HMix : An Efficient Hardware Accelerator for Quantized MLP-Mixer Inference

Dhananjay Rao Thallikar Shyam
Indian Institute of Science
Bengaluru, India
dhananjay.shyam@fsid-iisc.in

Shashank Nag
The University of Texas at Austin
Austin, USA
shashanknag@utexas.edu

Lizy K. John
The University of Texas at Austin
Austin, USA
ljohn@ece.utexas.edu

Abstract

Computer vision workloads are increasingly deployed at the edge, driving demand for efficient, reconfigurable accelerators that balance accuracy, performance, resource utilization, and power consumption. While Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) dominate current vision pipelines, the MLP-Mixer is a deviation by utilizing an all-MLP architecture while achieving competitive accuracy. However, the alternating Token- and Channel-mixing structure of MLP-Mixer introduces frequent tensor transpositions, resulting in irregular memory access patterns that pose significant challenges for efficient hardware implementation, particularly on resource-constrained edge platforms.

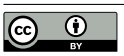
Field-Programmable Gate Arrays (FPGAs) are well-suited for edge deployment due to their energy efficiency and runtime reconfigurability, but their limited on-chip memory bandwidth and resources make supporting high-throughput, transpose-heavy workloads non-trivial. In this work, we present HMix, the first dedicated FPGA-based accelerator for quantized MLP-Mixer inference targeting low-power edge applications. HMix introduces a systolic-array-based dataflow co-designed with a novel "stall-free" transpose mechanism that enables parallel access along orthogonal matrix dimensions while minimizing memory bank conflicts. This design maximizes on-chip arithmetic intensity, and alleviates the performance penalties associated with off-chip DRAM accesses.

Additionally, our proposed dataflow reduces the on-chip memory footprint by 16% compared to naive implementations. Experimental results demonstrate that HMix achieves at least 50% lower resource utilization compared to previous FPGA-based ViT accelerators, while consuming a fraction of the DSP resources and power. These results establish HMix as an efficient solution for the deployment of MLP-based vision models on FPGA-based edge platforms.

Keywords

MLP-mixer, Hardware Accelerator, Systolic Array, FPGA

ACM Reference Format:
Dhananjay Rao Thallikar Shyam, Shashank Nag, and Lizy K. John. 2026. HMix : An Efficient Hardware Accelerator for Quantized MLP-Mixer Inference. In *Proceedings of the 23rd ACM International Conference on Computing Frontiers (CF '26), May 19–21, 2026, Catania, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3801487.3801834



CF '26, Catania, Italy

ACM ISBN 979-8-4007-2568-5/2026/05
https://doi.org/10.1145/3801487.3801834

1 Introduction

Computer vision, a subfield of Artificial Intelligence, has gained momentum in its practical applicability. Computer vision has established an utilitarian role in facial recognition, object detection, image reconstruction, and video motion analysis used in applications such as healthcare, advanced driver assistance systems (ADAS), quality control in manufacturing, and military surveillance [22, 35]. Another factor leading to the large-scale success of computer vision is the ability to deploy these models on resource-constrained low-power FPGAs and embedded systems, which is essential for real-time applications [8, 16].

The predominant state-of-the-art (SOTA) computer vision models are based on convolutions (Convolutional Neural Networks (CNNs) [14, 24, 33]), and more recently, attention-based Vision Transformers (ViTs) [9, 27, 37]. The CNN structure involves convolutional layers with differing filter sizes and strides, and the feature map's spatial resolution typically reduces across successive layers. Thus, any efficient hardware acceleration of CNNs requires a highly configurable hardware architecture paired with complex dataflows, as demonstrated in previous work [5, 6, 10, 20].

ViTs consist of alternating multi-head self-attention (MHSA) and multi-layer perceptron (MLP) blocks, both primarily implemented using general matrix multiplications (GEMMs). The MHSA exhibits quadratic complexity with respect to the input sequence length (number of image patches) and incurs significant memory traffic due to attention score computation and softmax operations, making it memory-bound despite being compute-intensive [11].

The MLP-Mixer [36], a recent computer vision model, aims to address these bottlenecks. It maintains an isotropic architectural design (the dimensions of the activation matrix remain the same across all layers) with a series of alternating Token- and Channel-mixing MLPs (TM and CM), using residual (skip) connections, Layer Normalization (LN) [3] and Gaussian Error Linear Unit (GeLU) [15] activation function (Figure 1).

The advantage that the MLP-Mixer brings to the table over CNNs and ViTs is its much simpler architecture. The MLP-Mixer has an isotropic design, where the dimensions of the feature and weight matrices remain the same throughout [36], potentially supporting a more straightforward hardware acceleration compared to CNNs. In addition, the MLP-Mixer model was demonstrated to be significantly more resilient to input perturbations compared to standard CNN models [36], such as ResNet50 [14]. When stacked against the ViT, the MLP-Mixer replaces the quadratic-complexity MHSA in the ViT with a linear-complexity Token-mixing (TM) MLP, offering a more scalable alternative to the ViT.

Though the MLP-Mixer architecture offers the potential for efficient hardware acceleration, there are certain considerations which make such an effort non-trivial and challenging, especially for resource constrained edge devices. The TM MLP operates along the patch dimension of the input (Figure 1), while the CM MLP operates on the channel dimension, which effectively necessitates a transpose of the entire input after each MLP due to the alternating TM and CM structure. When feeding data to a parallel architecture, such as a systolic array [17, 21], on an edge FPGA, this transpose can result in stalls due to limitations in the number of ports of on-chip block RAMs (BRAM) (two per BRAM block). Moreover, there is a need for a highly power-efficient and lightweight implementation of this transpose operation when targeting such edge devices.

Optimizing resource utilization, BRAM footprint, power consumption, and maximizing the arithmetic intensity of data transferred from off-chip memory (DRAM) must be addressed jointly with the chosen dataflow. Previous work on accelerating Vision Transformers [18, 26, 28, 30, 34, 40, 41] is not optimized for inferencing MLP-Mixers in edge environments. These designs are resource- and power-intensive or lack efficient mechanisms to support the frequent tensor transpositions required by the Token- and Channel-mixing operations in MLP-Mixer. This gap motivates HMix, a hardware accelerator designed for efficient, quantized MLP-Mixer inference on resource-constrained edge platforms.

