A Hardware-efficient Training-oriented Quadratic Approximation Softmax for Transformers

Zehao Cheng, Weichao Yi, Hui Chen, Chenggang Yan, Bi Wu, Ke Chen* and Weiqiang Liu
Nanjing University of Aeronautics and Astronautics, Nanjing, China
chen.ke@nuaa.edu.cn

***Abstract*—The softmax function constitutes a major computational bottleneck in Transformer architectures, especially during deep neural network training, where stringent numerical precision is required. This paper presents a hardware-efficient, training-oriented base-2 softmax architecture based on a novel quadratic approximation scheme, referred to as TQA-Softmax. Leveraging the Log-Sum-Exp formulation, coefficient quantization, and hardware-friendly approximate multipliers, the proposed design achieves high numerical accuracy, with a mean absolute error on the order of 10^{-7}. Compared with state-of-the-art softmax architectures, the proposed implementation reduces hardware area by up to 74.8% and improves hardware efficiency by 3.57×. Extensive experiments on ViT and BERT models demonstrate stable convergence and negligible performance degradation, validating the proposed design as an effective and scalable solution for high-performance Transformer training.**

***Index Terms*—Softmax, piecewise polynomial approximation, high precision, Transformer training.**

I. Introduction

In recent years, the Transformer architecture has emerged as the dominant standard for a wide spectrum of deep learning tasks, ranging from Natural Language Processing (NLP) to Computer Vision (CV) [1]–[3]. Central to this success is the self-attention mechanism, as illustrated in Fig. 1, which relies heavily on the softmax function to normalize attention scores into a probability distribution, allowing the network to capture dependencies within input sequences [1], [4], [5].

From a hardware design perspective, softmax is particularly challenging because it combines numerically sensitive nonlinear functions, such as exponentiation and normalization, with a global reduction and a division-like operation. These operators are expensive to implement at high throughput and are difficult to pipeline efficiently, since they require wide dynamic range and careful overflow/underflow handling. Furthermore, in multi-head self-attention, softmax is invoked repeatedly across layers, heads, and token positions, making its latency and energy consumption directly visible at the system level. As a result, a small improvement in the softmax datapath can yield a disproportionate gain in overall accelerator efficiency.

However, the rapid evolution from BERT and ViT models to Large Language Models (LLMs) has pushed model complexity to extreme levels, with parameter counts frequently exceeding hundreds of billions, thereby driving a surging demand for computing power [6]–[9]. As model scales and sequence lengths increase, softmax operation in self-attention emerges as a primary computational bottleneck, which increases the runtime significantly and severely impacts development efficiency in long-sequence tasks [10].

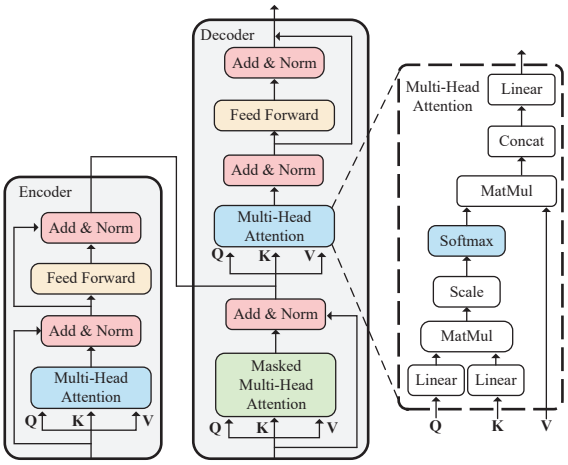

Fig. 1: Softmax in Transformer

In response to this bottleneck, the Log-Sum-Exp (LSE) technique was introduced in [11] and has been adopted as a foundational strategy. Combined with a down-scaling approach to mitigate overflow, the LSE formulation transforms the division into a subtraction in the logarithmic domain, which is more hardware-friendly. Nevertheless, most prior works primarily target neural network inference, where softmax precision requirements are relatively low. In contrast, as highlighted by [12], DNN training demands higher softmax precision, and the calculation error should be kept on the order of 10^{-6}. This requirement stems from the fact that, during training, softmax outputs directly affect both the forward attention distribution and the backward gradients. Inaccurate softmax evaluation can distort probability normalization, amplify quantization noise in gradient signals, and ultimately degrade convergence stability, especially for long sequences and large models. Moreover, modern Transformer training is constrained by tight latency and energy budgets. The attention module is executed repeat-

This work was supported by the National Nature Science Foundation of China(62425404, 62404102, and 62522406), Basic Research Program of Jiangsu(BK20253023), Jiangsu Province Major Scientific Project(BG2025012), and the High Performance Computing Platform of Nanjing University of Aeronautics and Astronautics.

edly across layers and heads, and its softmax computation is invoked for every token position. Therefore, even modest per-call latency or area overhead in softmax can translate into a substantial system-level cost. These observations motivate a training-oriented softmax design that simultaneously achieves (i) high numerical accuracy comparable to floating-point baselines and (ii) low hardware complexity and latency suitable for high-throughput accelerators.

Building on LSE-based formulations, extensive research has investigated LUT-based designs, piecewise linear approximations (PWL), and iterative methods (e.g., CORDIC) to accelerate the core operators in softmax [13]–[26]. Another widely used approach converts base-e exponentials into base-2 for implementation convenience ($e^x = 2^{x \cdot \log_2 e}$) [15]–[18], [22], [24], [25]; however, this conversion introduces extra hardware overhead. To avoid the conversion cost, base-2 softmax has been proposed and validated in DNNs [12], [14], [27].

