Algorithm-Hardware Co-design of a Unified Accelerator for Non-linear Functions in Transformers

Haonan Du[1], Chenyi Wen[1], Zhengrui Chen[1], Li Zhang[2], Qi Sun[1], Zheyu Yan[1,*], Cheng Zhuo[1]
[1]*Zhejiang University*, [2]*Hubei University Of Technology*

***Abstract*—Non-linear functions (NFs) in Transformers require high-precision computation consuming significant time and energy, despite the aggressive quantization schemes for other components. Piece-wise Linear (PWL) approximation-based methods offer more efficient processing schemes for NFs but fall short in dealing with functions with high non-linearities. Moreover, PWL-based methods still suffer from inevitably high latency introduced by the Multiply-And-Add (MADD) unit. To address these issues, this paper proposes a novel quadratic approximation scheme and a highly integrated, multiplier-less hardware structure, as a unified method to accelerate any unary non-linear function. We also demonstrate implementation examples for GELU, Softmax, and LayerNorm. The experimental results show that the proposed method achieves up to 5.41% higher inference accuracy and 60.12% lower area-delay product.**

***Index Terms*—Non-linear function, Transformer, Hardware-efficient accelerator, Approximate computing**

I. Introduction

Transformer-based models have achieved remarkable success in various fields, such as natural language processing and computer vision [1]. The recent boom in large language models (LLMs) further consolidates their dominance. However, these demanding models require a significant amount of computational power and energy, which hinders their further development, especially as model sizes scale up exponentially [2], [3].

To alleviate this computational burden, various approaches have been explored, such as model quantization [4], pruning [5], *etc*. These methods help reduce the model sizes, but cannot improve runtime performance unless they cooperate with specialized hardware [6]. Most hardware accelerators focus on the efficient processing of linear operations, typically matrix multiplications (MatMul). Non-linear function (NF) accelerations are largely under-explored due to their complexity, diversity, and high accuracy requirement [7]. However, as MatMul accelerators become increasingly efficient, the relative impact of NFs in transformers, including GELU, Softmax, and LayerNorm, becomes more significant.

In response to this challenge, some methods design dedicated datapaths for individual or specific series of NFs. For example, SOLE [8] uses log2 quantization and log-based division to approximate Softmax and low-precision statistic calculation for LayerNorm. ReAFM [9] proposes a reciprocal approximation optimization method and a precision adjustable exponential unit, supporting Swish, Sigmoid, Tanh and GELU, *etc.* However, these methods require designing and implementing different hardware components for each non-linear function. For transformer-based models containing multiple NFs, the

*Corresponding author: Zheyu Yan, zyan2@zju.edu.cn

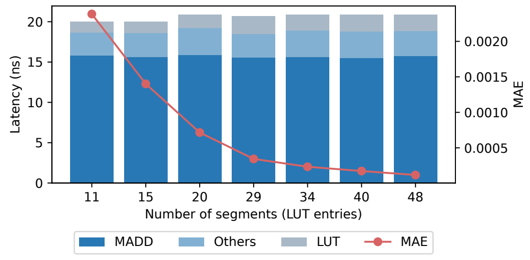


Fig. 1. Latency breakdown of different components in the conventional PWL method [11] for approximating the GELU function over the range $[-4, 4]$. The MADD unit accounts for the majority of the latency, which cannot be reduced by reducing the number of segments ($i.e.$, LUT entries).

accumulated area overhead and design effort to implement particular hardware accelerators for each NF is non-trivial.

Alternatively, methods leveraging the Piece-Wise Linear (PWL) technique offer a more unified scheme. In PWL, a non-linear function is segmented into several pieces, and each piece is approximated as a linear function. A Look-Up Table (LUT) can be used to store and identify which linear piece the input value falls into, and the corresponding coefficients are then passed to the multiply-and-add (MADD) unit for final computation [10]–[13].

Recent advances in PWL-based methods generally focus on improving segmentation strategies. For example, NN-LUT [12] employs a neural network to generate an efficient segmentation scheme, and GQA-LUT [13] uses genetic algorithms to find the optimal breakpoints in a quantization-aware approach. However, despite these advancements in segmentation strategies, these methods have not addressed the major bottleneck: the arithmetic logic component, specifically the MADD unit. As shown in Fig. 1, the MADD unit contributes to the majority of the computation latency and cannot be reduced by simply scaling down the LUT entries. This bottleneck limits the overall efficiency gains achievable by PWL methods, highlighting the need for optimization beyond segmentation strategies.

In short, the previous works highlight two major challenges. First, approaches with dedicated datapaths lack hardware reuse across different NFs, *requiring more resources and resulting in higher latency*. Second, while PWL/LUT methods offer a more unified approach, they are less efficient because *the major source of latency—the MADD unit—remains unoptimized.*

To overcome these limitations, in this paper, we **propose a Quadratic approximation-based Unified accelerator for Non-linear Functions (QUNF).** Through algorithm-hardware co-design, QUNF provides an efficient acceleration scheme for NFs, particularly those used in transformer models. QUNF uniformly segments the non-linear function and uses second-order Taylor expansions to fit the curve in each segment. While offering better generality than methods with dedicated datapaths, it achieves higher accuracy more easily than prior PWL-based methods. We further approximate the quadratic function computation arithmetic to enable hardware-friendly operations while preserving the precision of quadratic fitting. The incremental precision design in this approximated arithmetic provides greater flexibility in balancing accuracy and resource utilization. Moreover, with the proposed approximation scheme, we design a unified accelerator that achieves significantly higher efficiency than traditional PWL methods using MADD units. By incorporating segmentation-aware hardware pruning, QUNF eliminates redundant logic components without any accuracy degradation, further enhancing efficiency. Finally, QUNF provides a clear mathematical formulation of the approximate computing scheme, enabling stricter error analysis and facilitating optimizations focused on accuracy.