HMix utilizes the systolic-array architecture to achieve optimal parallelism in the MLP layers, while providing power and resource efficiency. To supply the necessary bandwidth of data for both the TM and CM MLPs operating on orthogonal dimensions of the activation matrix, we propose a "stall-free" transpose mechanism co-designed with the data access flow, with very minimal hardware and power overheads. Inspired by [4] and [30], we implement a general matrix multiplication (GeMM) dataflow that enables a 16% reduction in the BRAM footprint compared to a naive general matrix multiplication (GeMM) for MLP layers. With quantized implementation playing a key role in efficient low-power implementation on an FPGA, we provide an end-to-end quantized implementation, including the non-linear operations (GeLU and Layer Normalization). The results demonstrate that our design consumes a fraction of DSPs and power, achieving at least a 50% reduction in resource utilization compared to other ViT accelerators. We also show competitive energy efficiency (fps/W). In summary, the following are our contributions to this work:

- We propose HMix, the first dedicated MLP-Mixer accelerator for efficient quantized inference on the edge.
- We implement an efficient dataflow for the MLP-Mixer that minimizes hardware utilization and power consumption.
- We implement an efficient "stall-free" transpose unit to meet the bandwidth requirements of the systolic array, catering to the irregular data-access pattern of the MLP-Mixer.
- We provide a complete end-to-end quantized implementation of MLP-Mixer, including non-linear GeLU activation and Layer Normalization.

The remainder of the paper is organized as follows. Section 2 covers the background, and Section 3 presents the motivation for our work. Section 4 decribes our data flow and hardware design. Section 5 discusses the results, and Section 6 concludes the paper.

2 Background

2.1 MLP-Mixer

Processing of images using machine learning models involves identifying patterns (i) between different spatial locations and (ii) between features at a particular spatial location. The MLP-Mixer achieves this by using two types of MLPs: Token mixing (TM) and Channel mixing (CM).

The overall architecture of one layer of the MLP-Mixer architecture is described in Figure 1. The input image is first broken down into patches, and several filters are applied to each patch to encode the image into an activation table. This input table is then passed to the mixer stage.

In this work, we primarily consider the B/16 MLP-Mixer, with 12 mixer layers. Both the Token and Channel mixing MLP are two layered, with the output dimension equal to the input dimension, thus ensuring an isotropic flow of the input dimensions. The Token-mixing MLP (operating on the patch dimension) has a hidden-layer dimension of 384, and the Channel-mixing MLP (operating on the channel dimension) has a hidden-layer dimension of 3072. Both MLPs involve a GeLU activation layer after the first layer.

Each MLP operation is preceded by a layer normalization of the previous output. There is a residual connection at the output of each MLP. The output after the final mixer layer is passed through a classifier head, whose dimensions depend on the target dataset. In this work, we use the Imagenet1k [32] dataset with 1000 classes.

2.2 Inter-layer GeMM Computation

Generally, most GeMM optimizations focus on the operations of a single neural network layer. The work in [4] proposes an optimization between the layers of a neural network, or an inter-layer optimization, where the computed activation from the first layer in the MLP is broadcast to compute partial sums of the following layer. The advantage stated in [4] is that with sufficient hardware support, the computation of the two layers can be pipelined. However, in this work, we have used this scheme to reduce the resource utilization and on-chip BRAM footprint, as discussed in Section 4.1. This method essentially overlaps the execution of the two layers of the MLP, fitting the two layered MLP structure of the mixer.

2.3 Quantization

Quantization replaces single-precision floating-point GEMM operations with lower-precision integer arithmetic, reducing operand bit-widths and intermediate storage [12, 13]. This increases arithmetic intensity, improves compute throughput, and reduces memory bandwidth and power consumption, while typically preserving model accuracy [19, 23, 25].

In this work, we use the symmetric quantization methodology, which we elaborate. The objective is to use inputs and weights of INT8 precision for multiplications, while accumulating in INT32 precision. The symmetric representation of a real number R is $S \times q$, where S represents another real number, called the scale factor, and q being the quantized counterpart, is an integer quantity. Now, consider two real numbers $I = S_I \cdot q_I$ and $W = S_W \cdot q_W$ representing the input and weight in a matrix multiplication. An output element in the resultant matrix is given by the following equation:

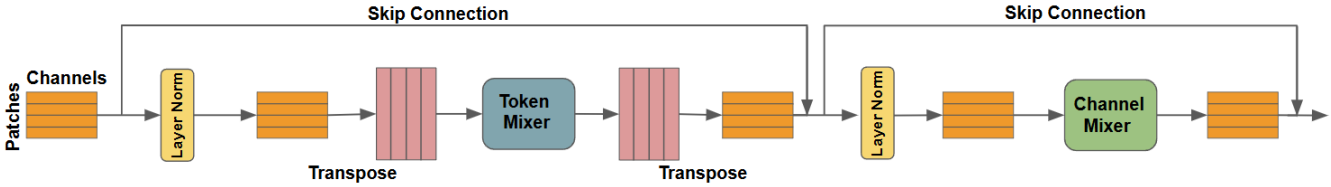


Figure 1: The architecture of one MLP-Mixer layer. The complete model involves the cascading of multiple layers, thus requiring alternate Token Mixing and Channel Mixing operations.

$$Out(i,j) = S_I \cdot S_W \sum_k q_I(i,k) \cdot q_W(k,j) \tag{1}$$

At the end of this computation, the resultant is a 32 bit integer. This has to be reduced to INT8 precision before forwarding it for next layer computation. This is done using the following re-quantization method [19, 23]:

$$\text{Out(i,j)} = S_{net} \cdot \lfloor \frac{S_I \cdot S_W}{S_{net}} \rfloor \cdot (q_{net}) = S_{net} \cdot ((b \cdot q_{net}) << c) \tag{2}$$

This effectively converts the ratio of the scaling factors into a multiplication by an integer b followed by a left-shift of c units.

3 Prior Work & Motivation

There have been several attempts to build hardware accelerators for ViT and its derivatives, and CNNs in the existing literature [5, 18, 26, 28–31, 34, 39–41]. Although these models are the SOTA in terms of accuracy, the MLP-Mixer offers its own set of merits, as highlighted in Section 1, while providing performance close to the accuracy of the SOTA, especially as the size of the dataset used to train the model increases [36]. Therefore, in this work, we provide the first attempt at a dedicated hardware accelerator for MLP-Mixer.

The overall architecture of one mixer layer is highlighted in Figure 1. As discussed in Section 1, the operations involve an irregular dataflow, with a transpose operation after each MLP. However, the BRAMs of typical edge FPGAs (e.g. Ultrascale ZCU106[1]) can only support this parallel read operation along one input-matrix dimension for a given data placement scheme due to port restrictions and bank conflicts.

