FMA Is (Almost) All You Need: Sharing FMA Resources for Linear and Nonlinear Function Acceleration in Attention-Based Networks

Arpan Suravi Prasad*, Andrea Belano†‡, Davide Rossi†‡, Francesco Conti†, Luca Benini*†
*IIS, ETH Zurich, Switzerland; †DEI, University of Bologna, Italy; ‡Chips-IT, Italy
{prasadar, lbenini}@iis.ee.ethz.ch, {andrea.belano2, davide.rossi, f.conti}@unibo.it

***Abstract*—Attention-based networks, namely Transformers and variants, are widely adopted across modern AI workloads due to their strong generalization capabilities. While execution in attention-based networks is dominated by linear operations that are efficiently accelerated by specialized matrix multiplication units, the growing diversity of nonlinear operations such as activation functions, softmax, and layer normalization has emerged as a key performance and energy bottleneck. Existing solutions mostly rely on dedicated nonlinear hardware units or software emulation, which incur high area or performance overhead. In this work, we present a fused-multiply-add (FMA)-centric approach for efficient nonlinear execution. Our approach leverages the abundant FMA units already present in GEMM accelerators to approximate a wide range of nonlinear functions, including reciprocal and inverse square root, using reconfigurable piecewise polynomial approximation (PWPA), without compromising native GEMM performance. We demonstrate the effectiveness of this approach by extending an open-source GEMM accelerator, enabling the efficient execution of softmax, a wide range of activation functions (GELU, SiLU, etc.), and layer normalization with low area overhead (10%). Implemented in TSMC 7 nm technology node and operating at 1 GHz, we achieve a peak throughput of 3.9, 3.9, and 7.8 GOp/s for softmax, layer normalization, and activation functions; corresponding energy efficiencies of 92, 98, and 172 GOp/J. Our flexible, low-overhead approach improves area efficiency by 1.7× and energy efficiency by 1.46× for softmax compared to existing work.**

I. Introduction

Transformers are widely deployed across Artificial Intelligence (AI) tasks due to their superior generalization capabilities and high accuracy, enabling sophisticated applications such as Generative AI (GenAI) and contextual AI while leveraging similar network architectures across multimodal settings. This has led to their widespread adoption from cloud to edge, transforming Natural Language Processing (NLP), vision, recommendation, and decision-making workloads.

While Convolutional Neural Networks (CNNs) [1], [2] dominated vision workloads for the past decade and drove the development of specialized matrix compute units [3]–[6], transformer models introduce a different computational profile centered on attention [7] mechanisms, where nonlinear softmax operations require exponentiation and division. Moreover, simple ReLU activations are replaced by more complex nonlinear functions [8], and inference-time batch normalization with static parameters is replaced by layer normalization [9], which computes normalization parameters dynamically and requires online division and square root operations.

The continued acceleration of linear operators has made nonlinear operations the new performance bottleneck [10], [11]. Modern attention-based networks [12]–[15] incorporate a wide range of nonlinear functions to improve model accuracy and generalization. These functions often involve transcendental operations, such as exponentiation, as well as expensive arithmetic operations, including division and square roots, all of which are challenging to accelerate efficiently in hardware.

While approximate hardware accelerators have been proposed to accelerate the softmax function, nonlinear activation functions remain inefficiently supported [11], [16] and require additional hardware support. Moreover, designing specialized accelerators for individual nonlinear functions is impractical due to their limited adaptability to newly emerging nonlinear functions [15]. Piecewise Polynomial Approximation (PWPA) has been widely studied and demonstrated high accuracy across diverse workloads [8], [17], leading to its adoption in RISC-V cores, Vector Processing Units (VPUs), as well as loosely coupled accelerators [8], [17]–[19].

While Piecewise Linear Approximation (PWLA) on VPU is a promising approach to accelerating nonlinear operations [8], it incurs higher area overhead due to the large number of partitions required to achieve acceptable accuracy, approaching the complexity of higher-degree approximations. On the other hand, dedicated accelerators employing a lightweight datapath for PWPA [19] have been proposed. However, they still require additional arithmetic units such as adders and multipliers to support softmax computation and necessitate explicit data movement and synchronization when operating alongside a general matrix-matrix multiplication (GEMM) accelerator. Furthermore, these approaches lack support for division and square root operations, necessitating additional hardware units (e.g., dedicated divider and square root units), thereby increasing area and design complexity.

In this work, we propose a novel method to extend an existing GEMM hardware accelerator to natively support softmax, activation, and layer normalization operations without compromising baseline GEMM performance. The proposed approach incurs low area overhead while achieving high accuracy, low latency, and energy-efficient execution of nonlinear operations. By leveraging the abundant Fused Multiply-Add (FMA) units already present in GEMM accelerators, our FMA-centric approach enables tightly coupled and efficient evaluation of higher-degree PWPAs for activation, exponential, inverse, and inverse square root functions. As a result, a single GEMM

accelerator can support both linear and nonlinear layers of Transformer models, eliminating synchronization overheads associated with loosely coupled accelerators and significantly reducing area overhead compared to dedicated nonlinear units. Our contributions are summarized as follows:

1) We propose a low-area-overhead (10%) approach for extending existing GEMM accelerators to support widely used nonlinear operations by reusing abundant FMAs units for nonlinear function computation (e.g., exponential, inverse, inverse square root, SiLU, GELU) via reconfigurable non-uniform higher-degree PWPA, as well as for auxiliary operations such as input shifting and reduction in softmax, and mean and variance computation in layer normalization.
2) We demonstrate the accuracy of our approach by replacing the GELU, exponential, and inverse operations in softmax, as well as the inverse square root operation in layer normalization, with our degree-3, 8-partition PWPA in pretrained Vision Transformer (ViT) variants (ViT-B and ViT-L), and evaluating Top-1 classification accuracy. We observe no measurable degradation in Top-1 accuracy compared to the pretrained baselines.
3) We analyze the hardware cost of implementing our approach on the open-source GEMM accelerator RedMulE [5], with 8×8 FMA array. Implemented in $7\,\mathrm{nm}$ technology and operating at $1\,\mathrm{GHz}$, the proposed design incurs a 10% area overhead without affecting the critical path of RedMulE. It achieves peak throughputs of 3.9 GOp/s, 3.9 GOp/s, and 7.8 GOp/s, with corresponding energy efficiencies of 92 GOp/J, 98 GOp/J, and 172 GOp/J for softmax, layer normalization, and activation functions, respectively. Overall, the proposed approach is 1.7× more area-efficient than existing work and achieves 1.46× higher energy efficiency for softmax acceleration.