Despite these advances, conventional approaches still face major challenges in training-oriented, high-precision scenarios: LUT/PWL methods require either prohibitive memory or overly complex segmentation logic to maintain accuracy, while CORDIC-based methods often incur long iterative latency. For example, the training-oriented design in [12] requires 26 clock cycles, with 21 cycles spent on CORDIC iterations. To address these limitations, this paper proposes TQA-Softmax, a high-precision, low-latency base-2 softmax architecture. The major contributions of this paper can be summarized as follows:

1) **High-Precision Softmax Architecture:** We propose a high-precision hardware design for softmax. Using the LSE technique, the softmax operation is reduced to accurate evaluations of 2^x for $x \in [0, 1)$ and $\log_2(x)$ for $x \in [1, 2)$. The core functions are efficiently computed using the TQA-Softmax scheme to ensure high numerical accuracy.

2) **Hardware-Oriented Algorithmic Optimization:** We introduce a hardware-aware coefficient quantization method to compensate for the quantization error caused by coefficient truncation. Additionally, we optimize the multiplication operations in the computation process, making them well-suited for our proposed hardware design.

3) **Synthesis and Application Evaluation:** The proposed design is synthesized using TSMC 28nm CMOS technology and compared with existing works. Network-level training experiments on ViT and BERT models verify its performance in Transformer training scenarios.

II. PRELIMINARIES

The softmax function plays an important role in the attention mechanism of Transformer networks by normalizing input logits into a probability distribution. The traditional softmax function for an input vector $X = [x_1, x_2, \ldots, x_N]$ is defined as (1). In hardware implementations, the natural exponential is often converted to base-2 form, which inevitably introduces additional overhead. To reduce complexity, we adopt the base-2 softmax, denoted as $f_2(x)$, as an efficient alternative, as formulated in (2).

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{1}$$

$$Base\text{-}2\ Softmax(x_i) = \frac{2^{x_i}}{\sum_{j=1}^{N} 2^{x_j}} \tag{2}$$

Direct implementation of (2) faces two primary challenges: (i) numerical overflow caused by exponentiation on large inputs, and (ii) the high hardware cost of division. To avoid overflow, we apply the down-scaling strategy by subtracting the maximum value x_{max} from all elements in X. After shifting, all exponential terms fall within $(0, 1]$, and the computation becomes:

$$f_2(x_i) = \frac{2^{x_i - x_{max}}}{\sum_{j=1}^{N} 2^{x_j - x_{max}}} \tag{3}$$

To mitigate the division overhead, we employ the Log-Sum-Exp (LSE) technique, which transforms the division into a subtraction in the logarithmic domain. Combining the base-2 formulation with down-scaling and LSE, we obtain:

$$f_2(x_i) = 2^{x_i - x_{max} - \log_2(\sum_{j=1}^{N} 2^{x_j - x_{max}})} \tag{4}$$

Equation (4) shows that softmax can be computed using two exponentiations and one logarithm operation.

The inputs to the exponentiation operator, denoted as x_{exp}, are $x_i - x_{max}$ and $x_i - x_{max} - \log_2(\sum_{j=1}^{N} 2^{x_j - x_{max}})$, both of which lie in $(-\infty, 0]$. We decompose x_{exp} into an integer part a and a fractional part b:

$$x_{exp} = a + b \tag{5}$$

where a is a negative integer and $b \in [0, 1)$. Then, the base-2 exponentiation can be expressed as:

$$2^{x_{exp}} = 2^a \cdot 2^b = 2^b \gg (-a) \tag{6}$$

(6) simplifies the computation logic: 2^a corresponds to a simple right-shift operation in hardware, leaving only the fractional term 2^b to be computed.

The input to the logarithmic operator is $\sum_{j=1}^{N} 2^{x_j - x_{max}}$, which lies in $(1, N]$. We utilize a Leading One Detector (LOD) to narrow the calculation range. $\sum_{j=1}^{N} 2^{x_j - x_{max}}$ is decomposed into k and m:

$$\log_2(\sum_{j=1}^{N} 2^{x_j - x_{max}}) = \log_2(m \cdot 2^k) = k + \log_2(m) \tag{7}$$

$$m = \sum_{j=1}^{N} 2^{x_j - x_{max}} \gg k \tag{8}$$

where k is an integer and $m \in [1, 2)$. k is directly obtained from the LOD logic and m can be obtained through a shift operation.

As derived above, the accuracy of the base-2 softmax ultimately depends on the precise evaluation of 2^x over $x \in [0, 1)$ and $\log_2(x)$ over $x \in [1, 2)$. To achieve high-precision computation while maintaining hardware efficiency, we build

on the PQA-FGS method proposed in [28] and develop an enhanced training-oriented quadratic approximation method, namely TQA-Softmax. While generalized arithmetic frameworks like FloPoCo [29] provide hardware-level polynomial approximations for elementary functions, meeting the strict high-precision requirements of Softmax requires specialized algorithmic-hardware co-design.

III. PROPOSED ALGORITHM FOR HARDWARE-ORIENTED BASE-2 SOFTMAX

In this section, we present the proposed hardware-oriented algorithm for base-2 softmax. Building on down-scaling and the Log-Sum-Exp (LSE) technique, the base-2 softmax computation is reduced to exponential and logarithmic evaluations over compact domains. We then apply the proposed TQA-Softmax method to accurately approximate these nonlinear operators. To further reduce hardware overhead, we introduce coefficient quantization and a hardware-friendly multiplier optimization tailored to TQA-Softmax.

