.*, “Detrs beat yolos on real-time object detection,” *arXiv preprint arXiv:2304.08069*, 2023.
- [46] A. Vaswani *et al.*, “Attention is all you need,” in *NeurIPS*, 2017.
- [47] Z. Dong *et al.*, “Hawq: Hessian aware quantization of neural networks with mixed-precision,” in *ICCVW*, 2019, pp. 293–302.
- [48] Y. Li *et al.*, “Brecq: Pushing the limit of post-training quantization by block reconstruction,” *arXiv preprint arXiv:2102.05426*, 2021.
- [49] A. H. Zadeh *et al.*, “Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference,” in *MICRO*. IEEE, 2020.
- [50] T. Dettmers *et al.*, “Llm.int8(): 8-bit matrix multiplication for transformers at scale,” *NeurIPS*, vol. 33, pp. 28 089–28 154, 2021.
- [51] Z. Yao *et al.*, “Zeroquant: Efficient and affordable post-training quantization for large-scale transformers,” *NeurIPS*, vol. 35, 2022.
- [52] Y. Li *et al.*, “Q-vit: Accurate and fully quantized low-bit vision transformer,” *NeurIPS*, vol. 34, pp. 28 092–28 103, 2022.
- [53] T. Chen *et al.*, “Chasing sparsity in vision transformers: An end-to-end exploration,” *NeurIPS*, vol. 34, pp. 19 974–19 988, 2021.
- [54] M. Zhu *et al.*, “Vision transformer pruning,” *arXiv preprint arXiv:2104.08500*, 2021.
- [55] S. Yu *et al.*, “Unified visual transformer compression,” in *International Conference on Learning Representations*, 2022.
- [56] S.-C. Kao *et al.*, “Flat: An optimized dataflow for mitigating attention bottlenecks,” in *ASPLOS*, 2023.
- [57] J. Fowers *et al.*, “A configurable cloud-scale dnn processor for real-time ai,” in *ISCA*. IEEE, 2018, pp. 1–14.
- [58] X. Zhang *et al.*, “Dnnexplorer: a framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator,” in *ICCAD*, 2020.
- [59] M. S. Bensaleh *et al.*, “Optimal task scheduling for distributed cluster with active storage devices and accelerated nodes,” *IEEE Access*, 2018.
- [60] D. Hendrycks *et al.*, “Gaussian error linear units (gelus),” *arXiv preprint arXiv:1606.08415*, 2016.
- [61] B. Jacob *et al.*, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *CVPR*, 2018.
- [62] J. Cai *et al.*, “A deep look into logarithmic quantization of model parameters in neural networks,” in *IAIT*, 2018, pp. 1–8.
- [63] Y. Lin *et al.*, “Fq-vit: Post-training quantization for fully quantized vision transformer,” in *IJCAI*, 2022, pp. 1173–1179.
- [64] W. contributors., “68–95–99.7 rule,” in *Wikipedia, The Free Encyclopedia*, 2022, pp. 0–1.
- [65] G. C. Cardarilli *et al.*, “A pseudo-softmax function for hardware-based high speed image classification,” *Scientific reports*, 2021.
- [66] W. Wang *et al.*, “Sole: Hardware-software co-design of softmax and layernorm for efficient transformer inference,” in *ICCAD*. IEEE, 2023.
- [67] S. Kim *et al.*, “I-bert: Integer-only bert quantization,” in *International conference on machine learning*. PMLR, 2021, pp. 5506–5518.
- [68] J. Deng *et al.*, “Imagenet: A large-scale hierarchical image database,” in *CVPR*. IEEE, 2009, pp. 248–255.
- [69] A. Krizhevsky *et al.*, “Learning multiple layers of features from tiny images,” 2009.
- [70] Z. Jiang *et al.*, “All tokens matter: Token labeling for training better vision transformers,” *arXiv preprint arXiv:2104.10858*, 2021.
- [71] Nvidia, “System Management Interface SMI — NVIDIA Developer.” [Online]. Available: <https://developer.nvidia.com/nvidia-system-management-interface>
- [72] Meta, “ViT Github.” [Online]. Available: <https://github.com/facebookresearch/deit>
- [73] AMD/Xilinx, “UG1573:Board evaluation and management Tool.”
- [74] R. Li *et al.*, “Fully quantized network for object detection,” in *CVPR*, 2019, pp. 2810–2819.
- [75] Y. Choukroun *et al.*, “Low-bit quantization of neural networks for efficient inference,” in *ICCVW*. IEEE, 2019, pp. 3009–3018.
- [76] S. K. Esser *et al.*, “Learned step size quantization,” *arXiv preprint arXiv:1902.08153*, 2019.
- [77] Nvidia, “NVIDIA H100 Tensor Core GPU Architecture.” [Online]. Available: <https://resources.nvidia.com/en-us-tensor-core>
- [78] L. Ma *et al.*, “Rammer: Enabling holistic deep learning compiler optimizations with {rTasks},” in *OSDI*, 2020, pp. 881–897.
- [79] A. Li *et al.*, “Automatic horizontal fusion for gpu kernels,” in *CGO*. IEEE, 2022, pp. 14–27.