II. Related Work

Attention-based networks employ multiple nonlinear functions throughout their execution pipeline, enabling higher model accuracy and improved generalization. Key nonlinear functions include the exponential operation used in softmax and activation functions such as SiLU and GELU. Additional complex operations include online inverse and inverse square root computations, which are commonly used in softmax and layer normalization.

Given the diversity of nonlinear functions, prior work has explored various approximation methods for transcendental functions, including Lookup Table (LUT), polynomial approximations, and PWPA. Many existing approaches focus on approximate computing hardware to accelerate the exponential operation in softmax [10], [11], [15], [16], either by proposing dedicated softmax accelerators or through Instruction Set Architecture (ISA) extensions [20] that leverage existing Floating Point Unit (FPU) compute resources. While attention is a key nonlinear kernel in Transformers, the approximation accuracy of such approaches is typically evaluated on a limited set of workloads. Moreover, accelerating softmax alone shifts the performance bottleneck to other nonlinear functions, necessitating additional hardware support for their acceleration [11].

On the other hand, PWPA has been widely studied and shown to provide high accuracy across diverse workloads for activation and exponential operations. For example, [8] employs PWLA and demonstrates that using 32 partitions achieves less than 1% error across more than 700 workloads. In contrast, [17] proposes an ISA extension that leverages higher-degree PWPA to achieve superior accuracy, while [19] introduces a lightweight datapath to perform PWPA without relying on full-fledged, area-intensive FMAs.

While these solutions enable highly accurate nonlinear function approximation, they incur significant area overhead. For example, [8] instantiates a partition detector and coefficient selector for each VPU lane, and the large number of partitions required for PWLA further increases area cost. Similarly, [19] incurs additional area overhead despite employing a lightweight datapath, due to the need for extra streamers, dedicated control, and specialized datapath support for PWPA.

Moreover, these works primarily report activation throughput and power consumption, without profiling other kernels such as softmax and layer normalization in isolation. While [8] can reuse the VPU for auxiliary operations, its efficiency on such kernels is not evaluated. In contrast, [19] requires additional hardware support to handle auxiliary operations beyond nonlinear function evaluation. Finally, none of the above works demonstrate support for inverse or inverse square root operations, which are integral to softmax and layer normalization and would require additional high-overhead hardware units, typically division or square root units [20].

In this work, we demonstrate higher-degree PWPA by leveraging the abundant FMAs available in GEMM accelerators to enable native, high-throughput execution of elementwise nonlinear functions. Employing higher-degree polynomials reduces coefficient storage and simplifies partition detection logic while maintaining high approximation accuracy. We further extend the GEMM datapath to support softmax by mapping auxiliary shift and reduction operations onto existing FMAs. By recomputing intermediate values instead of storing them, we reduce memory accesses for softmax operations by 25%. Overall, the proposed approach incurs only a 10% area overhead and achieves throughput of 3.9 GOp/s, 3.9 GOp/s, and 7.8 GOp/s, with corresponding energy efficiencies of 92 GOp/J, 98 GOp/J, and 172 GOp/J for softmax, layer normalization, and activation functions, respectively, without requiring additional hardware units. This tightly coupled, area-efficient design eliminates the synchronization overheads associated with loosely coupled accelerators, removes explicit data transfers, and enables higher-throughput operation.

III. Background

A. Softmax Computation

Softmax is a core operation in attention mechanisms, mapping a vector of scores to a normalized probability distribution. For an input vector $\mathbf{x} = x_i{}_{i=1}^{N}$, softmax is defined as

$$\mathrm{softmax}(x_i) = \frac{e^{x_i - x_{\max}}}{\sum_{j=1}^{N} e^{x_j - x_{\max}}}, \qquad x_{\max} = \max_j x_j. \quad (1)$$

The computation involves subtraction, exponentiation, reduction, and division, making softmax both computationally and

numerically intensive. Subtracting $x_{\max}$ ensures numerical stability by preventing overflow during exponentiation. Efficient hardware support for softmax therefore requires accurate and efficient implementations of exponentiation, accumulation, reciprocal, and multiplication.

B. Layer Normalization

Layer normalization is widely used in Transformer models to normalize activations across the feature dimension. Given an input vector $\mathbf{x} = x_i{}_{i=1}^{N}$, layer normalization computes the mean and variance over the feature dimension. The normalized output is then given by

$$\mu = \mathbb{E}[x], \quad \sigma^2 = \mathbb{E}[x^2] - \mathbb{E}[x]^2, \quad \mathrm{LN}(x_i) = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}, \tag{2}$$

where ε is a small constant for numerical stability. This operation requires online computation of the moments and an inverse square root.

