SoftAct: A High-Precision Softmax Architecture for Transformers Supporting Nonlinear Functions

Journal:	*IEEE Transactions on Circuits and Systems for Video Technology*
Manuscript ID	TCSVT-16113-2023
Manuscript Type:	Transactions Papers - Regular Issue
Date Submitted by the Author:	09-Nov-2023
Complete List of Authors:	Jiao, Hailong; Peking University Shenzhen Graduate School Fu, Yuzhe; Peking University Shenzhen Graduate School Zhou, Changchun; Peking University Shenzhen Graduate School Huang, Tianling; Peking University Shenzhen Graduate School Han, Eryi; Reconova Technologies Co Ltd He, Yifan; Reconova Technologies Co Ltd
EDICS:	6.1.9 ☐ Complexity/Power-Constrained Image/Video Processing < 6.1 ☐ Image/Video System Architectures < 6 ☐ IMAGE/VIDEO HARDWARE/SOFTWARE SYSTEMS, Low-Complexity Processing in Devices and Sensors < 6.1.9 ☐ Complexity/Power-Constrained Image/Video Processing < 6.1 ☐ Image/Video System Architectures < 6 ☐ IMAGE/VIDEO HARDWARE/SOFTWARE SYSTEMS, 6.2 ☐ Image/Video Circuits and Architectures < 6 ☐ IMAGE/VIDEO HARDWARE/SOFTWARE SYSTEMS, 6.2.2 ☐ VLSI Architectures and Implementations < 6.2 ☐ Image/Video Circuits and Architectures < 6 ☐ IMAGE/VIDEO HARDWARE/SOFTWARE SYSTEMS, 6.2.3 ☐ Low-Power Circuits and Architectures < 6.2 ☐ Image/Video Circuits and Architectures < 6 ☐ IMAGE/VIDEO HARDWARE/SOFTWARE SYSTEMS

SoftAct: A High-Precision Softmax Architecture for Transformers Supporting Nonlinear Functions

Yuzhe Fu, Changchun Zhou, Tianling Huang, Eryi Han, Yifan He, *Member, IEEE*, and Hailong Jiao, *Member, IEEE*

***Abstract*—Transformer-based deep learning networks are revolutionizing our society. The convolution and attention co-designed (CAC) Transformers have demonstrated superior performance compared to the conventional Transformer-based networks. However, CAC Transformer networks contain various nonlinear functions, such as softmax and complex activation functions, which require high precision hardware design yet typically with significant cost in area and power consumption. To address these challenges, *SoftAct*, a compact and high-precision algorithm-hardware co-designed architecture, is proposed to implement both softmax and nonlinear activation functions in CAC Transformer accelerators. An improved softmax algorithm with penalties is proposed to maintain precision in hardware. A stage-wise full zero detection method is developed to skip redundant computation in softmax. A compact and reconfigurable architecture with a symmetrically designed linear fitting module is proposed to achieve nonlinear functions. The *SoftAct* architecture is designed in an industrial 28-nm CMOS technology with the MobileViT-xxs network as the benchmark. Compared with the state of the art, *SoftAct* improves up to 5.87% network accuracy, 153.2× area efficiency, and 1435× overall efficiency.**

