Edge-MoE: Memory-Efficient Multi-Task Vision Transformer Architecture with Task-level Sparsity via Mixture-of-Experts

Rishov Sarkar[1], Hanxue Liang[2], Zhiwen Fan[2], Zhangyang Wang[2], Cong Hao[1]

[1]School of Electrical and Computer Engineering, Georgia Institute of Technology

[2]School of Electrical and Computer Engineering, University of Texas at Austin

rishov.sarkar@gatech.edu, lhx92505991@gmail.com, {zhiwenfan, atlaswang}@utexas.edu, callie.hao@ece.gatech.edu

Abstract—The computer vision community is embracing two promising learning paradigms: the Vision Transformer (ViT) and Multi-task Learning (MTL). ViT models show extraordinary performance over traditional convolution networks but are commonly recognized as computation-intensive, especially the self-attention with quadratic complexity. MTL uses one model to infer multiple tasks with better performance by enforcing shared representation among tasks, but a huge drawback is that, most MTL regimes require activation of the entire model even when only one or a few tasks are needed, causing significant computing waste. M³ViT is the latest multi-task ViT model that introduces mixture-of-experts (MoE), where only a small portion of subnetworks ("experts") are sparsely and dynamically activated based on the current task. M³ViT achieves better accuracy and over 80% computation reduction and paves the way for efficient real-time MTL using ViT.

Despite the algorithmic advantages of MTL, ViT, and even M³ViT, there are still many *challenges* for efficient deployment on FPGA. For instance, in general Transformer/ViT models, the self-attention is known as computational intensive and requires high bandwidth. In addition, softmax operations and the activation function GELU are extensively used, which unfortunately can consume more than half of the entire FPGA resource (LUTs). In the M³ViT model, the promising MoE mechanism for multi-task exposes new challenges for memory access overhead and also increases resource usage because of more layer types.

To address these challenges in both general Transformer/ViT models and the state-of-the-art multi-task M³ViT with MoE, we propose Edge-MoE, the *first end-to-end* FPGA accelerator for *multi-task ViT* with a rich collection of architectural innovations. <u>First</u>, for general Transformer/ViT models, we propose (1) a novel reordering mechanism for self-attention, which reduces the bandwidth requirement from proportional to constant regardless of the target parallelism; (2) a fast single-pass softmax approximation; (3) an accurate and low-cost GELU approximation, which can significantly reduce the computation latency and resource usage; and (4) a unified and flexible computing unit that can be shared by almost all computational layers to maximally reduce resource usage. <u>Second</u>, for the advanced multi-task M³ViT with MoE, we propose a novel patch reordering method to completely eliminate any memory access overhead. <u>Third</u>, we deliver *on-board implementation and measurement* on Xilinx ZCU102 FPGA, with verified functionality and open-sourced hardware design, which achieves 2.24× and 4.90× better energy efficiency comparing with GPU (A6000) and CPU (Xeon 6226R), respectively. A real-time video demonstration of our accelerated multi-task ViT on an autonomous driving dataset is available in GitHub,[1] together with our FPGA design using High-Level Synthesis, host code, FPGA bitstream, and on-board performance results.

I. Introduction

The Vision Transformer (**ViT**) [2], [7], [18], [28] has enjoyed great popularity in the computer vision field thanks to its impressive performance. Comparing to convolutional neural networks (CNN) which use pixel arrays, ViT divides images into fixed-size patches and treats them as "tokens" in NLP. Embeddings of the tokens will be learned and fed into the transformer encoder together with a positional embedding. ViT has shown extraordinary performance in object detection and semantic segmentation tasks [18], [28].

Meanwhile, Multi-task Learning (**MTL**) is a promising scenario where a single compact algorithm can simultaneously learn many different tasks with a much smaller model size than single-task learning (STL) [31]. In addition, MTL can learn an improved feature representation by sharing representations and utilizing regularizations between related tasks [20], [34]. MTL is important and yet challenging for real-world applications,

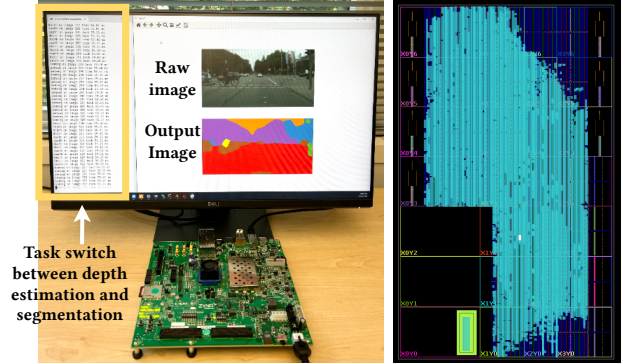[1]https://github.com/sharc-lab/Edge-MoE

Fig. 1. On-board implementation demo for our MTL ViT accelerator with guaranteed functionality and performance. The entire M³ViT runs on the ZCU102 FPGA board; outputs are streamed to laptop only for visualization.

especially when the model will be deployed in an environment with limited computational capability but with real-time requirement. For instance, autonomous driving [13] requires many tasks executed on the same platform, such as lane detection, pedestrian detection, segmentation, etc., where MTL is expected to deliver real-time performance for each task as well as swift task switch. Therefore, *real-time MTL with swift task switch* is in great demand for future AI systems.

While there is a rich amount of work exploring ViT and MTL separately, applying MTL to ViT also has emerged with attractive results. One prevailing type of MTL architectures [8], [17], [21], [25] adopt a shared backbone with independent task-specific head, while another type of MTL architectures [30], [32], [36], [37] make task predictions with a unified decoder and then make further improvements on top of the initial predictions. These models, however, usually have a large number of FLOPs [31] and are not suitable for real-time applications on resource-constrained or latency-sensitive systems. Chen et. al [3] propose a pre-trained multi-task ViT for image processing, composed of multiple pairs of head and tail and a shared transformer structures across the tasks, and Park et. al [22] propose a multi-task vision transformer for COVID-19 diagnosis and severity quantification. A most recent work M³ViT [16] introduces mixture-of-expert (MoE) into MTL ViT. While existing MTL models need to activate the entire backbone unconditionally even only one or a few tasks are needed [8], [17], [21], [31], M³ViT can dynamically and sparsely activate only a small portion of experts, selected by a gating network, for a specific task. M³ViT not only achieves state-of-the-art accuracy on multi-task datasets, but also greatly reduces the model size by more than 80% comparing to other models with similar accuracy, making it appealing for hardware deployment. Fig. 2 illustrates the differences between single-task learning (STL), multi-task learning (MTL), and MTL+MoE [16].

Despite the advancements in MTL with smaller model size and the sparsely activated experts in M³ViT, there are still many **challenges** to achieve real-time inference for multi-task ViT models on FPGA. <u>First</u>, transformers, including ViT models, are notoriously known for the quadratic complexity of self-attention layer computation [4], [7], [35]:
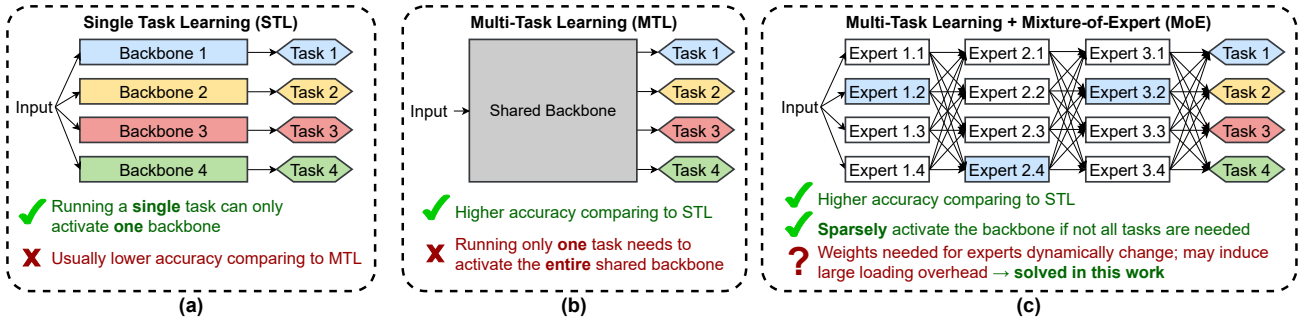
Fig. 2. Pros and cons of single-task learning (STL), multi-task learning (MTL), and multi-task learning with mixture-of-expert (MoE). The most significant advantage of MTL-MoE is **sparsely activated** backbone, which saves both computation and memory footprint. The challenge of **MTL-MoE** is, the activation of "experts" is dynamic, depending on the current image frame. Therefore, although it largely saves the computation and memory, it may induce large weights loading overhead and cancel the benefit. In this work, we propose a novel hardware architecture for **efficient MTL-MoE with zero overhead**.

