Refining Datapath for Microscaling ViTs

Can Xiao
Imperial College London
cx922@ic.ac.uk

Jianyi Cheng
University of Edinburgh
jianyi.cheng@ed.ac.uk

Yiren Zhao
Imperial College London
a.zhao@imperial.ac.uk

***Abstract*—Vision Transformers (ViTs) leverage the transformer architecture to effectively capture global context, demonstrating strong performance in computer vision tasks. A major challenge in ViT hardware acceleration is that the model family contains complex arithmetic operations that are sensitive to model accuracy, such as the Softmax and LayerNorm operations, which cannot be mapped onto efficient hardware with low precision. Existing methods only exploit parallelism in the matrix multiplication operations of the model on hardware and keep these complex operations on the CPU. This results in suboptimal performance due to the communication overhead between the CPU and accelerator. Can new data formats solve this problem?**

In this work, we present the first open-source ViT accelerator that maps all operations of the ViT models onto FPGAs. We exploit a new arithmetic format named Microscaling Integer (MXInt) for datapath designs and evaluate how different design choices can be made to trade off accuracy, hardware performance, and hardware utilization. Our contributions are twofold. First, we quantize ViTs using the MXInt format, achieving both high area efficiency and accuracy. Second, we propose MXInt-specific hardware optimization that map these complex arithmetic operations into custom hardware. Within 1% accuracy loss, our method achieves at least 93× speedup compared to Float16 and at least 1.9× speedup compared to related work.

I. Introduction

Hardware acceleration for transformers has shown significant performance benefits compared to general processors [1], [2], [3], among which Vision Transformers (ViTs) offer promising performance for capturing global image relationships [4]. Compared to traditional Convolutional Neural Networks (CNNs), ViTs present new model features: 1) these models often contain millions of parameters, leading to a large memory size; and 2) they contain non-linear operations, requiring complex hardware operator designs.

Traditional techniques for ViT acceleration focus on *1) integer quantization* and *2) datapath optimization*, exploiting the approximation tolerance of ViT models. First, integer quantization represents numbers as small integers, optionally with a scaling factor, leading to both smaller memory and circuit area [2], [3]. Second, datapath optimization determines new designs with simpler logic and similar results, leading to a smaller circuit area [5].

Still, non-linear operations in ViT, such as LayerNorm and Softmax, face challenges in efficient acceleration. These operations contain complex mathematical operations, such as *exp()* and *sqrt()*, and require large value ranges, restricting existing integer quantization. Existing design methods rely on the CPU and only accelerate part of the ViT models in FPGA fabric [2], [3]. This leads to a working but complex system design with suboptimal performance due to the communication overhead between the CPU and the accelerator.

TABLE I: Our MXInt design method maps all non-linear operations in ViTs into efficient hardware, achieving lower bitwidths than traditional fixed-point designs.

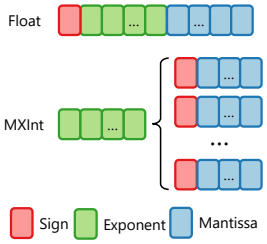
Methods	Precision on fabric	LayerNorm		GELU		Softmax	
		Fabric	Bitwidth	Fabric	Bitwidth	Fabric	Bitwidth
AutoViTAcc [3]	fixed-point	✗	8	✗	8	✗	8
[9]	fixed-point	✓	8	✓	8	✓	8
HeatViT [2]	fixed-point	✗	8	✓	8	✓	8
SDA [5]	fixed-point	✓	8	✓	8	✓	8
Our Work	**MXInt**	✓	**5**	✓	**5**	✓	**2**

In this work, we unlock this by exploiting a recently studied data format named Microscaling Integers (MXInt) [6]. The MXInt format shares an exponent among a block of integer values, forming a more compact floating-point format. This reduces memory size while maintaining high model accuracy [7]. The existing work in MXInt hardware mapping focuses on matrix multiplications [8], but it remains an open question regarding optimizing non-linear operations in MXInt.

In this work, we present fully quantized ViTs in MXInt and propose datapath optimization techniques for the efficient acceleration of both linear and non-linear operations. To ensure fairness in quantization, we focus on post-training quantization (PTQ) and compare accuracy without fine-tuning, as fine-tuning entails complex training techniques that may vary from model to model. Following prior work on MXInt quantization [7], we restrict the accuracy loss of the final design to within 1% to preserve high model accuracy. We then explore datapath optimization opportunities and show how to efficiently map MXInt operators into efficient hardware, including non-linear operations. This leads to fully hardware-accelerated ViTs, meaning that all operations are mapped into efficient hardware, as illustrated in Table I. Our contributions are as follows:

- We propose MXInt-based ViT accelerators, reducing the memory size up to 4.99× within 1% accuracy loss;
- We propose MXInt-specific datapath optimization for accuracy-sensitive arithmetic operators in the ViTs (e.g. GELU, Softmax and LayerNorm), reducing the area at least by 16× within 1% accuracy loss; and
- Over a set of models, we show our design methods achieve at least 93× speedup compared to Float16 and at least 1.9× speedup compared to related work.

The rest of the paper is organized as follows. Section II



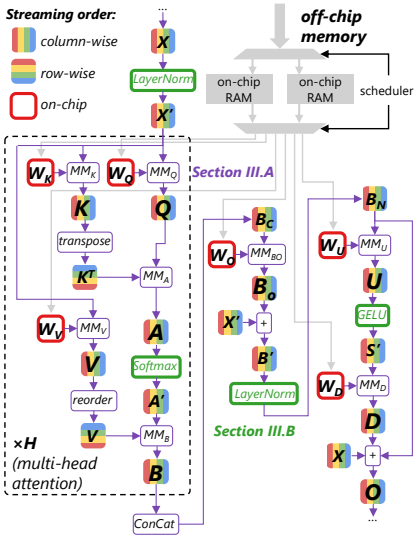
(a) A standard floating point format and a MXInt format.

Formats	Config	Accuracy	Memory Density
FP32	-	**81.80%**	1×
Int16	W16A16	80.05%	2×
MXInt8	W6.03/A8.5	**81.72%**	4.99×

(b) Quantization results of DeiT Base [10] on ImageNet [11]. MXInt balances model accuracy and memory efficiency, leading to a better choice of formats for ViT quantization. The decimal part derived from the shared exponent value.