In summary, our contributions can be concluded as follows:

- We propose **a quadratic piece-wise curve fitting method for unary non-linear functions** using second-order Taylor expansions, achieving higher accuracy with fewer segments than PWL methods and reducing LUT size and cost.
- We introduce **a hardware-aware approximation scheme** with incremental precision design, enabling highly efficient hardware designs and flexible trade-off options.
- We develop **a highly integrated hardware implementation** with a lossless pruning scheme, which achieves high efficiency while maintaining superior accuracy.
- Finally, we propose **a universal accuracy optimization technique** for our method based on analysis and modeling of approximation error.

Experimental results show that the proposed method achieves up to 5.41% higher accuracy when accelerating RoBERTa-large model, and up to 60.12%, 45.40%, and 35.42% times lower Area-Delay-Product (ADP) in GELU, nexp, and rsqrt functions, respectively.

II. Background and Related Work

A. Non-Linear Functions in Transformers

GELU, Softmax, and LayerNorm are the three major non-linear operations in recent transformer structures. GELU is used as the activation function in the feed-forward network layer [14], and Softmax is used in the attention mechanism to convert a set of scores into a probability distribution, while LayerNorm is used for normalization. The definitions of GELU, Softmax, and LayerNorm are as follows:

$$\mathrm{GELU}(x) = \frac{x}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{1}$$

$$\mathrm{Softmax}(X_i) = \frac{\exp(X_i - X_{\max})}{\sum_{j=1}^{n}\exp(X_j - X_{\max})} \tag{2}$$

$$\mathrm{LayerNorm}(x_i) = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} * \gamma + \beta \tag{3}$$

where $\mathrm{erf}(x)$ is the Error Function, and μ, σ^2 are the mean and variance value of the input vector. For Softmax, it is a common technique to subtract $X_{\max}$ to avoid exponential overflow [15], without affecting the numerical result.

It can be observed that these operations have unique and complicated mathematical representations. To perform the calculation, costly and slow FP32 arithmetic is required, taking up a non-negligible part of the execution time [16]. To address this issue, numerous approaches have been proposed to reduce the cost of non-linear operations, by leveraging the tolerance for approximation in these models and tasks [17].

B. Prior Non-Linear Implementation

Most recent works on non-linear approximation employ the PWL method. The principle of PWL [11] is:

$$f(x) \approx \begin{cases} k_1 x + b_1, & x \in [M, x_2) \\ \cdots \\ k_n x + b_n, & x \in [x_n, N] \end{cases} \tag{4}$$

By dividing the input range into segments, PWL approximates the function within each segment using a straight line. A LUT stores the boundaries, slopes, and intercepts. The segments can be uniform or non-uniform. Non-uniform segmentation often reduces LUT entries and improves accuracy by arranging segments according to the target function [10].

Several state-of-the-art works have utilized the PWL with LUT method to approximate non-linear operations in deep neural networks and transformers. Sun *et al.* [10] minimize the number of segments based on a specified mean absolute error (MAE) target. PLAC [11] further reduces segments by exploiting the discrete nature of inputs. NN-LUT [12] employs a simple neural network to approximate non-linear functions, and the weights are then transformed into PWL LUT entries. RI-LUT [18] addresses dynamic input ranges during fine-tuning by normalizing inputs using power-of-two scaling. GQA-LUT [13] proposes a quantization-aware algorithm using genetic methods to optimize breakpoints.

Alternatively, some works design dedicated datapaths to approximate specific non-linear functions. SOLE [8] implements Softmax and LayerNorm via log2 quantization and log-based division. ReAFM [9] approximates ReLU, Sigmoid, Tanh, and their variants—including GELU—using reciprocal approximation based on one-step Newton-Raphson iteration, combined with an exponential module based on Taylor expansion and base conversion from e to 2. I-BERT [4] proposes a polynomial-based approximation to accelerate GELU, Softmax, and LayerNorm under integer arithmetic with quantization but lacks hardware implementation.

Despite achieving notable results on their specific optimization targets, these methods have limitations. Dedicated datapath approaches support only specific non-linear functions and often consume more resources due to stacked submodules. PWL/LUT-based methods offer versatility, but typically require large LUT entries for acceptable precision [18]. Most prior works focus on segmentation strategies, but lack hardware integration to achieve maximum efficiency. Thus, our work aims to delve deeper into algorithm-hardware co-design, providing more efficient solutions for universal non-linear approximation.