***Index Terms*—Transformer-based networks, nonlinear functions, softmax, sparsity detection, overall efficiency.**

I. Introduction

TRANSFORMER-BASED models have demonstrated remarkable success in a range of artificial intelligence (AI) tasks, surpassing recurrent neural networks (RNNs) and convolutional neural networks (CNNs) in various domains, such as natural language processing and computer vision [1], [2], [3], [4], [5], [6]. The conventional Transformer networks rely on the attention mechanism and are typically characterized by enormous model size and high computational complexity (e.g., 307 M parameters and 190.7 G FLOPs in ViT-L [2]), making them infeasible for implementation on resource-constrained devices. A novel approach that combines attention with convolution has emerged in the field of lightweight Transformer networks [7], [8], [9], [10], [11], [12]. This convolution and attention co-designed (CAC) Transformer networks have the potential to significantly reduce the number of network parameters and computational complexity (e.g., 133.5× parameters and 272× FLOPs reductions in MobileViT [12] compared with ViT-L) while maintaining high accuracy, leading to better chance of deployment on edge devices. Unfortunately, these networks utilize a great number of nonlinear functions. These nonlinear functions have complex mathematical forms and are unfriendly for hardware implementation. For instance, every attention layer in the network includes a softmax operation. The convolution layers also have other nonlinear activation functions, such as Swish in MobileViT and both Swish and GeLU in Conformer [9]. In the inference stage, the delay of the nonlinear functions becomes significant due to the high data access to DRAM and low data reuse rate in hardware. For example, the softmax delay in GPT-2 accounts for ~33% of the overall network delay [13]. Processing those nonlinear functions therefore becomes a bottleneck when applying hardware acceleration to the CAC Transformer networks.

The hardware implementation of nonlinear functions is complex and challenging, especially for the softmax function. This is because softmax involves both exponential and division operations. Existing approaches do not utilize reconfigurable architecture, but instead rely on designing individual circuits to carry out these operations, which usually cost expensive area and high power consumption to support high precision in hardware implementation. Some works [14], [15] directly use CPU to implement nonlinear functions, which introduces significant delays in data communication. For the existing Transformer hardware architecture, Ham [16], Rizk [17], and Li [18] utilize the look-up table (LUT) or piecewise linear fitting (PLF) to implement the exponential function and rely on an area-expensive division unit to perform division operations. In [13], the softmax function is directly implemented based on floating-point (FP) computing units to provide high precision, yet also with serious area and power cost. A fixed-point softmax unit that utilizes LUT to implement the exponential function and division is proposed in [19], which incorporates a sparsity detection algorithm to reduce the number of softmax operations. However, the reconfigurability of the hardware is low, and the algorithm may not fully utilize the sparsity provided by the softmax function.

The aforementioned works are all based on the original softmax function, with no significant innovation in the hardware architecture. Some other researchers have modified the softmax function and proposed novel hardware architectures. The log-sum-exp softmax is proposed in [20] to avoid the complicated division operation through logarithmic conversion, yet still with high hardware cost. A constant multiplier strategy is proposed in [21] to calculate the exp-log-sum function within adjustable precision based on [20].

This work was supported in part by the National Natural Science Foundation of China 62074005. (*Corresponding authors: Hailong Jiao and Yifan He*.)

Yuzhe Fu, Changchun Zhou, Tianling Huang, and Hailong Jiao are with the School of Electronic and Computer Engineering, Shenzhen Graduate School, Peking University, Shenzhen, 518055, China (e-mail: jiaohailong@pku.edu.cn). Eryi Han and Yifan He are with Reconova Technologies Co., Ltd., Xiamen, 361015, China.

However, due to the approximate implementation method, there is still considerable room for improvement in terms of precision. Other approximate strategies [22], [23] are also proposed, such as the Coordinated Rotation Digital Computer (CORDIC) algorithm and Maclaurin series. However, sacrificing precision for better hardware performance can result in a significant reduction in network inference accuracy, which may not be acceptable for a variety of applications. Some works [24], [25] simplify the mathematical mechanism of softmax from base-e to base-2 to reduce the complexity of hardware design. However, mainstream Transformer-based networks do not utilize the base-2 softmax method.

Therefore, there are three issues with the existing softmax works: low precision in hardware implementation, high power consumption, and high area cost. To address these issues, we propose *SoftAct*, an algorithm and hardware co-designed architecture enabling efficient processing of softmax and nonlinear activation functions in network inference. To maintain the precision, we propose the improved softmax with penalties algorithm, which is based on the log-sum-exp format and includes penalty terms in linear fitting to mitigate the error caused by approximate calculations. To reduce the power consumption, we introduce a stage-wise full zero detection algorithm to avoid redundant operations in softmax with no loss in precision. Furthermore, to save the hardware area, we propose a reconfigurable hardware architecture to support both softmax and nonlinear activation functions. To verify the feasibility of our algorithm in the Transformer-based network, we use the MobileViT-xxs network, which contains both softmax and Swish activation functions, running with ImageNet-1k as the benchmark. The proposed *SoftAct* is designed and simulated by using an industrial 28-nm CMOS technology. Compared with the state of the art, the proposed *SoftAct* achieves up to 5.87% higher network accuracy, 153.2× higher area efficiency, and 1435× higher overall efficiency.

