electronics

MDPI

Article

Hardware Accelerator for Approximation-Based Softmax and Layer Normalization in Transformers

Raehyeong Kim, Dayoung Lee, Jinyeol Kim, Joungmin Park and Seung Eun Lee *

Department of Electronic Engineering, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea; kimraehyeong@seoultech.ac.kr (R.K.); leedayoung@seoultech.ac.kr (D.L.); kimjinyeol@seoultech.ac.kr (J.K.); parkjoungmin@seoultech.ac.kr (J.P.)
* Correspondence: seung.lee@seoultech.ac.kr; Tel.: +82-2-970-9021

Abstract: Transformer-based models have achieved remarkable success across various AI tasks, but their growing complexity has led to significant computational and memory demands. While most optimization efforts have focused on linear operations such as matrix multiplications, non-linear functions like Softmax and layer normalization (LayerNorm) are increasingly dominating inference latency, especially for long sequences and high-dimensional inputs. To address this emerging bottleneck, we present a hardware accelerator that jointly approximates these non-linear functions using piecewise linear approximation for the exponential in Softmax and Newton–Raphson iteration for the square root in LayerNorm. The proposed unified architecture dynamically switches operation modes while reusing hardware resources. The proposed accelerator was implemented on a Xilinx VU37P FPGA and evaluated with BERT and GPT-2 models. Experimental results demonstrate speedups of up to 7.6× for Softmax and 2.0× for LayerNorm, while maintaining less than 1% accuracy degradation on classification tasks with conservative approximation settings. However, generation tasks showed greater sensitivity to approximation, underscoring the need for task-specific tuning.

Keywords: transformer; non-linear function; approximation; FPGA; accelerator

Academic Editors: Ping-Feng Pai and Krzysztof Szczypiorski

Received: 18 April 2025
Revised: 3 June 2025
Accepted: 4 June 2025
Published: 7 June 2025

Citation: Kim, R.; Lee, D.; Kim, J.; Park, J.; Lee, S.E. Hardware Accelerator for Approximation-Based Softmax and Layer Normalization in Transformers. *Electronics* **2025**, *14*, 2337. https://doi.org/10.3390/electronics14122337

1. Introduction

The Transformer architecture, introduced by Vaswani et al. in 2017, marked a major breakthrough in addressing the limitations of traditional natural language processing (NLP) methods [1]. It has since served as the foundation for pre-trained models like bidirectional encoder representations from transformers (BERT) and generative pre-trained transformers (GPT), known for strong contextual understanding and text generation [2,3]. These models have seen widespread adoption not only in NLP but also in fields like computer vision, robotics, and bioinformatics [4–7]. However, as Transformer models grow in depth and complexity, their computational and memory demands are rising rapidly, highlighting the growing need for hardware acceleration to enable real-time inference [8–10]. Transformers consist of both linear and non-linear operations. Most existing hardware acceleration efforts have primarily targeted the optimization of linear components, such as matrix multiplications, by leveraging systolic arrays, operation decomposition, and low-precision arithmetic. These methods have been widely applied in both Transformer-specific accelerators and general-purpose AI processors for convolutional workloads or energy-efficient arithmetic units [11–13]. In contrast, non-linear functions like Softmax and layer normalization (LayerNorm) have received less attention, even though they are invoked repeatedly and are difficult to parallelize, making them key contributors to inference latency. This

bottleneck becomes more pronounced in large-scale language models and high-resolution vision Transformers, where the computational load of Softmax and LayerNorm scales with sequence length and model dimensionality [14,15]. These functions involve complex operations such as exponential and square root evaluations, along with mean and variance computations, which incur considerable latency and hardware cost when implemented with high precision [16,17]. To mitigate this, we adopt piecewise linear approximation (PLA) for the exponential in the Softmax and Newton–Raphson iteration for the square root in LayerNorm [18]. These methods reduce the computational burden while preserving acceptable accuracy in fixed-point hardware environments.

While hardware acceleration of normalization functions has been widely explored in convolutional neural networks (CNNs), the structural differences in computation flow and vector dimensionality make such techniques difficult to transfer directly to Transformer models. Although Softmax and LayerNorm have been individually approximated or accelerated in prior work, few studies have addressed both jointly within a single hardware design [19]. In particular, although these functions are executed sequentially in Transformer pipelines—making dynamic reconfiguration of shared hardware feasible—this architectural opportunity has been largely overlooked. To address this, we propose a unified hardware accelerator optimized for non-linear functions in Transformer models. The design identifies and exploits the shared computational structures of Softmax and LayerNorm, enabling both operations to be flexibly handled within a single fixed-point-friendly module. Approximation strength is tunable to support diverse accuracy–latency trade-offs. The contributions of this work are summarized as follows:

- We present a unified hardware architecture that integrates Softmax and LayerNorm operations into a single computational path within Transformer models. By analyzing their shared computational flows, the proposed design improves overall hardware utilization.
- PLA is applied to the exponential function in Softmax, while Newton–Raphson iteration is used for the square root computation in LayerNorm. The approximation strength is adjustable via parameters h and N, enabling fine-grained control over the trade-off between accuracy and speed.
- The proposed accelerator was implemented on a Xilinx VU37P FPGA and applied to BERT and GPT-2 models for practical evaluation. It achieved speedups of up to 7.6× for Softmax and 2.0× for LayerNorm, with Softmax delivering up to 4.6× faster execution compared to GPU baselines.
- To assess the impact on model accuracy, we performed extensive evaluation across eleven benchmark datasets spanning classification and generation tasks. When $h \leq 4$ and $N \geq 3$, classification performance was preserved with less than 1% degradation. In contrast, generation tasks were more sensitive to approximation, particularly in Softmax, necessitating more conservative parameter choices.

The remainder of this paper is organized as follows. Section 2 provides an overview of the Transformer architecture and presents a latency breakdown analysis that highlights the role of non-linear operations. Section 3 reviews prior studies on approximation techniques and hardware acceleration for Transformers. Section 4 introduces the proposed approximation methods with mathematical formulation, and Section 5 details the hardware architecture based on these methods. Section 6 presents experimental results from applying the proposed accelerator to various Transformer-based tasks. Finally, Section 7 concludes the paper and discusses potential future research directions.

2. Background

2.1. Transformer Overview

The Transformer is a neural architecture that introduced the self-attention mechanism to effectively model relationships between tokens within an input sequence [20]. This design overcomes the limitations of traditional recurrent neural networks (RNNs) that rely on sequential computation [21]. Figure 1 illustrates the structure of the Transformer [1]. It consists of stacked encoder and decoder blocks, each composed of a multi-head attention (MHA) module and a feed-forward network (FFN). LayerNorm is applied before each major block, and residual connections are established between the input and output of the MHA and FFN modules [22].

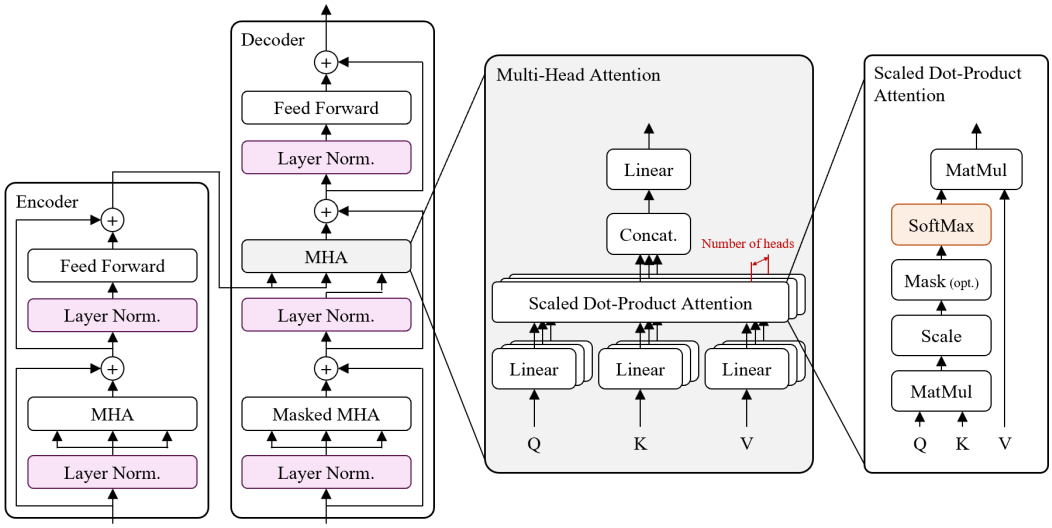


Figure 1. Overall Transformer architecture and attention mechanism breakdown.