Require: X ▷ Input Features
Require: H ▷ Number of heads
Require: L ▷ Number of hidden layers
$X_n \leftarrow \mathbf{LayerNorm}(X)$
for $i \in [0, H)$ **do**
  $Q_i \leftarrow W_{Q_i} \cdot X_n$
  $K_i \leftarrow W_{K_i} \cdot X_n$
  $V_i \leftarrow W_{V_i} \cdot X_n$
  $A_i \leftarrow \frac{Q_i \cdot {K_i}^T}{\sqrt{d_k}}$
  $\hat{A}_i \leftarrow \mathbf{Softmax}(A_i)$
  $B_i \leftarrow \hat{A}_i \cdot V_i$
end for
$B_c \leftarrow Concat(B_0..B_{H-1})$
$B_o \leftarrow W_0 \cdot B_c$
$B_n \leftarrow \mathbf{LayerNorm}(B_o + X_n)$
$U \leftarrow W_U \cdot B_n$
$D \leftarrow W_D \cdot (\mathbf{GELU}(U))$
$O \leftarrow D + B_n$
return O

(c) An algorithm view of a block in the DeiT model. Values highlighted in *blue* represent quantized values, and operations highlighted in **green** represent mxint specific approximated operations.



(d) An architecture view of the proposed hardware accelerator. The proposed architecture pipelines the model in a hierarchical dataflow, and tailors each operation for high area efficiency.

Fig. 1: Motivating example: dataflow hardware acceleration of a ViT in MXInt.

illustrates an overview of the proposed accelerator architecture. Section III describes our hardware architecture and design choices in detail. Section IV evaluates the effectiveness of our work and compares performance with related work. Section V compares our work with related work qualitatively.

II. Motivation

Why MXInt? Here we first introduce the MXInt format. Figure 1a compares the standard floating-point numerical format with the MXInt format. The top of the figure illustrates the standard floating-point format, which contains a sign bit, an exponent, and a mantissa [12]. The MXInt format has similar components to the standard floating-point format but allows a block of values to share the exponent, where the block size is user-defined. MXInt provides finer granularity compared to the traditional fixed-point format that shares a scaling factor across all tensor values. Prior work [7] shows that MXInt can achieve both high model accuracy and high memory density for software LLM quantization. Our observation from Figure 1b verifies that this still holds for ViT models.

How to map into MXInt? Figure 1 provides a high-level overview of the proposed accelerator architecture design. Figure 1c illustrates an algorithmic view of a transformer block in ViT models. There are various design choices on hardware architecture for the ViT accelerators, such as systolic arrays and custom dataflow architectures. The benefits of dataflow architectures have been widely studied in the literature [13]. Here we choose the dataflow architecture due to its high throughput and low control flow overhead.

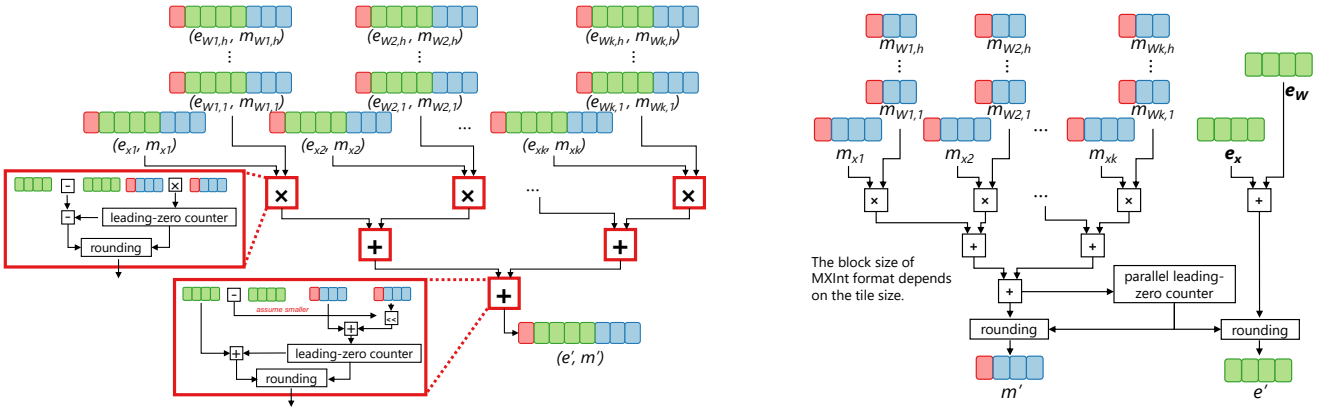
The corresponding accelerator architecture is shown in Figure 1d. In the figure, each operation in Figure 1c is mapped into a hardware operator unit with a dataflow interface. Each hardware unit is dedicated to computing a specific function for its inputs, minimizing control flow overhead. In our design, the parameters are initially stored off-chip due to the large size required by the ViT models. A predefined scheduler is implemented to prefetch the parameters to on-chip memory through a ping-pong buffer, as shown at the top right of Figure 1d. Similar to most dataflow architectures, the tensor is tiled due to its large size and streamed into the accelerators for deeply pipelined computation. Design choices for tiling sizes and streaming order have been widely studied in prior work [14], [15], [16]. These are outside our scope, while we focus on the datapath optimization of these operators. We will describe our MXInt quantization and linear operator implementation in Section III-A.

How to optimize MXInt hardware? The ViTs also contain non-linear operations, and the precision required by these operations is often high due to their high sensitivity to model accuracy. This leads to significant circuit area due to both high precision and arithmetic complexity. We will show how to lift this restriction by optimizing the datapath of non-linear operators in Section III-B.

III. Methodology

A. Opportunities and Challenges for MXInt

The hardware design of efficient MXInt operators faces several optimization opportunities regarding the block-sharing feature of the MXInt data format. Traditional operators compute on individual values independently because their values are presented in standalone formats. Such a design method is inefficient for MXInt values because the common results of



(a) Dot product in Float relies on area-expensive operators.

(b) Dot product in MXInt contains integer-only operators.

Fig. 2: Comparison between the standard floating-point format and MXInt on the dot product operator.

the shared exponent could be shared across the values in the same block.