C. PWPA

Given a target function $y(\cdot)$ over a bounded interval $[a, b)$, we partition the domain into P non-uniform sub-intervals and approximate y on each interval using a degree-d polynomial:

$$\hat{y}(x) = \sum_{i=0}^{d} c_{p,i}\, x^i, \qquad x \in I_p, \quad p \in \{0, \ldots, P-1\}, \tag{3}$$

where $I_p = [\alpha_p, \alpha_{p+1})$ denotes the p-th partition of the input domain, with $\{\alpha_p\}_{p=0}^{P-1}$ defining ordered partition boundaries. Enabling PWPA on a GEMM accelerator requires only a few comparators and multiplexers for coefficient selection, while polynomial evaluation reuses existing FMAs units.

D. Inverse Approximation

We approximate the division in softmax by computing the inverse of the denominator and multiplying it by the numerator. We assume the input to the inverse operation lies in a normalized range, which holds in practice since the denominator in softmax is a sum of positive values, and the maximum input $x_i = x_{\max}$ yields a normalized output of 1.

For a positive normalized input $x = 2^E \cdot m$ with $m \in [1, 2)$, we perform domain reduction by (i) extracting the unbiased exponent E from the floating-point exponent field and applying exponent negation ($E \leftarrow -E$), and (ii) approximating m^{-1} using PWPA over the reduced domain $m \in [1, 2)$:

$$x^{-1} = 2^{-E} \cdot m^{-1}, \qquad m^{-1} \approx \mathrm{PWPA}_{[1,2)}(m). \tag{4}$$

This approach enables efficient reciprocal computation using lightweight domain-reduction logic and PWPA, eliminating the need for a dedicated divider.

E. Inverse Square Root Approximation

Inverse square root is required in layer normalization. As in Section III-D, we assume a normalized input; however, this cannot always be guaranteed. We therefore add a small constant ε to ensure a well-defined positive normal input.

For a positive normalized value $x = 2^E \cdot m$, with $m \in [1, 2)$,

$$x^{-1/2} = 2^{-E/2} \cdot m^{-1/2}. \tag{5}$$

To avoid fractional exponents when E is odd, we fold the parity of E into the mantissa,

$$z = \begin{cases} m, & E \text{ even} \\ 2m, & E \text{ odd}, \end{cases} \qquad z \in [1, 4), \tag{6}$$

and define $k = \lfloor E/2 \rfloor$, yielding

$$x^{-1/2} = 2^{-k} \cdot z^{-1/2}. \tag{7}$$

We approximate $z^{-1/2}$ over $[1, 4)$ using PWPA, thereby enabling inverse square root computation using the existing PWPA datapath without requiring a dedicated square root unit.

$$z^{-1/2} \approx \mathrm{PWPA}_{[1,4)}(z), \tag{8}$$

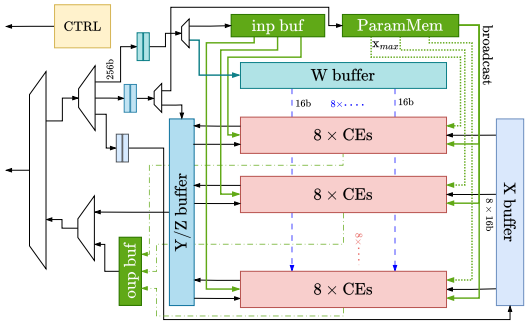

Fig. 1: RedMulE architecture adapted to support nonlinearity, with Nonlinearity blocks and paths highlighted in green

IV. Architecture

A. RedMulE Architecture Background

RedMulE is an open-source FP16 matrix multiplication accelerator designed to efficiently compute GEMM of the form

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathbf{Y}, \tag{9}$$

where $\mathbf{X} \in \mathbb{R}^{M \times N}$, $\mathbf{W} \in \mathbb{R}^{N \times K}$, $\mathbf{Y} \in \mathbb{R}^{M \times K}$, and $\mathbf{Z} \in \mathbb{R}^{M \times K}$. The accelerator employs a semi-systolic datapath to enable efficient data reuse. The main compute engine consists of FP16 FMA units organized into L rows and H columns. Within each row, the FMAs are connected in a systolic fashion. The input operand $\mathbf{X}$ is unique to each FMA and remains stationary for multiple cycles, while the weight operand $\mathbf{W}$ is broadcast across columns, enabling both temporal and spatial reuse of inputs and weights. The bias operand $\mathbf{Y}$ is stored in the Y/Z buffer, supplied to each row during accumulation, and the final output is written back to the Z buffer after processing.

Data movement from main memory is handled by dedicated streamers through a single interface multiplexed across different initiators. In this work, we use the RedMulE configuration with H=8, L=8, and a pipeline depth of P=2, resulting in 64 FP16 FMAs units capable of delivering a peak throughput of 128 Op/cycle. Control is implemented using a Finite State Machine (FSM), with configuration values stored in memory-mapped registers programmed by an external RISC-V host.