In the MHA module, the input matrices—query (Q), key (K), and value (V)—are split into multiple heads, typically denoted by h, with each head independently performing scaled dot-product attention (SDPA). This allows the model to capture diverse contextual relationships. In SDPA, token similarity is computed as the dot product of Q and K, followed by Softmax normalization to produce attention weights, which are then applied to V. The outputs from all heads are concatenated and linearly transformed to form the final attention output. Among the non-linear operations repeatedly applied in each Transformer layer, Softmax and LayerNorm are particularly critical for stabilizing training and preserving model performance.

2.2. Non-Linear Functions in Transformer

Non-linear functions in Transformer architectures are essential for stabilizing training dynamics, enriching model expressiveness, and preserving inference fidelity. Among these, Softmax and LayerNorm are particularly critical, as their row-wise computation and limited parallelism result in significant computational overhead, making them a primary bottleneck as both sequence length and model size increase [23].

2.2.1. Softmax

The Softmax function transforms an input vector into a probability distribution. Within the SDPA mechanism, it normalizes token-wise similarity scores into attention

weights, ensuring that the sum of all outputs equals one [24]. The function is formally defined in Equation (1).

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, \quad \text{for } i = 1, 2, \ldots, N \tag{1}$$

Here, x denotes the attention score vector obtained by the dot product of the Q and K matrices. For an input sequence of length N, the attention module computes an $N \times N$ score matrix per attention head. Since Softmax is applied row-wise, the exponential and summation operations must be performed N times over N elements, resulting in a computational complexity of $O(N^2)$ per head. For models with h attention heads, this scales to $O(hN^2)$. This quadratic growth makes Softmax a major computational bottleneck, particularly in long-sequence Transformer architectures [25].

2.2.2. Layer Normalization

LayerNorm stabilizes training by normalizing each input vector using its mean and variance, thereby mitigating issues such as exploding or vanishing gradients in deep networks [26]. The operation is defined in Equation (2).

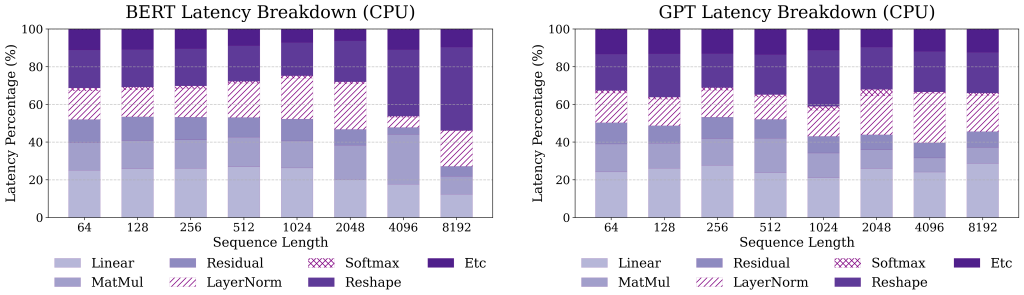
$$\text{LayerNorm}(x_i) = \gamma \cdot \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2}$$

In this formulation, μ and σ^2 represent the mean and variance of the input vector, while γ and β are learnable scaling and shifting parameters. Given a sequence length N and model dimension d, LayerNorm operates on each of the N vectors of size d, performing mean and variance computation, normalization, and an affine transformation. This yields an overall complexity of $O(Nd)$. Despite its relatively low theoretical number of floating-point operations per second (FLOPs), LayerNorm can often contribute disproportionately to inference latency due to its memory access patterns and element-wise operations.

2.3. Latency Breakdown Analysis

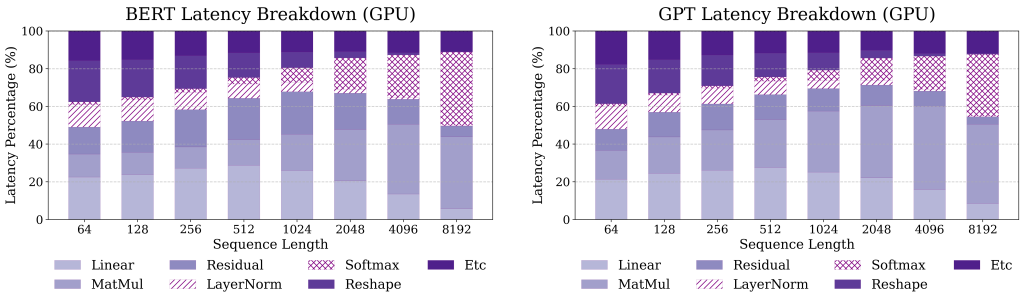
To understand the latency behavior of non-linear functions in Transformers, we analyzed operation-wise latency in two representative NLP models: BERT and GPT. The measurements were performed on both GPU and CPU platforms across a range of sequence lengths, from short inputs typical of classification tasks to long contexts used in generative models. This analysis highlights how the contributions of operations such as Softmax and LayerNorm vary with sequence length and hardware architecture.

Figure 2 shows that, on CPUs, LayerNorm consistently accounts for a larger share of inference latency than Softmax across various sequence lengths. This is primarily due to limited parallelism and irregular memory access patterns, which have a greater impact than arithmetic complexity in CPU environments. Figure 3 illustrates that, on GPUs, the latency contribution of Softmax increases significantly with sequence length, reaching over 30% at 8192 tokens. This trend reflects the $O(N^2)$ time complexity discussed in Section 2.2.1 and the data-dependent memory access behavior of Softmax, both of which contribute to persistent latency bottlenecks even in highly parallel environments. These results indicate that both LayerNorm and Softmax are critical latency bottlenecks in Transformer inference pipelines. LayerNorm is more dominant on CPUs, whereas Softmax has a greater impact on GPUs. With recent models such as GPT-4 Turbo supporting up to 128 K tokens, and others like Mistral and Claude handling tens of thousands, the computational burden of these non-linear functions has become increasingly significant. As sequence lengths continue to grow, hardware-level acceleration of both Softmax and LayerNorm is essential to ensure the scalability and real-time performance of future Transformer models.



(**a**) (**b**)

Figure 2. Latency breakdown of BERT and GPT models on CPU. (**a**) BERT latency breakdown by operation and sequence length. (**b**) GPT latency breakdown by operation and sequence length.



(**a**) (**b**)

Figure 3. Latency breakdown of BERT and GPT models on GPU. (**a**) BERT latency breakdown by operation and sequence length. (**b**) GPT latency breakdown by operation and sequence length.

3. Related Works

3.1. Normalization Acceleration in Transformers

Previous research on accelerating non-linear operations has largely focused on batch normalization (BatchNorm), a well-known bottleneck in CNNs. Due to its ability to enhance training stability and merge with linear operations, BatchNorm has been widely targeted for hardware optimization. Based on these advantages, several studies have explored applying BatchNorm to Transformer models or reusing existing CNN accelerators. Dong et al. proposed an FPGA-based accelerator that replaces LayerNorm with BatchNorm in Swin Transformers [17]. Their design enables fusion with linear layers and implements integer-based approximations for Softmax and GELU, achieving up to 14.6× and 5.05× energy efficiency improvements over CPU and GPU baselines, respectively. However, structural modifications were needed to maintain training stability, and issues such as batch dependency and slight accuracy loss remained. Similarly, Guo et al. introduced PRepBN, a progressive approach that gradually replaces LayerNorm with re-parameterized BatchNorm (RepBN) during training [27]. This method preserves training convergence and generalization but adds computational overhead and shows limited generality across Transformer variants.