III. Proposed Methodology

A. Piece-wise Quadratic Curve Fitting

Given a target function $g(x)$ on range $[L, R)$, we segment it into pieces, similarly to the PWL method, and fit each piece with a quadratic function. To design a hardware-friendly acceleration scheme, we adopt the following design principles:

- We assume L, R are integers.
- We employ uniform segmentation, *i.e.,* the segment width w remains constant for each segment.
- We set $w = \frac{1}{2^d}$, where d is a non-negative integer, serving as the segmentation factor.

In other words, the factor d is corresponding to the segmentation set $\{[l_k, r_k)\} = \{[L + \frac{k}{2^d}, L + \frac{k+1}{2^d})\}$, where k is the segment index from 0 to $\frac{L-R}{2^d}$. Fig. 2 illustrates the segmentation scheme and the related factors.

To approximate the target function using a quadratic curve, we need to find the best coefficients a_k, b_k, c_k, such that the difference between $f_k(x) = a_k + b_k x + c_k x^2$ and $g(x)$ in the range $[l_k, r_k)$ is minimized. For each segment, we perform a second-order Taylor series expansion on the target function $g(x)$ at the middle point $x_0 = \frac{l_k + r_k}{2}$, to obtain a coefficient group a, b, and c, as follows:

$$g(x) \approx g(x_0) + g'(x_0)(x - x_0) + \frac{g''(x_0)}{2}(x - x_0)^2 \tag{5}$$

$$\approx f(x) = a + bx + cx^2, \tag{6}$$

$$\text{where} \begin{cases} a = g(x_0) - g'(x_0)x_0 + \frac{g''(x_0)}{2}x_0^2 \\ b = g'(x_0) - g''(x_0)x_0 \\ c = \frac{1}{2}g''(x_0) \end{cases} \tag{7}$$

In practice, one can also directly solve the coefficients from the start, middle, and end point of the segment and obtain nearly the same result. However, by using Taylor expansion, we establish a relationship between the coefficients and the characteristics of the approximated function, which will aid in error analysis and optimization later in Section III-D and III-E.

The accuracy of quadratic approximation undoubtedly outperforms linear approximation. However, it is clear that the quadratic method involves more multiplications if implemented directly. In the following sections, we will address this issue through highly integrated hardware-software co-design, achieving significant resources savings compared to PWL implementation while still outperforming in precision.

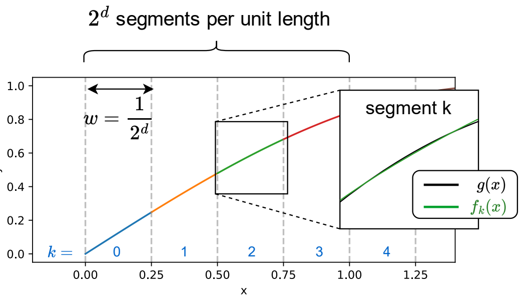


Fig. 2. Segmentation scheme used in QUNF.

B. Hardware-aware Approximation

For any input number x located in the segment $[l_k, r_k)$, we normalize the input range by subtracting l_k, acquiring $s = x - l_k \in [0, \frac{1}{2^d})$. In the rest part of Section III-B, we focus on the calculation within each segment. Therefore, the index subscript k of a_k, b_k, c_k, l_k is omitted for simplicity. Then we can transform $f(x)$ into $f(s)$:

$$\begin{aligned} f(s) &= a + b(l + s) + c(l + s)^2 \\ &= A + 2Bs + Cs^2 \end{aligned} \tag{8}$$

where $A = (a + bl + cl^2)$, $B = (0.5b + cl)$, and $C = c$ remain constants within each segment. To achieve adjustable precision, we quantize s to n bits, such that $s \in [\frac{t}{2^n}, \frac{t+1}{2^n})$, where $t = \lfloor s \cdot 2^n \rfloor$ is an integer within range $[0, 2^n)$. Denoting $s_1 = \frac{t}{2^n}, s_2 = \frac{t+1}{2^n}$, we use the integral average of $f(s)$ in the range $[s_1, s_2)$ to obtain the closest approximation after quantization, as in (9):

$$\begin{aligned} \hat{f}(s) &= \frac{1}{s_2 - s_1}\int_{s_1}^{s_2} f(s)ds \\ &= A + B(s_1 + s_2) + \frac{1}{3}C(s_1^2 + s_2^2 + s_1 s_2) \end{aligned} \tag{9}$$

Substitute s_1 and s_2 into t, and we acquire (10):

$$\hat{f}(t) = (A + \varepsilon) + B\frac{2t + 1}{2^n} + C\frac{t^2 + t}{2^{2n}} \tag{10}$$

where $\varepsilon = \frac{C}{3 \cdot 2^{2n}}$ is a function of n in each segment. ε is usually omitted to generate a consistent coefficient set that is adaptable to different precision levels, and this has a negligible influence on accuracy.