The rest of this paper is organized as follows. A brief review of the background and related works is given in Section II. The proposed algorithmic optimizations are presented in Section III. In Section IV, the proposed *SoftAct* hardware architecture is detailed. The experimental results of the proposed *SoftAct* are shown in Section V. This paper is summarized in Section VI.

II. Background and Related Works

In this section, the fundamental concepts of CAC Transformer networks, softmax function, and Swish function are provided. The existing hardware implementations of softmax and Swish functions are also introduced.

A. CAC Transformer Backbone and Self-Attention Mechanism

The recently emerged CAC Transformer models have made a breakthrough in the field of lightweight Transformer-based networks. The classical backbone of the CAC Transformer model is shown in Fig. 1(a), which is composed of a series of convolution and attention layers. The input data first passes through convolution layers to extract features, which are utilized as the input to the attention layers. In MobileViT [12], the nonlinear functions play critical roles. The softmax function is used in every attention layer. The Swish activation function is utilized in every convolution layer and the multi-layer perceptron (MLP) layer that follows the attention layer. The details of the attention layer with inputs Q, K, and V are shown in Fig. 1(b). $Q \times K^T$ is firstly calculated and followed by softmax with quantization to generate P_q. Then $P_q \times V$ is used to get the output data. The softmax function normalizes P to a range of (0, 1). Transformer networks typically employ 12-bit quantization to maintain network accuracy. Lower-bit quantization (such as 8-bit) tends to cause an accuracy loss (>1%) [13], [19]. After quantization, the near-zero values after softmax are rounded to 0, which leads to sparsity. Based on our statistical analysis of MobileViT-xxs, softmax generates 19.63% sparsity under 12b quantization, and 67.40% under 8b quantization, which can be explored for acceleration and power savings in hardware implementation.

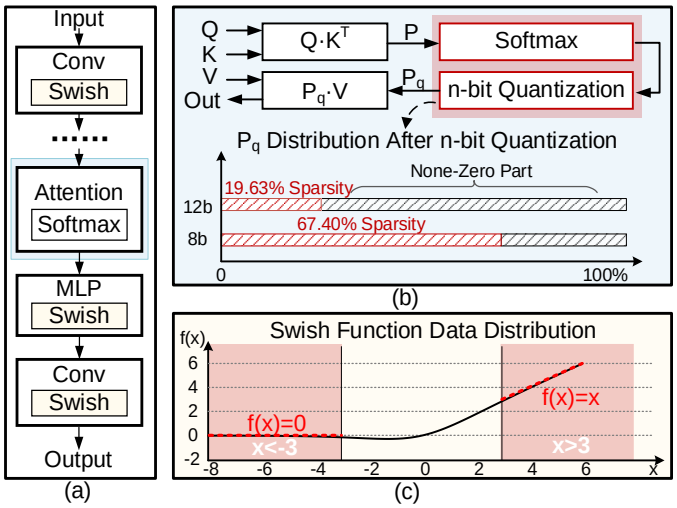


Fig. 1. (a) The backbone of CAC Transformer. (b) The mechanism of self-attention. (c) The curve of the Swish activation function.

The mathematical form of the Swish function is expressed in (1), with a wide input range and complex mathematical form. The existing works either use high resource cost and power consumption to meet the accuracy requirements of the Swish function [26], or sacrifice precision by simplifying calculations, in exchange for lower hardware complexity [27].

$$Swish(x) = x \cdot sigmod(x) = \frac{x \cdot e^x}{1+e^x}. \quad (1)$$

As shown in Fig. 1, when x is smaller than -3, the Swish output approaches 0. When x is greater than 3, the Swish output approaches x. By utilizing this observation, the complexity of the Swish function can be significantly reduced, which could be preferable for hardware implementation.