For simplicity, here we show a dot product design example to illustrate how mapping into MXInt causes hardware design improvements. Figure 2a shows a traditional implementation of a dot product unit with k values computing in parallel. The precision of the values is in standard floating-point to preserve high model accuracy. In the data path, standard floating-point multipliers and adders are used for computation. Using these operators often leads to significant area overhead because the normalization between values to the same dynamic range requires complex dynamic shift logic [17]. The normalization circuit is required at both the input and the output of each operator, leading to huge area overhead in total. There has been effort in optimizing normalization for individual floating-point operators [18], but they miss MXInt-specific optimizations such as exponent sharing.

We design fully customized operators for MXInt values, as illustrated in Figure 2b. Here, we assume that the exponent of x is shared by k values, and the exponent of W is shared by $h \times k$ values. In the dot product unit, all the operators perform either fixed-point or logical operations. The overhead caused by the expensive dynamic shift logic in standard floating-point is reduced from two aspects: 1) only one dynamic shift operator is required per block, where its result is shared within the block; and 2) the bitwidth of the mantissas in MXInt is often small after quantization, making the transformation into unrolled constant shifts affordable. This enables us to compute values with dynamic ranges in efficient hardware while preserving high accuracy.

A key challenge is to balance the precision and circuit area by determining an efficient mantissa bitwidth for MXInt values. A small bitwidth in MXInt mantissas could lead to quantization errors, while a large bitwidth could lead to a large operator area. Particularly, the bitwidth required by the accumulator used in the linear operations scales with the tensor sizes. A significantly large tensor with a large variance in element values could cause the accumulator to both underflow and overflow.

In this work, we focus on the hardware datapath optimization of non-linear operations, as linear operations have already been studied in related work [8]. For linear operations, we rely on greedy search in software quantization to determine the minimal bitwidth of the mantissa to preserve high accuracy within a 1% loss. We observed that the shared exponent can effectively handle overflow with negligible effects on model accuracy, but the underflow requires more bits. Therefore, we empirically determine the additional bits required by the accumulator operator and expand the bitwidth of the accumulator to minimize quantization loss. Particularly for DeiT models, we set the mantissa of the accumulator in all the linear operators to be 12 bits to perform lossless addition. A similar method is applied to other compute-intensive operators, such as convolution operators.

Finally, different shared block sizes in MXInt between layers could lead to additional hardware logic to ungroup and regroup these values between computations. This is more related to the control path of the circuit and is out of the scope of our work. In this work, we choose the block size to always be the same as the tile size, as shown in Figure 2b to minimize the control flow overhead on grouping values. For instance, a linear operator has exponents shared among 16 and 256 values in activation and weights, respectively. In the rest of the section, we will describe our optimization techniques for the datapath of non-linear operators.

B. Datapath Optimization in MXInt

In this section, we now show how to optimize all these non-linear functions in the mantissa domain with fewer numbers of bits. Our work focuses on optimizations for ViTs, but it could also be applied to MXInt accelerators for other ML models, leading to different results of bitwidth. Here we take three core operators from three representative classes of operations for illustration: LayerNorm, Softmax, and GELU. These operations are commonly seen in ViT models and require complex arithmetic circuits to compute.

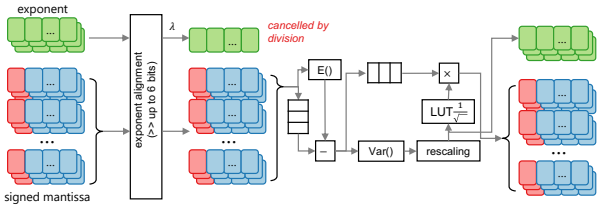


Fig. 3: Optimized datapath for MXInt LayerNorm.

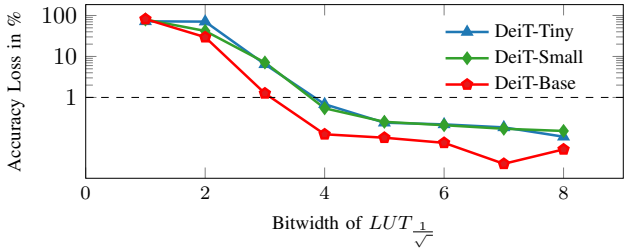


Fig. 4: Accuracy loss over different bitwidth of $LUT_{\frac{1}{\sqrt{}}}$.

1) LayerNorm Approximation: LayerNorm operations have been widely used in transformers, including ViTs, which scale values in a tensor to a fixed range [19]. The standard expression of the LayerNorm operation is presented as follows.

$$y = \frac{x - E(x)}{\sqrt{Var(x) + \epsilon}}\gamma + \beta \tag{1}$$

x and y denote the input and output tensors, and $E(.)$ computes the expectation and $Var(.)$ computes the variance, respectively. ϵ, γ, and β are constants in the model.

Existing approaches, including [9] and SDA [5], focus on integer-based datapath optimization for hardware acceleration, but their bitwidth is still large compared to our MXInt designs. This is because the absence of dynamic range in integer format requires more bits to represent the same range of values.

We now describe our optimization for LayerNorm in MXInt. Benefiting from the small mantissa bitwidth of MXInt, we separate the computation of the shared exponent and mantissas. An MXInt value can be represented as follows.

$$x = 2^{x_e} x_m \tag{2}$$

x_e and x_m are the unsigned shared exponent and signed mantissa for a value x, where x_e is shared by a group of values. When computing Softmax, we are effectively dealing with MXInt values coming from different blocks that are scaled by a set of different exponent values $(x_e^0, x_e^1, ...)$. As shown in Figure 3, we apply a re-quantization step that forces these values from different groups to use the same exponent (x_e^{max}) by dynamically right-shifting on mantissa values. Effectively, for each individual MXInt value, we have:

$$x = 2^{x_e} x_m = 2^{x_e^{max}} x_{m'} = \lambda x_{m'} \tag{3}$$

where $x_{m'}$ is the right-shifted version of x_m, and the right-shifting amount depends on $x_e^{max} - x_e$. As shown in Figure 3, since x_e^{max} is now shared across all values that are inputs to the Softmax function, we can effectively treat it as a constant $\lambda = x_e^{max}$ for the ease of expression.

We then substitute the expression of x into Equation 1 and extract the exponent as follows.

$$y = \frac{\lambda x_m - E(\lambda x_m)}{\sqrt{Var(\lambda x_m) + \epsilon}}\gamma + \beta \tag{4}$$

$$= \frac{\lambda (x_m - E(x_m))}{\sqrt{Var(\lambda x_m) + \epsilon}}\gamma + \beta \tag{5}$$

TABLE II: Comparison with related work on LayerNorm optimization for DeiT-Tiny. Our work achieves the minimal accuracy loss on fabric at the smallest bitwidth.