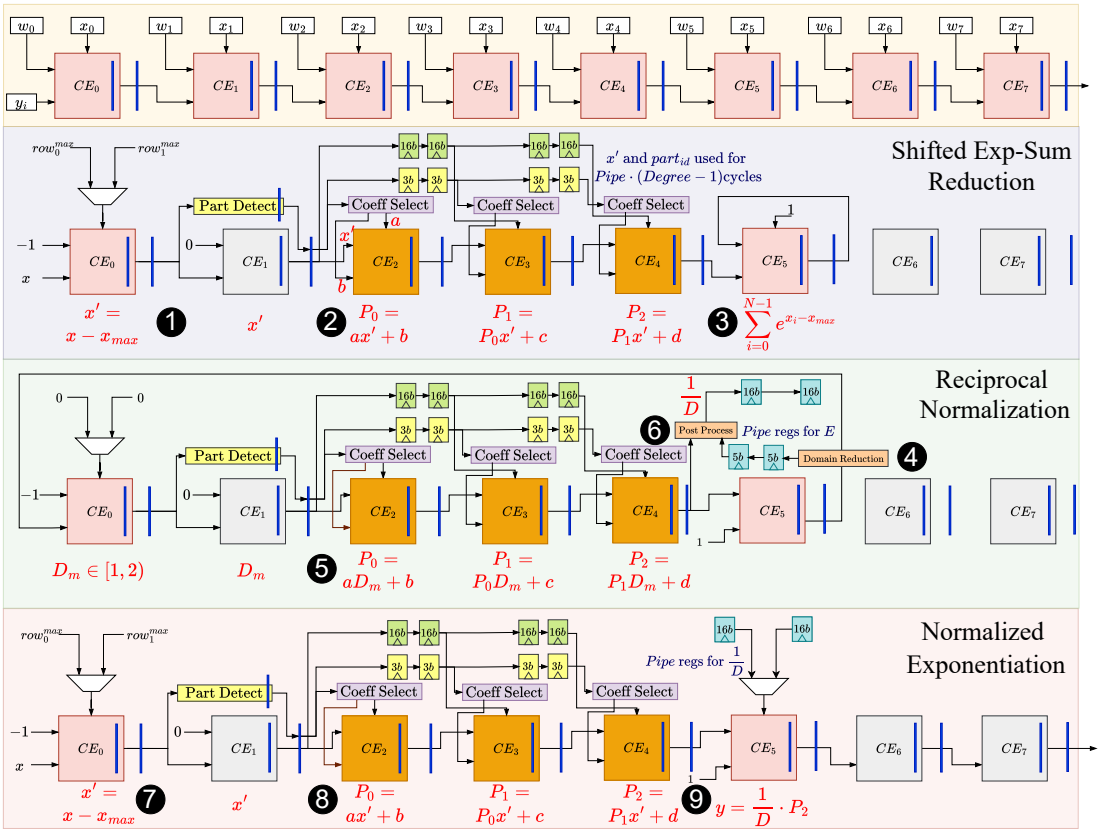

Fig. 2: Softmax computation in RedMulE. The top row illustrates the vanilla RedMulE row. The second row shows the input shifting, exponentiation, and reduction stages. The third row depicts the inverse computation, and the fourth row presents the normalized exponentiation (leveraging PWPA using Horner's scheme) and output streaming.

B. Support for Nonlinearity on RedMulE

The modifications introduced to support nonlinear operations are highlighted in green in Figure 1. A dedicated input buffer is added to stream data from the W FIFO in nonlinearity mode. The results of the nonlinearity operation are written to an output buffer and streamed out through the Z FIFO. The reuse of the existing RedMulE buffers is intentionally avoided, as this decoupled architecture enables the pipeline execution of linear and nonlinear operations without interference. The parameter memory stores the breakpoints and coefficients required for PWPA and broadcasts them to all rows. In contrast, values such as $x_{\max}$ in softmax are unique to each row and are connected per row. For each row, partition detection is implemented as a binary tree that compares the shifted input against the breakpoints broadcast from ParamMem and computes the final partition ID through a binary tree of multiplexers. This partition ID then drives mux-based coefficient selection for each polynomial stage.

Softmax and LayerNorm are executed according to the pseudocode in Algorithm 1 and Algorithm 2. The architectural modifications for single-row softmax execution with degree-3 PWPA and 8 partitions are illustrated in Figure 2. We choose this configuration to reduce partition-detection overhead while maintaining high accuracy. As shown in [17], degree-3 PWPA with 8 partitions achieves accuracy comparable to that of degree-1 PWPA with 64 partitions, which has been shown to provide high accuracy across a wide range of workloads [8].

ParamMem is first initialized with the breakpoints and coefficients for the exponent, along with $x_{\max}$. The value $x_{\max}$ is first broadcast to the first Computing Element (CE) of each row (❶), with P values selected in a ping-pong manner. The first CE performs input shifting to produce x', while the second CE forwards x'. In parallel, a binary-tree partition detector compares x' against the breakpoint set and resolves the 3-bit partition ID for the 8 partitions through mux stages; both the partition ID and 16-bit x' are then pipelined for polynomial

Algorithm 1: Softmax

$x_{\max} \leftarrow \max_i x_i$
for $i = 0$ to $N-1$ **do**
  $z_i \leftarrow x_i - x_{\max}$
  $e_i \leftarrow \mathrm{PwPA}(\exp(z_i))$
end for
$d \leftarrow \sum_i e_i$
$r \leftarrow \mathrm{PwPA}(d)$
for $i = 0$ to $N-1$ **do**
  $z_i \leftarrow x_i - x_{\max}$
  $e_i \leftarrow \mathrm{PwPA}(\exp(z_i))$
  $y_i \leftarrow e_i \cdot r$
end for

Algorithm 2: LayerNorm

$s \leftarrow 0,\ q \leftarrow 0$
for $i = 0$ to $N-1$ **do**
  $s \leftarrow s + x_i$
  $q \leftarrow q + x_i^2 \cdot (1/N)$
end for
$\mu \leftarrow s \cdot (1/N)$
$\sigma^2 \leftarrow q - \mu^2$
$r \leftarrow \mathrm{PwPA}(\sigma^2)$
for $i = 0$ to $N-1$ **do**
  $y_i \leftarrow (x_i - \mu) \cdot r$
end for

evaluation. In step ❷, the PWPA is initiated using coefficients selected by mux trees driven by the partition ID, with CE2–CE4 completing the remaining degree stages. In step ❸, CE5 accumulates the partial sums to form the denominator.

After processing the sequence across P rows, ParamMem is reloaded with the inverse coefficients. The accumulated denominator is forwarded to the domain reduction unit (❹), which normalizes the FP16 value to $[1, 2)$ and stores the negated exponent in a 5-bit register. Since the P rows are processed in a time-interleaved manner, no additional reduction stage is required. In step ❺, the reciprocal PWPA is evaluated, followed by ❻ post-processing, which recombines the reciprocal mantissa approximation with the stored exponent to produce $1/D$.