TABLE I
List of our proposed techniques and their applicable models and benefits.
(Transf.: Transformer; ViT: Vision Transformer; M³ViT: multi-task ViT with mixture-of-expert; Lat.: Latency; Res: Resource)

Proposed Techniques	Transf.	ViT	M³ViT	Benefit
❶ Attention reordering	✔	✔	✔	Lat.
❷ Softmax approximation	✔	✔	✔	Res., Lat.
❸ GELU approximation	✔	✔	✔	Res.
❹ Unified computing unit	✔	✔	✔	Res., Lat.
❺ Patch reordering (MTL)	–	–	✔	Res., Lat.

with N tokens in total, each token needs to compute attention factors with all N tokens including itself. The self-attention computation has became a major bottleneck even for models with smaller size such as DeiT [29] and T2T-ViT [33]. Second, although M³ViT proposes sparsely activated MoE with largely reduced computation, it introduces new challenges to memory access: since experts are dynamically determined on-the-fly, one has to either store all the weights for all experts on-chip with a large memory overhead, or load the required expert only with heavy off-chip data movement. Third, in Transformers and ViT models, softmax operations are extensively used after each attention layer (as opposed to convolution networks where softmax is only used in the final output), which requires a huge amount of non-linear FPGA-unfriendly computations that consume a large amount of resource and execution time (more details in Sec. III-A2). Fourth, in many state-of-the-art Transformers and ViT models, a new type of activation function, GELU (Gaussian Error Linear Unit) [11], has been widely used to improve training efficiency and accuracy [1], [7], [9], [27], [29]; however, its non-linearity introduces a large resource overhead and an obvious accuracy drop on FPGA (more details in Sec. III-A3).

To address these challenges, we propose **Edge-MoE**, a hardware accelerator with innovative architectural techniques, to deliver real-time performance for multi-task ViT model M³ViT, though the proposed techniques are generally applicable to standard Transformer/ViT models. Table I summarizes the proposed techniques, their applicable models, and the benefits. We summarize the contributions as follows:

- To the best of our knowledge, Edge-MoE is the *first end-to-end* accelerator for *multi-task* Vision Transformer, with *on-FPGA implementation and measurement*, verified functionality, and open-sourced hardware design using High-Level Synthesis (HLS). Fig. 1 depicts our on-board implementation with real-time performance on an autonomous driving dataset; a full video clip is available online.²

- For general Transformer/ViT models with common challenges (e.g., heavy self-attention, softmax, and GELU), we propose a collection of innovative techniques, including: ❶ a novel attention reordering mechanism to reduce the required bandwidth from proportion to

²https://github.com/sharc-lab/Edge-MoE/raw/main/demo.mp4

constant; ❷ a single-pass softmax approximation to achieve both high accuracy and fast computation speed; ❸ an accurate and low-cost GELU approximation with extremely low hardware resource; and ❹ a unified and flexible computing unit that can be shared by almost all linear layers, which drastically reduces the resource usage and thus leads to significant speedup. The proposed techniques can be directly applied to any Transformer/ViT model for resource and latency reduction.

- For the advanced multi-task ViT model with mixture-of-expert, M³ViT [16], we propose ❺ a novel patch reordering mechanism to eliminate memory overhead from off-chip data movement, achieving zero-overhead for expert switching and task switching. By deploying a M³ViT on a single FPGA, we demonstrate that it is highly possible for energy-efficient and real-time multi-task ViTs to run on edge devices.

- Since M³ViT is the state-of-the-art ViT model having all components in general Transformers and ViT models and also equipped with advanced MTL and MoE, we use it as our case study without losing any generality. Our accelerator achieves nearly 30 frames per second, 2.24× better energy efficiency than GPU (RTX A6000), and 4.90× better than CPU (Xeon 6226R). Results are measured on Xilinx ZCU102 FPGA evaluation board under 300 MHz.

- Our proposed techniques are *not specific* to FPGAs but can be applied to ASIC designs as well. We use FPGAs only for concrete evaluation.

II. PRELIMINARY AND RELATED WORK

A. Vision Transformers and M³ViT

The Vision Transformer (**ViT**) is first proposed by Dosovitskiy et. al [7] by adapting Transformers in Natural Language Processing (NLP) to processing images for computer vision tasks. Similar to the "tokens" in Transformers, each image is first split into "patches", where each patch is of size $P \times P$ and has P^2 pixels. Then patches will be flattened into vectors, projected to linear embeddings, and fed into to a standard transformer encoder in sequential order, usually with positional embeddings. Each block of the encoder is usually composed of a self-attention layer, normalization layers, a multi-layer perceptron (MLP) layer, and activation layers. In the self-attention layer (blue block in Fig. 3 left), we denote the input patch embeddings by $X \in \mathbb{R}^{L \times d}$ where d is the patch embedding dimension and L is the patch count. Three matrices, Q, K, and V, can be computed as: $Q = W^Q X$, $K = W^K X$, $V = W^V X$, where W^Q, W^K, and W^V are weights. Finally, output $Y = \text{softmax}(QK^T)V \in \mathbb{R}^{L \times d}$. This self-attention has a quadratic complexity in the softmax, where each patch needs to compute its attention score with all n patches.

Atop ViT, **M³ViT** [16] is the state-of-the-art *multi-task* ViT model, which innovatively introduces a mixture-of-expert (MoE) mechanism to sparsely activate only a small portion of the model for each task, aiming to reduce the computation complexity. Fig. 3 left illustrates the model structure of M³ViT. Like the standard ViT, M³ViT consists of a patch embedding module followed by twelve repeated blocks. Inside each block after the self-attention, however, there are two choices: even

2

blocks use a traditional ViT block (yellow block in Fig. 3 left), while odd blocks use an MoE block (pink block). A traditional **ViT block** uses a two-layer MLP with a GELU activation [11]. An **MoE block** is a collection of m "experts", each being a smaller MLP. The resultant output of an MoE layer is the summation of the selected top k experts from m expert candidates using a task-specific gating network. For instance, in the example shown in the figure, there are two gating networks for two tasks. When task A is being executed, it activates expert 1 and $m-1$, while other experts are not computed; when task B is being executed, it activates expert 1 and m and others are disabled. The advantage of this MoE approach is to *sparsely* activate the experts to largely reduce computation. Specifically, in real-world multi-task model inference, not all tasks are always needed. With MoE, only a small portion of task-related experts are selected for computation. Without MoE, however, the entire model must always be computed.

B. Existing ViT Accelerators on FPGA

Several prior studies target the acceleration of Transformer-based models on FPGA, such as VAQF [26], SPViT [12], FTRANS [14], Qi *et al.* [24], Peng *et al.* [23], and Auto-ViT-Acc [15]. These works note that Transformer models are computation- and memory-intensive and are too large to fit on an FPGA. Some solutions adopt various lossy model compression techniques: activation quantization, token pruning, block-circulant matrices (BCM) for weights, and block-balanced and column-balanced block weight pruning. All of these require the use of compression-aware training to avoid a drop in model accuracy. Tackling the ViT acceleration from different angles, our hardware novelties *do not rely on model compression or re-training* and, in fact, are orthogonal to many of the compression techniques proposed in prior works, which can be applied together with our proposed techniques.

In addition, instead of focusing on the entire Transformer/ViT model, Zhang *et al.* [35] propose an FPGA-based self-attention accelerator by weight pruning, while Lu et. al [19] propose a systolic-array design for the attention layer. By contrast, we propose an *end-to-end fully functional ViT model implementation*, which exposes additional challenges concealed by looking at only part of the model.

For *multi-task learning*, the state-of-the-art M³ViT [16] excels in both high algorithm accuracy and low computation complexity, thanks to the MoE mechanism. It suggests a hardware-friendly method for MoE computation, but unfortunately, it only provides an optimistic estimation using FPGA without any actual hardware implementation. Therefore, our work is, to our best knowledge, the *first FPGA accelerator for a multi-task ViT model with MoE*.