Methods	On Fabric	Bitwidth	Accuracy	Accuracy Loss
Original	✗	16	72.13%	-
Auto-ViT-Acc[3]	✗	16	72.13%	0
HeatViT [2]	✗	16	72.13%	0
[9]	✓	8	71.66%	0.472%
SDA [5]	✓	8	71.66%	0.472%
Vanilla MXInt	✓	13	72.078%	0.154%
Optimized MXInt	✓	**5**	71.89%	**0.242%**

We approximate ϵ to be zero so that we can extract the exponent from the square root function.

$$\approx \frac{\lambda (x_m - E(x_m))}{\lambda \sqrt{Var(x_m)}}\gamma + \beta \tag{6}$$

$$\approx \frac{(x_m - E(x_m))}{\sqrt{Var(x_m)}}\gamma + \beta \tag{7}$$

A key novelty of our work is that we convert an MXInt-based LayerNorm operation into an integer-only operator dealing with only the mantissa component (x_m) in the MXInt format. This significantly simplifies the circuitry required to implement the LayerNorm function.

Still, after computing the variance $Var(.)$, we end up with a large result in a larger bitwidth from the accumulator, determined by the tensor size. This is followed by a division and a square root operation ($\frac{1}{\sqrt{Var(.)}}$). Existing methods use large bitwidth in fixed-point numbers [9] or cast the values to floating-point numbers to preserve high precision in a small bitwidth. We combine the best of both approaches, casting the values to a small floating-point format followed by a LUT-based method to avoid high computational overhead.

Specifically, we rescale (*rescaling* in Figure 3) the variance ($Var(x_m)$), which is represented in the mantissa domain to a floating-point number $x_{m'}^v 2^{x_{e'}^v}$.

$$\frac{1}{\sqrt{x^v}} = (x^v)^{-1/2} = (x_m^v \cdot 2^{x_e^v})^{-1/2} = 2^{-x_e^v/2} \cdot (x_m^v)^{-1/2}. \tag{8}$$

Here only x_m^v needs complex operations $\frac{1}{\sqrt{}}$ and $x_e^v/2$ can be handled by shift. We use a LUT to represent the function $\frac{1}{\sqrt{}}$

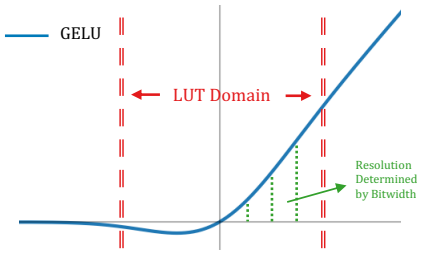


Fig. 5: The LUT domain covers the non-linear region of the GELU function and its bitwidth affects the precision.

to reduce the computation overhead.

$$x = \begin{cases} LUT_{\frac{1}{\sqrt{}}}(x_m^v)2^{(-\frac{x_e^v}{2})}, & x_e^v \bmod 2 = 0, \\ LUT_{\frac{1}{\sqrt{}}}(\frac{x_m^v}{2})2^{(-\frac{x_e^v+1}{2})}, & x_e^v \bmod 2 \neq 0. \end{cases} \quad (9)$$

A key benefit of our approach is that the required LUT entries are significantly smaller due to excluding $x_e^v/2$, and the area of the dynamic shift operator in rescaling is small due to the low precision of mantissas.

Figure 4 illustrates the accuracy loss of the model over different LUT entries. We observe that a minimum of 4 bits is required by x_m^v to preserve the model accuracy within a 1% loss. Table II compares our optimization with related work on accuracy loss at the system level. We observed that 5 bits are required for MXInt ViTs when combined with optimizations on other layers. Still, our optimization saves a significant number of LUT entries compared to the vanilla MXInt operator implementation. Compared to related work using integer-based datapath optimization, our work achieves minimal accuracy loss with a significantly smaller bitwidth, thanks to the dynamic ranges provided by the shared exponent.

2) GELU Approximation: The mathematical definition of the GELU function involves complex arithmetic operations with high quantization sensitivity, particularly for small input values [20]. The standard formulation of GELU is as follows.

$$\text{GELU}(x) = \frac{x}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt \quad (10)$$

Existing work approximates it into a polynomial-based error function (erf) to reduce the computational overhead [2], [9].

$$\text{GELU}(x) \approx \frac{x}{2}\left[1 + L_{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \quad (11)$$

Benefiting from the small bitwidth of mantissas in MXInt, we further push the boundaries of such an approximation and propose a LUT-based method to map GELU into efficient hardware operators in MXInt.

A key challenge in the LUT-based optimization is that traditional designs still require input values containing both the exponent and the mantissa, leading to a significant number of entries. However, this overhead may be affordable for MXInt designs, thanks to the small mantissa bitwidth of MXInt values. We present an efficient LUT-based optimization that partitions its input domain into three parts, as illustrated in Figure 5. The absolutely higher ranges on both ends are approximated to the ReLU function, and the small value range is mapped into a LUT.

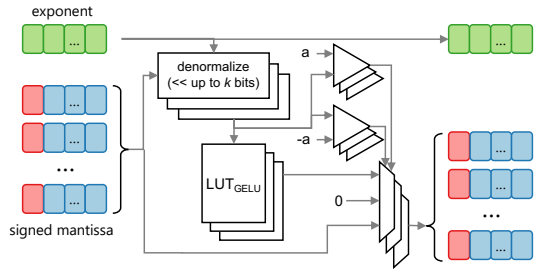


Fig. 6: Optimized datapath for MXInt GELU. k = LUT bitwidth + log_2(LUT domain) - 1.