C. Incremental Precision

To provide further design space for resource-accuracy tradeoff, (10) can be divided into multiple stages as follows:

$$\hat{f}^{(n)}(t_n) = \hat{f}^{(1)}(t_1) + \sum_{i=2}^{n} \Delta\hat{f}^{(i)}(t_i) \tag{11}$$

$$\hat{f}^{(1)}(t_1) = \begin{cases} A + 0.5B, & t_{[1]} = 0 \\ A + 1.5B + 0.5C, & t_{[1]} = 1 \end{cases} \tag{12}$$

$$\begin{aligned} \Delta\hat{f}^{(i)}(t_i) &= \hat{f}^{(i)}(t_i) - \hat{f}^{(i-1)}(t_{i-1}) \\ &= \begin{cases} -B\frac{1}{2^i} - C\frac{t_i}{2^{2i}}, & t_{[i]} = 0 \\ B\frac{1}{2^i} + C\frac{t_i + 1}{2^{2i}}, & t_{[i]} = 1 \end{cases} \end{aligned} \tag{13}$$

where t_i denotes the i-most significant bits (MSBs) of t, while $t_{[i]}$ represents the i-th MSB.

More accumulated stages provide higher precision, but also increase resource utilization [19]. The subsequent sections will show that, under most circumstances, a total precision level n of 4$\sim$ 8 can achieve a competitive approximation accuracy.

D. Error Analysis

Simulation results indicate that the error from theoretical quadratic curve-fitting is less than 1×10^{-7}. The primary error arises from the quantization in each segment, as described in (14):

$$\left|\hat{f}(t) - f(s)\right| = \left|B\left(\frac{2t + 1}{2^n} - 2s\right) + C\left(\frac{t^2 + t}{2^{2n}} - s^2\right)\right| \tag{14}$$

The upper bound of the error is approached when s gets close to $\frac{t+1}{2^n}$ and deviates the most from $\frac{t}{2^n}$:

$$\begin{aligned}&\left|\hat{f}(t) - f(s)\right| < \left|\hat{f}(s \cdot 2^n - 1) - f(s)\right| \\ &= \cdots = \left|-\frac{B}{2^n} - \frac{Cs}{2^n}\right| = \frac{1}{2^n}\left|B + Cs\right|\end{aligned} \tag{15}$$

By expanding B, C back to (7), we obtain:

$$\begin{aligned}\left|\hat{f}_k(t) - f_k(s)\right| &< \frac{1}{2^n}\left|B_k + C_k s\right| = \frac{1}{2^{n+1}}\left|b_k + 2c_k x\right| \\ &= \frac{1}{2^{n+1}}\left|g'(x_{0_k}) + (x - x_{0_k})g''(x_{0_k})\right| \\ &< \frac{1}{2^{n+1}}\left|g'(x_{0_k}) + \frac{1}{2^{d+1}}g''(x_{0_k})\right|\end{aligned} \tag{16}$$

Equation (16) demonstrates the relationship between the error and the derivatives of $g(x)$. Specifically, the approximate error can be reduced by selecting a target $g(x)$ with a smaller derivative magnitude.

E. Non-Linear Implementation and Optimization

The proposed method can approximate most non-linear functions directly, *e.g.*, Sigmoid, Tanh, and GELU. Others (*e.g.*, rsqrt in LayerNorm) can be approximated after hardware-friendly transformations. Moreover, in this work, we introduce a function transformation methodology to improve the approximation precision. As proved in Section III-D, this can be achieved by transforming the target function into one with a smaller derivative magnitude.

Therefore, the non-linear functions in Softmax, LayerNorm and GELU are implemented as follows:

Nexp in Softmax: According to (2), the input to the exponential function in Softmax is non-positive. We can identify the key non-linear function here as $\text{nexp}(x) = \exp(-x)$, where $x \geqslant 0$. The nexp function approaches zero as x increases. For $x \geqslant 7$, the value of nexp is less than 1×10^{-4}, so it can be approximated as zero for larger values.

Rsqrt in LayerNorm: i.e., $\text{rsqrt}(x) = \frac{1}{\sqrt{x}} = x^{-\frac{1}{2}}$. Rsqrt has a wide input/output range. We adopt the method proposed in [18], as follows:

$$(m \cdot 2^E)^{-0.5} = \begin{cases} m^{-0.5} \cdot 2^{-0.5E}, & \text{if E is even} \\ (2m)^{-0.5} \cdot 2^{-0.5(E-1)}, & \text{if E is odd} \end{cases} \tag{17}$$

where m and E are the mantissa and exponent part of the floating-point representation of x. Therefore, we only need to approximate rsqrt in the range of $[1, 4]$, while supporting a wide dynamic range at the cost of two additional simple shift operations.

GELU: From (1) we derive two auxiliary functions:

$$V(x) = x \cdot \text{erf}\left(\frac{x}{\sqrt{2}}\right); \quad H(x) = V(x) - |x| \tag{18}$$

Notice that $V(x)$ is an even function. One optimization is to approximate $V(x)$ instead of GELU, leveraging its symmetry to halve the number of LUT entries. Moreover, for $V'(x)$ in the positive input range, it approaches 1 as x increases. Therefore, by simply subtracting $|x|$ from $V(x)$, we acquire $H(x)$, where $|H'(x)|$ is smaller than $|V'(x)|$ and approaches zero.