We considered mapping the MLP-Mixer onto prior ViT accelerators, and we observe that none of them could fully exploit the Mixer's structure for efficient computation and resource utilization, nor support the data layout transpositions between the TM and CM MLPs. The works [26, 28, 40] do not enable a mechanism to regularly change the data layout without stalling the hardware pipeline. Although the dataflow of [41] can be transpose friendly, and the transpose unit in [18] is a possible solution, both these designs consume much more power than desired for our target edge setting. Although [30] is extremely power efficient and considers the arithmetic intensity of the on-chip data, there is no mechanism for high-bandwidth transpositions.

To address these limitations, we propose a transpose unit, co-designed with the dataflow, that ensures the required bandwidth of data is supplied to the compute unit every cycle without any stalls. To target resource constrained edge devices, we propose a design that significantly reduces power consumption and resource utilization compared to other works. Moreover, our aim with this work is to provide a complete end-to-end efficient quantized hardware implementation of the MLP-Mixer, including the non-linear activations and operations. To achieve these goals, we have considered optimizations on several fronts, maximizing the arithmetic intensity of data transferred from off-chip, minimizing the BRAM footprint, and minimizing the power and resource utilization through the dataflow selection.

4 Design of HMix

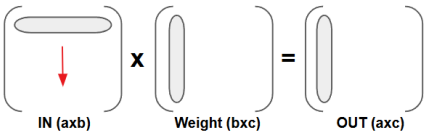
In this section, we describe our proposed design in detail. Firstly, we look at the design choices through which we arrive at our dataflow. Following that, we describe the detailed architecture of our systolic array, and the transpose mechanism using the staggered dataflow pattern of a systolic array. Lastly, we describe the design of the non-linear units (GeLU and Layer Normalization).

4.1 Dataflow Design

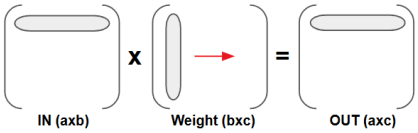
The HMix dataflow is designed to optimize the MLP-mixer on several fronts by considering the typical hardware accelerator setup of a CPU offloading the inference to an FPGA at the edge. Our dataflow and design are motivated by analyzing the considerations of the following:

(1) Arithmetic Intensity of model parameters and input image transferred from the off-chip DRAM to the on-chip BRAM.
(2) Minimizing the on-chip BRAM footprint, which is key in the deployment on memory-constrained edge devices.
(3) Minimizing LUT utilization, BRAM accesses, and DSP usage, resulting in reduced power consumption.
(4) Ensuring sufficient bandwidth to feed the processing core for both the input and its transpose, considering the TM and CM MLPs, while preventing stalls.
(5) A simple and uniform dataflow across the design, in keeping with the motivations of the model [36].

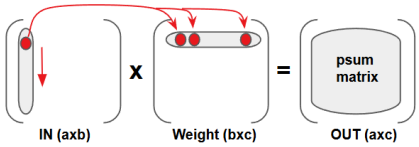
We gradually develop the design through each of the considerations, while considering the three types of GeMM dataflows described in Figure 2. These dataflows are chosen as representatives for three different reuse patterns in GeMM computation, reusing a weight column while sliding through all rows of the input matrix to compute the output one column at a time Figure 2a, reusing an input row by sliding across the weight columns Figure 2b, or a balance between the input and weight reuse, accumulating partial



(a) Dataflow with higher weight reuse pattern.



(b) Dataflow with higher input reuse pattern.



(c) Dataflow with balanced reuse pattern.

Figure 2: GeMM dataflows for different data reuse patterns.

sums (psums) of the output matrix, increasing the accesses (for successive accumulations) Figure 2c. We have also considered previous work based on systolic arrays [18, 28, 34] and processing element (PE) array [30, 39] for our analysis.

Initially, it is imperative to understand and estimate the memory footprint of the model, which includes the input and model parameters. We consider variant B/16 of the MLP-mixer, which has a total of 59 million parameters, and consider the Imagenet1k dataset [32]. Table 1 shows the memory requirement of the input, hidden activation, and the weights associated with an MLP operation. To ensure item 5, we consider the same dataflow for both CM and TM MLPs. From Table 1, it is evident that the CM operation (particularly the weights) strains the on-chip memory BRAM resources on relatively small-scale FPGAs, such as the Ultrascale ZCU106 [1] used in this work, if the dataflow requirements are not carefully considered. Therefore, we consider optimizing the CM MLP while choosing the dataflow, by ensuring the tiled reuse for the weight matrix.

Table 1: Layerwise memory requirements for B/16 mixer.

Layer (Input image - 224 × 224 × 3)	KiloBytes (KB)
Size of input matrix	147
TM - Total weight size	147
TM - Hidden activation layer size	288
CM - Total weight size	4608
CM - Hidden activation layer size	588

Another observation in the MLP-mixer is that, similar to ViT, it has an isotropic design (the size of the activation matrix remains constant throughout the model) and the output of one TM/CM MLP is the input to the successive CM/TM MLP. Moreover, the input size is 147 kB, as shown in Table 1, which can easily be accommodated on the on-chip BRAM. To maximize the arithmetic intensity (item 1) and minimize the memory requirements for the weights, we choose to implement the scheme in [30], where one tile of the weight matrix is transferred on-chip and is reused until all its corresponding operations are complete. This results in the reduction of reuse opportunities for inputs, which would not be an issue as the entire input matrix is always on-chip.

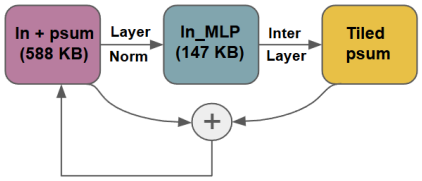
Based on the discussion in the previous paragraph, we eliminate the second dataflow (Figure 2b). Before choosing between the first and third options (Figure 2a and Figure 2c), we highlight a key constraint that arises when accumulating psums in a quantized implementation.

Accumulation constraint: Section 2.3 talks about the symmetric quantization, which we apply in our work. Since the dot products involved in computing the GeMM are accumulated in 32 bits, they require requantization, which involves multiplying by a scaling factor and then performing a left shift, with these two parameters being fixed offline. This procedure was described in Section 2.3. The caveat is that this procedure will not work for dataflows that partially accumulate the output matrix, as left-shifting a premature psum (accumulated as 32 bits) would result in the loss of the value. Therefore, they must be stored as 32-bit integers in memory. With this constraint, the third dataflow (Figure 2c) would require more than 2 MB to store the hidden layer for the CM MLP. Another constraint is that optimizations like [4] cannot be applied to psums of the hidden layer, since a non-linear GeLU operation requires the final accumulated value to be used. Therefore, we also eliminate the third dataflow (Figure 2c) and adopt the first weight reuse scheme (Figure 2a).