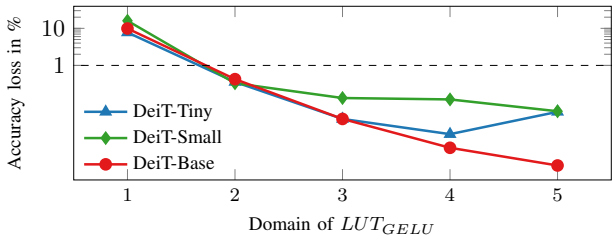


Fig. 7: Accuracy loss over different LUT domains for GELU. Bitwidth = 8.

$$y = \begin{cases} x, & x \geq a \\ LUT_{GELU}(x) & \text{if } -a < x < a \\ 0, & x \leq a \end{cases} \quad (12)$$

The optimized datapath is illustrated in Figure 6. Since there is only a small difference between the input and the output of the GELU function, the exponent value does not change and is directly forwarded to the output. For the small values, each MXInt value as a floating-point number is cast to a fixed-point number and passes through the LUT. The proposed design involves two design constraints. First, a defines the LUT domain that covers the non-linear range for small values, and the bitwidth of the LUT determines the resolution of the curve. Second, the bitwidth of the LUT determines the resolution of the curve.

The design space explorations of both the LUT domain and its bitwidth are shown in Figure 7 and Figure 8. In Figure 7, we evaluate the tradeoff between accuracy and LUT entries in the LUT domain. A larger domain covers a wider non-linear range; however, more bits are required to preserve the same resolution. A similar tradeoff is observed with the LUT entries. For a fixed LUT domain, a smaller bitwidth leads to fewer LUT entries, reducing circuit area at the cost of accuracy loss. In the figures, we show that a minimal domain, $a = 3$, and a minimal bitwidth of 4 are required for MXInt ViTs. These small values make our LUT-based approach amenable to mapping onto an area-efficient datapath.

We compare our optimization with related work in Table III. Both Huang *et al.* [9] and HeatViT [2] exploit the integer-

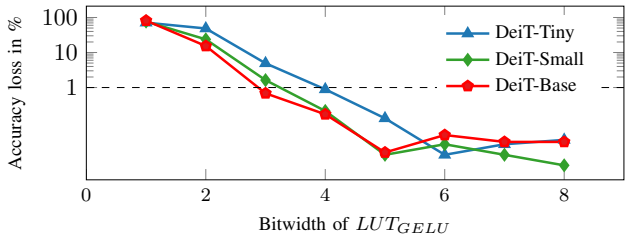


Fig. 8: Accuracy loss over different bitwidths for GELU. LUT domain = 3.

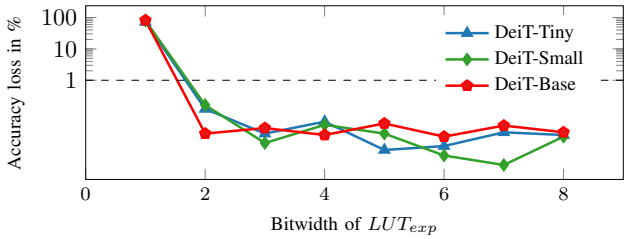


Fig. 9: Accuracy loss over different bitwidths of LUT_{exp}.

TABLE III: Comparison with related work on GELU optimization for DeiT-Tiny. Our work achieves the minimal accuracy loss on fabric at the smallest bitwidth.

Methods	On Fabric	Bitwidth	Accuracy	Accuracy Loss
Original	✗	16	72.13%	-
Auto-ViT-Acc[3]	✗	16	72.13%	0
HeatViT [2]	✓	8	71.95%	0.137%
[9]	✓	8	71.95%	0.137%
SDA [5]	✓	8	70.03%	2.132%
Vanilla MXInt	✓	14	72.102%	0.030%
Optimized MXInt	✓	**5**	72.084%	**0.048%**

TABLE IV: Comparison with related work on Softmax optimization for DeiT-Tiny. Our work achieves the minimal accuracy loss on fabric at the smallest bitwidth.

Methods	On Fabric	Bitwidth	Accuracy	Accuracy Loss
Original	✗	16	72.13%	-
Auto-ViT-Acc[3]	✗	16	72.13%	0
HeatViT [2]	✓	8	71.65%	0.482%
[9]	✓	8	71.916%	0.216%
SDA [5]	✓	8	72.12%	0.012%
Vanilla MXInt	✓	16	72.148%	+0.016%
MXInt to match SDA	✓	5	72.118%	0.014%
Optimized MXInt	✓	**2**	72.01%	**0.122%**

based approximation in Equation (11). They preserve high accuracy, but the bitwidth remains large. SDA [5] approximates GELU into the ReLU6 function [21] for Stable Diffusion models but loses significant accuracy for ViTs due to their sensitivity to precision. In comparison, our approach achieves the best accuracy at the smallest bitwidth. Our MXInt-specific optimization also saves a significant number of entries compared to the vanilla LUT-based MXInt implementation.

3) Softmax Approximation: The Softmax function rescales the elements in a tensor to be in the range between 0 and 1, with the sum of the output tensor being 1 [22]. The standard arithmetic expression of the Softmax operation is as follows.

$$y_i = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{13}$$

x_i and y_i represent the ith elements of the input and output tensors, respectively. This operation is expensive in circuit area because it requires hardware units to perform exponential functions and divisions. These hardware units must be implemented in a general form to compute with arbitrary values.

Integer-based datapath optimization for Softmax has been widely studied. A popular approximation is to replace it with the subtraction of the maximum element of the sum dimension [23], [24], [9], [2], [25], [5]. However, this method cannot be directly applied to MXInt values due to their shared exponent. The `llama.cpp` project [24] proposes an optimization that separates the mantissa and exponent parts of a floating-point value for efficient computation. However, this method targets the CPU architecture and still relies on its efficient exp hardware peripherals.

We adopt the approach by `llama.cpp` and extend it to MXInt datapath optimization. Specifically, we propose a LUT-based approach that transforms the exponent e^x into an expression of integers n and a fixed-point number r as follows.

$$e^x = 2^{x \log_2^e} \tag{14}$$

$$n = \lfloor fp2Int(x \log_2(e)) \rfloor \tag{15}$$

$$r = x \log_2(e) - n \tag{16}$$

$$e^x = 2^{n+r} = 2^n \cdot 2^r \tag{17}$$

$fp2Int$ casts the floating-point format to fixed-point numbers. Instead of computing 2^r using arithmetic operators, we choose to map the function into a LUT, shown as follows.

$$2^r \approx LUT_{pow2}(r) \tag{18}$$

$$e^x \approx 2^n \cdot LUT_{pow2}(r) \tag{19}$$

Benefiting from the efficient representation of MXInt, only a small number of bits are required to represent n and r. n is handled by the shift operation, and its shift bits are determined by the bitwidth of r. An efficient r could significantly save both the unrolled shift operator area and the number of LUT entries.