B. The Original Softmax Function

The softmax function is commonly utilized as a normalization function in deep neural networks. The input to the softmax layer is an N-sized vector $X = [x_0, x_1, \dots, x_{N-1}]$. The mathematical form of the softmax function is expressed in (2). The softmax function is invariant to the subtraction operation with x_i. Therefore, the value of x_i is often subtracted

by x_{max}, which is the maximum value in X, to reduce the computational complexity.

$$f(x_i) = \frac{exp(x_i)}{\sum_{j=0}^{N-1} exp(x_j)} = \frac{exp(x_i - x_{max})}{\sum_{j=0}^{N-1} exp(x_j - x_{max})}. \quad (2)$$

[13], [16], [17], [18], [19] adopt (2) and use a two-stage hardware architecture that typically comprises an exponential unit and a division unit. However, these works can only support the softmax function with high hardware cost. Take [19] as an example, $exp(x_i - x_{max})$ and $1/\sum_{j=0}^{N-1} exp(x_j - x_{max})$ are implemented based on 16-segment PLF. These two terms are multiplied to obtain the result of the softmax function. However, the input range of this method is not constrained. A large number of segments for PLF is therefore required to ensure high precision, thereby resulting in high hardware area cost.

C. Log-Sum-Exp Softmax Function

The log-sum-exp softmax is a simplified softmax algorithm which is first proposed in [20] and expressed as

$$\begin{aligned} f(x_i) &= \frac{exp(x_i - x_{max})}{exp(ln(\sum_{j=0}^{N-1} exp(x_j - x_{max})))} \\ &= exp(x_i - x_{max} - ln(\textstyle\sum_{j=0}^{N-1} exp(x_j - x_{max}))). \end{aligned} \quad (3)$$

The main idea is to eliminate the division by incorporating the logarithm function and reusing the exponential function, which is hardware-friendly. Various methods have been proposed for implementing the exponential and logarithmic functions in hardware, such as CORDIC implementation [22] and linear approximation [28]. However, most of these methods face challenges such as large area, low operating frequency, or precision loss, and therefore cannot achieve a good balance between energy efficiency and precision.

A hardware-friendly implementation of exponential and logarithmic functions is proposed in [21], which converts them into base-2 expressions through constant multiplication. The mathematical transformation for the exponential function is

$$exp(-x_i) = 2^{-x_i \cdot log_2(e)}, x_i \geq 0. \quad (4)$$

By dividing the exponent term $x_i \cdot \log_2(e)$ into two parts, an integral part n_i and a fractional part f_i, (4) can be further simplified as

$$\begin{aligned} exp(-x_i) &= 2^{-(n_i + f_i)} = 2^{-f_i} \cdot 2^{-n_i} \\ &= 2^{-f_i} \gg n_i, f_i \in [0, 1). \end{aligned} \quad (5)$$

The exponential function is thereby converted to a constant multiplication, 2^{-f_i}, and a shift operation, which are easy for hardware implementation.

The mathematical transformation for logarithmic function is

$$ln(S) = ln(2) \cdot log_2(S), \quad (6)$$

where $S = \sum_{i=0}^{N-1} exp(x_i - x_{max})$ and $exp(x_i - x_{max}) \in (0, 1]$. Suppose $S = u \cdot 2^v$, where u is a real number in [1, 2) and v is an integer. Then the logarithmic function can be simplified as

$$ln(S) = ln(2) \cdot log_2(u \cdot 2^v) \quad (7)$$

By utilizing $\log_2(u) \approx u - 1$, $u \in [1, 2)$, (7) is further simplified as

$$ln(S) = ln(2) \cdot (v + u - 1) \quad (8)$$

(8) can reduce the hardware complexity but introduce ~14.78% average error in the implementation of $log_2(u)$, which negatively impacts the softmax precision. The constant multiplication in hardware, which is based on approximate calculations, further degrades the softmax precision. Therefore, there is still a large room for performance improvement in [21].