Next, we optimize item 2 by considering either a regular GeMM layer computation (Figure 3a) or by employing the inter-layer scheme described in [4] and Section 2.2 (Figure 3b), in keeping with the dataflow in Figure 2a. We analyze the maximum BRAM footprint required for each of these. The MLP-mixer involves a residual connection between the output of each MLP and the output from the preceding MLP (Figure 1). There is also a requirement of computing the LayerNorm of the output of the preceding MLP, which is then fed as input to the current MLP. When we consider the regular GeMM with high weight reuse pattern, the maximum BRAM footprint is equal to twice the input memory footprint added that of the hidden layer as depicted in Figure 3a. However, if we employ the optimization in [4], there is scope of a 17% reduction in the BRAM footprint. Since the psum of the second layer of the MLP is accumulated over multiple iterations, we can ensure the psum of the first iteration is added directly to the output of the previous MLP (Figure 3b). This would reduce the BRAM footprint to that of the input plus the output of the MLP. The constraint mentioned in the previously holds here for psum accumulation. Therefore, the output psum cannot be requantized to INT8 until the final accumulation.

Finally, we address item 3 and item 4. The number BRAM accesses and DSP utilization from item 3 can be optimized using a Systolic-array-based compute unit. A systolic array is inherently built for optimal data reuse, once the data is fetched from the BRAM. Both the inputs and weights are fetched once from BRAM, but their flow is orchestrated such that the fetched unit of data is used for multiple computations. We address reducing the LUT consumption in Section 4.2, by utilizing just one systolic array capable of computing both the Output and Input stationary schemes to handle both

(a) Regular GeMM dataflow.



(b) Dataflow optimization in [4], enabling a 17% reduction in BRAM footprint.

Figure 3: Two types of dataflows for GeMM.

MLP layers for our dataflow. Moreover, the staggered data access pattern of the systolic array enabled us to introduce a "stall-free transpose" unit with minimal resource overhead (Section 4.3), addressing item 4 concerns due to the repeated transpose operations.

In conclusion, we choose a systolic-array based architecture employing the [4] dataflow. This translates to an Output Stationary scheme for the first MLP layer, followed by an Input Stationary scheme for the second MLP layer. Figure 4 describes the detailed hardware architecture of HMix.

4.2 Systolic-Array Design

HMix uses a systolic array based architecture, to reduce the number of BRAM accesses for fetching data, while also providing the necessary parallelism for GeMM computation. As explained in Section 4.1, HMix employs the methodology of [4], where the computed results of the intermediate layer are forwarded for use in the computation of the second layer of the two-layer MLP structure recurring in the MLP-Mixer architecture (Figure 1).

For the first MLP layer, the results are computed using an output stationary scheme [38], where the input tile consisting of a set of rows in the input matrix and weight tile consisting of a set of columns in the weight matrix is pumped in perpendicular directions of a systolic array, orchestrated in a fashion where the correct set of input and weight tuple for multiplication in GeMM meet at a processing element in the same clock-cycle, and their product is accumulated in an accumulator register.

The accumulated values of the first MLP layer should now be forwarded for computations on the second MLP layer, through the GeLU unit which we describe later in subsubsection 4.4.2. However, since the number of accumulated values in a systolic array grows as the square of the size of the systolic array, all these values (256 for a 16x16 systolic array of size 16) cannot be drained out at the same time due to the constraints on the number of wires (256 × 32bits each) and the availability of GeLU units. Therefore, we use a shifting operation between adjacent accumulators belonging to the same row of the systolic array. This shifting mechanism follows the "cyan" datapath in Figure 5, where the weight and input registers are flushed with zero at the end of pumping the respective tile into the systolic array.

From Figure 4, the Systolic array feeds the GeLU block, which consists of 16 individual GeLU units working in parallel. An observation we made was that this same shifting logic can actually be used for computing the values of the next MLP layer. The output of GeLU, through the re-quantization module, is fed in the same systolic array through the input shifting scheme already used for output stationary operation ("Red" in Figure 5). At the end of this, the same systolic array can be used with the hidden layer now occupying the input registers and the weights of the second layer being pumped in, effectively using an input stationary dataflow. The product of an input and weight is forwarded through the "cyan connection to be added with the product of adjacent input and weight, with the output of the last column of the systolic array being the accumulated partial sum (psum) that will be stored in memory (accumulated with the previous psum).

This enables us to use a single systolic array for the entire MLP computation, resulting in resource and power efficiency, while also adopting a dataflow which minimizes on-chip BRAM utilization.

4.3 Data Access and Transpose

In this section, we describe the data access scheme for handling both the Token and Channel mixing MLP, co-designed with the dataflow described in Section 4.1. This scheme is designed to handle the irregular data access pattern in the MLP-Mixer, due to the alternating Token and Channel Mixing (TM and CM) operations, each computing on orthogonal dimensions of the input matrix, while working with the constraint of the number of ports available on BRAMs. The considerations for optimizing this operation are (i) minimizing latency, to prevent stalls in the systolic array (ii) minimizing overhead of buffers, required for re-ordering the input from row major to column major (transpose) and (iii) minimizing any overheads by the address generator in the controller (Figure 4), due to reading different addresses from different BRAM banks for re-ordering.

The scheme we develop is highlighted in Figure 6 for a simple example of a 4×4 systolic array, for both the TM and CM MLPs. A 4×4 systolic array requires 4 rows of input and 4 columns of weight matrices to be pumped in, with 4 new inputs and 4 new weights every cycle for complete utilization without stalls. We observe that delaying the address by one cycle for each successive BRAM block provides both the staggered data access ([38]) and the ability to use just a single address from the address generator (AGU in Figure 4) to meet the bandwidth requirements.

The "red" arrow depicts all the data items required to be pumped in the same clock cycle. There are four BRAM blocks (B0, B1, B2, and B3) each storing a row of the input matrix. The CM MLP takes the input as rows, while the TM MLP takes the input as columns.