III. CHALLENGES

Despite the algorithmic advantages of ViT models and the computational sparsity of M³ViT for multi-task learning, there are still severe challenges to actually deploy ViT and M³ViT models on FPGA to achieve real-time performance, where many of them are being overlooked by existing studies. We highlight a few from two perspectives: challenges that are *general to almost all Transformer/ViT models*, and that are *specific to multi-task M³ViT*.

A. General Challenges for Transformer/ViT

1) High Bandwidth Requirement for Self-attention: It is already well-known that self-attention is computation-intensive [19], [35] and is bandwidth constrained. Without repeating previous statements, we highlight that, achieving low latency either requires an extremely high bandwidth or large on-chip memory to buffer as many weights as possible. Neither is realistic for edge devices with limited resources.

2) Massive Softmax Operations with Unwanted Overflow: M³ViT and other Transformer models make extensive use of softmax operators to compute attention scores for pairs of tokens. M³ViT also uses softmax to compute MoE expert weights for each token. Unfortunately,

softmax uses the $\exp(\cdot)$ function, which is very hardware unfriendly: low-precision computation may yield a large error, but high-precision computation consumes significant resources and latency.

Furthermore, overflows in $\exp(\cdot)$ result in catastrophic errors in softmax results, as they propagate through softmax and throughout the attention mechanism, resulting in downstream task accuracy of almost zero.

Since $\exp(\cdot)$ must be computed many times in softmax, using a fixed-point datatype is preferred on FPGA. However, even small inputs can easily overflow such a datatype. For instance, $\exp(7) \approx 1096.63$, which requires 12 or more integer bits in a signed fixed-point datatype.

3) Expensive Hardware Cost of GELU Activation: Modern Transformers and ViT models, including M³ViT, make use of the GELU (Gaussian Error Linear Unit) activation function [11] for fully connected layers and MLPs instead of ReLUs. GELU can be regarded as a smoother ReLU but its better non-linearity leads to faster and better convergence of neural networks [6], [7], [16], [18]. $\text{GELU}(x)$ is computed as follows:

$$\text{GELU}(x) = x\Phi(x) = x\cdot 0.5(1+\text{erf}(x/\sqrt{2})) \qquad (1)$$

where $\Phi(x)$ is the standard Gaussian cumulative distribution function and $\text{erf}(\cdot)$ is the Gauss error function.

Implementing GELU activation using Eq. (1) is extremely expensive on FPGA: one instance uses 161.8k LUTs (59% of those available on the ZCU102) on 32-bit fixed-point data; thus an approximation must be used.

It is suggested that GELU can be approximated using $\tanh(\cdot)$ [11], but this still consumes significant hardware resources: one instance consumes 18.7k LUTs (6% of available LUTs on ZCU102), while thousands are expected to be computed in parallel. The same work also suggests a sigmoid-based approximation, which uses only 4.7k LUTs (1%) for one instance, but is significantly less accurate.

Thus it is challenging to maintain both high accuracy and low complexity and hardware resources in computing the GELU activation.

4) Duplication of Linear Layer Logic: Transformers and ViT models extensively use fully connected linear layers, including in MLP, patch embedding for self-attention, and linear projection at the end of self-attention. Creating a dedicated hardware module for each linear operation consumes too many resources and limits parallelism; thus aggressive resource sharing across linear layers in different module blocks is needed. However, though the computational nature of the linear layers is the same, their structures differ between different blocks. Variations include: input and output sizes, weight fetch and store address, preferred data parallelism mechanism (partition), and whether to use activations.

The M³ViT model aggravates the challenge by introducing another block type: the MoE block alongside the traditional ViT block (see Fig. 3 left). The ViT blocks use MLPs with a larger hidden dimension than that of the MLPs used in the MoE blocks. Furthermore, due to the nature of mixture-of-experts computation, expert MLPs do not process all tokens; instead, they must accept sparse inputs of only a subset of input tokens.

Therefore, designing a unified computing unit accommodating all types of linear layers to encourage resource sharing and thus to enable larger parallelism is a critical challenge.

B. Model Specific Challenges for MTL M³ViT

1) Memory Accesses for Mixture-of-Experts: A defining feature of the M³ViT algorithm is its use of mixture-of-experts (MoE) to sparsify a large ViT model by selecting, for each input token, a set of "experts" to use to compute its output representation. A gating network scores all m experts with respect to each token, and the k highest-scoring experts determine that token's output. Fig. 9(a–b) depicts how the input image is split into patches and a subset of experts is selected for each patch.

A natural approach to perform the MoE computation is to treat it like any other MLP: first, load all weights for all experts into on-chip BRAM. Then, computation of each token's outputs is straightforward by providing the weights of each of the k selected experts from BRAM as the weight inputs to an MLP. However, this approach requires
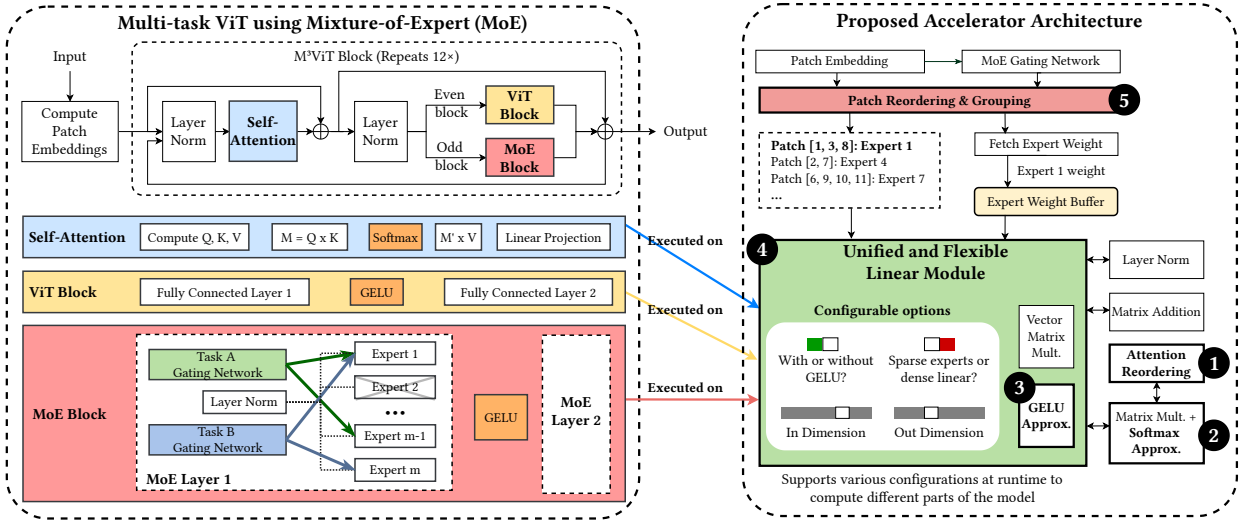
Fig. 3. Left: the multi-task M³ViT architecture with mixture-of-expert (MoE) [16]. It consists of a patch embedding module followed by 12 blocks, each of which contains a self-attention module followed by either a ViT block (on even-numbered blocks) or an MoE block (on odd-numbered blocks). Right: our proposed FPGA architecture with novel techniques, labeled by ❶~❺. ❶~❹ is applicable to general Transformer/ViT models; ❺ is specialized for multi-task MoE inside M³ViT.
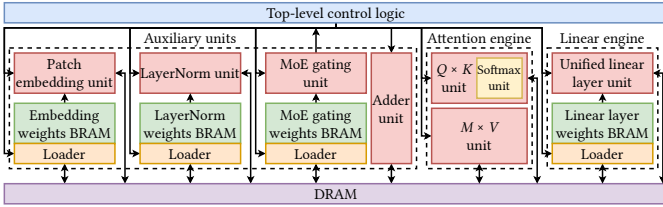

Fig. 4. The system architecture of Edge-MoE.

loading all m experts on-chip. In M³ViT, $m=16$, and whole expert weights cannot fit into the available BRAM resources.

An alternative approach is to compute the patches one by one and fetch experts only when needed, as in Fig. 9(c). For instance, when computing patch 1, since its selected experts are 1 and 4, expert 1's weights are first loaded, followed by expert 4; next, when computing patch 2, expert 1's weights have to be reloaded since they were swapped out. This sidesteps the issue of limited BRAM, but it incurs severe memory delays due to constant reloading of expert weights.