ParamMem is subsequently reloaded with the exponent coefficients. In step ❼, the shifted input undergoes exponent evaluation. Using PWPA, ❽ computes the exponent, and ❾ multiplies the result by the stored $1/D$ to generate the final softmax output. During softmax execution, $e^{x'}$ is not written back to memory; instead, once the reciprocal is available, the exponent is recomputed and fused with the multiplication, eliminating one intermediate memory write.

While softmax requires two passes through RedMulE, activation functions require only a single pass. In activation mode, $x_{\max}$ is set to zero; CE1 and CE5–CE7 bypass the input to the output, while CE2–CE4 perform the PWPA evaluation.

Layer normalization, in contrast, requires two iterations. The first pass computes the statistics (mean and second moment), and the second pass applies normalization. Unlike softmax, ParamMem is initialized only once, since only the inverse square root operation is needed. Softmax, however, requires multiple ParamMem reloads for exponentiation, reciprocal, and final exponentiation.

C. System-Level Integration

We integrate RedMulE into the Snitch cluster [21], which consists of a single RISC-V core with an FPU for control and a dedicated RISC-V Direct Memory Access (DMA) core to manage data movement. Both cores are connected to a multi-bank shared L1 memory of 128 KiB. The system also includes a data-mover IP to support data shuffling operations, such as matrix transposition. In this work, we profile RedMulE both in standalone mode and at the system level to evaluate overall performance and power consumption. RedMulE is programmed and controlled by the Snitch compute core.

V. Results

A. Approximation Accuracy Evaluation

We analyze the accuracy of our PWPA (approximations for reciprocal and inverse square root operations, along with other nonlinear functions), using FP16 precision, end-to-end to match RedMulE. To assess the efficacy of our approach across different network depths and embedding dimensions, we evaluate two pretrained ViT variants [13], ViT-B and ViT-L, both with an input resolution of 224. We evaluate the accuracy of our approach by replacing the nonlinear operations with PWPA, using coefficients and breakpoints derived according to the methodology in [17]. The approximation is configured with polynomial degree 3 and 8 partitions.

TABLE I: Classification accuracy and output similarity under nonlinear approximations for ViT-B / ViT-L. Seq. len. = 197; layers = 12 / 24; embed dim. = 768 / 1024.

Configuration	Top1-Accuracy(%)	Top1-Similarity(%)
Baseline	81.33 / 82.52	100.00 / 100.00
Softmax (e^x)	81.33 / 82.53	99.89 / 99.85
Softmax (e^x & $1/x$)	81.33 / 82.54	99.87 / 99.86
GELU	81.34 / 82.55	99.89 / 99.87
LayerNorm ($1/\sqrt{x}$)	81.31 / 82.52	99.94 / 99.94
All	81.35 / 82.55	99.86 / 99.82

The pretrained networks are cast to FP16 to match RedMulE's precision, and the resulting FP16 Top-1 classification accuracy on the ImageNet validation set [22] serves as the baseline. The result is reported using Top-1 accuracy,

$$\text{AccTop-1} = \frac{1}{N}\sum i = 1^N \mathbb{I}(\hat{y}i = y_i), \tag{10}$$

where y_i and $\hat{y}i$ denote the ground-truth and predicted labels. To quantify fidelity, we also report Top-1 similarity, which measures sample-wise agreement between the FP16 baseline and the approximated model beyond overall accuracy.

$$\text{SimTop-1} = \frac{1}{N}\sum i = 1^N \mathbb{I}(\hat{y}_i^{\text{approx}} = \hat{y}_i^{\text{base}}), \tag{11}$$

The accuracy results are summarized in Table I. All experiments are conducted on an NVIDIA GeForce GTX 1080 Ti using Python 3.11.3. Among all configurations, only one case shows a very minor degradation (0.018%) when applying layer normalization approximation in ViT-B. Even in this case, the Top-1 similarity remains high at 99.936%. No accuracy loss is observed in any other configuration, including ViT-L. Across all settings, the similarity score exceeds 99.8% and reaches up to 99.94%, confirming that the proposed approximations preserve baseline prediction behavior.

B. RedMulE area breakdown

We performed the place-and-route of the Snitch cluster with RedMulE integration in TSMC 7 nm technology at an operating frequency of 1 GHz, using Synopsys Fusion Compiler. The resulting area breakdown is shown in Figure 5.

In RedMulE, approximately 60% of the total area is allocated to the compute engine, followed by 16% for buffers and 15% for the streamer. Adding nonlinearity support introduces a ParamMem to store PWPA coefficients and breakpoints, along with additional input and output buffers for streaming data, contributing 15.8% of the total buffer area. The modifications within the engine account for 9.8% of the total engine area, while the additional control overhead is negligible.

A detailed breakdown of the engine overhead shows that the three coefficient selection stages and the partition detector contribute 19% and 15% of the nonlinearity overhead, respectively. The domain reduction logic accounts for only 2.6%, while the remaining 63% corresponds to additional multiplexers and flip-flops introduced to support multiple execution modes. Compared to vanilla RedMulE, these adaptations introduce a 10% area overhead at the accelerator level, and at the system level, these translate to only 1.6% overhead.