The accuracy of Transformer-based models is highly sensitive to nonlinear functions, thereby requiring high-precision hardware implementations [29]. However, most of the existing works compromise the precision of softmax for higher frequency and lower hardware complexity, while not supporting any other functions. Therefore, we propose an algorithm-hardware codesigned *SoftAct* architecture to address the above challenges. In terms of algorithm, we propose an improved softmax with penalties algorithm to maintain the softmax precision in hardware implementation. Furthermore, we introduce a stage-wise full zero detection algorithm to exploit the sparsity caused by quantization of softmax to avoid redundant operations. In terms of hardware, we design a compact and reconfigurable hardware architecture, which supports various nonlinear functions with low hardware cost. While many existing works verify their approaches using simple CNNs, these methods often fail on Transformer-based networks. We employ the MobileViT-xxs network as our benchmark to affirm the feasibility of our proposed methods in Transformer-based networks.

III. Improved Algorithms for Softmax and Swish

In this section, the software innovations of the proposed *SoftAct* are introduced. Three algorithmic optimization techniques are proposed to improve the precision of the softmax function and enable the reconfigurable hardware design to support the Swish function. Firstly, to overcome the precision degradation caused by the approximate calculations in hardware, an improved softmax function with penalties is proposed. Secondly, a stage-wise full zero detection algorithm is introduced to eliminate redundant calculations during softmax operations. Thirdly, to support various nonlinear functions, a unified computation framework that incorporates a low-bit-width Swish is proposed.

A. Improved Softmax with Penalties (ISP)

The high-precision hardware implementation of softmax encounters two main challenges. Firstly, the input range of exponential function and division in traditional softmax is large, posing challenges for accurate calculation even with a large-segment PLF. Secondly, although the existing log-sum-exp softmax algorithm saves division and constrains the input range, the hardware implementation introduces errors caused by approximate constant multiplications. Simply increasing the bit width of the system results in a small accuracy gain compared to the caused significant bandwidth and area losses to the hardware. For instance, increasing the bit width from 16b to 32b can reduce the mean average error (MAE) of softmax by 20%, but with a 61% area overhead. Therefore, we propose an

improved softmax with penalties (*ISP*), which is based on the log-sum-exp softmax to constrain the input range. Furthermore, we introduce penalty terms by incorporating the error terms caused by the approximate constant calculation into (4) and (7). These penalty terms are then used to correct the coefficients of PLF, thus reducing the MAE by up to 99.29% in 16b as shown in Section V-A with nearly no area overhead. The details of *ISP* are shown below.

The original softmax is firstly transformed into (3), with two exponential functions and one logarithmic function. When implementing the exponential function, the error is defined as

$$err_0 = log_2(e) - log_2(e)_h, \tag{9}$$

where $log_2(e)$ is the full precision value and $log_2(e)_h$ is the approximate value obtained by the hardware. By introducing err_0 into (4), we can obtain

$$\begin{aligned} exp(-x_i) &= 2^{-x_i\cdot(log_2(e)_h+err_0)} \\ &= 2^{-x_i\cdot err_0} \times 2^{-x_i\cdot log_2(e)_h}. \end{aligned} \tag{10}$$

Suppose $-x_i\, log_2(e)_h = -(n_i + f_i)$, where n_i is the integer part and f_i is the fractional part. The exponential calculation can be converted as

$$\begin{aligned} exp(-x_i) &= 2^{\frac{-err_0\cdot(n_i+f_i)}{log_2(e)_h}} \times 2^{-(n_i+f_i)} \\ &= 2^{-n_i\cdot\frac{err_0}{log_2(e)_h}} \times 2^{-f_i\cdot\left(1+\frac{err_0}{log_2(e)_h}\right)} \gg n_i, \end{aligned} \tag{11}$$