A. TQA-Softmax Approximation for 2^x and $\log_2 x$

Prior work [28] introduced a universal piecewise quadratic approximation framework applicable to a wide class of nonlinear functions, including exponentiation and logarithm. In this paper, we develop TQA-Softmax, which explicitly targets the stringent numerical accuracy and hardware efficiency requirements of Transformer training. TQA-Softmax systematically incorporates coefficient quantization and bit-width truncation into the approximation model, enabling precise control of numerical error under fixed hardware constraints.

Specifically, for a given input value x and a nonlinear function $y(x)$, the input is decomposed into three segments, denoted as M_0, M_1, and M_2, with corresponding bit widths N_0, N_1, and N_2, respectively. For example, the mantissa of a single-precision floating-point number can be expressed as:

$$\begin{aligned}
&x = m_0 m_1 m_2 ... m_{21} m_{22} \\
&N_0 + N_1 + N_2 = 23 \\
&M_0 = m_0 m_1 ... m_{N_0-1} \times 2^{-N_0} < 1 \\
&M_1 = m_{N_0} m_{N_0+1} ... m_{N_0+N_1-1} \times 2^{-(N_0+N_1)} < 2^{-N_0} \\
&M_2 = m_{N_0+N_1} ... m_{N_0+N_1+N_2-1} \times 2^{-N_0-N_1-N_2} < 2^{-N_0-N_1} \\
&x = M_0 + M_1 + M_2 = \{m_0 m_1 ..., ..., ... m_{N_0+N_1+N_2-1}\}
\end{aligned} \tag{9}$$

where $m_0, m_1 ... m_{22}$ are the mantissa bits.

Within each segment, the nonlinear function is approximated using a second-order polynomial:

$$\begin{aligned}
y(x) \approx w_{i,2} M_1^2 + w_{i,1}(M_1 + M_2) + w_{i,0} \\
x \in [x_i, x_{i+1})
\end{aligned} \tag{10}$$

where $w_{i,2}$, $w_{i,1}$, and $w_{i,0}$ denote the quadratic, linear, and constant coefficients in the i-th segment, respectively. In this formulation, M_0 serves as the index to select the coefficient set, while the remaining bits are used for arithmetic evaluation. The input domain is thus partitioned into 2^{N_0} segments.

When directly computing the quadratic term $(x - M_0)^2$, the resulting bit width becomes excessively large, which necessitates aggressive truncation and leads to inefficient hardware utilization. To mitigate this issue, the remaining bits of x are further decomposed into M_1 and M_2, where M_1 is used to compute the quadratic term and $M_1 + M_2$ is used for the linear term. This decomposition effectively limits bit growth while preserving approximation accuracy.

Considering coefficient quantization and bit-width truncation in hardware implementation, [29] performed an error analysis for the approximation process. In this work, we further extend the analysis to determine the optimal bit-width configuration that truncates unnecessary fractional bits under training-level precision requirements.

First, we analyze the truncation of $M_1 + M_2$. The total approximation error consists of model error (e_m), coefficient quantization error (e_c), and truncation error (e_t), where the model error (e_m) represents the inherent mathematical difference between the exact transcendental function and our ideal unquantized quadratic polynomial. When the number of segments is sufficiently large, the model error becomes negligible. Incorporating e_c and e_t, the approximation can be expressed as:

$$\begin{aligned}
y(x) \approx\ & (w_{i,2} + e_{c,2})(M_1^2 + e_{t1}) \\
& + (w_{i,1} + e_{c,1})(M_1 + M_2 + e_{t2}) \\
& + (w_{i,0} + e_{c,0}), \quad x \in [M_i, M_{i+1}) \\
e_{t1} = 0.5(2M_1M_2 + M_2^2) &\approx M_1 M_2
\end{aligned} \tag{11}$$

Subtracting (10) from (11), the resulting approximation error can be expressed as:

$$\begin{aligned}
Error \approx\ & w_{i,2} M_1 M_2 + w_{i,1} e_{t2} \\
& + M_1^2 e_{c,2} + (M_1 + M_2) e_{c,1} + e_{c,0}
\end{aligned} \tag{12}$$

Based on (9), an upper bound of the approximation error is derived as:

$$\begin{aligned}
Error <\ & |w_{i,2}| \times 2^{-(2N_0+N_1)} + |w_{i,1}| \times 2^{-(N_0+N_{12})} \\
& + 2^{-(2N_0+Nw_2)} + 2^{-(N_0+Nw_1)} + 2^{-(Nw_0)} \\
\le\ & 2^{-N_{ulp}}
\end{aligned} \tag{13}$$

where N_{w2}, N_{w1}, N_{w0}, N_{12} denote the bit width for $w_{i,2}$, $w_{i,1}$, $w_{i,0}$, $M_1 + M_2$ and ulp represent the unit of the last place. We evenly allocate $2^{-N_{ulp}}$ among these five terms:

$$\begin{aligned}
|w_{i,2}| \times 2^{-(2N_0+N_1)} < 2^{-(2N_0+N_1)} &\le \frac{1}{5} \times 2^{-N_{ulp}} \\
|w_{i,1}| \times 2^{-(N_0+N_{12})} &\le \frac{1}{5} \times 2^{-N_{ulp}} \\
2^{-(2N_0+Nw_2)} &\le \frac{1}{5} \times 2^{-N_{ulp}} \\
2^{-(N_0+Nw_1)} &\le \frac{1}{5} \times 2^{-N_{ulp}} \\
2^{-(Nw_0)} &\le \frac{1}{5} \times 2^{-N_{ulp}}
\end{aligned} \tag{14}$$

where $|w_{i,2}| < 1$ has been verified in software simulation. In (15), $w_{i,1}$ serves only to affect the value of N_{12}. Although the

five error components cannot be strictly equally distributed within $2^{-N_{ulp}}$ theoretically, we can marginally adjust N_1, N_{12}, N_{w2}, N_{w1}, and N_{w0} around their theoretical bounds from (15) during software simulation to achieve the minimum approximation error.