C. Nonlinear Operation Performance

We benchmark the throughput and energy efficiency of softmax, activation, and layer normalization (without affine

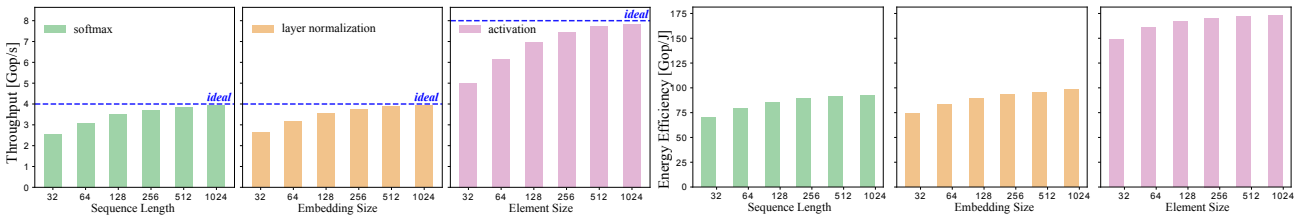

Fig. 3: Throughput and energy efficiency of RedMulE for softmax, activation, and layer normalization under varying workload dimensions. The ideal throughput assumes 100% utilization of RedMulE.

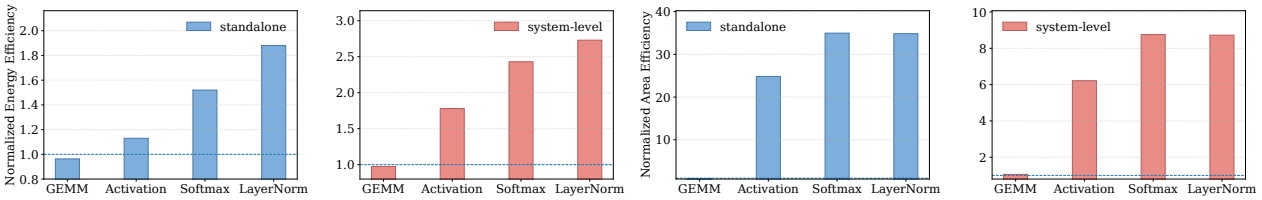

Fig. 4: Normalized energy and area efficiency relative to the baseline RedMulE system. The GEMM baseline corresponds to RedMulE, whereas nonlinear kernels are compared against a PWPA enhanced 64-bit FPU integrated in the Snitch core. Area efficiency accounts for the standalone FPU area in the baseline and only the incremental area overhead in the proposed system. GEMM is evaluated on a kernel with 256k FMAs, while nonlinear kernels are evaluated on 16k elements.

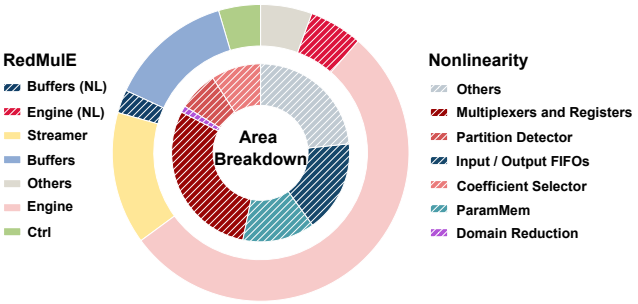

Fig. 5: The outer ring shows RedMulE, highlighting the hardware overhead introduced for nonlinearity support (buffers include ParamMem and input/output FIFOs). The inner ring details the distribution of this additional area across nonlinearity components. Buffer overhead is split into ParamMem and input/output FIFOs, while the remainder corresponds to the compute engine and others.

transformation) by sweeping the sequence length, element count, and embedding dimension from 32 to 1024, respectively. We use 16 rows to utilize 8 rows of RedMulE across two pipeline stages. The throughput at 1 GHz frequency is shown in Figure 3.

We observe that throughput increases with input dimensions, approaching the theoretical peak as the relative overhead of parameter loading and systolic latency is amortized over larger workloads. The peak throughput reaches 3.9 GOp/s for softmax and layer normalization, and 7.8 GOp/s for activations, which require only a single pass through RedMulE.

Energy efficiency is obtained from back-annotated switching activity at 0.75 V, by extracting power for both standalone RedMulE and system-level configurations, as shown in Figure 3. Similar to throughput, energy efficiency increases with input dimensions as overheads are amortized over larger workloads. The standalone peak energy efficiency reaches 92 GOp/J, 98 GOp/J, and 172 GOp/J for softmax, layer normalization, and activation, respectively, corresponding to system-level energy efficiencies of 66 GOp/J, 69 GOp/J, and 121 GOp/J.

D. Comparison to Baseline

We evaluate the overhead of our modifications to RedMulE for a pure GEMM workload at both the standalone and system levels. As shown in Figure 4, the energy efficiency decreases by only 4.2% at the standalone level and 2.8% at the system level due to the additional logic and buffers. With a 10% area overhead, standalone area efficiency decreases by 9.1%. At the system level, area efficiency improves by 1.05% because the baseline RedMulE system includes an FPU for nonlinear operations, whereas the modified system does not. If the FPU is retained in both systems, the system-level area-efficiency loss is below 0.1%.

For workloads requiring nonlinear operations, the cluster with baseline RedMulE relies on the Snitch-integrated FPU, consistent with [11], [20]. Since the original FPU does not support PWPA, we extend it with degree-3 PWPA capability using 8 partitions by adding a partition detector and a PACE memory that stores polynomial coefficients and breakpoints, initialized via DMA and broadcast across four 16-bit lanes within the FP64 FPU. The degree-3 polynomial is evaluated by reusing the existing FMA unit in the FPU across three iterations, thereby avoiding the area overhead of additional FMAs. Activation functions are implemented as degree-3 polynomials through iterative issuance of FMAs across four lanes. To maximize efficiency and match RedMulE's degree-3 capability, we leverage FREP, SSR, hardware loops, and streaming registers during kernel execution.

Our adaptation achieves 1.13× higher energy efficiency at the standalone level and 1.78× at the system level, with higher gains for layer normalization and softmax. These improvements arise from better utilization of RedMulE, elimination of emulation overhead, and high-throughput pipelined execution using abundant FMAs.