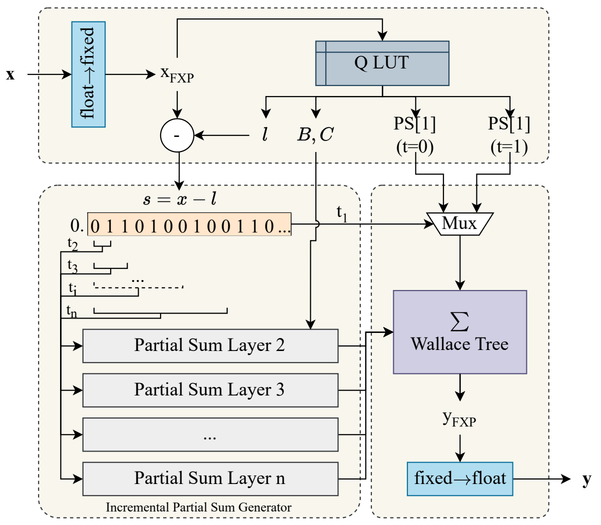


Fig. 3. Overview of QUNF hardware. The logic for extra transformations required by some non-linear functions is not included.

Now GELU can be calculated from $H(x)$ as follows:

$$\text{GELU}(x) = \begin{cases} 0.5H(|x|) + x, & x \geqslant 0 \\ 0.5H(|x|) + 0, & x < 0 \end{cases} \tag{19}$$

If we only approximate $H(x)$, we further reduce the approximate error of GELU, as described in Section III-D. An additional adder and shifter are required. In the remainder of this paper, the proposed QUNF implementation uses the function $H(x)$ for GELU by default, unless stated otherwise.

IV. Hardware Design

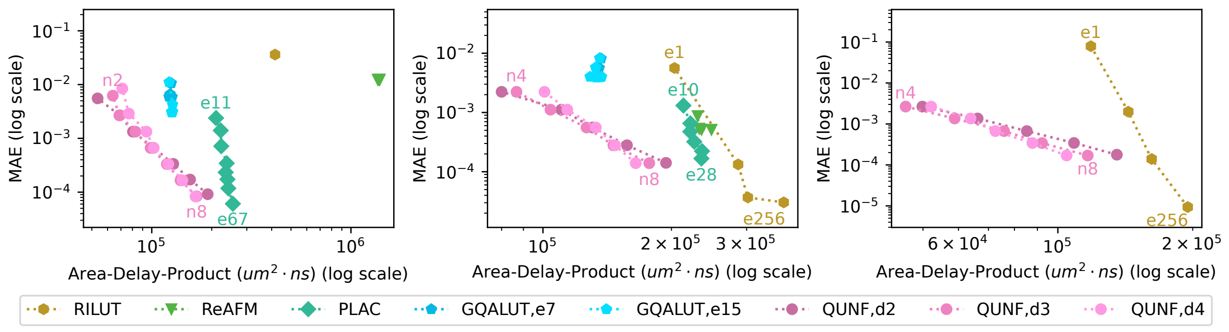
A. Overview

Fig. 3 presents an overall hardware architecture of the proposed method. Despite using floating-point input and output, the design employs fixed-point numbers internally to further reduce resource consumption [20]. The design includes a Q LUT to store the quadratic coefficients defined in (8), an incremental partial sum generation module, and a modified Wallace tree [21], [22] structure to accumulate all partial sums (PSs). The partial sum items from layer $i = 1$ defined in (12) are pre-computed and stored in the Q LUT to reduce the number of additions. However, the designer may choose not to do so if the LUT is external or limited.

B. Partial Sum Generation

This module cooperates with the approximation scheme and generates partial sums. We adopt the CSD Encoding and propose a novel lossless segment-aware hardware pruning technique to reduce resource consumption. For simplicity, the partial sums corresponding to B and C are denoted as PSBs and PSCs, respectively.

CSD Encoding: According to (13), PSBs can be calculated using a shift operation. For PSCs, we denote the divisor $T = t + (t \bmod 2)$, and convert it into Canonical Signed Digit



(a) GELU (b) nexp (c) rsqrt

Fig. 4. (a) - (c) Comparison between QUNF and prior methods. The X-axis represents Area-Delay-Product (ADP) and the Y-axis represents Mean Absolute Error (MAE). QUNF clearly demonstrates Pareto-optimal performances.

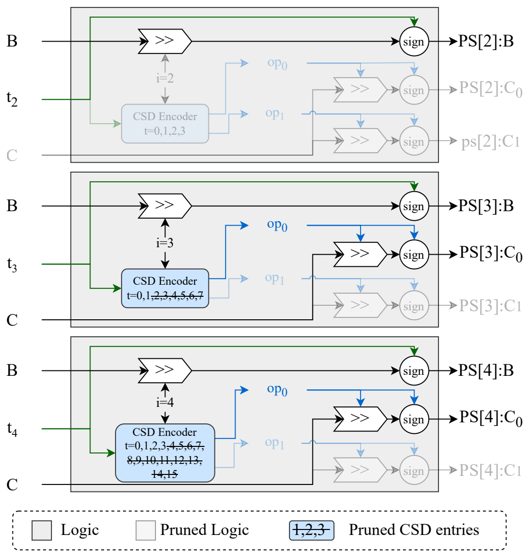


Fig. 5. Example of partial sum generation and pruning when $d = 2$ and $n = 4$.