By (14), N_1, N_{12}, N_{w2}, N_{w1}, and N_{w0} can be effectively determined:

$$\begin{aligned} N_1 &\geq N_{ulp} + \log_2(5) - 2N_0 \\ N_{12} &\geq N_{ulp} + \log_2(5) + \log_2(w_{i,1}) - N_0 \\ N_{w2} &\geq N_{ulp} + \log_2(5) - 2N_0 \\ N_{w1} &\geq N_{ulp} + \log_2(5) - N_0 \\ N_{w0} &\geq N_{ulp} + \log_2(5) \end{aligned} \tag{15}$$

Through N_{ulp} and N_0, we can get N_1, N_{12}, N_{w2}, N_{w1}, and N_{w0}, where the values of N_{ulp} and N_0 depend on our accuracy requirement.

When computing $A \times B$, least significant bits (LSBs) of B can be discarded if the product of A and the discarded bits is negligible to target accuracy. Let e_{a1} and N_{a1} denote the new error and bit-width of M_1^2. Then:

$$\begin{aligned} e_{a1} &= w_{i,2} \times 2^{-N_{a1}} < 2^{-N_{ulp}} \\ N_{a1} &> N_{ulp} + \log_2(\max(w_{i,2})) \end{aligned} \tag{16}$$

When performing the addition operations in (10), it is necessary to truncate LSBs prior to summation. The error analysis is presented as follows:

$$\begin{aligned} y(x) &\approx (w_{i,2}M_1^2 + e_{a2}) \\ &\quad + [w_{i,1}(M_1 + M_2) + e_{a3}] + (w_{i,0} + e_{a4}) \\ e_{a2} &< 2^{-N_{a2}} \leq \frac{1}{3} \times 2^{-N_{ulp}} \\ e_{a3} &< 2^{-N_{a3}} \leq \frac{1}{3} \times 2^{-N_{ulp}} \\ e_{a4} &< 2^{-N_{w0}} \leq \frac{1}{3} \times 2^{-N_{ulp}} \\ N_{a2} &\geq \log_2(3) + N_{ulp} \\ N_{a3} &\geq \log_2(3) + N_{ulp} \\ N_{w0} &\geq \log_2(3) + N_{ulp} \end{aligned} \tag{17}$$

where e_{a2}, e_{a3}, e_{a4} denote the truncation error, and N_{a2}, N_{a3} denote the bit widths of $w_{i,2}M_1^2$ and $w_{i,1}(M_1 + M_2)$ after truncation. And N_{a2}, N_{a3} are also marginally adjusted during software simulation to minimize the approximation error.

Thus, through TQA-Softmax, we can perform error analysis and determine the bit widths for each variable.

B. Hardware-Oriented Algorithmic Optimization

Although the proposed TQA-Softmax framework achieves high numerical accuracy through quadratic approximation, directly implementing the formulation in hardware would still incur non-negligible overhead due to coefficient precision and multiplication complexity. To further reduce hardware cost while preserving training-level accuracy, we introduce a set of hardware-oriented algorithmic optimizations, including coefficient quantization and truncated multiplication, which are specifically tailored to the characteristics of the TQA-Softmax computation.

To obtain the polynomial coefficients $w_{i,2}, w_{i,1}$, and $w_{i,0}$ for each segment, the Least Squares Method (LSM) is employed.

The input domain is first discretized into 2^{N_0} sub-intervals according to the high-order bits M_0. Within each segment, the variable $M_1 + M_2$ represents the fine-grained fractional component. Accordingly, the matrix D is constructed with columns $[(M_1 + M_2)^2, M_1 + M_2, 1]$ and the coefficients $w_{i,2}, w_{i,1}$, and $w_{i,0}$ are obtained by $[w_{i,2}, w_{i,1}, w_{i,0}] = D \setminus Y_{acc}$, where Y_{acc} contains the accurate values of 2^x or $\log_2(x)$.

Based on the bit widths N_{w2}, N_{w1}, and N_{w0} determined by TQA-Softmax, we quantize the coefficients $w_{i,2}$, $w_{i,1}$, and $w_{i,0}$. For a given coefficient (e.g., $w_{i,2}$) with a fractional bit-width of N_{w2}, two quantization candidates, namely $floor$ and $ceil$, are considered. These candidates are mathematically defined as:

$$\begin{aligned} w_{i,2\_}floor &= \frac{floor(w_{i,2} \times 2^{N_{w2}})}{2^{N_{w2}}} \\ w_{i,2\_}ceil &= \frac{ceil(w_{i,2} \times 2^{N_{w2}})}{2^{N_{w2}}} \end{aligned} \tag{18}$$

In hardware implementation, the $floor$ operation corresponds to directly truncating the coefficient to N_{w2} fractional bits, whereas the $ceil$ operation corresponds to truncation followed by adding one to the least significant bit (LSB) of the truncated coefficient.

For the coefficients $w_{i,2}$, $w_{i,1}$, and $w_{i,0}$, each coefficient can adopt either $floor$ or $ceil$ quantization, resulting in a total of eight possible combinations. The combination that yields the smallest approximation error is selected through software simulation. In this manner, the most suitable quantized coefficients are obtained while maintaining fixed bit widths and avoiding excessive deviation from the original values.

Following coefficient quantization, the multiplier design is further optimized. When computing $A \times B$, the LSBs of the product are typically truncated for subsequent calculations. However, these discarded partial products are still generated by the hardware, leading to unnecessary hardware overhead.