While these works aim to leverage BatchNorm acceleration in Transformers, structural differences between the two normalization schemes impose constraints. As shown in Figure 4a, BatchNorm normalizes across batch and channel dimensions, whereas LayerNorm operates at the token level, resulting in fundamentally different data shapes and computation flows. These limitations highlight the need for dedicated, lightweight LayerNorm acceleration techniques tailored to Transformer architectures—an increasingly critical requirement for efficient real-time hardware inference.

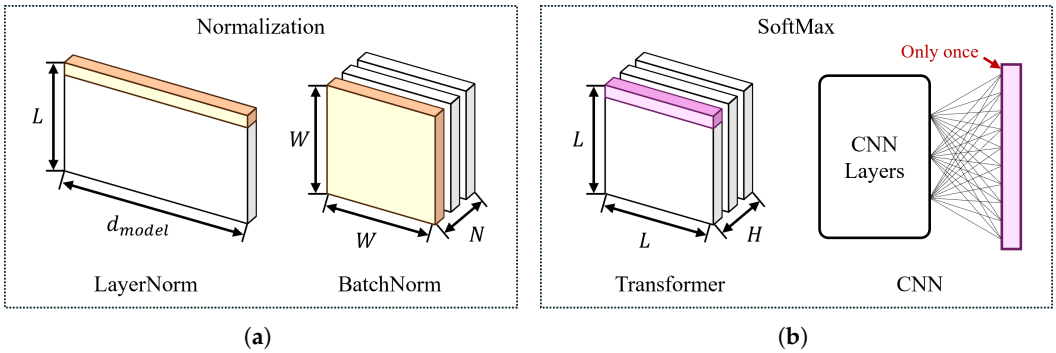


Figure 4. Comparison of Softmax and normalization usage between Transformers and CNNs. (**a**) Operation granularity of Softmax: Repeated per token in Transformers vs. Single application in CNNs (**b**) Normalization scope per operation: Token-wise LayerNorm vs. Channel-wise BatchNorm.

3.2. Softmax Approximation Techniques

As shown in Figure 4b, Softmax was traditionally applied only once in the final classification layer of CNNs, leading to minimal computational load and limited hardware acceleration efforts. However, with the increasing adoption of Transformer architectures, its usage has grown significantly. This shift has elevated Softmax into a major computational bottleneck, thereby increasing the demand for effective hardware acceleration strategies. Stevens et al. proposed Softermax, which replaces the natural base e with 2 in the exponential calculation and combines low-precision fixed-point arithmetic with online normalization [28]. This LUT- and shift-based design achieved a 2.35× energy efficiency and 0.90× area efficiency improvement without retraining, with less than 0.5% accuracy loss. Zhu et al. introduced a simplified fixed-point Softmax architecture with a precision-tuning parameter P, allowing dynamic control over the accuracy–efficiency trade-off, thus making the design adaptable to a wide range of applications [29]. Mei et al. proposed TEA-S, a Softmax accelerator based on piecewise linear approximation computation (PLAC) [30]. By reducing pipeline delay from 2N to N cycles in a single-pass streaming structure, it halved execution latency while keeping accuracy loss within 2%. Gao et al. combined segment-wise LUT-based exponential approximation with low-order Taylor expansions, and designed a log-domain shift–subtract normalization unit [31]. Their 16-bit fixed-point implementation preserved accuracy while reducing computation overhead.

3.3. Joint Optimization of Softmax and LayerNorm

More recently, joint optimization of both Softmax and LayerNorm has been explored. Wang et al. proposed SOLE, a hardware–software co-optimized design for both functions [32]. It incorporates E2Softmax, a LUT-free, multiplication-free approximation based on binary logarithms, and AILayerNorm, an integer-based statistical computation unit. These components are implemented as separate hardware modules, and while they enable deployment without retraining and achieve significant speedups (36.2× for Softmax and 61.3× for LayerNorm), the architecture lacks dynamic configurability and does not support datapath sharing between the two functions. Cha et al. introduced a dedicated vector processing unit (VPU) for accelerating both Softmax and LayerNorm in transformer inference [33]. Their design parallelizes mean and variance computation, reduces loop iterations from six to three for LayerNorm, and employs FP32 accumulators to maintain high numerical precision. However, this structure results in high hardware resource usage, including increased LUT and FF counts, and follows a fixed execution flow that limits flexibility and scalability—especially for lightweight or embedded inference scenarios.

Table 1 summarizes and compares prior work on the hardware acceleration of nonlinear functions. Most existing works target either Softmax or LayerNorm individually or process them through separate hardware modules, which limits opportunities for resource sharing and integrated scheduling. Architectures such as SOLE demonstrate the benefits of low-precision approximation, while VPU-based designs achieve high accuracy with pipeline efficiency—but often at the cost of increased complexity or lack of tunability. To address these limitations, our work proposes a unified accelerator architecture that executes both Softmax and LayerNorm within a single reconfigurable datapath. By analyzing the operations common to both functions, the proposed design shares hardware components through a lightweight FSM-based mode switch. Furthermore, the use of PLA for exponentials and Newton–Raphson iteration for square roots enables tunable approximation strength, allowing trade-offs between latency and accuracy to be adjusted according to task requirements. This flexibility, combined with a compact hardware design, makes the proposed architecture well suited for real-time and resource-constrained Transformer inference.

Table 1. Comparison of prior works on Softmax and LayerNorm acceleration.

Paper	Target	Technique	Precision	HW-Int.	Config.
Dong et al. [17]	LN	Replace LN with BN for inference	Fixed-point	No	No
Guo et al. [27]	LN	Gradual LN-to-BN replacement during training	Mixed (BN + LN)	No	Yes
Stevens et al. [28]	SM	Use base-2 exp and online normalization	Fixed-point	No	No
Zhu et al. [29]	SM	Simplified log-sum-exp with tunable parameter	Fixed-point	No	Yes
Mei et al. [30]	SM	PLAC-based approximation in one-pass pipeline	INT8	No	No
Gao et al. [31]	SM	Use LUT and Taylor series for exp/log	Fixed-point	No	No
Wang et al. [32]	SM + LN	Log2-based SM and integer stats LN	INT4–INT8	No	No
Lu et al. [34]	SM + LN	Log-sum-exp SM and parallel LN accumulation	Fixed-point	No	No
Cha et al. [33]	SM + LN	Parallel mean/var and loop fusion optimization	FP32 Accumulator	Yes	No

SM: Softmax, LN: LayerNorm, BN: BatchNorm. HW-Int.: Whether Softmax and LayerNorm are integrated in a shared hardware architecture. Config.: Whether the design supports tunable precision or approximation parameters. "Mixed (BN + LN)" refers to progressive replacement of LayerNorm with BatchNorm during training.

4. Proposed Methods

Softmax and LayerNorm operations involve exponential and square root computations, respectively, both of which are computationally demanding and challenging to implement efficiently in hardware. To address these challenges, this study adopts piecewise linear approximation (PLA) for the exponential function in Softmax and the Newton–Raphson iterative method for square root computation in LayerNorm. These techniques were chosen not only for their ability to maintain acceptable numerical precision while significantly reducing computational complexity and hardware resource consumption, but also for their fast approximation speed [35,36].