(CSD) representation [23]. The CSD representation minimizes the non-zero terms in the binary representation, thereby reducing the numbers to accumulate. For example, $2025_d = 11111101001_b = 100000(-1)01001_{csd} = 2^{11} - 2^5 + 2^3 + 2^0$. In this way, we reduce the number of non-zero bits from 8 to 4. In our cases, it can be observed that for any $i \leqslant 4$ (*i.e.*, $t = 0 \sim 15$), we only need to add 2 items to calculate T, and 3 items for $i \leqslant 5$, *etc.*

Therefore, for any $T_{csd} = \sum_j 2^j$ where $T_{csd}[j] \neq 0$, the term $C\frac{T}{2^{2i}}$ can be calculated as follows:

$$C\frac{T}{2^{2i}} = C\frac{\sum 2^j}{2^{2i}} = \sum_j C \cdot 2^{j-2i} = \sum_j [C \gg (2i - j)] \quad (20)$$

The operator $\gg$ represents the bitwise right shift operation. When $T = 0$, this term is directly assigned a value of zero.

Segment-Aware Pruning: In Section III-B, the quantization of s assumes a natural range of $[0, 1)$. However, for a segmentation factor $d > 0$, we have $s \in [\frac{t}{2^n}, \frac{t+1}{2^n}) \subseteq [0, \frac{1}{2^d})$, implying $t \in [0, 2^{n-d})$. This segmentation mismatch enables lossless pruning of partial sum generation logic as follows:

1) PSC Pruning: For the i-th layer where $i \leqslant d$., $t_i \in [0, 2^{i-d})$ remains zero. Consequently, $C\frac{t_i}{2^{2i}}$, will be zero according to (13). Therefore, PSCs only need be computed for layer $1 + d$ to n.
2) Encoder Pruning: For layers where $i > d$, since $t_i \in [0, 2^{i-d})$, higher values never occur. This allows for the removal of redundant branches in the CSD Encoder.

TABLE I
DESIGN AND IMPLEMENTATION SETUP

Method	**gelu**	**nexp**	**rsqrt**
ReAFM [9]	-	L=2,4,6,8	/
PLAC [11]	$MAE_{target} = 0.0001$ to 0.0040	$MAE_{target} = 0.0001$ to 0.0040	/
RI-LUT [18]	SimpleGELU	entries=1,16,64,256	entries=1,16,64,256
GQA-LUT [13]	entries=7,15 (6 pretrained instances each)	entries=7,15 (6 pretrained instances each)	/
QUNF w/o H	$\rightarrow$	d=2,3,4;n=4∼8	$\leftarrow$
QUNF w/ H	d=2,3,4; n=2∼8	/	/

'-': default implementation; '/': not supported or not implemented

Fig. 5 illustrates the structure of the partial sum generation module with the pruning mechanism, which effectively eliminates redundant logic without compromising accuracy.

In this pruning design, a higher value of factor d increases the number of Q LUT entries while reducing hardware logic. Conversely, a partial sum generation module synthesized for a smaller d consumes more resources but remains compatible with coefficients generated for higher d values.

Therefore, while the overall precision level n controls the total number of partial sum layers, the factor d offers a trade-off between the resource consumption of Q LUT and the partial sum generation logic. This expansion of the design space allow developers to optimize the design based on specific applications or hardware resource constraints.

V. EXPERIMENTAL RESULTS

A. *Performance on Individual Non-Linear Functions*

To evaluate our design along with the most representative prior works, we implemented all designs for accelerating GELU, nexp, and rsqrt using Verilog (for rsqrt, only RILUT and our method are included, due to poor accuracy without dynamic range handling for others). The detailed configurations for each implementation are shown in Table I.

We synthesized these designs using Synopsys Design Compiler with the UMC 40nm technology library to obtain hardware resource utilization and performance metrics. For accuracy results, we uniformly sampled input data from the typical input range of these functions. We then measured the Mean Error (ME) and the Mean Absolute Error (MAE) for all designs.

TABLE II
COMPARISON BETWEEN QUNF AND PRIOR METHODS ON FINAL INFERENCE ACCURACY OF DIFFERENT TRANSFORMER MODELS*

Method	**Settings**	**T5-base**								**RoBERTa-large**							
		CoLA	STSB	MRPC	RTE	MNLI	QQP	QNLI	SST2	CoLA	STSB	MRPC	RTE	MNLI	QQP	QNLI	SST2
Exact	-	58.36	88.77	91.19	69.31	86.58	89.62	92.82	94.84	65.06	91.91	91.14	82.31	90.09	90.76	94.51	96.22
RI-LUT	entry-1	+0.02	**0.00**	**-0.41**	**-0.72**	**+0.08**	**+0.02**	0.00	**0.00**	**+1.05**	**-0.31**	**-0.73**	**-4.33**	**-0.49**	**-0.18**	**-0.40**	**0.00**
RI-LUT	entry-16	0.00	**0.00**	**0.00**	**0.00**	+0.02	0.00	0.00	**0.00**	**-0.37**	**+0.06**	-0.14	-3.97	-0.01	-0.03	-0.13	**-0.11**
RI-LUT	entry-64	0.00	**0.00**	**0.00**	-0.36	0.00	0.00	0.00	**0.00**	+0.18	**+0.06**	-0.14	-3.97	-0.07	-0.04	-0.09	**0.00**
QUNF	d2-n4	**+0.27**	**0.00**	-0.20	**-0.72**	+0.07	0.00	**-0.04**	**0.00**	-0.01	0.00	**+0.20**	**+1.08**	-0.07	**+0.03**	**+0.07**	**-0.11**
QUNF	d2-n6	0.00	**0.00**	-0.20	**-0.72**	**-0.01**	0.00	**+0.02**	**0.00**	+0.25	0.00	0.00	0.00	-0.01	0.00	-0.02	**0.00**
QUNF	d2-n8	0.00	**0.00**	**0.00**	-0.36	+0.01	0.00	**+0.02**	**0.00**	0.00	0.00	0.00	0.00	**+0.02**	+0.02	+0.04	**0.00**

*: Bold and red text for the best and worst accuracy in each task, respectively.