To mitigate this issue, a truncated multiplier design is proposed. Specifically, the multiplier input operands A and B are partitioned into their MSB and LSB components:

$$A = A_{MSBs} + A_{LSBs} \qquad B = B_{MSBs} + B_{LSBs} \tag{19}$$

Based on (19), the full multiplication can be expanded as:

$$\begin{aligned} A \times B = &A_{MSBs} \times B_{MSBs} + A_{MSBs} \times B_{LSBs} + \\ &A_{LSBs} \times B_{MSBs} + A_{LSBs} \times B_{LSBs} \end{aligned} \tag{20}$$

The term $A_{LSBs} \times B_{LSBs}$ corresponds to the product of the least significant bits and falls entirely within the truncation range. Therefore, only the first three terms in the expansion need to be retained. Moreover, for the computation of M_1^2, an additional multiplier can be eliminated by using a left-shift operation when computing $2M_{1\_MSBs}M_{1\_LSBs}$. This optimization effectively reduces the multiplier area and logic depth without introducing a significant impact on the approximation error. The error result is shown in Table II of Section V.

IV. The High-Precision Softmax Architecture

Guided by the theoretical framework and hardware-oriented algorithmic optimizations established in Section II and Section III, we present the efficient hardware implementation of the proposed design. The overall hardware architecture is illustrated in Fig. 2. It primarily consists of several key modules, including a comparison module, a subtraction module, an adder tree, and exponentiation and logarithm modules based on TQA-Softmax. Each submodule is implemented using a pipelined design to support high-throughput operation.

Initially, the input sequences X is processed by the comparison module, which performs parallel comparisons to determine the maximum value x_{max}. Subsequently, the subtraction module computes the shifted values x_i' by subtracting x_{max} from each input element.

For a signed fixed-point negative value x_i', the integer part a_i and fractional part b_i can be obtained directly, since a_i intrinsically corresponds to the integer portion and b_i to the fractional portion. This direct mapping requires no additional hardware operations, making it highly hardware-friendly. As a result, x_i' can be efficiently decomposed into a_{1i} and b_{1i}.

Following the decomposition, the fractional component b_{1i} is fed into the exponentiation module to compute $2^{b_{1i}}$. A subsequent right-shift operation produces exp_{1i}. An adder tree then accumulates all exp_{1i} values to generate the summation result, denoted as sum.

Next, a Leading One Detector (LOD) is applied to sum to determine the integer value k. The mantissa m is then obtained through a right-shift operation and forwarded to the logarithm module to compute $\log_2(m)$. Since $\log_2(m)$ lies in the interval $[0, 1)$, and k is an integer, $\log_2(sum)$ can be obtained via a simple concatenation operation, thereby eliminating the need for an additional adder.

Pipeline registers are inserted to store intermediate values during the computation process, such as x_i', ensuring proper data alignment across pipeline stages. Finally, the subtraction and exponentiation modules are reused to compute $x_i' - \log_2(sum)$ and generate the final output $f_2(x_i)$, improving hardware utilization efficiency.

The detailed hardware implementation of the core TQA-Softmax arithmetic unit is illustrated in Fig. 3. To maximize system throughput and satisfy high-frequency timing constraints, the unit adopts a fully pipelined architecture.

The input operand is first partitioned into three bit segments: the most significant bits M_0, the middle segment M_1, and the LSBs M_2. The segment M_0 drives the coefficient selection logic, which is implemented using multiplexers to efficiently retrieve the corresponding pre-computed coefficients $[w_{i,2}, w_{i,1}, w_{i,0}]$ for the current sub-interval.

The arithmetic execution is decoupled into two parallel computational branches to reduce latency. In the quadratic computation branch, the segment M_1 is first squared and subsequently multiplied by the second-order coefficient $w_{i,2}$ to capture the curvature of the function. Simultaneously, the linear computation branch processes the concatenated segment $\{M_1, M_2\}$, performing multiplication with the first-order coefficient $w_{i,1}$ and then adds $w_{i,0}$. The outputs of these parallel branches are finally accumulated to generate the result, effectively balancing hardware complexity with high-precision approximation requirements.

V. Experimental Results and Analysis

In this section, we present a comprehensive evaluation of the proposed base-2 softmax. The assessment covers three dimensions: numerical approximation accuracy, hardware implementation overhead, and network-level application. First, we analyze the error through extensive software simulations and compare the precision of our method against state-of-the-art designs. Second, we synthesize the Verilog RTL code and provide a comparison. Finally, we apply the designed base-2 softmax to Transformer training and evaluate its effectiveness by observing the network's training performance.

A. Numerical Approximation Accuracy

To guarantee stable network convergence during training, strict constraints must be imposed on the computational pre-