We can observe that the address used to read the data from the first BRAM bank (B0) is the same address required to read data from the adjacent BRAM bank (B1) the next cycle. This essentially implies that just one address line can be used to read data for both the CM and TM data access. Further, we can observe that for CM, the order of data items read (B0, B1, B2 followed by B3) is the same order required to be fed into the systolic array (Figure 6a). However, for TM, the the data items must be re-ordered every cycle. This is achieved through a Transpose Unit (TU in Figure 4) that shuffles the

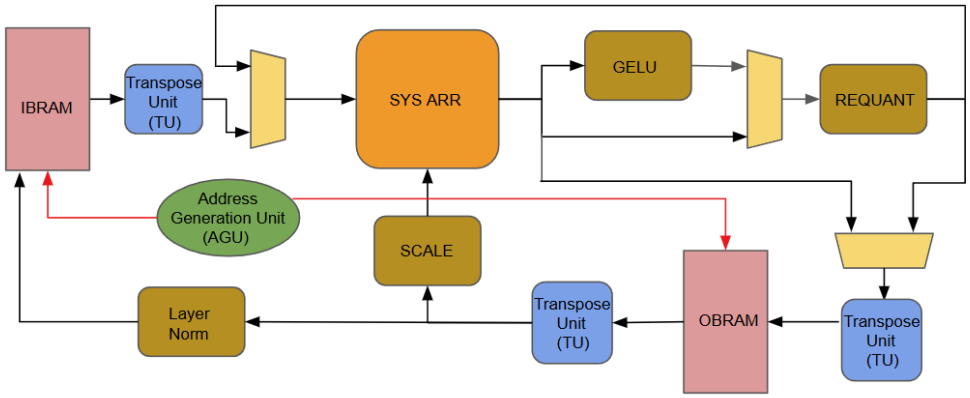


Figure 4: Overall Hardware Architecture of HMix. IBRAM refers to the Input BRAM unit which stores the Layer Normed input to the MLP and OBRAM refers to the output BRAM unit which stores the accumulated psum.

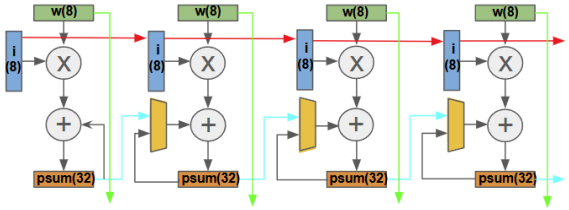
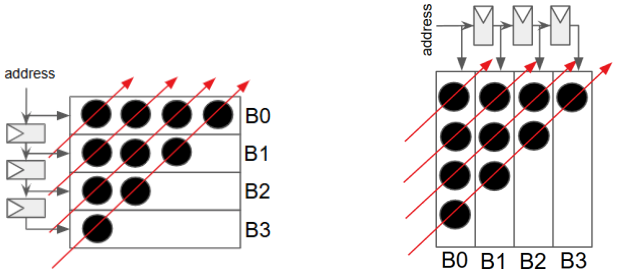


Figure 5: Systolic array capable of handling both OS and IS dataflow. The figure represents one row of the systolic array grid, where the colored arrows represent the flow of a data type (input, output and psum) in a particular direction.



(a) Data access for CM. **(b) Data access for TM.**

Figure 6: Data access schemes for CM and TM, enabling a "stall-free" transpose (B is a BRAM bank). Each "red" arrow represents all the data units accessed in the same clock cycle.

data read every cycle. This unit ensures the "stall-free" transpose mechanism. We show this scheme in Table 2, where the input into each row of the systolic matrix is in the format (BnAm/BpAq) before and after the permutation respectively, where Bp and Bn represent the output of one of the BRAM banks and Am and Aq represent the address of the data element in that bank. The R's in Table 2 represent the output of the Transpose Unit where Ri represents position i in the output order. This scheme ensures all the three initial considerations for the transpose unit – minimum latency (single cycle buffering), minimum buffers (16 for a 16x16 systolic array) and just one address line from the address generator. The transpose unit will be disabled during the CM operation. Moreover, since it can be coupled to the dataflow, the same unit can also be used to write to the output BRAM (OBRAM) during the TM operation seamlessly.

Table 2: Data permutation for TM for successive cycles.

Cycle	R1	R2	R3	R4
Cycle 1	(B0A0/B0A0)	(–/–)	(–/–)	(–/–)
Cycle 2	(B0A1/B1A0)	(B1A0/B0A1)	(–/–)	(–/–)
Cycle 3	(B0A2/B2A0)	(B1A1/B1A1)	(B2A0/B0A2)	(–/–)
Cycle 4	(B0A3/B3A0)	(B1A2/B2A1)	(B2A1/B1A2)	(B3A0/B0A3)
Cycle 5	(–/–)	(B1A3/B3A1)	(B2A2/B2A2)	(B3A1/B1A3)
Cycle 6	(–/–)	(–/–)	(B2A3/B3A2)	(B3A2/B2A3)
Cycle 7	(–/–)	(–/–)	(–/–)	(B3A3/B3A3)

The weight matrix is loaded in tile-wise, and does not involve any transpose operations. For both the TM and CM, one set of columns of the weight matrix is loaded in, and reused until all its corresponding operations are completed. The fetching of the subsequent set of the weight columns is overlapped with the computations.

4.4 Other components

4.4.1 Residual Connection. The MLP-Mixer, similar to ViT, has a residual connection, in this case after each TM and CM MLP (Figure 1). As described in Figure 3b, we accumulate the latest psum from the systolic array into the memory where the output of the previous MLP are stored, effectively achieving the residual addition while also saving memory. However, there is a caveat that the scaling factors for the quantized implementations are not the same for the psum and previous layer output for direct addition. Therefore, we scaled up the previous output to 32 bits before adding them to the psum. Note that this is required only for the first set of psums, and not for further accumulations.

The outputs of the systolic array are 32 bit psums, with $\mathbf{PSUM} = \mathbf{S_{in}} \cdot \mathbf{S_w} \cdot p$ representing their symmetic-quantized format. $\mathbf{S_{in}}$ and $\mathbf{S_w}$ are the scaling factors of the hidden activation after GeLU, and the weight matrix, respectively, and p is the accumulated partial sum in INT32. However, the output of the previous MLP layer can be represented as $\mathbf{Out} = \mathbf{S_{out}} \cdot o$, where $\mathbf{S_{out}}$ is the scaling factor and o is the INT8 value.