where $\gg$ is a right shift operation. There are two error terms in (11), where $2^{-f_i\cdot\left(\frac{err_0}{log_2(e)_h}\right)}$ represents the effect of err_0 on the fractional part, and $2^{-n_i\cdot\frac{err_0}{log_2(e)_h}}$ reflects the impact of err_0 on the integer part. Incorporating these error terms into the PLF can complete the correction process. Suppose the internal penalty $P_{in} = \frac{err_0}{log_2(e)_h}$ and overall penalty $P_{ov} = 2^{-n_i\frac{err_0}{log_2(e)_h}}$ as shown in (12). Note that P_{in} is determined by $log_2(e)_h$ and P_{ov} is determined by P_{in} and n_i. We introduce an external coefficient p_0 to replace n_i, thereby achieving the idea of manually customizing the penalty strength of P_{ov} by adjusting p_0. Therefore, $P_{ov} = 2^{-p_0\cdot P_{in}}$. The optimal value for p_0 can be iteratively determined through software-based statistical analysis of input data. Then, by performing PLF on $2^{-f_i\cdot(1+P_{in})}$, we can get

$$\begin{aligned} exp_{p_0}(-x_i) &= P_{ov} \times 2^{-f_i\cdot(1+P_{in})} \gg n_i \\ &= P_{ov} \times (k\cdot f_i + b) \gg n_i, \end{aligned} \tag{12}$$

where *k* and *b* represent the coefficients after linear fitting.

The high-precision logarithmic function is also implemented based on PLF, similar to the implementation of the exponential function. Firstly, the error err_1 is defined as

$$err_1 = ln(2) - ln(2)_h, \tag{13}$$

where $ln(2)$ represents the full-precision value and $ln(2)_h$ represents the approximate value used in hardware. After introducing err_1 into (7),

$$\begin{aligned} ln(S) &= (ln(2)_h + err_1)\cdot log_2(u\cdot 2^v) \\ &= (ln(2)_h + err_1)\cdot v + ln(2)\cdot log_2(u), \end{aligned} \tag{14}$$

where u and v are explained in Section II-C. By performing PLF on $log_2(u)$, we obtain

$$ln_{p_1}(S) = ln(2)_h\cdot v + ln(2)\cdot(k'\cdot u + b') + P_{ln}, \tag{15}$$

where the logarithmic penalty term $P_{ln} = err_1\cdot v$. The coefficients k' and b' are obtained by linear fitting. err_1 is determined by $ln(2)_h$. We introduce another external coefficient p_1 to replace v and to manually adjust the penalty strength of P_{ln}. p_1 is determined by analyzing the range of input data. Therefore, $P_{ln} = err_1\cdot p_1$, which can be determined before hardware implementation. To sum up, the final expression of *ISP* is

$$f_{p_0p_1}(x_i) = exp_{p_0}(x'_i - ln_{p_1}(\textstyle\sum_{j=0}^{N-1} exp_{p_0}(x'_j))), \tag{16}$$

where $x'_i = x_i - x_{max}$.

For the determination of p_0 and p_1, we have made the following optimizations. If the input data range is fixed, the optimal values of p_0 and p_1 can be determined by an iterative algorithm. However, for real network data, the data ranges can vary significantly among different layers. Frequent changes in the values of p_0 and p_1 lead to changes in the linear fitting coefficients, which in turn require frequent updating in hardware and ultimately harm the latency and power consumption of the system. To solve this issue, we propose a constant-adaptive penalty algorithm. The constant penalty is applied to (12). P_{in} is obtained by linear fitting with no impact on the latency and power consumption of the hardware, while P_{ov} is determined by p_0. Therefore, we set p_0 as a constant number to fix *k* and *b* of the exponential function. For (15), P_{ln} is determined by p_1 and can be implemented through constant addition in hardware. We apply (17) as the adaptive penalty on p_1. th is the threshold value determined by the network data.

$$p_1 = \begin{cases} 0, & if\ v < th \\ th, & if\ v \geq th \end{cases}. \tag{17}$$

By analyzing the real network data, it is found that the range of v is [0, 6]. Therefore, we choose $th = 3$ in this case. By using the fixed and adaptive penalty algorithm, reloading the linear fitting coefficients of the softmax function is unnecessary when the input ranges vary, and the precision of the function is still maintained.