TABLE III
COMPARISON BETWEEN QUNF AND PRIOR METHODS ON GELU

Method	**Settings**	**Area** (μm^2)	**Power** (mW)	**Delay** (ns)	**ME**	**MAE**
RILUT	SimpleGELU	15856	4.10	26.23	3.03e-2	3.61e-2
GQALUT	e15,inst5	8556	1.09	14.70	1.64e-3	3.04e-3
ReAFM	L4	33066	4.44	41.34	5.23e-3	1.23e-2
PLAC	e20	10699	4.65	20.88	-1.12e-4	7.16e-4
PLAC	e34	11274	4.61	20.86	-3.92e-5	2.34e-4
PLAC	e48	11598	4.65	20.89	-2.10e-5	1.18e-4
QUNF w/o H	d2,n4	8091	1.33	10.92	4.82e-5	8.48e-3
QUNF w/o H	d2,n6	10770	1.80	12.26	2.78e-5	2.12e-3
QUNF w/o H	d2,n8	15346	2.61	12.67	8.22e-6	5.40e-4
QUNF w/ H	d2,n4	**7558**	**1.02**	**10.64**	-8.42e-5	1.33e-3
QUNF w/ H	d2,n6	10871	1.52	11.80	-6.34e-6	3.36e-4
QUNF w/ H	d2,n8	14492	2.13	13.17	**-1.99e-6**	**9.16e-5**

According to Fig. 4a, ReAFM and SimpleGELU from RI-LUT demonstrate worse performance than others. They calculate GELU via Sigmoid, introducing intrinsic error and also more complicated calculation logic. The PWL-based methods, *i.e.,* GQA-LUT and PLAC have better accuracy and resource utilization than those with dedicated datapath. From Fig. 4a, 4b and 4c, the proposed method outperforms any prior work on both accuracy and resource utilization for these operations. In several rare cases, RI-LUT and PLAC can achieve higher precision in exchange for higher resource utilization. This trend is further demonstrated in Table III.

For design parameter choices within QUNF, changing the precision level parameter n offers a higher design space in trading off accuracy against hardware efficiency. The optimal selection of segmentation factor d differs in different n values, where smaller n prefers smaller d, and vice versa. This trend validated the effectiveness of the pruning mechanism.

B. Transformer Inference

To demonstrate the effectiveness of QUNF, we evaluated the accuracy of our approximation in real-world transformer models. We selected T5-base [24] and RoBERTa-large [25] models and fine-tuned them on tasks from the GLUE dataset [26]. The performance of fine-tuned models for each task without approximation is shown in the first row of Table II as “Exact”. We employed the software implementation of our method using PyTorch to approximate the NFs in GELU, Softmax, and LayerNorm. We then replaced the original functions in the models with them and re-executed the inference on the tasks. Finally, we calculated the difference in accuracy for each task compared to the baseline. This process was repeated using RI-LUT for comparison. The 1-entry SimpleGELU implementation is used for all RI-LUT settings.

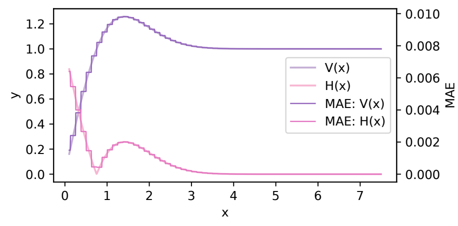


Fig. 6. MAE in each segment of approximation using function H and V, compared to $H'(x)$ and $V'(x)$. The first derivative of the target function is a precise indicator of its approximation error.

The result is displayed in Table II. For most tasks, our method achieved minor degradation in accuracy or even positive impact. The accuracy degradation on the T5-base and RoBERTa-large model is within 0.72%. RI-LUT also achieved good results on the T5-base model but introduced more errors for larger and more accurate ones. Results from separate tests demonstrate that the majority of the error originates from SimpleGELU, further highlighting the superiority of our unique GELU implementation.

C. Validation of Error Optimization

We measured the MAE of approximating $V(x)$ and $H(x)$ in each segment, observing that the error using $H(x)$ decreases as expected, as shown in Fig. 6. Table III further outlines the differences between the implementation of the proposed method directly on GELU or using the auxiliary function $H(x)$. It can be observed that the version using $H(x)$ achieved better accuracy while also slightly reducing resources due to the exploitation of the symmetry.

Notably, this optimization is not limited to the proposed approximation method and is generally effective in most curve-fitting-based approximation techniques. When applied to PLAC, it significantly improves the accuracy of GELU approximation, reducing the error measured in MAE by 70%∼80%, while maintaining the same number of LUT entries.