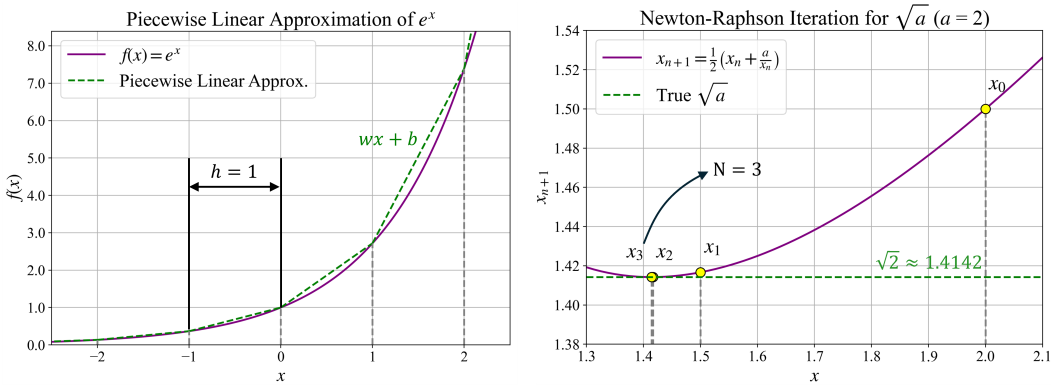
4.1. Piecewise Linear Approximation

The exponential function e^x, used in the Softmax operation, is one of the most computationally intensive and hardware-expensive operations. Direct implementation incurs high latency and considerable circuit complexity, necessitating efficient approximation techniques. To address this, we adopt a piecewise linear approximation (PLA) method [35]. In this study, the input domain of x is fixed to the range $[-16, 16]$, uniformly divided into intervals of length h. Figure 5a compares the exact exponential function with its piece-wise linear approximation when $h = 1$. A smaller h improves approximation accuracy but increases the number of segments and the memory required to store the corresponding

slope w_i and intercept b_i for each segment. Each interval $[x_i, x_{i+1}]$ is approximated by a linear function, as shown in Equation (3):

$$e^x \approx w_i x + b_i, \quad \text{for } x \in [x_i, x_{i+1}] \tag{3}$$

Here, w_i and b_i are the slope and intercept for the i-th interval, respectively. These coefficients are precomputed and stored in on-chip memory. Given an input x, the hardware identifies the corresponding interval and performs one multiplication and one addition, enabling simplified dataflow, low latency, and reduced hardware overhead. Constraining the input range aligns well with the characteristics of the self-attention mechanism in Transformers. Attention scores, computed via dot products between Q and K, are typically normalized and scaled during the Transformer pipeline, which suppresses extreme values and confines most Softmax inputs to a moderate range. As a result, most values fall within -10 to 10, making the fixed domain assumption both practical and efficient. These characteristics support the use of linear approximations over a bounded domain. When evaluated over the range $[-16, 16]$ with a PLA interval of $h = 1$, the mean relative error was 8.62%. Despite this moderate error, the Softmax function primarily depends on the relative differences between input values to construct the output distribution. By segmenting the input space into uniform intervals and applying linear approximations, this method achieves low memory usage and minimal control overhead, making it well-suited for parallel hardware architectures.



(**a**) (**b**)

Figure 5. Approximation techniques for non-linear functions (**a**) PLA of the exponential function with interval width $h = 1$. (**b**) Newton–Raphson iteration for square root calculation with convergence shown for $a = 2$ and $N = 3$ iterations.

4.2. Newton–Raphson Method

LayerNorm requires computing the square root of the variance term to normalize input vectors. This operation is computationally intensive and hardware-expensive, especially in fixed-point arithmetic where direct square root computation is impractical. To address this, we adopt the Newton–Raphson method—a well-known numerical technique with fast convergence—for approximating square roots [36]. It estimates the root of the function $f(x) = x^2 - a$, where a is the value whose square root is sought [37]. The iterative update rule is given in Equation (4).

$$x_{n+1} = \frac{1}{2} \cdot \left(x_n + \frac{a}{x_n} \right) \tag{4}$$

This recurrence reliably converges to $\sqrt{a}$, and the convergence rate depends on the choice of the initial guess x_0. In our design, we simplify the hardware by setting the initial value $x_0 = a$, thereby eliminating the need for lookup tables or conditional logic. Given that the variance term a in LayerNorm is generally bounded within $[0, 2]$, the Newton–Raphson

method converges rapidly under this initialization. Empirical evaluation over the interval $a \in [0, 2]$, sampled at a resolution of 0.001, shows that the average relative error after 2, 3, 4, and 5 iterations is 3.551%, 1.030%, 0.530%, and 0.403%, respectively. This demonstrates that even with a small number of iterations, sufficient accuracy can be achieved for practical deployment. This yields a compact and uniform dataflow suitable for hardware realization. Each iteration involves one addition, one multiplication, and one division. In fixed-point implementations, the division operation a/x_n is particularly costly. Despite the approximation error introduced by an uninformed initial guess, the Newton–Raphson algorithm rapidly converges to high-accuracy estimates. For example, as illustrated in Figure 5b, computing $\sqrt{2}$ with an initial guess of 2 yields an approximation of 1.4142 after only three iterations. This efficiency enables a balance between hardware simplicity and sufficient numerical precision, making it well-suited for latency-sensitive acceleration of LayerNorm operations.

5. Hardware Implementation

Before implementing the non-linear operations in hardware, we conducted a detailed analysis of the internal computation flow for both Softmax and LayerNorm. As shown in Figure 6, both functions exhibit similar structural patterns at the micro-operation level, except for the exponential and square root computations. Since Transformer models execute Softmax and LayerNorm sequentially during inference, there is no need for concurrent hardware support for both operations.

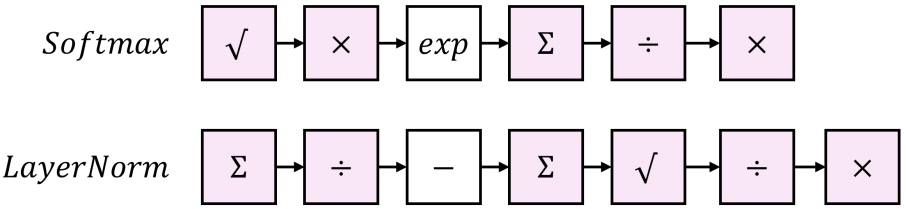


Figure 6. Micro-level operation flow for Softmax and LayerNorm.

Leveraging this sequential behavior, we designed a unified hardware architecture that reuses the same PE array for both Softmax and LayerNorm computations. The accelerator dynamically switches its operation mode depending on the task, thereby minimizing area overhead without requiring separate processing units for each function. Accordingly, we used a Q5.26 fixed-point format for all operations except the exponential function in Softmax, which employed a Q26.5 format to ensure adequate numerical precision. The accelerator core is organized into three functional components: a configuration register that stores key model parameters such as input dimensionality and operation modes; an exponential_mem block containing the precomputed slopes and intercepts required for PLA of the exponential function; and an Operator module that executes computations in accordance with the configuration. Figure 7 shows the architecture of the accelerator core, which consists of three major components: a 32-element processing element (PE) array, an accumulator unit, and a square root module. Each module is described in the following subsections.

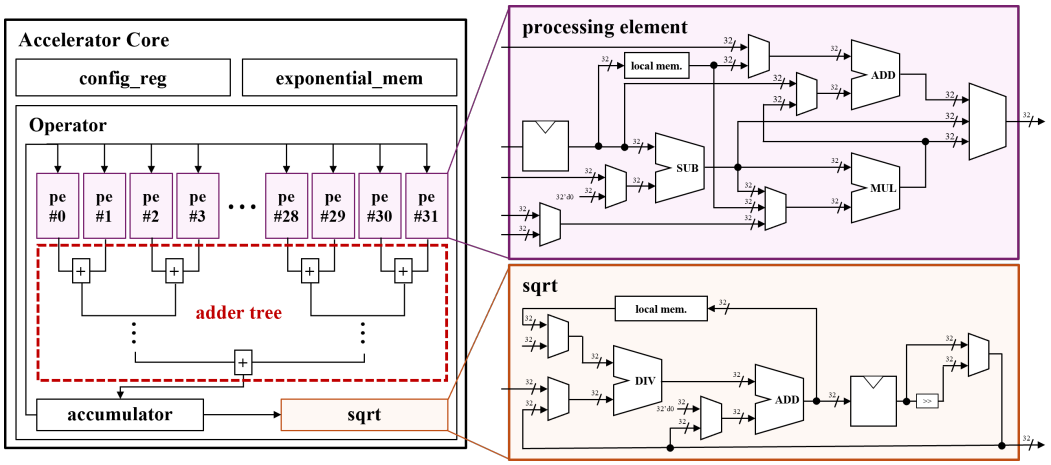


Figure 7. Architecture of the proposed accelerator core.