Compared with the original function (2), we use the log-exp-sum algorithm to avoid the division operation and constrain the input range of linear fitting. Furthermore, we introduce penalty terms to mitigate the errors caused by approximate constant multiplication, and external variables p_0 and p_1 to control the strength of the penalty terms, which are used for correcting linear fitting coefficients. p_0 and p_1 are determined by an iterative algorithm for a fixed input range, or by the constant-adaptive penalty algorithm for a variable network data range. Therefore, the corrected coefficients, which are $P_{ov}\cdot(k, b)$ in (12) and $ln(2)\cdot(k', b')$ in (15), are fixed before hardware implementation. *ISP* does not affect the hardware performance while improving precision.

B. Stage-wise Full Zero Detection (SFZD)

Global attention contains massive small attention scores [13], which are $Q\times K^T$ in Fig. 1(b). The softmax layer normalizes

these scores to clarify the relevance of inputs, which exponentially reduces the small attention score to a near-zero number. After quantization, these near-zero numbers are rounded to 0, resulting in a sparse softmax. Skipping these zeros causes no accuracy loss while saving power consumption. We propose a stage-wise full zero detection (*SFZD*) algorithm based on log-sum-exp softmax to fully skip zeros in softmax.

The conventional n-bit quantization method, as shown in Fig. 2, first performs a range check on the input P to get its max and min values. Then P is mapped to the quantization domain to obtain P_Q, which is 0 when

$$\frac{P-min}{max-min} < 2^{-n}. \quad (18)$$

P is the output of the softmax, whose range is [0, 1). Therefore, we can directly set $max = 1$ and $min = 0$ in (18). Substituting (3) into (18) with further simplification, we can obtain the core idea of *SFZD*: after n-bit quantization, if $exp(in) < 2^{-n}$, where in is the input value of the exponential function, the output of softmax is 0. The *SFZD* contains two-stage detection (Stage1 and Stage3 shown in Fig. 2) in the log-sum-exp softmax to skip redundant computation. In Stage1, *SFZD* is applied before $e^{x_i - x_{max}}$ calculation as shown in (19). In Stage3, (20) is used to detect the sparsity of $exp(x_i - x_{max} - ln(S))$.

$$exp(x_i - x_{max}) < 2^{-n} \Rightarrow x_i - x_{max} < -n \cdot ln(2). \quad (19)$$

$$x_i - x_{max} - ln(S) < -n \cdot ln(2). \quad (20)$$

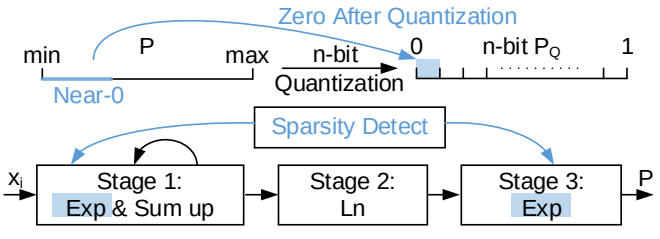


Fig. 2. The mechanisms of quantization and stage-wise full zero detection.

After evaluation in Section V-B, our proposed *SFZD* can fully utilize the sparsity provided by softmax to skip the redundant computations. Note that *SFZD* does not create new sparsity but rather leverages the existing sparsity, thereby not affecting the network accuracy.

C. Unified Computation Framework

The Swish activation function is shown in (1), which includes both multiplication and sigmoid calculations. Due to the complex mathematical expression, the Swish function is difficult to be directly implemented in hardware. To solve this issue, we propose a low-bit width Swish (*LBS*), which replaces the Swish function with (21) to constrain the input data range and reduce the bit width requirement based on a hardware-friendly Swish (hswish) [30].

$$LBS = \begin{cases} 0, & x \in (-\infty, -3) \\ PLF(x^2/6 + x/2), & x \in [-3, 3] \\ x, & x \in (3, \infty) \end{cases}. \quad (21)$$

For *LBS*, the range of input x is first checked. If $x < -3$ or $x > 3$, no further calculation is needed. Therefore, *LBS* reduces the calculation range from $(-\infty, \infty)$ of the original Swish function to [-3, 3] and achieves the low-bit width feature. When $-3 \le x \le 3$, a PLF is used to calculate $x^2/6 + x/2$. As the number of segments increases, the precision increases but with more hardware cost.