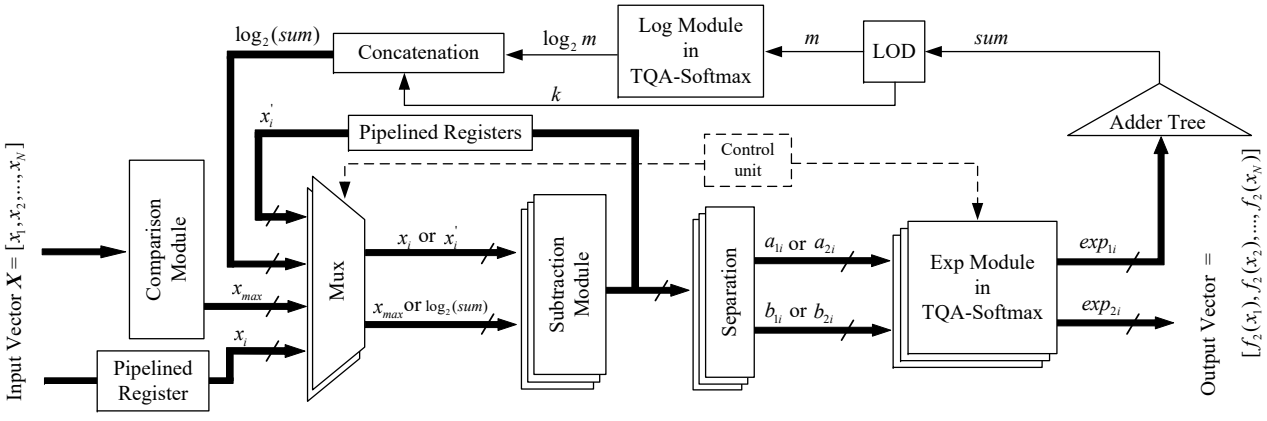

Fig. 2: The overall architecture of softmax based on TQA-Softmax

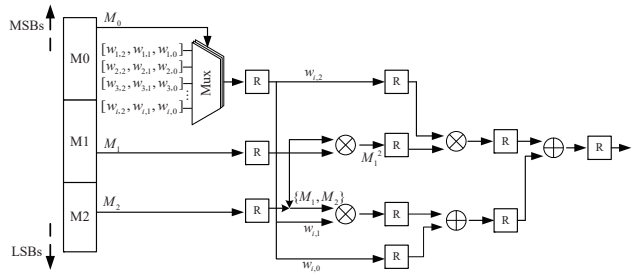

Fig. 3: The architecture of unit based on TQA-Softmax

cision of the softmax operator. As demonstrated in [12], for base-2 softmax approximation, a Maximum Absolute Calculation Error (MACE) on the order of 10^{-6} is required to achieve training stability and accuracy comparable to the baseline. Accordingly, we adopt a stringent target error threshold of $\epsilon_{target} \leq 1.0 \times 10^{-6}$ for our hardware design.

To satisfy the target error precision ϵ_{target}, the unit-in-the-last-place parameter N_{ulp} must satisfy:

$$N_{ulp} \geq -\log_2(\epsilon_{target}) \approx 19.93 \tag{21}$$

Thus, the value of N_{ulp} is set to 21 in our design.

To determine the value of N_0 for the exponentiation and logarithm modules, we conduct MATLAB simulations using the LSM without coefficient quantization or bit-width truncation.

TABLE I: Model Error of Exponentiation and Logarithm

N_0	Exponentiation		Logarithm	
	MACE	MAE	MACE	MAE
3	1.04e-5	2.54e-6	3.96e-5	5.74e-6
4	**1.33e-6**	**3.17e-7**	5.42e-6	7.15e-7
5	1.68e-7	3.97e-8	**7.05e-7**	**8.94e-8**

As shown in Table I, where Model Error denotes e_m in Section III and MAE denotes Mean Absolute Error, selecting $N_0 = 5$ would double the coefficient storage requirements and result in a substantial increase in hardware overhead due to the parallel exponentiation modules. Therefore, we set N_0 to 4 for the exponentiation operation. In contrast, the logarithm module adopts N_0 = 5 to meet the required accuracy. The bit widths of other variables can therefore be determined through the equations in Section III.

We evaluate the accuracy of the proposed design by applying 800,000 randomly distributed single-precision floating-point inputs over different input ranges as the baseline. The measured MAE and MACE are summarized in Table II, where *TQA-Acc* denotes results obtained using accurate multipliers, and *TQA-App* represents those using approximate multipliers. It is evident that the adoption of approximate multipliers does not introduce a significant increase in error. Moreover, compared with state-of-the-art designs [12], the proposed method achieves lower numerical errors.

TABLE II: Numerical Error Analysis

Method	Input Range	MACE	MAE
TQA-Acc	$[-1, 1]$	8.34e-7	2.15e-7
TQA-App		1.04e-6	2.26e-7
TQA-Acc	$[-5, 5]$	2.08e-6	2.30e-7
TQA-App		2.50e-6	2.60e-7
TQA-Acc	$[-10, 10]$	2.92e-6	2.52e-7
TQA-App		2.98e-6	2.75e-7
[12]	—	3.81e-6	—

Although MACE marginally exceeds 10^{-6}, MAE remains below 3×10^{-7}, implying that majority of inputs satisfy the strict precision threshold. Furthermore, given that $N_{ulp} = 21$ corresponds to a theoretical approximation error of 4.7×10^{-7}, the result that our MAE surpasses this limit validates the superior efficiency of TQA-Softmax.

B. Hardware Implementation Overhead

To evaluate the hardware efficiency of the proposed design, the architecture is implemented in Verilog HDL and synthesized using Synopsys Design Compiler (DC) with a TSMC 28-nm CMOS standard cell library. For a fair comparison with the state-of-the-art work presented in [12], we adopt an identical 8-way parallel configuration, in which each input consists of 4 integer bits and 21 fractional bits. The comprehensive synthesis results are summarized in Table III.

As shown in Table III, our final design achieves a hardware area of 24891.80μm^2 and a power consumption of 27.44mW at a working frequency of 1GHz. Compared with *TQA-Acc*, the use of approximate multipliers reduces the circuit area by 4.37% and power consumption by 4.06% while maintaining the required numerical precision. In addition, a 4.27% reduction in Area–Delay Product (ADP), together with an 8.93% improvement in hardware efficiency, demonstrates the effectiveness of the truncated approximate multipliers.