IV. PROPOSED METHODS

In this section, we describe our novel solutions to alleviate the previously mentioned challenges. The proposed overall architecture is presented in Fig. 3 with our proposed novel techniques, labeled from ❶ to ❺, where ❶~❹ are generally applicable to standard Transformers and ViT models, and ❺ is specific to the multi-task M³ViT with MoE. Fig. 4 shows another perspective, including memory hierarchy.

The proposed design is a *fully functional end-to-end* accelerator including the following major components and key techniques:

- Initial patch embedding computation;
- A task-specific gating network to generate expert selection;
- Patch reordering and grouping to consolidate expert weight loading and to eliminate memory access overhead (❺ in Sec. IV-D);
- A unified and flexible linear module which consolidates a large portion of linear layers across the entire model (❹ in Sec. IV-E);
- An efficient attention reordering module for self-attention computation with largely reduced memory access and increased parallelism, overcoming the bandwidth bottleneck (❶ in Sec. IV-A);
- A novel single-pass, accurate, and low-cost softmax approximation module used for self-attention (❷ in Sec. IV-B);
- An accurate and low-cost GELU approximation module being integrated into the unified linear module (❸ in Sec. IV-C);

Note that although the depicted accelerator is for M³ViT, it is easily applicable to standard Transformers and ViT models simply by removing the MoE layer and patch reordering module. In the experiments, we will evaluate performance on standard ViT models. In the following sections, we discuss our proposed key techniques.

A. Attention Reorder for Bandwidth Reduction

Without reordering. The attention mechanism in Transformer models as well as M³ViT is computation-intensive and bandwidth-constrained. Figure 5 (top) illustrates the bandwidth limitation by showing the computation flow of $Q_i \times K_j$ for all the tokens, where $1 \le i \le N$, $1 \le j \le N$, and N is the total number of tokens. To finish such computation for all tokens, N^2 pairs of vector multiplication is needed. A straightforward way is to load Q_i tokens in order, from Q_1 to Q_N; then, for each Q_i, we load K_j tokens from K_1 to K_N, as demonstrated in Fig. 5 (top). The limitation of this method is, increasing attention parallelism also requires an increase in DRAM bandwidth by the same factor. In this example, if one wants to achieve a parallelism of 4, then each Q_i token must be multiplied by four K_j tokens in every iteration; therefore, it requires the four K_j tokens being loaded from DRAM every iteration, requiring four times the bandwidth to access K. As a result, each K_j token is loaded N times, resulting in the transfer of N^2 blocks of data, and the Q_i tokens are loaded once, requiring an additional N blocks. Therefore, to compute the attention in this fashion will need N^2+N times of data transfer to achieve a total latency of N^2/p; therefore, the **bandwidth requirement is proportional to parallelism**, approximately p. The memory requirement is one buffer for Q token and p buffers for K token, totaling $p+1$ buffers.

Our proposed reordering. Addressing this challenge, we introduce a novel reordering strategy, shown in Figure 5 (bottom), that **achieves arbitrary parallelism with a constant bandwidth** for the input matrices. Specifically, for a desired parallelism factor p (4 in the figure), we first cache one batch of p tokens of Q_i in a local buffer (e.g., Q_1 to Q_4 in the first batch); then, during each iteration, we load a new K_j token and multiply it with the p tokens of Q in the local buffer. The memory required is p buffers for Q token and 1 buffer for K token, also $p+1$ buffers in total.

Notice that some certain tokens of Q are not aligned with the start of the K matrix, such as Q_2 in the figure. Thus certain outputs are initially "missing," like $Q_2 \times K_1$, since Q_2 was not in the buffer during the iteration in which K_1 was fetched from DRAM. These "missing" outputs are revisited after the rest of the K matrix is processed, while the next batch of Q is being read. Conveniently, the last output for

Naive matrix multiplication for 4× parallelism
✗ Transfers (**N² + N**) **blocks** of data from DRAM
✗ Requires 4× **bandwidth** to *K* memory

Iter.	Load	Load ×4	Compute	Compute	Compute	Compute
1	Q_1	$K_{1...4}$	$Q_1 \times K_1$	$Q_1 \times K_2$	$Q_1 \times K_3$	$Q_1 \times K_4$
2		$K_{5...8}$	$Q_1 \times K_5$	$Q_1 \times K_6$	$Q_1 \times K_7$	$Q_1 \times K_8$
⋮		⋮	⋮	⋮	⋮	⋮
N/4		$K_{N-3...N}$	$Q_1 \times K_{N-3}$	$Q_1 \times K_{N-2}$	$Q_1 \times K_{N-1}$	$Q_1 \times K_N$
N/4+1	Q_2	$K_{1...4}$	$Q_2 \times K_1$	$Q_2 \times K_2$	$Q_2 \times K_3$	$Q_2 \times K_4$
N/4+2		$K_{5...8}$	$Q_2 \times K_5$	$Q_2 \times K_6$	$Q_2 \times K_7$	$Q_2 \times K_8$
⋮		⋮	⋮	⋮	⋮	⋮
N²/4		$K_{N-3...N}$	$Q_N \times K_{N-3}$	$Q_N \times K_{N-2}$	$Q_N \times K_{N-1}$	$Q_N \times K_N$

Reordered to minimize DRAM bandwidth
✓ Transfers (**N²/4 + N + 3**) **blocks** of data from DRAM
✓ Requires **no extra bandwidth** to *K* memory

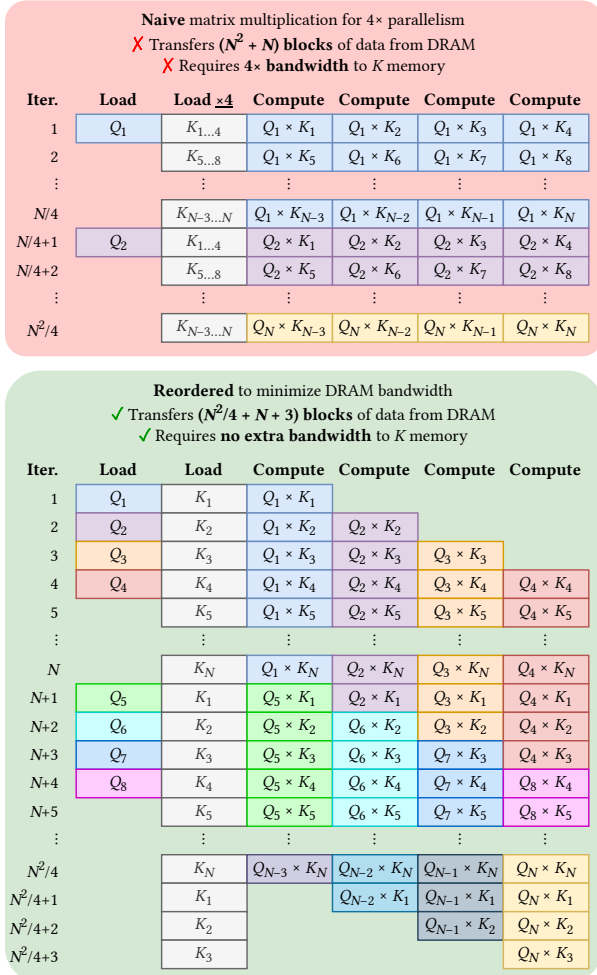| Iter.	Load	Load	Compute	Compute	Compute	Compute
1	Q_1	K_1	$Q_1 \times K_1$			
2	Q_2	K_2	$Q_1 \times K_2$	$Q_2 \times K_2$		
3	Q_3	K_3	$Q_1 \times K_3$	$Q_2 \times K_3$	$Q_3 \times K_3$	
4	Q_4	K_4	$Q_1 \times K_4$	$Q_2 \times K_4$	$Q_3 \times K_4$	$Q_4 \times K_4$
5		K_5	$Q_1 \times K_5$	$Q_2 \times K_5$	$Q_3 \times K_5$	$Q_4 \times K_5$
⋮		⋮	⋮	⋮	⋮	⋮
N		K_N	$Q_1 \times K_N$	$Q_2 \times K_N$	$Q_3 \times K_N$	$Q_4 \times K_N$
N+1	Q_5	K_1	$Q_5 \times K_1$	$Q_2 \times K_1$	$Q_3 \times K_1$	$Q_4 \times K_1$
N+2	Q_6	K_2	$Q_5 \times K_2$	$Q_6 \times K_2$	$Q_3 \times K_2$	$Q_4 \times K_2$
N+3	Q_7	K_3	$Q_5 \times K_3$	$Q_6 \times K_3$	$Q_7 \times K_3$	$Q_4 \times K_3$
N+4	Q_8	K_4	$Q_5 \times K_4$	$Q_6 \times K_4$	$Q_7 \times K_4$	$Q_8 \times K_4$
N+5		K_5	$Q_5 \times K_5$	$Q_6 \times K_5$	$Q_7 \times K_5$	$Q_8 \times K_5$
⋮		⋮	⋮	⋮	⋮	⋮
N²/4		K_N	$Q_{N-3} \times K_N$	$Q_{N-2} \times K_N$	$Q_{N-1} \times K_N$	$Q_N \times K_N$
N²/4+1		K_1		$Q_{N-2} \times K_1$	$Q_{N-1} \times K_1$	$Q_N \times K_1$
N²/4+2		K_2			$Q_{N-1} \times K_2$	$Q_N \times K_2$
N²/4+3		K_3				$Q_N \times K_3$

Fig. 5. The proposed computation reordering for parallelism in self-attention.