5.1. Processing Element (PE)

The operator consists of 32 PEs, each containing a 32-bit adder, subtractor, multiplier, local memory, and six multiplexers (MUXes). The MUXes are controlled by a finite-state machine (FSM) that dynamically selects operands based on the current computation phase. Distinct FSM sequences are defined for Softmax and LayerNorm operations, as summarized in Table 2. At runtime, the accelerator receives 32 parallel 32-bit input values per cycle from an external controller. For vectors longer than 32 elements, data are streamed over multiple cycles. The operation mode is specified via a mode selection signal provided at the beginning of execution. Based on this selection, the FSM automatically progresses through a predefined sequence of states:

- LayerNorm: the FSM performs mean and variance computation (LN_MEAN, LN_VAR), computes the standard deviation (LN_STD), normalizes the input (LN_NORM), and applies the learnable scale and bias (LN_WGT, LN_BIAS).
- Softmax: the FSM applies the exponential function (SM_EXP), computes the normalization term via division (SM_DIV), and outputs the result (SM_RSLT).

When LayerNorm mode is selected, the external controller streams the learnable scale and bias vectors, which are stored in each PE's local memory before processing begins. These parameters are then reused as segments of the input vector are processed in parallel across multiple cycles. Each PE processes a segment of the input vector in parallel, and the intermediate results are aggregated via an adder tree and passed to the accumulator.

Table 2. FSM states and their corresponding operations.

State	Operation Description
LayerNorm	
LN_MEAN	Compute mean of input vector
LN_VAR	Compute variance using squared deviation
LN_STD	Apply square root to obtain standard deviation
LN_NORM	Normalize vector by subtracting mean and dividing by std
LN_WGT	Multiply normalized values by scale (weight)
LN_BIAS	Add learned bias term
Softmax	
SM_EXP	Apply exponential function to each element
SM_DIV	Divide each exponential by the total exponential sum
SM_RSLT	Output final Softmax result

5.2. Accumulator

To support varying sequence lengths and embedding dimensions across different Transformer models, the accelerator includes an accumulator that enables flexible vector processing beyond the 32-element PE width. The accumulator receives the adder tree outputs from the PE array and accumulates partial results over multiple cycles if the input exceeds 32 elements. Once accumulation is complete, the result is conditionally routed to the next stage depending on the operation type: if a square root or division is required—such as for variance normalization in LayerNorm or probability normalization in Softmax—the value is sent to the square root module. Otherwise, the result is returned to the PE array for further computation, such as scaling, bias addition, or result aggregation. This conditional routing mechanism allows the shared hardware datapath to support both functions without duplication.

5.3. Square Root and Division

The square root and division module handles Newton–Raphson-based square root approximation and reciprocal-based division for normalization. It is composed of local memory, a 32-bit divider, adder, and input MUXes, and operates under FSM control. To minimize hardware cost, all divisions are performed using reciprocal multiplication, i.e., a/b is computed as $a \times (1/b)$. Precomputed reciprocal values are stored in dedicated lookup tables (LUTs), with different configurations for LayerNorm and Softmax:

- LayerNorm: The input to the square root is generally less than 2. The LUT stores reciprocals of values in the range $[0, 2]$ using uniform intervals or linear interpolation.
- Softmax: The normalization term can be large, ranging from 1 to 10^6, depending on sequence length and input scale. A separate LUT covering $[10^{-6}, 1]$ is used, with log-uniform spacing to maintain precision across a wide dynamic range.

Notably, the entire architecture contains only a single hardware divider. By using only one divider across both operations, the design avoids duplicating costly division hardware and achieves better area efficiency.

6. Experiments and Results

6.1. Experimental Setup

The proposed nonlinear function accelerator was designed at the RTL level using Verilog HDL and synthesized using Vivado 2023.2. The design was implemented on a Xilinx VU37P HBM FPGA and evaluated on a VCU128 development board operating at a clock frequency of 50 MHz. During evaluation, input vectors and parameters were preloaded into on-chip BRAMs via a UART interface. Table 3 summarizes the hardware resource utilization of the accelerator core, which includes 22,654 LUTs, 128 DSP slices, and 4345 flip-flops (FFs). Compared to prior works, the proposed core design achieved reductions of at least 30% in LUT usage and 88% in FF usage, primarily due to its unified datapath and resource-sharing structure.

Baseline software experiments were conducted on an Intel Xeon Gold 6242R CPU and an NVIDIA RTX A4000 GPU. All evaluations were performed using PyTorch 2.5.1 with CUDA 12.1. For model-level accuracy and perplexity evaluation, we used PyTorch's standard implementations of Softmax and LayerNorm under the default FP32 precision setting. These results served as the baseline for assessing the impact of the proposed approximations on downstream task performance. For execution time benchmarking, we re-implemented the approximate Softmax and LayerNorm functions—based on PLA and Newton–Raphson iteration—in PyTorch using FP32 precision on both CPU and GPU. These matched the structure and computation flow of the FPGA implementation, which

used 32-bit fixed-point arithmetic, enabling consistent and fair architectural comparisons across platforms.

Table 3. Hardware resource utilization for different Softmax/LayerNorm accelerator implementations.

Paper	Hardware Platform	LUT	FF	DSP
Cha et al. [33]	Xilinx ZCU106 FPGA	129,772	117,365	1033
Lu et al. [34]	Xilinx XCVU13P FPGA	32,741	37,948	129
Ours	Xilinx VCU128 FPGA	22,654	4345	128
Wang et al. [32]	ASIC (simulated)	N/A	N/A	N/A

N/A: Not applicable for ASIC-based designs.

6.2. Accuracy and Perplexity

The proposed accelerator was integrated into BERT and GPT-2 models to evaluate its impact on both classification and generation tasks. For classification, we used five benchmark datasets: SST-2 [38], AG News [39], IMDb [40], QQP [38], and RTE [38]. To assess the effect of the approximation methods, we conducted experiments across varying levels of approximation strength, configured by the PLA interval size h and the number of Newton–Raphson iterations N for square root computation. This allowed us to evaluate the trade-off between approximation accuracy and hardware performance, and to derive guidelines for appropriate parameter settings based on application scenarios. These datasets were selected to cover a broad range of classification tasks with varying input lengths, linguistic complexity, and label types, ensuring a representative evaluation of the approximation effects across different use cases.

For generation tasks, we used six widely adopted benchmark datasets: OpenWebText, WikiText-103 [41], and Penn Treebank [42] for next-token prediction, and PIQA [43], CBT [44], and LAMBADA [45] for comprehension-based multiple-choice evaluation. These datasets were chosen to reflect diverse generative scenarios, including both open-domain language modeling and context-aware reasoning. They vary in sequence length, domain specificity, and dependency on long-range contextual understanding, thereby allowing us to comprehensively evaluate the sensitivity of generative models to numerical approximations under different linguistic and structural demands.

Figure 8 illustrates the performance variation of BERT-based classification tasks under different approximation strengths. Experimental results show that when $h \leq 4$ or $N \geq 3$, accuracy degradation was generally limited to within 1% for most datasets, and the overall error rate remained stable. For instance, even at $h = 4$, the accuracy loss was only 0.6% for SST-2 and 0.15 for AG News. For IMDb, QQP, and RTE, the error rate was also contained within the 1–1.5% range. However, under more aggressive approximation settings such as $h = 8$ or $N = 2$, significant accuracy drops were observed. Specifically, IMDb and RTE showed sensitivity to approximation with losses of 11.38% and 15.92%, respectively. Tasks requiring fine-grained sentence-level understanding, such as IMDb and RTE, exhibited higher vulnerability to LayerNorm approximation. Similarly, QQP experienced an accuracy drop of up to 11.40% at $h = 16$. In contrast, shorter sentence classification tasks like SST-2 and AG News maintained robust performance even under higher approximation levels.