As summarized in Table III, the proposed design shows comprehensive advantages over state-of-the-art works [12], [14], [30]. Specifically, compared with the best-performing baseline [12], the hardware area is reduced by approximately 74.80% (from 98,787.43 μm^2 to 24,891.80 μm^2). Because of the pipeline structure described in Section IV, the computation latency is optimized to 18 clock cycles, comprising 3 clock cycles for the adder tree and 4 cycles each for the exponentiation and logarithm modules. Consequently, the proposed design achieves a lower ADP of 0.448 than 2.568 in [12] and an overall hardware efficiency of 11.71, which is 3.57× higher than that of [12]. While the power consumption is slightly higher, a detailed power analysis report indicates that this is primarily attributed to the extensive use of pipeline registers required to ensure high throughput and the storage overhead for TQA-Softmax coefficients.

TABLE III: Comparison of Hardware Synthesis Results

[1]Design	Area (μm^2)	Power (mW)	[2]Latency	[3]ADP ($mm^2 \cdot ns$)	[4]Efficiency (G/s/$mm^2 \cdot mW$)
TQA-Acc	26,028.20	28.60	18	0.468	10.75
TQA-App	24,891.80	27.44	18	0.448	11.71
CORDIC-based [12]	98,787.43	24.27	26	2.568	3.276
PWL-based [30]	166804.48	40.70	17	2.836	1.178
LUT-based [14]	482304.12	45.95	17	8.199	0.361

[1]Design: All designs are synthesized at TSMC28nm with a 1GHz clock frequency and a throughput of 8 G/s.
[2]Latency: Computation Latency (clock cycles).
[3]ADP: Area-Delay Product ($mm^2 \cdot ns$).
[4]Efficiency: Throughput / (Area $\cdot$ Power), Throughput = Inputs $\times$ Frequency.

C. Network-Level Training Evaluation

To evaluate the effectiveness of the proposed softmax in practical Transformer training, we integrate our design into two representative frameworks covering both computer vision and natural language processing domains. For the CV task, we employ a Vision Transformer variant based on the VTs-Drloc framework [31] and train it from scratch on the CIFAR-100 dataset. For the NLP task, we utilize a BERT-based model from the Chinese-Text-Classification-PyTorch repository [32], which is fine-tuned on the THUCNews dataset consisting of 200,000 samples across 10 categories.

For each network, training is conducted using three softmax configurations: exact base-e softmax, exact base-2 softmax, and the proposed approximate base-2 softmax. For the approximate base-2 softmax, to ensure numerical stability and effective weight updates during backpropagation, the gradient computation is defined as follows:

$$\frac{\partial f_2(x_i)}{\partial x_j} = \begin{cases} \ln 2 \times f_2(x_i)\left(1 - f_2(x_j)\right), i = j \\ \ln 2 \times f_2(x_i)\left(0 - f_2(x_j)\right), i \neq j \end{cases} \tag{22}$$

Training loss and validation accuracy are monitored across epochs to assess convergence behavior and performance. Fig. 4(a) illustrates the learning curves of the VTs-Drloc model trained from scratch over 180 epochs. The training loss curves under all three configurations exhibit similar convergence trends, indicating stable optimization behavior. Moreover, the validation accuracy achieved by the proposed design closely matches the exact baselines. Specifically, at the 180th epoch, our method attains an accuracy of 74.705%, which is comparable to 74.617% obtained with the exact base-e softmax and 75.177% with the exact base-2 softmax, demonstrating negligible performance degradation.

Similarly, Fig. 4(b) depicts the fine-tuning process of the BERT-based model over 20 epochs. Benefiting from the pre-trained weights, the training loss converges rapidly. Throughout the fine-tuning process, the proposed design maintains a validation accuracy in the range of 93%–94%, which is statistically equivalent to that of the exact softmax counterparts.

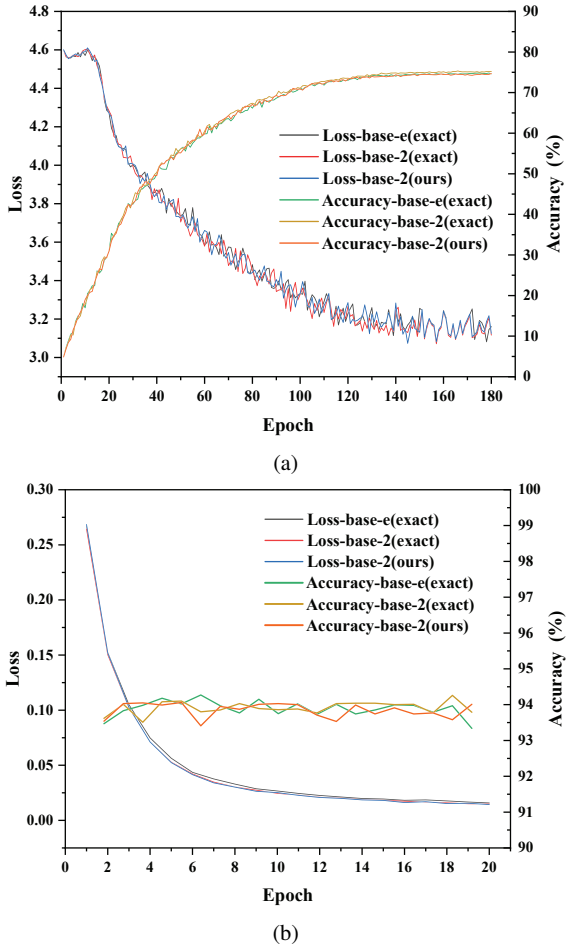

Fig. 4: (a)Training Loss and Validation Accuracy in different networks. (a) VTs-Drloc; (b) BERT-based Model

The experimental results in Fig. 4 validate the applicability of base-2 softmax in Transformer models and demonstrate that the proposed approximation closely matches the exact baselines in both convergence behavior and final accuracy.