VI. CONCLUSIONS

In this work, we propose a novel unified approximation module for non-linear functions in transformer models. Through quadratic approximation and a customized hardware structure, the proposed module can approximate all unary non-linear functions, including GELU, nexp, and rsqrt. The experimental results show that the proposed method achieved better accuracy and hardware efficiency than prior works in both standalone circuit evaluations and real transformer model inferences.

ACKNOWLEDGMENT

This work was supported by the NSFC (Grant W2412034).

References

[1] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 9992–10 002.

[2] M. Wang, Z. Xu, B. Zheng, and W. Xie, "BinaryFormer: A Hierarchical-Adaptive Binary Vision Transformer (ViT) for Efficient Computing," *IEEE Transactions on Industrial Informatics*, pp. 1–12, 2024.

[3] L. Chen, Y. Wu, C. Wen, S. Wang, and L. Zhang, "An Agile Framework for Efficient LLM Accelerator Development and Model Inference," in *2024 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, Oct. 2024.

[4] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-BERT: Integer-only BERT Quantization," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021, pp. 5506–5518.

[5] M. Xia, Z. Zhong, and D. Chen, "Structured Pruning Learns Compact and Accurate Models," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1513–1528.

[6] P. Ganesh, Y. Chen, X. Lou, M. A. Khan, Y. Yang, H. Sajjad, P. Nakov, D. Chen, and M. Winslett, "Compressing Large-Scale Transformer-Based Models: A Case Study on BERT," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 1061–1080, Sep. 2021.

[7] K. T. Chitty-Venkata, S. Mittal, M. Emani, V. Vishwanath, and A. K. Somani, "A survey of techniques for optimizing transformer inference," *Journal of Systems Architecture*, vol. 144, p. 102990, Nov. 2023.

[8] W. Wang, S. Zhou, W. Sun, P. Sun, and Y. Liu, "SOLE: Hardware-Software Co-design of Softmax and LayerNorm for Efficient Transformer Inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, Oct. 2023, pp. 1–9.

[9] X. Wu, S. Liang, M. Wang, and Z. Wang, "ReAFM: A Reconfigurable Nonlinear Activation Function Module for Neural Networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 7, pp. 2660–2664, Jul. 2023.

[10] H. Sun, Y. Luo, Y. Ha, Y. Shi, Y. Gao, Q. Shen, and H. Pan, "A Universal Method of Linear Approximation With Controllable Error for the Efficient Implementation of Transcendental Functions," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 1, pp. 177–188, Jan. 2020.

[11] H. Dong, M. Wang, Y. Luo, M. Zheng, M. An, Y. Ha, and H. Pan, "PLAC: Piecewise Linear Approximation Computation for All Nonlinear Unary Functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, pp. 2014–2027, Sep. 2020.

[12] J. Yu, J. Park, S. Park, M. Kim, S. Lee, D. H. Lee, and J. Choi, "NN-LUT: Neural approximation of non-linear operations for efficient transformer inference," in *2022 59th ACM/IEEE Design Automation Conference (DAC)*. New York, NY, USA: Association for Computing Machinery, Aug. 2022, pp. 577–582.

[13] P. Dong, Y. Tan, D. Zhang, T. Ni, X. Liu, Y. Liu, P. Luo, L. Liang, S.-Y. Liu, X. Huang, H. Zhu, Y. Pan, F. An, and K.-T. Cheng, "Genetic Quantization-Aware Approximation for Non-Linear Operations in Transformers," in *2024 61th ACM/IEEE Design Automation Conference (DAC)*, 2024.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 6000–6010.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[16] 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)*, Dec. 2021, pp. 469–474.

[17] C. Wen, Y. Wu, X. Yin, and C. Zhuo, "Approximate Floating-Point FFT Design with Wide Precision-Range and High Energy Efficiency," in *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2023, pp. 134–139.

[18] J. Kim, J. Lee, J. Choi, J. Han, and S. Lee, "Range-Invariant Approximation of Non-Linear Operations for Efficient BERT Fine-Tuning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, Jul. 2023, pp. 1–6.

[19] C. Wen, H. Du, J. Wang, Z. Chen, L. Zhang, Q. Sun, and C. Zhuo, "PACE: A Piece-Wise Approximate Floating-Point Divider with Runtime Configurability and High Energy Efficiency," *ACM Trans. Des. Autom. Electron. Syst.*, Dec. 2024.

[20] Y. Wu, C. Chen, W. Xiao, X. Wang, C. Wen, J. Han, X. Yin, W. Qian, and C. Zhuo, "A Survey on Approximate Multiplier Designs for Energy Efficiency: From Algorithms to Circuits," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 29, no. 1, pp. 23:1–23:37, Jan. 2024.

[21] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.

[22] C. Chen, W. Qian, M. Imani, X. Yin, and C. Zhuo, "PAM: A Piecewise-Linearly-Approximated Floating-Point Multiplier With Unbiasedness and Configurability," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2473–2486, Oct. 2022.

[23] A. Avizienis, "Signed-Digit Numbe Representations for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 389–400, Sep. 1961.

[24] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[25] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," Jul. 2019, arXiv:1907.11692.

[26] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding," Feb. 2019, arXiv:1804.07461.