Algorithm 1 Quantized GELU

Input: q_{acc}, S_{acc} : quantized input and scaling factor
Output: q_{gelu}, S_{gelu} : quantized output and scaling factor

function I-POLY(q, S) ▷ $qS = x$
  $q_b \leftarrow \left\lfloor \frac{b}{S} \right\rfloor$
  $q_c \leftarrow \left\lfloor \frac{c}{aS^2} \right\rfloor$
  $S_{out} \leftarrow aS^2$
  $q_{out} \leftarrow (q + q_b)^2 + q_c$
  return q_{out}, S_{out} ▷ $q_{out}S_{out} \approx a(x+b)^2 + c$
end function

function I-ERF(q, S) ▷ $qS = x$
  $a, b, c \leftarrow -0.2888,\ -1.769,\ 1$
  $q_{sgn}, q \leftarrow \text{sgn}(q),\ \text{clip}(|q|, \max = -b/S)$
  $q_L, S_L \leftarrow$ I-POLY(q, S)
  $q_{out}, S_{out} \leftarrow q_{sgn}q_L, S_L$
  return q_{out}, S_{out} ▷ $q_{out}S_{out} \approx \text{erf}(x)$
end function

function I-GELU(q, S) ▷ $qS = x$
  $q_{erf}, S_{erf} \leftarrow$ I-ERF($q, S/\sqrt{2}$)
  $q_1 \leftarrow \lfloor 1/S_{erf} \rfloor$
  $q_{out}, S_{out} \leftarrow q(q_{erf} + q_1),\ SS_{erf}/2$
  return q_{out}, S_{out} ▷ $q_{out}S_{out} \approx \text{GELU}(x)$
end function

$q_{gelu}, S_{gelu} \leftarrow$ I-GELU(q_{acc}, S_{acc})

To tackle the addition of residual (skip) connection, we use re-scaling in Equation 3. Since we are re-scaling from an 8 bit into a 32 bit value, we use the inverse procedure of what was described in Section 2.3, since $\mathbf{S_{in}} \cdot \mathbf{S_w}$ would be of a smaller magnitude than $\mathbf{S_{out}}$, we can effectively translate the operation into a multiplication of o with a fixed-point scaling factor b, and then right-shifting by a value c. This operation is performed by the SCALE unit in Figure 4.

$$\text{Out} = \mathbf{S_{in}} \cdot \mathbf{S_w} \cdot \lfloor \frac{\mathbf{S_{out}}}{\mathbf{S_{in}} \cdot \mathbf{S_w}} \rfloor \cdot o = \mathbf{S_{in}} \cdot \mathbf{S_w} \cdot ((b \cdot o) << c) \tag{3}$$

4.4.2 GeLU. The Gaussian-error linear unit (GeLU) [15] is a non-linear activation function that is applied on the activation of the first layer in both the TM and CM MLP. With our aim to implement the entire inference operation in the quantized domain, we adopt the SOTA GeLU implementation from [23]. This entails a polynomial approximation for the gaussian error function (erf) involved in the GeLU computation. The entire algorithm for quantized computation of GeLU activation is described in Algorithm 1. This operation is handled by the GELU unit in Figure 4, which comprises of parallel GeLU operations equal to the size of the systolic array.

4.4.3 Layer-norm. The layer-norm [3] is involved before every MLP operation in the MLP-Mixer (Figure 1). This normalization operation is performed along the channel dimension of the input matrix, irrespective of whether it is TM or CM. This operation poses two major hurdles to the integer-only implementation theme. One is the square root operation, and the other being the repeated divisions involved. The overall formula for the layer-norm operation is:

$$\tilde{x} = \frac{x - \mu}{\sigma}, \quad \mu = \frac{1}{C}\sum_{i=1}^{C} x_i, \quad \sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i - \mu)^2}. \tag{4}$$

Let the input x be represented by the quantization parameters S and q. Then, the quantized mean can be represented by the parameters $\frac{S}{C}$ and $\sum_{i=1}^{C} q_i$. For the standard deviation, we expand the the above formula into:

$$\sigma = \sqrt{\frac{1}{C}\left(\sum_{i=1}^{C} x_i^2 - C\mu^2\right)} = \frac{S}{C}\sqrt{\left(C\sum_{i=1}^{C} q_i^2 - (\sum_{i=1}^{C} q_i)^2\right)} \tag{5}$$

The standard deviation can now be calculated in one pass of the input row, where both the sum of squares and mean is calculated simultaneously. The computation for square root follows the method proposed in [7] and adopted by [23]. Finally, the layer-norm of an input is calculated as:

$$\tilde{x} = \frac{Cq - \sum_{i=1}^{C} q_i}{\sqrt{\left(C\sum_{i=1}^{C} q_i^2 - (\sum_{i=1}^{C} q_i)^2\right)}} \tag{6}$$

To efficiently handle the division with minimal hardware, power and latency overhead, we make use of the fact that the standard deviation is constant across an entire row. Therefore, we calculate the scaled reciprocal of the standard deviation as $\frac{(2<<b)}{\sigma}$. The scaling is later adjusted during requantization. The layer-normed values are calculated through another pass across the input matrix row – subtracting the mean and multiplying with the scaled reciprocal.

5 Evaluation and Results

5.1 MLP-Mixer Quantization

Here, we outline the challenges faced in the quantization of MLP-Mixer B/16 model pre trained on Imagenet-1k [32]. We observe that merely performing a post-training quantization resulted in a significant accuracy drop on the Imagenet-1k validation dataset. Therefore, we explored the method of quantization-aware fine tuning, using the Imagenet-1k training dataset for the same. In the forward pass, the same setup as the quantized inference is used. However, this is not feasible during back-propagation as there is a requirement for all the functions to be differentiable, which is not the case for multiple floor functions used for quantization of non-linear operations like GeLU and Layer normalization. To implement this, straight-through estimators provided by the JAX framework is utilized, which replace the quantized implementations with the regular real valued functions for backpropagation. The accuracy drop was observed to be minimal after finetuning for a few epochs.

5.2 Hardware Evaluation

We evaluate HMix with respect to hardware utilization and power consumption. We also consider on-chip memory accesses and consumption compared with naive implementations. Our experiments are based on the synthesis on Ultrascale ZCU106 FPGA ([1]), using AMD's Vivado suite [2], at 200 MHz.

HMix breakdown: Table 5 presents a breakdown of the modular utilization of the HMix accelerator. The results show that the

Table 3: Memory accesses and storage requirements for the Channel-mixing operation, for different dataflow architectures.

Dataflow	Compute Size	Memory Accesses				Memory Size (kB)			
		Input	Weight	Output	Total	Input	Weight	Output	Total
PE-array based	16 × 16	924M	57M	0.75M	981M	294	2	588	882
Output-Stationary	16 × 16	57M	57M	0.75M	115M	294	24	588	904
Proposed	16 × 16	29M	57M	57M	143M	147	24	588	759

Table 4: Comparison of HMix with Vision Transformer accelerators.