A key benefit of this optimization is that the output values are already represented in a floating-point number format ($2^n \cdot LUT_{pow2}(r) = 2^{x_e} x_m$), and can be passed to perform a division efficiently, as shown below:

$$\frac{x_1}{x_2} = \frac{x_{m1} 2^{x_{e1}}}{x_{m2} 2^{x_{e2}}} = \frac{x_{m1}}{x_{m2}} 2^{x_{e1} - x_{e2}} \tag{20}$$

This leads to a significant reduction in circuit area compared to the vanilla approach that casts all values to floating-point before computing exp.

Figure 9 shows the design space exploration for the bitwidth of r over different models. We observed that the bitwidth required for r is small, only requiring two bits to preserve

TABLE V: Model accuracy over different quantization techniques and precisions. MXInt quantization achieves lower bits compared to traditional approaches. MXInt$N$ means N-bit mantissa. The exponent is always 8-bit in MXInt. The block size is 16 for activations and 256 for parameters.

Precision		DeiT Tiny		DeiT Small		DeiT Base	
Parameters	Activations	%	Δ%	%	Δ%	%	Δ%
Float32	Float32	72.13	0.00	79.83	0.00	81.80	0.00
Float8	Float8	71.26	-0.87	79.26	-0.57	81.74	-0.06
Int16	Int16	71.85	-0.28	79.34	-0.49	80.05	-1.75
Int8	Int8	0.10	-72.03	0.11	-79.72	0.10	-81.70
MXInt8	MXInt8	72.04	-0.09	79.80	-0.03	81.84	0.04
MXInt6	MXInt8	71.56	-0.57	79.42	-0.41	81.72	-0.08
MXInt6	MXInt6	70.97	-1.16	79.13	-0.70	81.70	-0.10
MXInt4	MXInt6	54.61	-17.52	70.21	-9.62	77.53	-4.27

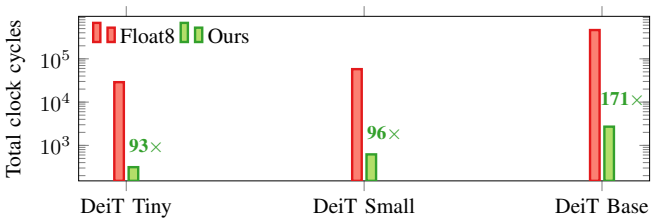


Fig. 10: Speedups achieved by our work (shown in green).

high model accuracy. This leads to an efficient datapath design for the exponential function in MXInt. Table IV compares our design with related work. We present an intermediate design point to compare with the state-of-the-art design by SDA [5] and show that our approach achieves the highest accuracy with the smallest bitwidth. The optimized datapath in our final designs only requires two bits under the accuracy loss budget.

IV. Experiments

In this section, we evaluate the proposed MXInt datapath optimization by addressing the following questions.

1) What are the bitwidth saving breakdown by quantization the whole ViTs in MXInt?
2) What are the system speedup achieved by MXInt datapath optimization bring?
3) What are the insights for designing future MXInt accelerators?

We evaluated our work on the DeiT family [10], including DeiT Tiny, DeiT Small, and DeiT Base, on the ImageNet dataset [11]. All of them were obtained directly from PyTorch Image Models [26]. We evaluated the model accuracy on PTQ results to show the scalability of our approach so that no GPU computing time is required for fine-tuning. We target Alveo U250 FPGAs due to their availability, but our datapath optimization results are independent of the FPGA platform. The version of Xilinx software used was 2023.2. Our results were obtained from cycle-accurate simulation and the implementation reports from Vivado. The results of related work were obtained from their publications. To ensure fairness, we compare our work with the non-pruned HeatViT [2] and focus on MXInt datapath optimization.

TABLE VI: Area saving in LUT entries and accuracy loss by our approximation techniques. The LUT entries are reduced by at least 16×. The bitwidths are determined by greedy search.

Operation	Approach	LUT entry bits	Accuracy loss in %		
			DeiT Tiny	DeiT Small	DeiT Base
	Float32	-	79.83	72.13	81.80
GELU	vanilla LUT	14	0.03	0.04	0.01
	Our work	5	0	0.07	0.06
Softmax	vanilla LUT	16	0.01	0	0.03
	Our work	2	0.03	0.02	0.03
LayerNorm	vanilla LUT	13	0.11	0.14	0.06
	Our work	5	0.55	0.43	0.08

1) Bitwidth Saving Evaluation: We first discuss the effectiveness of our work based on model accuracy. Table V shows the model accuracy over a set of quantization approaches. As mentioned before, we focus on PTQ results to ensure fairness, as different approaches may have their own fine-tuning optimizations. In the table, integer quantization struggles to preserve high accuracy with smaller bits. This is because the numerical variance between tensor elements is huge and requires a large range to be represented. On the other hand, both the standard floating-point format and the MXInt format have an exponent to scale these values, leading to a large dynamic range. With the same bitwidth, MXInt shows better accuracy compared to Float8 because it has five more mantissa bits at the price of sharing the exponent. We apply a greedy quantization search and determine the minimal bitwidth required for lossless quantization in each ViT model, as shown in the table. The final precision is used for hardware mapping in Table VII. Finally, it can also be seen that larger models tend to have better tolerance to low-bit quantization, which suggests our approach will be more effective as we expect future model sizes will continue to increase. Also, the PTQ results avoid GPU time on retraining, making our quantization approach more applicable when training resources are limited.

The evaluation of our datapath optimization on non-linear operations at the system level is shown in Table VI. In the table, it can be seen that a significant number of entries are saved by our datapath optimizations, where reducing a bit leads to half of the LUT size. Besides high area efficiency, our approximation techniques preserve high model accuracy, and the accuracy loss is negligible.

2) System Performance Evaluation: We also compared our work with the corresponding floating-point implementation Figure 10. The red bars represent the same architecture in Float8, the green bars represent our work without and with inter-layer pipelining, respectively. We chose Float8 rather than Float32 as our baseline because the 8-bit floating-point format has a similar bitwidth to our MXInt format, leading to a fair comparison. We compare our work with the same architecture in Float8 to show the effectiveness of MXInt mapping. The independent exponent operators in Float8 lead to significant overhead in both area and memory size, which limits the design space for data parallelism and inter-layer

TABLE VII: Comparing our work with related work on ViT accelerators is unfair due to the differences in hardware microarchitectures (e.g. systolic arrays and dataflow circuits) and algorithm optimizations (e.g. fine-tuning and PTQ). In this work, we focus on MXInt datapath optimization and only report our system results of MXInt ViTs to be self-contained.