TABLE II
Data loading amount, latency, memory requirement, and bandwidth requirement with and without our proposed attention reordering under parallelism p.

Approach	Data Load	Latency	Bandwidth	Memory
w/o reorder	N^2+N	N^2/p	$\sim p$	$p+1$
w/ reorder	$N^2/p+N+p-1$	$N^2/p+p-1$	~ 1	$p+1$

each token is computed just before a new token is to be read into the buffer, thus incurring no idle time between batches. At the end, once all of Q has been read, up to $p-1$ iterations are added to process any remaining "missing" outputs in the last batch.

Since Q is processed in batches of p tokens at a time, each token of K is reused across p tokens, meaning that the whole K matrix is read only $\frac{N}{p}$ times. This results in the transfer of $\frac{N^2}{p}$ blocks of the K matrix, plus up to $p-1$ blocks additionally transferred at the end to account for missing outputs. The Q matrix is still read only once, resulting in a total transfer of $\frac{N^2}{p}+N+p-1$ blocks of data. Given a total latency of $\frac{N^2}{p}+p-1$, the **required bandwidth is approximately 1 regardless the parallelism** p. This reuse of the K matrix for p blocks of Q is also exactly why the baseline strategy needs p times the bandwidth for K to achieve the same parallelism.

Table II summarizes performance without and with our proposed reorder with parallelism p, including the total number of data loading, latency, required bandwidth, and on-chip memory. Without reordering, the bandwidth requirement is proportional to parallelism, while our proposed mechanism reduces the bandwidth to a constant.
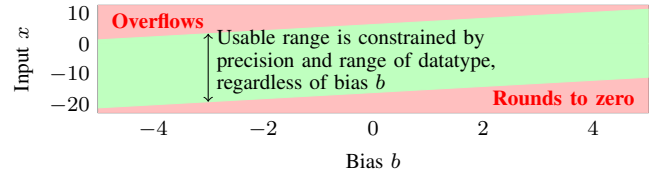

Fig. 6. The usable range (marked in green) of the function $\exp(x-b)$ in 32-bit signed fixed-point with 10 integer bits. Optimal b value depends on the input x.

While we only depict the optimization on $Q \times K$, it can be applied similarly to $M' \times V$ **in reverse:** instead of *producing* up to four elements of $Q \times K$ in each iteration, as shown in Fig. 5 (bottom), the four elements at those indices are *loaded* every iteration. They are then processed using softmax, multiplied by the V matrix (which is loaded the same way as K in Fig. 5 bottom), and accumulated into a cache of four tokens. After every N iterations, rather than *loading* four tokens of Q, the contents of the four-token cache are *written back* to DRAM as the final result of four of the tokens in the $M' \times V$ computation.

B. Single-Pass Softmax Accumulation

1) Dynamic bias for accuracy compensation: As discussed in Sec. III-A2, the biggest challenge of softmax computation on hardware is the overflow of the exponential term $\exp(x)$, which largely affects model accuracy since softmax is extensively used in Transformer/ViT models.

To address the overflow problem to compensate accuracy loss, we propose a *compensation bias*, denoted by b, to re-adjust the output range for $\exp(x_i)$ and $\exp(x_j)$ while computing $\frac{\exp(x_i)}{\sum_{j=1}^N \exp(x_j)}$. Specifically, we subtract b from the arguments of both exponential functions $\exp(x_i)$ and $\exp(x_j)$ to reduce the magnitude of the exponential terms; thanks to the additive property of exponential functions, the compensation does not change the softmax result:

$$\frac{\exp(x_i-b)}{\sum_{j=1}^N \exp(x_j-b)} = \frac{\exp(-b)\exp(x_i)}{\exp(-b)\sum_{j=1}^N \exp(x_j)} = \frac{\exp(x_i)}{\sum_{j=1}^N \exp(x_j)} \quad (2)$$

The challenge is, however, selecting larger bias value b can make the exponential term $\exp(x-b)$ less likely to overflow, but it also introduces accuracy loss. If $x-b$ decreases too much, the value $\exp(x-b)$ may be too small to be accurately represented by fixed-point numbers. Figure 6 demonstrates this trade-off: when the input x is small, larger b will make the value of $\exp(x-b)$ round to zero; when x is large, smaller b will result in overflow. Therefore, an optimal bias b heavily depends on the input value x, and thus *a "one-size-fits-all" approach using a predetermined bias b is not feasible for fixed-point datatype* for softmax computation.

Therefore, we propose to *dynamically* decide the bias value b for each token (with embedding x_i), where $b = \max_{j \in \{1,...,N\}}(x_j)$ which ensures that $\exp(x_j-b) \leq 1$ for all $j \in \{1,...,N\}$, preventing overflows while also preserving accuracy. With this dynamic bias, even if a particular term $\exp(x_j-b)$ experiences loss of precision or rounds to zero, accuracy is still maintained. The reason is, even if $\exp(x_j-b)$ rounds to zero, it implies that x_j is much smaller than $\max(x_j)$, and thus its contribution to the overall softmax calculation is negligible and can be ignored without loss of accuracy.

2) From three-pass softmax to single-pass softmax: Using a dynamic bias, however, implies **an expensive three-pass approach** to compute Eq. (2): **Pass 1** over all tokens $j \in \{1,...,N\}$ to scan all the input values find the optimal bias b, **Pass 2** to compute the sum of exponential terms in the denominator $s = \sum_{j=1}^N \exp(x_j - b)$, and **Pass 3** to compute $\exp(x_i-b)/s$ for each token to finish the softmax computation.

To reduce the computation from three-pass to one-pass to reduce the computation latency, we propose an online algorithm to compute the bias b and softmax denominator summation s simultaneously, effectively combining **Pass 1** and **Pass 2**. The proposed process is shown in Algorithm 1. We begin by initializing the sum to 0 and the bias to the most negative value representable by the fixed-point datatype (symbolized by $-\infty$) (line 1). For each element x_j in the sequence,

Algorithm 1 Online algorithm to compute softmax bias b and denominator summation s simultaneously

Require: x, a list of N elements
Ensure: $b = \max(x)$, $s = \sum_{j=1}^{N} \exp(x_j - b)$
1: $b \leftarrow -\infty$, $s \leftarrow 0$
2: **for** $j \in \{1, ..., N\}$ **do**
3:     **if** $x_j > b$ **then**
4:         $s \leftarrow s \cdot \exp(b - x_j) + 1$
5:         $b \leftarrow x_j$
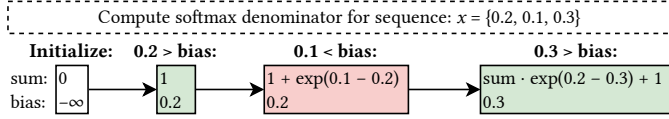6:   **else**
7:         $s \leftarrow s + \exp(x_j - b)$
8:     **end if**
9: **end for**

Compute softmax denominator for sequence: $x = \{0.2, 0.1, 0.3\}$

Initialize:	0.2 > bias:	0.1 < bias:	0.3 > bias:
sum: 0	1	$1 + \exp(0.1 - 0.2)$	$\text{sum} \cdot \exp(0.2 - 0.3) + 1$
bias: $-\infty$	0.2	0.2	0.3

Fig. 7. An example with three elements demonstrates how our online softmax algorithm works no matter the order in which the elements are sorted.

we first determine whether this element is the new maximum and thus should be the new bias. If so, we scale the current sum by $\exp(b - x_j)$, thereby effectively updating the bias from b to x_j for all previous elements in the summation without explicitly doing so, and add 1, which is simply $\exp(x_j - b)$ for $b = x_j$ (line 4–5). Otherwise, we add $\exp(x_j - b)$ to the sum, where b is the existing bias (line 7). Figure 7 shows an example of this algorithm on three elements, $\{0.2, 0.1, 0.3\}$. It shows that, while we are processing the elements one by one, both the sum and the bias b are updated simultaneously in a single pass.