Accelerator	Device	Precision	Model	Params (M)	Freq. (MHz)	BRAM/DSP/LUT/FF	FPS	FPS/W	FPS/DSP
ViA [40]	Alveo U50	FP16	Swin-T	28	300	1002/2420/258k/257k	—	—	—
Auto-ViT [26]	ZCU102	W8 A8	DeiT-B	86	150	—/2066/128k/—	25.9	2.76	0.12
Hybrid SA [34]	ZCU102	W8 A8	ViT-B/16	86	—	637/2185/169k/166k	19.5	1.62	0.009
OBS [41]	VC707	INT8	TNT-S	24	200	1024/2048/156k/77k	67.6	5.41	0.033
ME-ViT [28]	Alveo U200	—	DeiT-B	86	300	288/1024/192k/132k	23.98	2.57	0.023
Group Vector [18]	ZCU102	W8 A8	ViT-tiny	22	300	648/1268/144k/168k	86.95	2.94	0.068
HMix (Ours)	ZCU106	W8 A8	Mixer-B/16	59	200	232/150/77k/40k	3.64	2.81	0.024

proposed transpose units incur hardware overhead of only 12% (all three TU in Figure 4 combined) of the total LUT consumption. This transpose implementation enables stall-free data movement during TM and CM operations. This directly satisfies one of the primary design constraints targeted by HMix (item 4).

Table 5: Hardware utilization of individual components.

Module	LUTs	BRAMs	DSPs
Top (Overall)	**77736**	**232**	**150**
Systolic Array (With additional logic)	56980	0	0
Controller (AGU)	432	0	2
Layer-norm	5640	0	48
GeLU	2848	0	96
IBRAM	79	48	0
OBRAM	240	168	0
WBRAM	0	16	0
Transpose at IBRAM output	1380	0	0
Transpose at OBRAM input	4119	0	0
Transpose at OBRAM output	4119	0	0

Dataflow Analysis: Table 3 compares the HMix dataflow with two other representative dataflows to highlight memory utilization per MLP operation. The PE-array-based dataflow represents the row-based compute unit described in [30, 39]. The second dataflow represents a generic systolic-array-based output stationary scheme for GeMM. The memory size represents the peak value during the CM MLP while the memory accesses are also computed for the CM MLP. It can be seen that the constraint item 2 is met, with up to 16% reduction in BRAM utilization compared to a naive implementation.
Resource and Power: The main objective of HMix is to meet the resource utilization and power consumption constraints for edge deployment. The analysis in Figure 7 demonstrates the effectiveness of our design from the power perspective. Our design consumes an order of magnitude less power with just 1.28W. This can be attributed to the limited use of DSPs. Furthermore, from Table 4, HMix demonstrates a competitive efficiency of 2.81 fps/W. Although HMix trails in efficiency against OBS [41] and Group Vector [18], their observations are made on models of much smaller

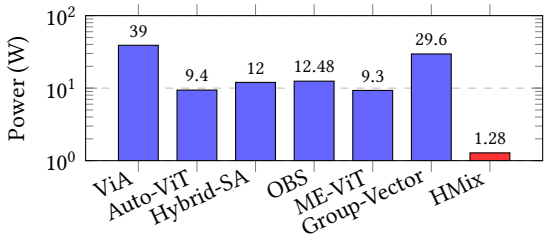


Figure 7: Power consumption comparison. The values on the bars represent the absolute power values.

scale. Moreover, HMix demonstrates substantially higher efficiency than inference on the Jetson TX2, which achieves only 0.64 FPS/W for the DeiT-B model [26].

6 Conclusion

The MLP-Mixer is an alternative to the standard CNN and ViT based models for vision applications, involving a simple and regular dataflow without the involvement of quadratic-complexity attention in ViT. In this work, we propose HMix, the first quantized hardware accelerator for MLP-Mixer and show an efficient design with low power consumption and resource utilization for embedded applications. As part of the implementation, we propose an efficient "stall-free" transpose unit with minimal hardware and latency overhead. We also ensure that the quantization has minimal effects on the inference accuracy by implementing a post-quantization fine tuning on the ImageNet dataset. With the growing utility of computer vision in applications, this work evaluates the hardware acceleration of a novel model to investigate its potential benefits, while tailoring the hardware for efficient inference.

Acknowledgements

This research was supported in part by NSF Grants #2326894 and #2425655, Center for Generative AI and Texas Advanced Computing Center (TACC), and an NVIDIA Applied Research Accelerator Program Grant. Any opinions, findings, conclusions, or recommendations are those of the authors and not of the funding agencies.

References

[1] AMD (formerly Xilinx) 2019. *Zynq UltraScale+ MPSoC ZCU106 Evaluation Kit User Guide, UG1244.* AMD (formerly Xilinx). https://docs.amd.com/v/u/en-US/ug1244-zcu106-eval-bd

[2] AMD Xilinx. 2022. *Vivado Design Suite.* https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html Version 2022.1.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. arXiv:1607.06450 [stat.ML] https://arxiv.org/abs/1607.06450

[4] Shenggang Chen and Zhonghai Lu. 2019. Hardware Acceleration of Multilayer Perceptron Based on Inter-Layer Optimization. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. 164–172. doi:10.1109/ICCD46524.2019.00028

[5] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138. doi:10.1109/JSSC.2016.2616357

[6] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308. doi:10.1109/JETCAS.2019.2910232

[7] R. Crandall and C. Pomerance. 2005. *Prime Numbers: A Computational Perspective.* Springer. https://books.google.co.in/books?id=RbEz-_D7sAUC

[8] Dana Diaconu, Xue Lin, Michaela Blott, and Miriam Leeser. 2025. A Survey of FPGA-based 3D CNN Accelerators and Hardware-aware Algorithmic Optimizations. *ACM Comput. Surv.* 58, 6, Article 155 (Dec. 2025), 35 pages. doi:10.1145/3777366

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV] https://arxiv.org/abs/2010.11929

[10] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: shifting vision processing closer to the sensor. *SIGARCH Comput. Archit. News* 43, 3S (June 2015), 92–104. doi:10.1145/2872887.2750389

[11] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. 2023. On The Computational Complexity of Self-Attention. In *Proceedings of The 34th International Conference on Algorithmic Learning Theory (Proceedings of Machine Learning Research, Vol. 201)*, Shipra Agrawal and Francesco Orabona (Eds.). PMLR, 597–619. https://proceedings.mlr.press/v201/duman-keles23a.html

[12] Julian Faraone, Nicholas Fraser, Michaela Blott, and Philip H. W. Leong. 2018. SYQ: Learning Symmetric Quantization for Efficient Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4300–4309.