VI. CONCLUSION

In this paper, we propose a high-precision and low-latency base-2 softmax design for Transformer training, using TQA-Softmax to implement the critical nonlinear operators in softmax. Through detailed error analysis and hardware-oriented algorithmic optimizations, we achieve a robust architecture. Furthermore, network-level training experiments on Transformer network confirm that our design demonstrates negligible performance degradation compared to the exact baseline, validating its suitability for high-performance neural network training.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[3] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[4] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in neural information processing systems*, vol. 35, pp. 16 344–16 359, 2022.

[5] Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, Z. Zhao, and C. Zheng, "Synthesizer: Rethinking self-attention for transformer models," in *International conference on machine learning*. PMLR, 2021, pp. 10 183–10 192.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[7] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[8] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[9] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

[10] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 469–474.

[11] B. Yuan, "Efficient hardware architecture of softmax layer in deep neural network," in *2016 29th IEEE International System-on-Chip Conference (SOCC)*, 2016, pp. 323–326.

[12] Y. Zhang, L. Peng, L. Quan, Y. Zhang, S. Zheng, and H. Chen, "High-precision method and architecture for base-2 softmax function in DNN training," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 8, pp. 3268–3279, 2023.

[13] G. Du, C. Tian, Z. Li, D. Zhang, Y. Yin, and Y. Ouyang, "Efficient softmax hardware architecture for deep neural networks," in *Proceedings of the 2019 Great Lakes Symposium on VLSI*, 2019, pp. 75–80.

[14] Y. Zhang, Y. Zhang, L. Peng, L. Quan, S. Zheng, Z. Lu, and H. Chen, "Base-2 softmax function: Suitability for training and efficient hardware implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 9, pp. 3605–3618, 2022.

[15] Y. Wu, Z. Xie, H. Pan, and Y. Wang, "MBS: A High-Precision Approximation Method for Softmax and Efficient Hardware Implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2025.

[16] M. Wang, S. Lu, D. Zhu, J. Lin, and Z. Wang, "A high-speed and low-complexity architecture for softmax function in deep learning," in *2018 IEEE asia pacific conference on circuits and systems (APCCAS)*. IEEE, 2018, pp. 223–226.

[17] D. Zhu, S. Lu, M. Wang, J. Lin, and Z. Wang, "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.

[18] Z. Mei, H. Dong, Y. Wang, and H. Pan, "TEA-S: A tiny and efficient architecture for PLAC-based softmax in transformers," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 9, pp. 3594–3598, 2023.

[19] Q. Sun, Z. Di, Z. Lv, F. Song, Q. Xiang, Q. Feng, Y. Fan, X. Yu, and W. Wang, "A high speed softmax VLSI architecture based on basic-split," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. IEEE, 2018, pp. 1–3.

[20] Y. Gao, W. Liu, and F. Lombardi, "Design and implementation of an approximate softmax layer for deep neural networks," in *2020 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[21] K. Chen, Y. Gao, H. Waris, W. Liu, and F. Lombardi, "Approximate softmax functions for energy-efficient deep neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 1, pp. 4–16, 2022.

[22] J. Kim, S. Kim, K. Choi, and I.-C. Park, "Hardware-efficient SoftMax architecture with bit-wise exponentiation and reciprocal calculation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[23] A. Kagalkar and S. Raghuram, "CORDIC based implementation of the softmax activation function," in *2020 24th International Symposium on VLSI Design and Test (VDAT)*. IEEE, 2020, pp. 1–4.

[24] Y. Cao, W. Xiao, J. Jia, D. Wu, and W. Zhou, "Cordic-based softmax acceleration method of convolution neural network on FPGA," in *2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS)*. IEEE, 2020, pp. 66–70.

[25] Q. Zhang, J. Cao, S. Zhang, Q. Zhang, Y. Zhang *et al.*, "Efficient FPGA implementation of softmax layer in deep neural network," in *2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP)*. IEEE, 2020, pp. 935–939.

[26] S. Raghuram, A. S. Bharadwaj, S. Deepika, M. S. Khadabadi, and A. Jayaprakash, "Digital implementation of the softmax activation function and the inverse softmax function," in *2022 4th international conference on circuits, control, communication and computing (I4C)*. IEEE, 2022, pp. 64–67.

[27] G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, A. Nannarelli, M. Re, and S. Spanò, "A pseudo-softmax function for hardware-based high speed image classification," *Scientific reports*, vol. 11, no. 1, p. 15307, 2021.

[28] L. Quan, B. Wang, F. Lyu, and H. Chen, "PQA-FGS: Piecewise Quadratic Approximation with Fine-grained Segmentation for High-Precision Non-linear Computation," in *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2025, pp. 1–5.

[29] F. De Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.

[30] F. Lyu, X. Xu, Y. Wang, Y. Luo, Y. Wang, and H. Pan, "Ultralow-latency VLSI architecture based on a linear approximation method for computing Nth roots of floating-point numbers," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 2, pp. 715–727, 2020.

[31] Y. Liu, E. Sangineto, W. Bi, N. Sebe, B. Lepri, and M. De Nadai, "Efficient Training of Visual Transformers with Small Datasets," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[32] JackHCC, "Chinese-Text-Classification-PyTorch," https://github.com/JackHCC/Chinese-Text-Classification-PyTorch, 2018.