Finally, to avoid a separate **Pass 3** to compute the final softmax results $\exp(x_i - b)/s$ for all the tokens, we keep the original scores x_i stored alongside the bias b and denominator s. The next process that needs the softmax result, such as the self-attention subprocess that reads attention scores and multiplies them with the V matrix, includes the exponentiation and division hardware to compute the softmax result $\exp(x_i - b)/s$ as it reads each incoming element x_i. With pipelining, the added latency for the exponentiation and divison is negligible.

C. Accurate Low-Cost GELU Approximation

As discussed in Sec. III-A3, direct GELU computation is extremely expensive, while existing approximation methods using tanh or sigmoid are either resource-heavy or largely inaccurate. To address this challenge, we propose an accurate and hardware-friendly approximation for GELU with extremely low hardware cost. The proposed approximation method includes four steps of optimization, depicted in Fig. 8.

<u>First</u>, we notice that the value of GELU is similar to ReLU, which is easily computed as $\text{ReLU}(x) = \max(x, 0)$. Therefore, we use ReLU as the base approximation with a calibration $\delta(x)$ based on the input x:
$$\text{GELU}(x) \approx \text{ReLU}(x) - \delta(x) \quad (3)$$
where $\delta(x)$ can be pre-computed, uniformly sampled to fixed-point, and stored in ROM look-up tables. Fig. 8 shows three curves: the ReLU curve in orange, the GeLU curve in blue, and their difference δ in green.

<u>Second</u>, we notice that the $\text{erf}(\cdot)$ in GELU is an odd function holding $\text{erf}(-z) = -\text{erf}(z)$. For $x > 0$, we have:
$$\delta(x) = \text{ReLU}(x) - \text{GELU}(x) = x - x \cdot \tfrac{1}{2}(1 + \text{erf}(\tfrac{x}{\sqrt{2}})) \quad (4)$$
$$\delta(-x) = 0 - (-x) \cdot \tfrac{1}{2}(1 + \text{erf}(\tfrac{-x}{\sqrt{2}})) = x \cdot \tfrac{1}{2}(1 - \text{erf}(\tfrac{x}{\sqrt{2}})) = \delta(x) \quad (5)$$
Therefore, the difference between GELU and ReLU is symmetric about zero, i.e., $\delta(x)$ is an even function. This allows us store only values of $\text{ReLU}(x) - \text{GELU}(x)$ where $x \geq 0$. Fig. 8 shows that only half of the $\delta(x)$ curve is sampled and stored on-chip.

<u>Third</u>, to store the look-up table for discretized values for $\delta(x)$, since $0 \leq \text{ReLU}(x) - \text{GELU}(x) < 1$ for all $x \in \mathbb{R}$, we can store only
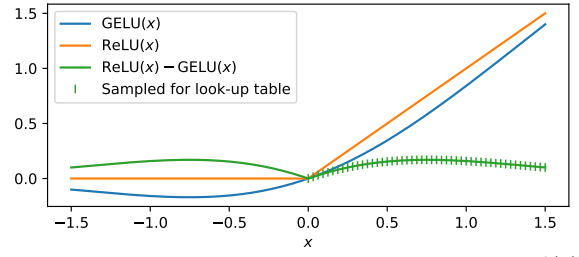

Fig. 8. GELU approximation using ReLU and a calibration function $\delta(x)$. $\delta(x)$ is an even function, so we uniformly sample $\delta(x) > 0$ and store the discretized values in a look-up table.

the unsigned 22 fractional bits of the high-precision 32-bit fixed-point datatype used for GELU approximation, which maintains full precision without requiring storage of any of the integer bits.

<u>Fourth</u>, we truncate the look-up table at the point where $\text{GELU}(x)$ rounds to $\text{ReLU}(x)$ based on the 32-bit fixed-point datatype; for any query value x outside this range, we simply use $\text{ReLU}(x)$ as a sufficiently precise approximation of $\text{GELU}(x)$. In addition, the look-up table step size is chosen to be a negative power of two. This makes the division operation required to compute the look-up table index equivalent to a bit shift, making it very efficient and cheap to implement in hardware.

D. Expert-by-Expert Computation Reordering

As described in Sec. III-B1, MoE models such as M³ViT have an additional challenge beyond standard Transformer-based models, which is to process multiple experts in an unpredictable access pattern without constantly reloading expert weights. Our proposed method is to reorder the token-by-token computations to be performed expert-by-expert instead, as shown in Figure 9(d). During the processing of the gating network, when the top-k expert MLPs are selected for each token, the token indices are added to per-expert queues for later processing.

We construct a metaqueue of all experts whose queues have nonzero length. This allows us to skip the loading step of any experts not used in the current MoE block, such as expert 3 in the figure. Then, for each expert in the metaqueue, we load the expert weights and biases into our unified linear layer module and generate all outputs for that expert. The gating network assigns scores for each token/expert pairing, and each expert's output for a given token is weighted by the score before being accumulated onto the existing partial output for that token. Tokens not in the expert's previously generated queue are skipped entirely.

By pre-aggregating a queue of all tokens that an expert MLP needs to compute, we consolidate the loading of each expert's weights and biases and maximize their reuse. Furthermore, with ping-pong buffering, we can completely hide the loading latency of most expert MLPs, except in the case of the first expert in the metaqueue or any cases of workload imbalance (where the preceding expert has only a few tokens to compute and finishes its computation early while the next expert is still loading).

E. Unified Linear Layer Module

Across the entire M³ViT model, linear layers with different input/output dimensions are present in the following blocks: (1) on dense inputs, from input dimension to ViT block hidden dimension; (2) on dense inputs, from ViT block hidden dimension to output dimension; (3) on sparse inputs, from input dimension to MoE block hidden dimension; (4) on sparse inputs, from MoE block hidden dimension to output dimension; (5) on dense inputs, from input dimension to output dimension. Letting each linear layer consume its own dedicated hardware will result in a large resource waste, particularly DSPs, which will largely constrain the maximum parallelism and harm latency. We consolidate all linear layers in the model into one module, which enables all linear layers to take advantage of high parallelism. As shown in Fig. 3 right, the unified linear module (green block) can process multiple
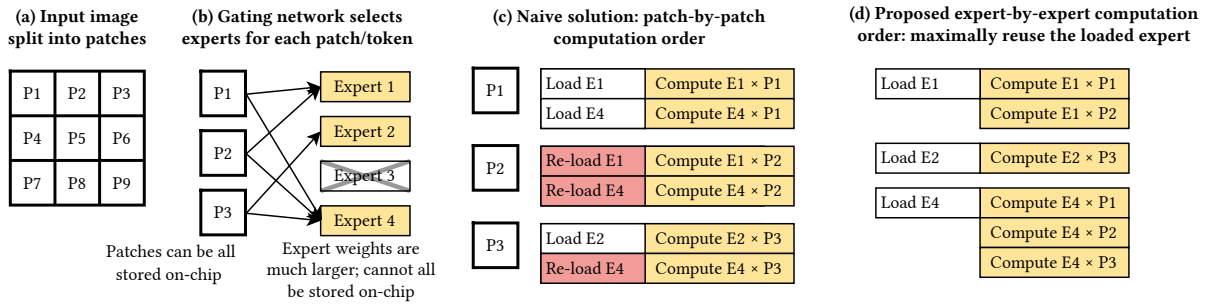
6

Fig. 9. Our expert-by-expert computation reordering strategy (d). Comparing with the naive patch-by-patch method, the proposed method maximally reuse each loaded expert and thus each expert only needs to be loaded once. This method eliminates all potential memory and data movement overhead introduced by MoE.