Models	Methods	Platforms	Hardware resources								Performance				
			kLUTs	%	DSPs	%	BRAM	%	URAM	%	Power (W)	Fmax (MHz)	GOPs/s	FPS	FPS/W
DeiT Tiny	[9]	ZCU102	144	52.7	1268	50.3	-	-	-	-	-	300	616.10	245.28	-
	HeatViT [2]	ZCU102	116	42	1739	69	289	32	-	-	8.01	150	197.86	78.30	9.77
	Ours	U250	1163	67	24	0.2	330	12	529	41	40.08	183	1488.34	589.44	14.71
DeiT Small	[9]	ZCU102	144	52.7	1268	50.3	-	-	-	-	-	300	762.70	89.69	-
	HeatViT [2]	ZCU102	130	48	1754	70	493	54	-	-	10.10	150	239.81	25.90	2.57
	Auto-ViT-Acc [3]	ZCU102	185	67	1552	62	-	-	-	-	9.63	150	907.80	99.70	10.35
	Ours	U250	860	50	12	0.1	1476	55	820	64	28.37	192	2861.03	309.21	10.89
DeiT Base	HeatViT [2]	ZCU102	145	53	1786	71	664	73	-	-	11.04	150	395.80	11.20	1.01
	Auto-ViT-Acc [3]	ZCU102	186	68	1556	62	-	-	-	-	9.31	150	1181.50	34.00	3.65
	Ours	U250	744	43	8	0.1	497	18	1089	85	25.25	179	2332.37	66.06	2.61

pipelining. The latency is significantly worse than that of the MXInt design in the green bars. Our approach achieves at least a 93× speedup by translating Float8 operators into efficient MXInt hardware in a lossless form. Still, the speedup for large models is restricted by the available hardware resources on the FPGA and can be further exploited in larger FPGAs such as the V80 device [27].

3) Discussion on Limitations: We include related work on ViT accelerators in Table VII with our work. The related work cannot be directly compared to our results due to two main reasons. First, they map part of their models on the CPU, while we map the whole models on the fabric, leading to different energy efficiencies. Second, they fine-tune the model using different software optimization techniques, while we focus on datapath optimization without fine-tuning. Our quantization results offer a lower bound for our approach and can be significantly improved with fine-tuning.

In the table, existing work focuses on fixed-point quantization and acceleration. Their fixed-point operators fully exploit existing DSP blocks for acceleration, as the DSPs are hardened on-chip and amenable to efficient fixed-point/floating-point multiplication and addition. In our work, the DSP blocks are significantly underutilized as we focus on general datapath optimization for both ASIC and FPGA designs. The proposed accelerator is bounded by the LUT resources as most of the operators are mapped into LUTs. Efficient logic synthesis for MXInt operators to FPGA-specific IPs, such as DSP blocks, is outside our scope but is key future work, where our work provides an initial baseline.

V. Related Work

Microscaling Quantization: Sharing certain components for a block of values has been widely recognized as the state-of-the-art technique for quantizing CNNs [28], [29]. Further explorations within this line of research have investigated grouping numbers at various granularities, including layer-wise [30], channel-wise [31], and vector-wise quantization [32]. In addition, many block floating-point variants [33], [32], [6] have been proposed, with the core idea of grouping values into multiple blocks and elements within each block sharing common digits. The closest piece related to our work is by Darvish *et al.* [6] that proposes MXInt quantization for DNN accelerators. This work is later extended to multi-level MX quantization, also known as MXFP, where the shared component can be non-integers [34]. They focus on MXInt quantization and overlook hardware optimization, while our work proposes MXInt-specific datapath optimization with design space exploration.

Quantized Transformer Accelerators: Quantization for efficient ML inference on accelerators has been widely studied [35], [36], [37], [38], especially using fixed-point numbers [39], [40], [41], [42], [43], [44]. Other work customizes hardware architectures for efficient inference [45], [46], [47], [48], [49], [50]. GOBO [51] and EdgeBERT [52] exploit software and hardware co-designs for accelerating transformers. FACT [53] and FlightLLM [1] exploit mixed-precision quantization using fixed-point numbers on linear layers. They only exploit quantization with fixed-point numbers, while we target MXInt quantization.

In the domain of ViT accelerators, existing work focuses on fixed-point quantization [3], [2], [9], while we propose MXInt quantization with hardware optimizations. They only accelerate part of the models on the FPGA, while our hardware accelerator computes the complete workload of the model.

VI. Conclusions

In this work, we propose fully hardware-accelerated ViTs in MXInt and customize hardware datapath optimization for MXInt operators. We explore the tradeoff between model accuracy and area efficiency of the MXInt hardware operators and determine a balance for efficient acceleration. Given the block sharing property of the MXInt format, our proposed datapath optimization techniques further tailor operators with complex circuitry to efficient LUTs. Our optimization realizes all operations mapped to efficient hardware on the fabric and has shown significant improvements compared to vanilla floating-point designs.

Acknowledgments

This work was supported by the Advanced Research + Invention Agency (ARIA), UK. We also thank ARIA for their research network.

REFERENCES

[1] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang, *et al.*, "Flightllm: Efficient large language model inference with a complete mapping flow on fpga," *arXiv preprint arXiv:2401.03868*, 2024.

[2] P. Dong, M. Sun, A. Lu, Y. Xie, K. Liu, Z. Kong, X. Meng, Z. Li, X. Lin, Z. Fang, *et al.*, "Heatvit: Hardware-efficient adaptive token pruning for vision transformers," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 442–455, IEEE, 2023.

[3] 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," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 109–116, IEEE, 2022.

[4] D. Alexey, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv: 2010.11929*, 2020.

[5] G. Yang, Y. Xie, Z. J. Xue, S.-E. Chang, Y. Li, P. Dong, J. Lei, W. Xie, Y. Wang, X. Lin, *et al.*, "Sda: Low-bit stable diffusion acceleration on edge fpgas," in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 264–273, IEEE, 2024.