We observe even higher area-efficiency gains, ranging from 24× to 34× at the standalone level and from 6× to 8× at the system level for activation functions, layer normalization, and softmax. The general-purpose FPU incurs hardware

and pipeline overhead, which limits throughput, whereas our approach executes nonlinear operations within the existing datapath with minimal additional area.

E. Performance on ViT

We characterize latency and energy per inference using a single attention block of ViT-B. The linear layers are tiled to fit within half of the L1 memory (64 KiB) to enable double buffering. For nonlinear operations, we use workload sizes of $16\times$ the embedding dimension, element count, and sequence length for layer normalization, activation, and softmax, respectively. Tile latency is obtained through cycle-accurate simulation, and total layer latency and energy are computed by scaling with the number of tiles and aggregating across all layers. We measure an execution time of $24.4\,\mathrm{ms}$ and an inference energy of $1.21\,\mathrm{mJ}$ at the system level, with nonlinear operations (softmax, activation, and layer normalization) accounting for 1.2%.

VI. Comparison with SoA

We compare our approach with the existing work, both qualitatively and quantitatively, and capture the results in Table II. We use the RedMulE's peak throughput, area, and energy efficiencies for Softmax, Layer Normalization, and Activation kernels.

Reggiani et al. [8] propose a PWLA implementation on top of a VPU with support for multiple precisions. While similar multi-precision capability is feasible in our approach, we focus on FP16 to match the RedMulE datatype and a commonly used precision for AI workloads. For comparison, we consider their 64-partition configuration, which achieves accuracy comparable to our degree-3, 8-partition PWPA. Although a direct area comparison across different technology nodes is not fair, our architectural choices inherently lead to higher area efficiency. Their design requires a larger parameter memory, partition detector, and coefficient selector due to the higher number of partitions. In contrast, we share the parameter memory across multiple RedMulE rows, reducing storage overhead.

Furthermore, while certain softmax operations can be executed using the FMA units of the VPU, division and square root operations typically use division square root units, which can cost as high as 100kGE per FPU [20]. While the reported raw energy-efficiency numbers appear superior in [8], this is because the reported power does not include the VPU power consumption. When restricting the comparison to only the partition detector and coefficient selection logic, our approach achieves a $5\times$ higher energy efficiency.

Zhu et al. [23] propose a softmax approximation and validate their approach using quantization-aware fine-tuning. In contrast, our approach does not require any fine-tuning and extends beyond softmax to support diverse nonlinear activation functions as well as layer normalization, which is not addressed in [23]. Although a direct area comparison is not fair due to differences in technology nodes, their architecture would require additional dedicated hardware to support activation functions and inverse square root operations needed for layer normalization. From an architectural perspective, this leads to higher area overhead and reduced area efficiency compared to our shared FMA-centric design.

Wang et al. [20] propose an ISA extension to an existing RISC-V core to accelerate softmax. While this approach can be area efficient in isolation, it achieves a $2.73\times$ lower area efficiency compared to our design. Furthermore, the extension targets exponentiation only and relies on the existing FMA units and a division/square-root unit to complete the softmax computation. Additionally, this solution does not support native activation execution and would require either additional dedicated hardware or high-latency software emulation. It can be area efficient for softmax when a suitable RISC-V core is already present in the system; however, in the absence of such a core, the overhead becomes substantial, as a typical core occupies approximately 420 kGE of area. Adding such a core would cause the effective area efficiency to decrease, reaching approximately 0.1% of the area efficiency achieved by our approach. From an energy-efficiency standpoint, our approach is $1.4\times$ more energy efficient in standalone operation. Furthermore, by recomputing the exponent value and reducing memory transfers from four to three, our approach is expected to further improve energy efficiency. This optimization is estimated to provide an additional $1.05\times$ energy-efficiency improvement at the system-level.

Belano et al. [11] propose a dedicated softmax accelerator. Compared to their approach, our solution achieves a $22\times$ higher area efficiency. This improvement stems from two key factors. First, we reuse the existing streamers and FMAs already present in the accelerator, whereas their design requires a standalone instantiation. Second, while they approximate the exponent for softmax, they rely on general-purpose cores to execute activation functions such as GELU and SiLU. As discussed earlier, such cores incur significant area overhead, and introducing them for activation support further reduces overall area efficiency. From an energy efficiency standpoint, our approach achieves a $2.2\times$ higher energy efficiency for the softmax operation.

Prasad et al. [19] propose a lightweight datapath for PWPA. While their design is more area efficient than instantiating full-fledged FMA units, it remains $4.5\times$ less area efficient than our approach. This is primarily due to the overhead of introducing a dedicated nonlinear datapath, along with additional accelerator components such as control and streaming units. From a standalone accelerator perspective, their design achieves $1.55\times$ higher energy efficiency for activation functions. However, this comes at the cost of explicit data movement and synchronization between separate linear and nonlinear units. When accounting for this overhead, the effective energy advantage reduces to approximately $1.1\times$. In contrast, our approach can fuse activation directly with the preceding GEMM operation, eliminating additional synchronization and minimizing data movement. Furthermore, extending such a standalone nonlinear accelerator to support softmax and layer normalization would require additional hardware units, further reducing overall area efficiency.