For GPT-2, we used six datasets. For next-token prediction tasks such as OpenWebText, WikiText-103 [41], and Penn Treebank [42], perplexity was used as the primary evaluation metric, as it reflects how well the model assigns high probability to the correct next token. In contrast, comprehension-based generation tasks such as PIQA [43], CBT [44], and LAMBADA [45] were evaluated using accuracy. Figure 9 presents the results of these generation tasks. Compared to classification tasks, the GPT-2 model shows much higher sensitivity to the approximation of Softmax and LayerNorm, particularly in perplexity-

based metrics. Even moderate approximation levels resulted in substantial increases in perplexity. For instance, on WikiText-103 and Penn Treebank, perplexity increased by 335.14% and 261.27%, respectively, with a PLA interval of $h = 4$, and surged up to 993.90% when $h = 8$. Similarly, reducing the Newton–Raphson iteration count to $N = 2$ led to sharp error increases—1141.13% for Penn Treebank and 1867.75% for WikiText-103—highlighting the strong sensitivity to reduced precision in language modeling tasks.

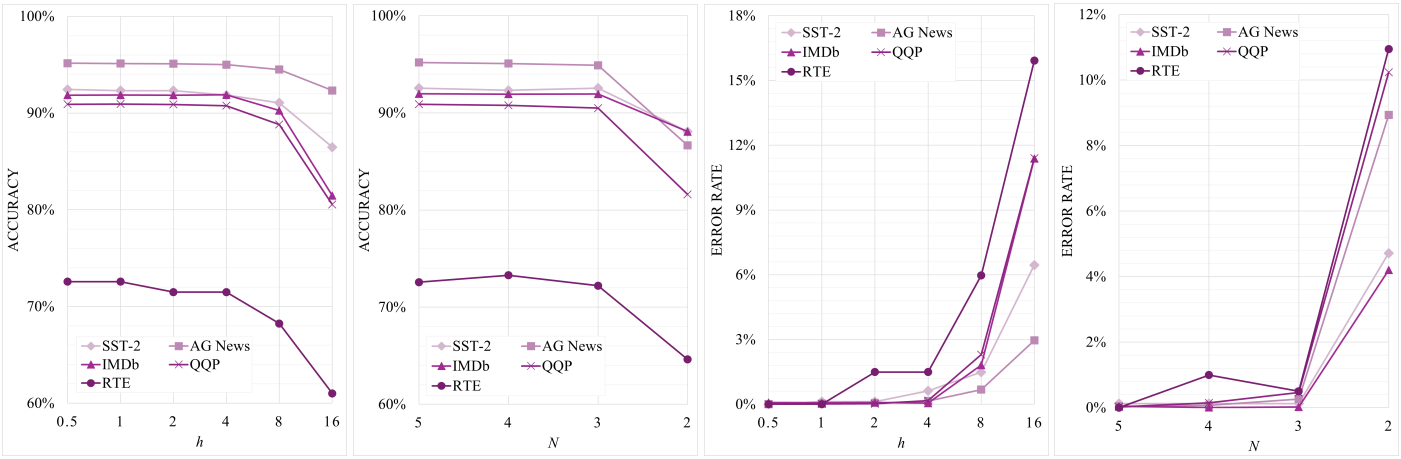


Figure 8. Impact of Softmax and LayerNorm approximation on the BERT model. (**a**,**b**) Accuracy under varying PLA interval h and Newton–Raphson iteration count N. (**c**,**d**) Corresponding error rates under different approximation levels. Evaluated on five benchmark classification datasets: SST-2, IMDb, AG News, QQP, and RTE.

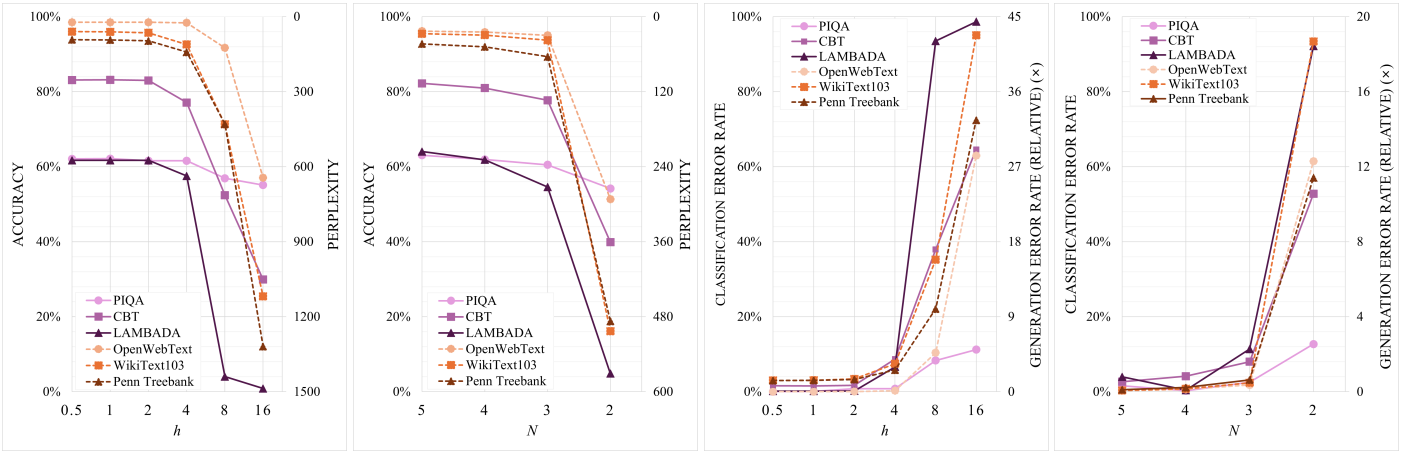


Figure 9. Impact of Softmax and LayerNorm approximation on the GPT model. (**a**,**b**) Accuracy and perplexity under varying PLA interval h and Newton–Raphson iteration count N. (**c**,**d**) Corresponding error rates, including relative degradation in perplexity and classification accuracy. Evaluated on six benchmark datasets: PIQA, CBT, LAMBADA, OpenWebText, WikiText103, and Penn Treebank.

In contrast, multiple-choice generation tasks such as PIQA, CBT, and LAMBADA maintained relatively stable accuracy under mild approximation settings. However, under more aggressive conditions ($h = 2$–3, $N = 2$–3), accuracy degradation exceeded 10–12% in several cases. CBT and LAMBADA, which require long-range context modeling, were particularly sensitive to normalization errors in Softmax, showing accuracy drops of 37.95% and 39.49%, respectively, at $h = 8$. These results underscore the stronger dependence of generation tasks on precise probability estimation compared to classification.

This heightened sensitivity stems from the autoregressive nature of language generation, where prediction errors can accumulate and propagate across multiple tokens, significantly degrading output quality. Based on these findings, moderate approximation settings such as $h = 4$ and $N = 3$ offer a favorable trade-off for classification tasks, preserving accuracy while reducing computation. In contrast, generative tasks—particularly those involving autoregressive decoding—require smaller interval sizes (e.g., $h = 0.5$) and higher iteration counts (e.g., $N \geq 5$) to avoid accuracy loss. These observations serve as practical guidelines for selecting approximation parameters according to task sensitivity and performance requirements.

6.3. Execution Time

The execution time was measured under moderate approximation settings, with a PLA interval of $h = 4$ and Newton–Raphson iteration count of $N = 3$, which offer a balanced trade-off between accuracy and performance. Table 4 summarizes the measured execution times in microseconds (μs). The FPGA implementation achieved the lowest latency in most cases, with Softmax completing in 69.10 μs for BERT and 32.75 μs for GPT-2. For LayerNorm, the FPGA achieved 109.45 μs and 66.70 μs, respectively. Compared to CPU and GPU baselines, this represents a substantial reduction in latency, especially for Softmax on long sequences.

Table 4. Measured execution time (μs) for approximate Softmax and LayerNorm operations.