[6] B. Darvish Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner, *et al.*, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," *Advances in neural information processing systems*, vol. 33, pp. 10271–10281, 2020.

[7] C. Zhang, J. Cheng, I. Shumailov, G. A. Constantinides, and Y. Zhao, "Revisiting block-based quantisation: What is important for sub-8-bit llm inference?," 2023.

[8] E. Samson, N. Mellempudi, W. Luk, and G. A. Constantinides, "Exploring fpga designs for mx and beyond," in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 304–310, IEEE, 2024.

[9] M. Huang, J. Luo, C. Ding, Z. Wei, S. Huang, and H. Yu, "An integer-only and group-vector systolic accelerator for efficiently mapping vision transformer on edge," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[10] H. Touvron, M. Cord, and H. Jégou, "Deit iii: Revenge of the vit," in *European conference on computer vision*, pp. 516–533, Springer, 2022.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[12] W. Kahan, "Ieee standard 754 for binary floating-point arithmetic," *Lecture Notes on the Status of IEEE*, vol. 754, no. 94720-1776, p. 11, 1996.

[13] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, "Understanding the potential of fpga-based spatial acceleration for large language model inference," *ACM Transactions on Reconfigurable Technology and Systems*, 2024.

[14] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 40–47, IEEE, 2016.

[15] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, pp. 65–74, 2017.

[16] H. Ye, H. Jun, and D. Chen, "Hida: A hierarchical dataflow compiler for high-level synthesis," *arXiv preprint arXiv:2311.03379*, 2023.

[17] S. Coward, G. A. Constantinides, and T. Drane, "Automating constraint-aware datapath optimization using e-graphs," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[18] K. Alexandridis and G. Dimitrakopoulos, "Online alignment and addition in multiterm floating-point adders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2024.

[19] PyTorch LayerNorm, 2025.

[20] PyTorch GELU, 2025.

[21] PyTorch ReLu6, 2025.

[22] PyTorch Softmax, 2025.

[23] Z. Li and Q. Gu, "I-vit: Integer-only quantization for efficient vision transformer inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17065–17075, 2023.

[24] llama.cpp, 2025.

[25] P. Dong, L. Lu, C. Wu, C. Lyu, G. Yuan, H. Tang, and Y. Wang, "Packqvit: Faster sub-8-bit vision transformers via full and packed quantization on the mobile," *Advances in Neural Information Processing Systems*, vol. 36, pp. 9015–9028, 2023.

[26] Pytorch Image Models, 2023.

[27] AMD Alveo V80 Compute Accelerator, 2025.

[28] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," 2017.

[29] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," 2018.

[30] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," *arXiv preprint arXiv:1802.04680*, 2018.

[31] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[32] S. Dai, R. Venkatesan, M. Ren, B. Zimmer, W. Dally, and B. Khailany, "Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 873–884, 2021.

[33] S. B. Harma, C. Sönmez, B. Falsafi, M. Jaggi, and Y. Oh, "Accuracy boosters: Epoch-driven mixed-mantissa block floating-point for dnn training," *arXiv preprint arXiv:2211.10737*, 2022.

[34] B. Darvish Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, A. More, L. Melnick, M. Golub, G. Varatkar, *et al.*, "With shared microexponents, a little shifting goes a long way," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pp. 1–13, 2023.

[35] R. Andri, B. Bussolino, A. Cipolletta, L. Cavigelli, and Z. Wang, "Going further with winograd convolutions: Tap-wise quantization for efficient inference on 4x4 tiles," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 582–598, IEEE, 2022.

[36] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "Drq: dynamic region-based quantization for deep neural network acceleration," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1010–1021, IEEE, 2020.

[37] A. H. Zadeh, M. Mahmoud, A. Abdelhadi, and A. Moshovos, "Mokey: Enabling narrow fixed-point inference for out-of-the-box floating-point transformer models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 888–901, 2022.

[38] Y. Zhao, C. Liu, Z. Du, Q. Guo, X. Hu, Y. Zhuang, Z. Zhang, X. Song, W. Li, X. Zhang, *et al.*, "Cambricon-q: A hybrid architecture for efficient training," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 706–719, IEEE, 2021.

[39] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Llm. int8 (): 8-bit matrix multiplication for transformers at scale," *arXiv preprint arXiv:2208.07339*, 2022.

[40] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[41] G. Xiao, J. Lin, M. Seznec, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," *arXiv preprint arXiv:2211.10438*, 2022.

[42] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27168–27183, 2022.

[43] F. Liu, N. Yang, and L. Jiang, "Psq: An automatic search framework for data-free quantization on pim-based architecture," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*, pp. 507–514, IEEE, 2023.

[44] F. Liu, N. Yang, H. Li, Z. Wang, Z. Song, S. Pei, and L. Jiang, "Spark: Scalable and precision-aware acceleration of neural networks via efficient encoding," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1029–1042, IEEE, 2024.

[45] J. Wu, J. Zhou, Y. Gao, Y. Ding, N. Wong, and H. K.-H. So, "Msd: Mixing signed digit representations for hardware-efficient dnn acceleration on fpga with heterogeneous resources," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 94–104, IEEE, 2023.

[46] H. Fan, T. Chau, S. I. Venieris, R. Lee, A. Kouris, W. Luk, N. D. Lane, and M. S. Abdelfattah, "Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 599–615, IEEE, 2022.

[47] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee, *et al.*, "Aˆ 3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 328–341, IEEE, 2020.

[48] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 616–630, IEEE, 2022.

[49] S.-C. Kao, S. Subramanian, G. Agrawal, A. Yazdanbakhsh, and T. Krishna, "Flat: An optimized dataflow for mitigating attention bottlenecks," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 295–310, 2023.

[50] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 977–991, 2021.

[51] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 811–824, IEEE, 2020.

[52] T. Tambe, C. Hooper, L. Pentecost, T. Jia, E.-Y. Yang, M. Donato, V. Sanh, P. Whatmough, A. M. Rush, D. Brooks, and G.-Y. Wei, "Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, (New York, NY, USA), p. 830–844, Association for Computing Machinery, 2021.

[53] Y. Qin, Y. Wang, D. Deng, Z. Zhao, X. Yang, L. Liu, S. Wei, Y. Hu, and S. Yin, "Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pp. 1–14, 2023.