[13] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. A Survey of Quantization Methods for Efficient Neural Network Inference. arXiv:2103.13630 [cs.CV] https://arxiv.org/abs/2103.13630

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV] https://arxiv.org/abs/1512.03385

[15] Dan Hendrycks and Kevin Gimpel. 2023. Gaussian Error Linear Units (GELUs). arXiv:1606.08415 [cs.LG] https://arxiv.org/abs/1606.08415

[16] Seyed Hani Hozhabr and Roberto Giorgi. 2025. A Survey on Real-Time Object Detection on FPGAs. *IEEE Access* 13 (2025), 38195–38238. doi:10.1109/ACCESS.2025.3544515

[17] Kung H.T. 1982. Why systolic architectures? *Computer* 15, 1 (1982), 37–46. doi:10.1109/MC.1982.1653825

[18] Mingqiang Huang, Junyi Luo, Chenchen Ding, Zikun Wei, Sixiao Huang, and Hao Yu. 2023. An Integer-Only and Group-Vector Systolic Accelerator for Efficiently Mapping Vision Transformer on Edge. *IEEE Transactions on Circuits and Systems I: Regular Papers* 70, 12 (2023), 5289–5301. doi:10.1109/TCSI.2023.3312775

[19] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. arXiv:1712.05877 [cs.LG] https://arxiv.org/abs/1712.05877

[20] Junye Jiang, Yaan Zhou, Yuanhao Gong, Haoxuan Yuan, and Shuanglong Liu. 2025. FPGA-based Acceleration for Convolutional Neural Networks: A Comprehensive Review. arXiv:2505.13461 [cs.LG] https://arxiv.org/abs/2505.13461

[21] Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. arXiv:2304.01433 [cs.AR] https://arxiv.org/abs/2304.01433

[22] Md Mohsin Kabir, Ashifur Rahman, Md Nahid Hasan, and M.F. Mridha. 2025. Computer vision algorithms in healthcare: Recent advancements and future challenges. *Computers in Biology and Medicine* 185 (2025), 109531. doi:10.1016/j.compbiomed.2024.109531

[23] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. I-BERT: Integer-only BERT Quantization. arXiv:2101.01321 [cs.CL] https://arxiv.org/abs/2101.01321

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[25] Zhikai Li and Qingyi Gu. 2023. I-ViT: Integer-only Quantization for Efficient Vision Transformer Inference. arXiv:2207.01405 [cs.CV]

[26] Zhengang Li, Mengshu Sun, Alec Lu, Haoyu Ma, Geng Yuan, Yanyue Xie, Hao Tang, Yanyu Li, Miriam Leeser, Zhangyang Wang, Xue Lin, and Zhenman Fang. 2022. Auto-ViT-Acc: An FPGA-Aware Automatic Acceleration Framework for Vision Transformer with Mixed-Scheme Quantization. In *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. 109–116. doi:10.1109/FPL57034.2022.00027

[27] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. arXiv:2103.14030 [cs.CV]

[28] Kyle Marino, Pengmiao Zhang, and Viktor K. Prasanna. 2023. ME- ViT: A Single-Load Memory-Efficient FPGA Accelerator for Vision Transformers. In *2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. 213–223. doi:10.1109/HiPC58850.2023.00039

[29] Shashank Nag, Alan T. L. Bacellar, Zachary Susskind, Anshul Jha, Logan Liberty, Aishwarya Sivakumar, Eugene B. John, Krishnan Kailas, Priscila M. V. Lima, Neeraja J. Yadwadkar, Felipe M. G. Franca, and Lizy K. John. 2025. LL-ViT: Edge Deployable Vision Transformers with Look Up Table Neurons. arXiv:2511.00812 [cs.LG] https://arxiv.org/abs/2511.00812

[30] Shashank Nag, Gourav Datta, Souvik Kundu, Nitin Chandrachoodan, and Peter A. Beerel. 2023. ViTA: A Vision Transformer Inference Accelerator for Edge Applications. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. doi:10.1109/ISCAS46773.2023.10181988

[31] Anouar Nechi, Lukas Groth, Saleh Mulhem, Farhad Merchant, Rainer Buchty, and Mladen Berekovic. 2023. FPGA-based Deep Learning Inference Accelerators: Where Are We Standing? *ACM Trans. Reconfigurable Technol. Syst.* 16, 4, Article 60 (Oct. 2023), 32 pages. doi:10.1145/3613963

[32] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[33] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV] https://arxiv.org/abs/1409.1556

[34] Qingzeng Song, Yao Dai, Hao Lu, and Guanghao Jin. 2024. High-throughput systolic array-based accelerator for hybrid transformer-CNN networks. *Journal of King Saud University - Computer and Information Sciences* 36, 8 (2024), 102194. doi:10.1016/j.jksuci.2024.102194

[35] Richard Szeliski. 2022. *Computer Vision: Algorithms and Applications.* doi:10.1007/978-3-030-34372-9

[36] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. 2021. MLP-Mixer: An all-MLP Architecture for Vision. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 1857, 12 pages.

[37] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. arXiv:2012.12877 [cs.CV]

[38] Bo Wang, Sheng Ma, Guoyi Zhu, Xiao Yi, and Rui Xu. 2022. A novel systolic array processor with dynamic dataflows. *Integration* 85 (2022), 42–47. doi:10.1016/j.vlsi.2022.03.002

[39] Hong-Yi Wang and Tian-Sheuan Chang. 2022. Row-wise Accelerator for Vision Transformer. arXiv:2205.03998 [cs.AR] https://arxiv.org/abs/2205.03998

[40] Teng Wang, Lei Gong, Chao Wang, Yang Yang, Yingxue Gao, Xuehai Zhou, and Huaping Chen. 2022. ViA: A Novel Vision-Transformer Accelerator Based on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4088–4099. doi:10.1109/TCAD.2022.3197489

[41] Zhongyu Zhao, Rujian Cao, Ka-Fai Un, Wei-Han Yu, Pui-In Mak, and Rui P. Martins. 2023. An FPGA-Based Transformer Accelerator Using Output Block Stationary Dataflow for Object Recognition Applications. *IEEE Transactions on Circuits and Systems II: Express Briefs* 70, 1 (2023), 281–285. doi:10.1109/TCSII.2022.3196055