Platform	BERT		GPT-2	
	Softmax	**LayerNorm**	**Softmax**	**LayerNorm**
CPU	272.30	154.10	249.05	130.30
GPU	160.75	117.35	150.85	103.15
Ours (FPGA)	**69.10**	**109.45**	**32.75**	**66.70**

The normalized speedup relative to the CPU baseline is shown in Table 5. The proposed FPGA accelerator achieved up to 7.6× and 2.0× speedups for Softmax and LayerNorm, respectively. It also outperformed GPU performance by a factor of up to 4.6× in Softmax and 1.5× in LayerNorm.

Table 5. Relative runtime speedup compared to CPU baseline (CPU = 1.000×).

Platform	BERT		GPT-2	
	Softmax	**LayerNorm**	**Softmax**	**LayerNorm**
CPU	1.000×	1.000×	1.000×	1.000×
GPU	1.694×	1.313×	1.651×	1.263×
Ours (FPGA)	**3.940**×	**1.408**×	**7.605**×	**1.954**×

The speedup achieved for Softmax was consistently higher than that of LayerNorm across all test cases. This difference is primarily due to the relative simplicity of the approximation used for the exponential function in Softmax. The piecewise linear segments were precomputed and stored in advance, allowing each exponential computation to be performed immediately with a simple arithmetic operation. In contrast, LayerNorm involves a more complex sequence of computations, including the calculation of mean and variance followed by square root estimation using Newton–Raphson iteration. These operations introduce additional latency, especially since the iterative square root approximation requires multiple cycles to converge. As a result, the overall speedup from approximation is more significant for Softmax than for LayerNorm.

7. Conclusions

In this paper, we presented a unified hardware accelerator targeting two major non-linear bottlenecks in Transformer models: Softmax and LayerNorm. To reduce the computational cost associated with exponential and square root operations, the accelerator employs PLA and Newton–Raphson iteration, respectively—methods that enable efficient fixed-point implementation. By reusing shared datapaths and dynamically switching operation modes, the proposed architecture maximizes resource utilization while minimizing area overhead. Furthermore, the design introduces tunable approximation parameters—interval width h and iteration count N—to balance the trade-offs between accuracy, latency, and resource usage. The accelerator was implemented on a Xilinx VU37P HBM FPGA and evaluated with BERT-base and GPT-2 models. Under conservative settings ($h \leq 4$, $N \geq 3$), it achieved up to 7.6× and 2.0× speedup for Softmax and LayerNorm, respectively, while maintaining classification accuracy within 1% of the baseline. Although generation tasks exhibited greater sensitivity—particularly in Softmax—the use of tighter approximation parameters effectively mitigated performance degradation, as demonstrated in our experimental analysis. These findings highlight the practicality of jointly accelerating non-linear functions for Transformer inference using a unified, fixed-point-based design. The current implementation uses 32 PEs, which is sufficient for models with 768-dimensional embeddings such as BERT-base and GPT-2. However, recent models adopt significantly larger dimensions (e.g., 4096 or more), which would increase the number of processing cycles proportionally unless the PE array is scaled accordingly. In addition, fixed-point arithmetic reduces hardware resource usage but restricts the representable dynamic range. This limitation may affect numerical stability in cases where input distributions have large variance or require high precision. Exploring alternative formats such as FP8 or FP16 could offer a better balance between precision and resource efficiency. In addition, the current design operates at a relatively low clock frequency due to non-pipelined datapaths involving shared arithmetic resources. Future revisions may achieve better timing performance through architectural and implementation-level optimizations.

Author Contributions: Conceptualization, R.K. and S.E.L.; methodology, R.K. and D.L.; software, R.K. and J.K.; validation, R.K. and D.L.; formal analysis, R.K. and J.P.; investigation, D.L. and J.P.; resources, D.L. and S.E.L.; data curation, J.K.; writing—original draft preparation, R.K.; writing—review and editing, R.K., D.L., J.K, J.P. and S.E.L.; visualization, R.K. and D.L.; supervision, S.E.L. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by SeoulTech (Seoul National University of Science and Technology).

Data Availability Statement: All datasets used in this study are publicly available via the Hugging Face Datasets library. The following datasets were used: GLUE-QQP (glue, qqp), GLUE-RTE (glue, rte), GLUE-SST2 (glue, sst2), AG News (ag_news), IMDb (imdb), Penn Treebank (ptb_text_only), PIQA (piqa), WikiText-103 (wikitext, wikitext-103-raw-v1), CBT-CN (cbt, CN), LAMBADA (lambada), and OpenWebText (openwebtext). These datasets can be accessed through https://huggingface.co/datasets (accessed on 6 June 2025).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
BatchNorm	Batch Normalization
BN	Batch Normalization
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network

CPU	Central Processing Unit
DSP	Digital Signal Processor (or Digital Signal Processing slice)
FF	Flip-Flop
FFN	Feed-Forward Network
FPGA	Field-Programmable Gate Array
FP32	32-bit Floating Point
GPT	Generative Pre-trained Transformers
GPU	Graphics Processing Unit
GELU	Gaussian Error Linear Unit
INT8	8-bit Integer
LayerNorm	Layer Normalization
LUT	Look-Up Table
LN	Layer Normalization
LAMBADA	LAnguage Modeling Benchmark with Analysis of Discourse Across Discourse
LUT	Look-Up Table
MHA	Multi-Head Attention
NLP	Natural Language Processing
PE	Processing Element
PLA	Piecewise Linear Approximation
Q	Query
K	Key
V	Value
RNN	Recurrent Neural Network
RTE	Recognizing Textual Entailment
SDPA	Scaled Dot-Product Attention
SM	Softmax
SST-2	Stanford Sentiment Treebank 2
TEA-S	Transformer Exponential Approximation-Softmax
VU37P	Xilinx Virtex UltraScale+ VU37P FPGA
"intN"	N-bit Integer (e.g., int8, int16, etc.)

References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Long Beach, CA , USA, 4–9 December 2017; pp. 6000–6010.
2. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2019**, arXiv:1810.04805. [CrossRef]
3. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
4. Khan, A.; Rauf, Z.; Sohail, A.; Khan, A.R.; Asif, H.; Asif, A.; Farooq, U. A survey of the vision transformers and their CNN-transformer based variants. *Artif. Intell. Rev.* **2023**, *56*, 2917–2970. [CrossRef]
5. Li, Y.; Liu, M.; Wu, Y.; Wang, X.; Yang, X.; Li, S. Learning adaptive and view-invariant vision transformer for real-time UAV tracking. In Proceedings of the 41st International Conference on Machine Learning, ICML'24, Vienna, Austria, 21–27 July 2024.
6. Anwar, A.; Khalifa, Y.; Coyle, J.L.; Sejdic, E. Transformers in biosignal analysis: A review. *Inf. Fusion* **2025**, *114*, 102697. [CrossRef]
7. Vafaei, E.; Hosseini, M. Transformers in EEG analysis: A review of architectures and applications in motor imagery, seizure, and emotion classification. *Sensors* **2025**, *25*, 1293. [CrossRef]
8. Kachris, C. A survey on hardware accelerators for large language models. *Appl. Sci.* **2025**, *15*, 586. [CrossRef]
9. Du, J.; Jiang, J.; Zheng, J.; Zhang, H.; Huang, D.; Lu, Y. Improving computation and memory efficiency for real-world transformer inference on GPUs. *ACM Trans. Archit. Code Optim.* **2023**, *20*, 46. [CrossRef]
10. Chen, Q. Research on inference and training acceleration of large language model. In Proceedings of the 2024 7th International Conference on Computer Information Science and Artificial Intelligence, CISAI '24, Shaoxing, China, 13–15 September 2024; pp. 303–307. [CrossRef]
11. Wu, J.; Song, M.; Zhao, J.; Gao, Y.; Li, J.; So, H.K.H. TATAA: Programmable mixed-precision transformer acceleration with a transformable arithmetic architecture. *ACM Trans. Reconfig. Technol. Syst.* **2025**, *18*, 14. [CrossRef]