Yu et al. [18] implement PWPA with 16 partitions and validate accuracy on two workloads, whereas [8] indicates that at least 32 partitions are required to ensure error-safe operation across 700+ workloads. Their reported area and power results are post-synthesis, while ours are post-layout. Even so, our approach achieves $1.7\times$ higher area efficiency. Although [18]

TABLE II: State-of-the-art comparison for Softmax (Soft.), Layer Normalization (LN) and Activation (Act)

Reference	Precision	Kernel Support			Tech	Freq	Area	Throughput (GOp/s)			Area Eff. (GOp/s/mm^2)			Energy Eff. (GOp/J)		
		Soft.	LN	Act	(nm)	(GHz)	(μm^2)	Soft	LN	Act	Soft	LN	Act	Soft	LN	Act
Reggiani et al. [8]	I/FP32/16/8	✗	✗	✓	28	0.6	14857†	–	–	1.2	–	–	81†	–	–	324†
Zhu et al. [23]	FX16§	✓	✗	✗	28	1.64	18392	13.12	–	–	713	–	–	–	–	–
Wang et al. [20]	BFP16	✓	✗	✗	12	1	968‡	0.45	–	–	465‡	–	–	63	–	–
Belano et al. [11]	BFP16	✓	✗	✓	12	1.12	39000	2.25	–	–	57.6	–	–	42	–	–
Prasad et al. [19]	FP32/16	✗	✗	✓	12	1	28178	–	–	15.6	–	–	553	–	–	269
Yu et al. [18]	I32/FP16/32	✓	✓	✓	7	0.74	498	–	–	0.37	–	–	1486	–	–	–
This Work	FP16	✓	✓	✓	7	1	3095‡	3.93	3.94	7.85	**1270**‡	**1273**‡	**2536**‡	**92**	**98**	**172**

† Area and power reported only for the partition detector and coefficient memory; FMA functionality is implemented separately in a VPU. 64 partitions.
‡ Area overhead includes additional hardware required to achieve the reported area efficiency.
§ Requires quantization; high-accuracy format is used for comparison.
⋆ Post-synthesis power and area results. Throughput assumes one element per cycle.

reports $0.025\,\mathrm{mW}$ post-synthesis power, the lack of voltage, activity, and frequency details prevents a fair energy-efficiency comparison. Moreover, loosely coupled accelerators require more area, explicit synchronization and data movement, and supporting softmax would require multiple passes, increasing buffer requirements and data transfers.

In summary, our unified FMA-centric design delivers native, high-accuracy support for softmax, activation functions, and layer normalization without additional nonlinear hardware, achieving $1.7\times$ higher area efficiency and $1.46\times$ higher energy efficiency than prior softmax acceleration approaches.

VII. Conclusion

We presented an FMA-centric approach for accelerating nonlinear operations in attention-based networks by reusing the abundant FMAs already available in GEMM accelerators. By combining lightweight architectural extensions with higher-degree PWPA, the proposed design supports softmax, activation functions, and layer normalization without compromising baseline GEMM functionality. Implemented on top of RedMulE, the proposed approach incurs only 10% area overhead while preserving high approximation fidelity across pretrained ViT models. It achieves peak throughputs of 3.9 GOp/s, 3.9 GOp/s, and 7.8 GOp/s for softmax, layer normalization, and activation functions, respectively, with corresponding energy efficiencies of 92 GOp/J, 98 GOp/J, and 172 GOp/J. Overall, this tightly coupled design eliminates the overheads of loosely coupled nonlinear accelerators and provides an efficient path toward flexible support for evolving Transformer nonlinearities.

Acknowledgment

This work was supported by Meta.

References

[1] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, arXiv:1704.04861.
[2] M. Tan and Q. Le, "EfficientNetV2: Smaller models and faster training," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.
[3] Y.-H. Chen *et al.*, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.
[4] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," 2017, arXiv:1704.04760.
[5] Y. Tortorella *et al.*, "RedMule: A mixed-precision matrix-matrix operation engine for flexible and energy-efficient on-chip linear algebra and TinyML training acceleration," *Future Generation Computer Systems*, vol. 149, pp. 122–135, 2023.
[6] N. Malaya *et al.*, "Accelerating matrix processing with GPUs," in *Proceedings of the IEEE 24th Symposium on Computer Arithmetic (ARITH)*, 2017.
[7] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.
[8] E. Reggiani *et al.*, "Flex-SFU: Accelerating DNN activation functions by non-uniform piecewise approximation," in *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.
[9] J. L. Ba *et al.*, "Layer normalization," 2016, arXiv:1607.06450.
[10] J. R. Stevens *et al.*, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC)*, 2021.
[11] A. Belano *et al.*, "A flexible template for edge generative AI with high-accuracy accelerated softmax and GELU," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 15, no. 2, pp. 200–216, 2025.
[12] S. Mehta and M. Rastegari, "MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer," in *International Conference on Learning Representations (ICLR)*, 2022.
[13] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021.
[14] H. Touvron *et al.*, "LLaMA: Open and efficient foundation language models," 2023, arXiv:2302.13971.
[15] J. Zhu, X. Chen, K. He, Y. LeCun, and Z. Liu, "Transformers without normalization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
[16] G. Islamoglu *et al.*, "ITA: An energy-efficient attention and softmax accelerator for quantized transformers," in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2023.
[17] A. S. Prasad *et al.*, "PACE: An optimal piecewise polynomial approximation unit for flexible and efficient transformer nonlinearity acceleration," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2025.
[18] J. Yu *et al.*, "NN-LUT: Neural approximation of nonlinear operations for efficient transformer inference," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022.
[19] A. S. Prasad *et al.*, "PACE-Lite: Compact and efficient piecewise polynomial approximation for transformer nonlinearity acceleration," in *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, 2025.
[20] R. Wang *et al.*, "VEXP: A low-cost RISC-V ISA extension for accelerated softmax computation in transformers," in *Proceedings of the IEEE 32nd Symposium on Computer Arithmetic (ARITH)*, 2025.
[21] F. Zaruba *et al.*, "Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1845–1860, 2021.
[22] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
[23] D. Zhu *et al.*, "Efficient precision-adjustable architecture for softmax function in deep learning," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3382–3386, 2020.