TABLE III
Prevailing ViT models and multi-task M³ViT with mixture-of-expert (MoE), and their latency reduction by applying the proposed techniques.

| Model | Layers | Dimensions | | Heads | Params | Latency (ms) | | |
		Hidden	MLP			w/o opt.	w/ opt.	Speedup
ViT-Base [7]	12	768	3072	12	86M	4061.1	414.32	9.80×
ViT-Large	24	1024	4096	16	307M	14264	1450.6	9.83×
ViT-Huge	32	1280	5120	16	632M	29502	2997.9	9.84×
DeiT-Small [28]	12	384	1536	6	22M	1063.6	109.00	9.76×
DeiT-Base	12	768	3072	12	86M	4061.1	414.32	9.80×
M³ViT+MoE [16]	12	192	768	3	7M	353.44	34.64	10.20×

TABLE IV
CPU and GPU comparisons. The batch size is 1 image.

Device	Latency	Power	Energy	Frequency	Δ_m Acc.
CPU	169.72 ms	14.53 W	2.466 J (4.90×)	2500 MHz	+0.76%
GPU	13.73 ms	82.24 W	1.129 J (2.24×)	1800 MHz	+0.76%
Edge-MoE	34.64 ms	14.54 W	0.504 J (1.00×)	300 MHz	+0.67%

types of layers with flexible run-time configuration, including variable input/output dimension, dense or sparse input, and whether to use GELU. **Handling variable input/output dimensions**. In HLS, a linear layer is usually implemented as a nested loop, where the outer loop iterates over the output dimension and the inner loop iterates over the input dimension. HLS can create one pipeline from both loops only when the inner loop has a constant bound. But since linear layers in Edge-MoE have differing input dimensions, this bound cannot be constant. Thus the loop cannot be flattened into one pipeline, resulting in severe delays, as an entire output dimension must be processed before the next can start.

To address this limitation, we propose to use a single pipeline that tracks virtual loop indices in separate registers, incremented manually inside the loop. For instance, a register `j` can represent the inner loop index, starting at 0 and incrementing every loop until the input dimension is reached, at which point it is reset to 0 and another register `i`, representing the output dimension, is incremented.

The unified linear layer accepts weights and biases in a blocked format, where a single block represents all weights and biases needed for a single cycle of computation. A unified weight loading module is used to read weights from their sequential form from off-chip DRAM into BRAM in the required blocked format. The unified module consists of a streaming dataflow supporting inter-token pipelining to minimize latency. **Handling hybrid fixed-point quantization schemes.** While unifying all linear layers, the bias loading modules are also unified. However, different layers have different fixed-point datatypes for biases to maintain accuracy. For instance, self-attention uses 16-bit biases with 7 integer bits, but the ViT and MoE blocks use only 5 integer bits for higher precision. We separately convert these to a single wider bias type, with enough integer bits to cover the range and enough fractional bits to cover the precision of both datatypes.

Handling sparse and dense inputs. To be able to handle both sparse and dense inputs simultaneously for the MoE and non-MoE layers, the unified linear layer directly controls the processes reading and writing from DRAM. The reader and writer processes each contain two submodules for direct (dense) and indirect (sparse, indexed by per-expert queues) DRAM accesses, and the top-level module passes a flag to indicate which should be used at any given time. The indirect writer submodule also supports a weighted accumulation of the linear layer output atop the existing output buffer rather than overwriting it, thus enabling accumulation of

per-expert outputs directly without an additional aggregation step.

Additionally, a flag controls whether the writer process should apply the GELU function, which allows the MLPs in the ViT and MoE blocks to incorporate the activation function without an extra step. Thanks to pipelining, the latency added by this activation function is negligible.

F. Gating Network for Multi-Task

The use of Mixture-of-Experts is particularly important in M³ViT, as it is the key to enabling efficient Multi-Task Learning. Figure 3 (left) demonstrates how M³ViT uses MoE in a multi-task scenario: separate gating networks are used for each task in order to select the best experts for a given combination of token and downstream task. Our implementation loads gating network weights from DRAM only as needed for computation, enabling easy zero-overhead switching between tasks simply by updating the pointer to the task-specific gating network.

V. EXPERIMENTS

We conduct several *on-board, verified* experiments to demonstrate the effectiveness of our proposed methods. All on-board code is implemented through High-Level Synthesis through Xilinx Vitis HLS 2021.1. We deploy bitstreams to Xilinx ZCU102 FPGA and use the PYNQ library for host code. The clock frequency of our design is 300 MHz. All experiments use the Cityscapes dataset [5], with images of size 128×256 split into patches of size 16×16.

Our main comparison is with CPU and GPU baselines and the FPGA design without our proposed techniques. We find it difficult to make a fair comparison with prior works for two main reasons. First, we target ViT acceleration from a different angle than prior works and do not rely on model compression or re-training; these techniques are orthogonal to ours and can be applied together. Second, prior works in ViT acceleration lack an end-to-end on-board implementation, which exposes difficulties to compare with.

A. Comparison with CPU and GPU on M³ViT

We first evaluate our accelerator for M³ViT since it is the state-of-the-art multi-task ViT model, which uses standard ViT as its backbone but also is equipped with MoE features, making it more challenging than standard ViT models. Therefore, we choose M³ViT as our primary baseline, evaluated on the semantic segmentation and depth estimation tasks. We deploy our accelerator onto ZCU102 FPGA and compare against CPU (Intel Xeon 6226R) and GPU (NVIDIA RTX A6000) baselines implemented in PyTorch using FastMoE [10], a library that uses scatter/gather techniques for optimized MoE computation. Power is measured

7

TABLE V
Ablation study of our proposed techniques. All the latency, resource, and accuracy values are measured on-board. The baseline is a fully functional M³ViT accelerator design without our proposed techniques. Applying all six techniques can result in more than 18× speedup and no accuracy drop.

| Architecture	Latency (Speedup)	Hardware resources				Sem. seg. (mIoU ↑)	Depth est. (RMSE ↓)	MTL acc. gain Δ_m (↑)
		BRAM	DSP	LUT	FF			
Baseline w/o our proposed techniques	650.3 ms (1.00×)	84.6%	65.9%	60.8%	46.9%	63.8066	0.0373	+0.58%
+ Expert-by-expert reordering (§IV-D)	433.4 ms (1.50×)	84.3%	65.9%	59.0%	45.4%	63.8066	0.0373	+0.58%
+ Single-pass dynamic bias softmax (§IV-B)	353.4 ms (1.84×)	84.0%	67.9%	59.0%	45.7%	63.8066	0.0373	+0.58%
+ Accurate, low-cost GELU (§IV-C)	212.9 ms (3.05×)	64.1%	63.0%	45.5%	34.9%	63.8491	0.0372	+0.67%
+ Unified linear layer module (§IV-E)	104.3 ms (6.23×)	47.5%	57.3%	44.1%	27.0%	63.8491	0.0372	+0.67%
+ Attention reordering on $Q \times K$ (§IV-A)	59.2 ms (10.98×)	48.1%	61.6%	46.5%	27.5%	63.8491	0.0372	+0.67%
+ Attention reordering on $M' \times V$ (§IV-A)	34.6 ms (18.77×)	50.1%	76.3%	46.8%	29.4%	63.8491	0.0372	+0.67%
Software baseline accuracy (PyTorch)						63.9450	0.0372	+0.76%
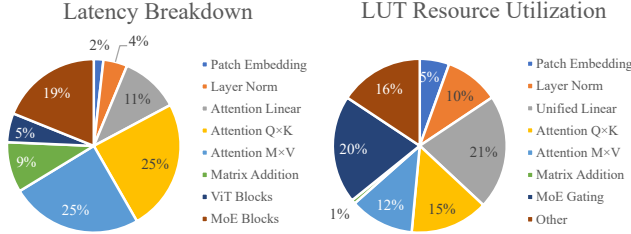
Fig. 10. A breakdown of latency and LUT usage in our FPGA implementation.

using Intel RAPL and NVIDIA SMI, respectively. We use 16-bit fixed-point weights and 32-bit fixed-point activations. Results are in Table IV.

First, in terms of accuracy, both the software baseline M³ViT and Edge-MoE outperform single-task learning (STL) baselines (Δ_m Acc. in Table IV): software M³ViT outperforms STL by +0.76%, and Edge-MoE on FPGA performs +0.67% better – a drop of only 0.09%. Second, in terms of energy efficiency, Edge-MoE on FPGA achieves significant savings: 4.90× over CPU and 2.24× over GPU.

B. Standard ViT Models

We also evaluate how well our optimizations can reduce the latency of several different ViT models when implemented on FPGA.