12. Park, J.; Shin, J.; Kim, R.; An, S.; Lee, S.; Kim, J.; Oh, J.; Jeong, Y.; Kim, S.; Jeong, Y.R.; et al. Accelerating strawberry ripeness classification using a convolution-based feature extractor along with an edge AI processor. *Electronics* **2024**, *13*, 344. [CrossRef]
13. Jeong, Y.; Park, J.; Kim, R.; Lee, S.E. SEAM: A synergetic energy-efficient approximate multiplier for application demanding substantial computational resources. *Integration* **2025**, *101*, 102337. [CrossRef]
14. Wang, J.; Zhang, L.; Li, X.; Yang, H.; Liu, Y. ULSeq-TA: Ultra-long sequence attention fusion transformer accelerator supporting grouped sparse softmax and dual-path sparse layernorm. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* **2024**, *43*, 892–905. [CrossRef]
15. Sadeghi, M.E.; Fayyazi, A.; Azizi, S.; Pedram, M. PEANO-ViT: Power-efficient approximations of non-linearities in vision transformers. In Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED '24, 5–7 August, Newport Beach, CA, USA, 2024; pp. 1–6. [CrossRef]
16. Bao, Z.; Li, H.; Zhang, W. A vision transformer inference accelerator for KR260. In Proceedings of the 2024 8th International Conference on Computer Science and Artificial Intelligence, CSAI '24, Beijing China, 6–8 December 2024; pp. 245–251. [CrossRef]
17. Dong, Q.; Xie, X.; Wang, Z. SWAT: An efficient swin transformer accelerator based on FPGA. In Proceedings of the 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC), Incheon, Republic of Korea, 22–25 January 2024; pp. 515–520. [CrossRef]
18. Amin, H.; Curtis, K.; Hayes-Gill, B. Piecewise linear approximation applied to nonlinear function of a neural network. *IEE Proc.-Circuits Devices Syst.* **1997**, *144*, 313–317. [CrossRef]
19. Geng, X.; Lin, J.; Zhao, B.; Kong, A.; Aly, M.M.S.; Chandrasekhar, V. Hardware-aware softmax approximation for deep neural networks. In *Computer Vision—ACCV 2018, Proceedings of the 14th Asian Conference on Computer Vision, Perth, Australia, 6–8 December 2018*; Jawahar, C., Li, H., Mori, G., Schindler, K., Eds.; Springer: Cham, Switzerland, 2019; pp. 107–122.
20. Soydaner, D. Attention mechanism in neural networks: Where it comes and where it goes. *Neural Comput. Appl.* **2022**, *34*, 13371–13385. [CrossRef]
21. Schuster, M.; Paliwal, K. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* **1997**, *45*, 2673–2681. [CrossRef]
22. Xiong, R.; Yang, Y.; He, D.; Zheng, K.; Zheng, S.; Xing, C.; Zhang, H.; Lan, Y.; Wang, L.; Liu, T.Y. On layer normalization in the transformer architecture. *arXiv* **2020**, arXiv:2002.04745. [CrossRef]
23. Tay, Y.; Dehghani, M.; Bahri, D.; Metzler, D. Efficient transformers: A survey. *ACM Comput. Surv.* **2022**, *55*, 109. [CrossRef]
24. Pearce, T.; Brintrup, A.; Zhu, J. Understanding softmax confidence and uncertainty. *arXiv* **2021**, arXiv:2106.04972. [CrossRef]
25. Holm, A.N.; Wright, D.; Augenstein, I. Revisiting softmax for uncertainty approximation in text classification. *Information* **2023**, *14*, 420. [CrossRef]
26. Ba, J.L.; Kiros, J.R.; Hinton, G.E. Layer normalization. *arXiv* **2016**, arXiv:1607.06450. [CrossRef]
27. Guo, J.; Chen, X.; Tang, Y.; Wang, Y. SLAB: Efficient transformers with simplified linear attention and progressive re-parameterized batch normalization. In Proceedings of the 41st International Conference on Machine Learning, ICML'24, Vienna, Austria, 21–27 July 2024.
28. Stevens, J.R.; Venkatesan, R.; Dai, S.; Khailany, B.; Raghunathan, A. Softermax: Hardware/software co-design of an efficient softmax for transformers. In Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 5–9 December 2021; pp. 469–474. [CrossRef]
29. Zhu, D.; Lu, S.; Wang, M.; Lin, J.; Wang, Z. Efficient precision-adjustable architecture for softmax function in deep learning. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 3382–3386. [CrossRef]
30. Mei, Z.; Dong, H.; Wang, Y.; Pan, H. TEA-S: A tiny and efficient architecture for PLAC-based softmax in transformers. *IEEE Trans. Circuits Syst. II Express Briefs* **2023**, *70*, 3594–3598. [CrossRef]
31. Gao, Y.; Liu, W.; Lombardi, F. Design and implementation of an approximate softmax layer for deep neural networks. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 12–14 October 2020; pp. 1–5. [CrossRef]
32. Wang, W.; Zhou, S.; Sun, W.; Sun, P.; Liu, Y. SOLE: Hardware-software co-design of softmax and layernorm for efficient transformer inference. In Proceedings of the 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), San Francisco, CA, USA, 28 October–2 November 2023; pp. 1–9. [CrossRef]
33. Cha, M.; Lee, K.; Nguyen, X.T.; Lee, H.J. A low-latency and scalable vector engine with operation fusion for transformers. In Proceedings of the 2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS), Abu Dhabi, United Arab Emirates, 22–25 April 2024; pp. 307–311. [CrossRef]
34. Lu, S.; Wang, M.; Liang, S.; Lin, J.; Wang, Z. Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. In Proceedings of the 2020 IEEE 33rd International System-on-Chip Conference (SOCC), Las Vegas, NV, USA, 8–11 September 2020; pp. 84–89. [CrossRef]
35. Thorp, J.; Lewine, R. Exponential approximation with piecewise linear error criteria. *J. Frankl. Inst.* **1968**, *286*, 308–320. [CrossRef]
36. Korzilius, S.; Schoenmakers, B. Divisions and square roots with tight error analysis from newton–raphson iteration in secure fixed-point arithmetic. *Cryptography* **2023**, *7*, 43. [CrossRef]
37. Galántai, A. The theory of Newton's method. *J. Comput. Appl. Math.* **2000**, *124*, 25–44. [CrossRef]

38. Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; Bowman, S. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*; Linzen, T., Chrupała, G., Alishahi, A., Eds.; Association for Computational Linguistics: Brussels, Belgium, 2018; pp. 353–355. [CrossRef]
39. Zhang, X.; Zhao, J.; LeCun, Y. Character-level convolutional networks for text classification. In Proceedings of the 29th International Conference on Neural Information Processing Systems, NIPS'15, Montreal, QC, Canada, 7–12 December 2015; Volume 1, pp. 649–657.
40. Årup Nielsen, F. A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. *arXiv* **2011**, arXiv:1103.2903. [CrossRef]
41. Merity, S.; Xiong, C.; Bradbury, J.; Socher, R. Pointer sentinel mixture models. *arXiv* **2016**, arXiv:1609.07843. [CrossRef]
42. Marcus, M.P.; Santorini, B.; Marcinkiewicz, M.A. Building a large annotated corpus of English: The Penn Treebank. *Comput. Linguist.* **1993**, *19*, 313–330.
43. Bisk, Y.; Zellers, R.; Bras, R.L.; Gao, J.; Choi, Y. PIQA: Reasoning about physical commonsense in natural language. *arXiv* **2019**, arXiv:1911.11641. [CrossRef]
44. Hill, F.; Bordes, A.; Chopra, S.; Weston, J. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv* **2016**, arXiv:1511.02301. [CrossRef]
45. Paperno, D.; Kruszewski, G.; Lazaridou, A.; Pham, N.Q.; Bernardi, R.; Pezzelle, S.; Baroni, M.; Boleda, G.; Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Erk, K., Smith, N.A., Eds.; Association for Computational Linguistics: Berlin, Germany, 2016; pp. 1525–1534. [CrossRef]