Results in Table III show consistent improvement from our proposed methods ranging from 9.76× to 9.84× on non-M³ViT models, as well as over 10× speedup on M³ViT with Mixture-of-Experts. The reliable speedup gained from our optimizations shows that, besides our model-specific techniques for M³ViT, our methods are generally applicable and effective when applied to any ViT model.

C. Latency and Resource Breakdown

To understand which parts of our model are the most expensive and which take the most time to compute, we measure the on-board latency and resource usage of the different components of our M³ViT implementation. Figure 10 displays our findings.

Even at 4× parallelism, the attention multiplications $Q \times K$ and $M' \times V$ take half of the total computation time, demonstrating the necessity of accelerating this computation.

Additionally, the effectiveness of our unified linear layer stands out: it takes the largest portion of LUT resources, but as a result, it greatly accelerates the attention linear layers, ViT blocks, and MoE blocks, which take only 35% of the overall latency *combined*.

D. Ablation Study

In Table V, we conduct an ablation study to evaluate the effect of our optimizations on latency, resource usage, and accuracy. We show the incremental progression of these factors as each of our key techniques are applied, including the indirect effect of lower-cost hardware resources enabling greater increases in parallelism.

We find that each of our features leads to a noticeable speedup. Additionally, most hardware resources follow a downward trend over

time, demonstrating empirically that our techniques are hardware-friendly. However, the number of DSPs used remains roughly the same throughout most of the ablation study, as consistent increases in parallelism roughly negated the effect on the number of DSPs.

We call attention to three rows of Table V in particular:

1) Single-pass softmax with dynamic bias: Due to the challenges caused by overflow of $\exp(\cdot)$ described in Sec. III-A2, all architectures use a dynamic bias, as in Sec. IV-B1. This row introduces our single-pass softmax from Sec. IV-B2, rather than three passes.

2) Accurate, low-cost GELU: Most of our optimizations do not affect the downstream task accuracy at all and are mathematically equivalent to their unoptimized versions. However, our accurate, low-cost GELU implementation supersedes a less accurate sigmoid-based approximation from previous architectures, which is described in Sec. III-A3. Not only does our calibration-based approach increase the approximation accuracy, it also further reduces resource usage, allowing for higher parallelism which results in 1.66× speedup over the previous architecture.

3) Unified linear layer module: Our proposed unified linear layer module led to the single largest relative change in latency in the table, over 2× faster than the previous architecture, while also reducing the usage of all four hardware resources. As described in Sec. IV-E, the unified linear layer supersedes five dedicated hardware modules for separate linear layers, thereby dramatically reducing the number of resources consumed. This, in turn, allowed for significantly higher parallelism in the unified linear layer, which benefited all parts of M³ViT that were previously using a dedicated module, while still leaving reduced resource utilization.

VI. CONCLUSION

In this paper, we proposed a novel FPGA accelerator with a collection of innovative techniques for Vision Transformer (ViT) models and a state-of-the-art multi-task ViT using Mixture-of-Expert (MoE). To the best of our knowledge, this is the first end-to-end FPGA implementation for multi-task ViT, with verified functionality on-board. We propose five key techniques: (1) an attention computation reordering mechanism which reduces the bandwidth requirement from proportion to constant, regardless of parallelism; (2) a single-pass softmax approximation achieving both high accuracy and speed; (3) an accurate GELU activation approximation with extremely low hardware cost; (4) a unified and flexible linear layer for aggressive resource sharing; (5) a novel expert-by-expert MoE computing mechanism requiring zero memory and data transfer overhead. The proposed techniques can reduce latency by more than 18× applying to M³ViT and more than 9× applying to standard ViT models. Comparing with CPU and GPU, our design achieves 4.90× and 2.24× better energy efficiency, respectively. On the multi-task self-driving car dataset, we achieve nearly 30 frames per second, showing promising performance for MTL ViT in real-world applications.

ACKNOWLEDGEMENTS

References

[1] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "Vivit: A video vision transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6836–6846.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[3] H. Chen, Y. Wang, T. Guo, C. Xu, Y. Deng, Z. Liu, S. Ma, C. Xu, C. Xu, and W. Gao, "Pre-trained image processing transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 299–12 310.

[4] Q. Chen, C. Sun, Z. Lu, and C. Gao, "Enabling energy-efficient inference for self-attention mechanisms in neural networks," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2022, pp. 25–28.

[5] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[8] Y. Gao, J. Ma, M. Zhao, W. Liu, and A. L. Yuille, "Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3205–3214.

[9] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, "Transformer in transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 908–15 919, 2021.

[10] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, "FastMoE: A fast mixture-of-expert training system," Mar. 2021.

[11] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[12] Z. Kong, P. Dong, X. Ma, X. Meng, W. Niu, M. Sun, B. Ren, M. Qin, H. Tang, and Y. Wang, "SPViT: Enabling faster vision transformers via soft token pruning," Dec. 2021.

[13] D.-G. Lee, "Fast drivable areas estimation with multi-task learning for real-time autonomous driving assistant," *Applied Sciences*, vol. 11, no. 22, p. 10713, 2021.

[14] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "FTRANS: Energy-efficient acceleration of transformers using FPGA," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '20. New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 175–180.

[15] Z. Li, M. Sun, A. Lu, H. Ma, G. Yuan, Y. Xie, H. Tang, Y. Li, M. Leeser, Z. Wang *et al.*, "Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization," *arXiv preprint arXiv:2208.05163*, 2022.

[16] H. Liang, Z. Fan, R. Sarkar, Z. Jiang, T. Chen, Y. Cheng, C. Hao, and Z. Wang, "M³vit: Mixture-of-experts vision transformer for efficient multi-task learning with model-accelerator co-design," in *36th Annual Conference on Neural Information Processing System (NeurIPS 2022)*, December 2022.

[17] S. Liu, E. Johns, and A. J. Davison, "End-to-end multi-task learning with attention," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 1871–1880.

[18] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[19] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer," in *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*. IEEE, 2020, pp. 84–89.

[20] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi, "Modeling task relationships in multi-task learning with multi-gate mixture-of-experts," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1930–1939.

[21] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3994–4003.

[22] S. Park, G. Kim, Y. Oh, J. B. Seo, S. M. Lee, J. H. Kim, S. Moon, J.-K. Lim, and J. C. Ye, "Multi-task vision transformer using low-level chest x-ray feature corpus for covid-19 diagnosis and severity quantification," *Medical Image Analysis*, vol. 75, p. 102299, 2022.

[23] H. Peng, S. Huang, T. Geng, A. Li, W. Jiang, H. Liu, S. Wang, and C. Ding, "Accelerating transformer-based deep learning models on FPGAs using column balanced block pruning," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, Apr. 2021, pp. 142–148.

[24] P. Qi, Y. Song, H. Peng, S. Huang, Q. Zhuge, and E. H.-M. Sha, "Accommodating transformer onto FPGA: Coupling the balanced model compression and FPGA-implementation optimization," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*. New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 163–168.

[25] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, "Latent multi-task architecture learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4822–4829.

[26] M. Sun, H. Ma, G. Kang, Y. Jiang, T. Chen, X. Ma, Z. Wang, and Y. Wang, "VAQF: Fully automatic software-hardware co-design framework for low-bit vision transformer," Feb. 2022.

[27] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit *et al.*, "Mlp-mixer: An all-mlp architecture for vision," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 261–24 272, 2021.

[28] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*, vol. 139, July 2021, pp. 10 347–10 357.

[29] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 347–10 357.

[30] S. Vandenhende, S. Georgoulis, and L. V. Gool, "Mti-net: Multi-scale task interaction networks for multi-task learning," in *European Conference on Computer Vision*. Springer, 2020, pp. 527–543.

[31] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool, "Multi-task learning for dense prediction tasks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[32] D. Xu, W. Ouyang, X. Wang, and N. Sebe, "Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 675–684.

[33] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 558–567.

[34] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3712–3722.

[35] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, "Algorithm-hardware co-design of attention mechanism on fpga devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[36] Z. Zhang, Z. Cui, C. Xu, Z. Jie, X. Li, and J. Yang, "Joint task-recursive learning for semantic segmentation and depth estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 235–251.

[37] Z. Zhang, Z. Cui, C. Xu, Y. Yan, N. Sebe, and J. Yang, "Pattern-affinitive propagation across depth, surface normal and semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4106–4115.