We substitute *LBS* with different PLF configurations into the network for training and finally determine that 8-segment PLF for *LBS*, which is shown in Fig. 3, can guarantee network accuracy (<0.5% degradation).

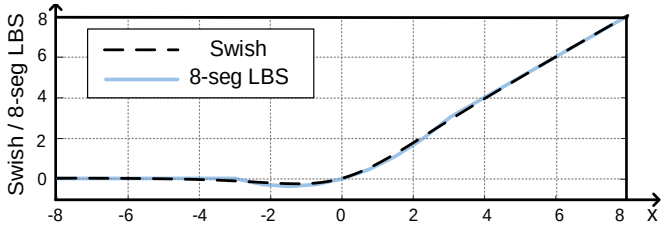


Fig. 3. Comparison between Swish and our 8-segment *LBS*.

To accommodate both the softmax function and other nonlinear functions, we propose a unified computation framework for *ISP* and *LBS* in Algorithm 1. The method first configures the inner data flows according to the input *Mode*. Before the calculation, the sparsity is firstly calculated, which are Line 5 and Line 12 for *ISP* based on the *SFZD*, and Line 18 for *LBS* in Algorithm 1, to skip the redundant operations. The computation process for *ISP* involves three stages.

- Stage1: Detect sparsity, calculate $P_{ov} \times 2^{-f_i \cdot (1+P_{in})}$ in (12) and sum up $exp_{p_0}(x_i')$, which are from Line 2 to Line 8.
- Stage2: Calculate $ln_{p_1}(S)$, which is Line 9.
- Stage3: Repeat Stage1 except summing up $exp_{p_0}(x_i')$, which are Line 10 to Line 15.

One stage is needed for *LBS*:

- Stage4: Check the input range and get the corresponding output. If the input is in [-3, 3], then calculate $x^2/6 + x/2$ in (21), which are Line 17 to Line 21.

Algorithm 1 Unified Computation Framework

Input: $Mode$, $X = [x_0, x_1, ..., x_{N-1}]$, x_{max}
Output: $Y = [y_0, y_1, ..., y_{N-1}]$

1: **if** $Mode = ISP$ **then**
2: $S = 0$
3: **for** $i = 0$ to $N - 1$ **do**
4: $x_i' = x_i - x_{max}$
5: **if** $x_i' < -n \cdot ln(2)$ **then**
6: Skip sparsity
7: **else**
8: $S = S + exp_{p_0}(Mode, x_i')$
9: $lnOut = ln_{p_1}(Mode, S)$
10: **for** $i = 0$ to $N - 1$ **do**
11: $x_i'' = x_i' - lnOut$
12: **if** $x_i'' < -n \cdot ln(2)$ **then**
13: Skip sparsity
14: **else**
15: $y_i = exp_{p_0}(Mode, x_i'')$
16: **else** # $Mode = LBS$
17: **for** $i = 0$ to $N - 1$ **do**
18: **if** $x_i < -3$ or $x_i > 3$ **then**
19: $y_i = 0$ or $y_i = x_i$
20: **else**
21: $y_i =$*linearfit*$(Mode, x_i)$
22: **return** Y

Algorithm 2 Calculation for Exponential Function

Input: $Mode$, x_i
Output: $exp_{p_0}(x_i)$
1: $x_i' = x_i \cdot log_2(e)_h$
2: $n_i = floor(x_i')$
3: $f_i = x_i' - n_i$
4: $2_{p_0}^{-f_i} = linearfit(Mode, f_i)$
5: $exp_{p_0}(x_i) = 2_{p_0}^{-f_i} >> n_i$
6: **return** $exp_{p_0}(x_i)$

Algorithm 3 Calculation for Logarithmic Function

Input: $Mode$, S
Output: $ln_{p_1}(S)$
1: $v = LOD(S)$
2: $u = S >> v$
3: $v = v \cdot ln(2)_h$
4: $log_2(u)_{p_1} = linearfit(Mode, u)$
5: $ln_{p_1}(S) = v + log_2(u)_{p_1}$
6: **return** $ln_{p_1}(S)$

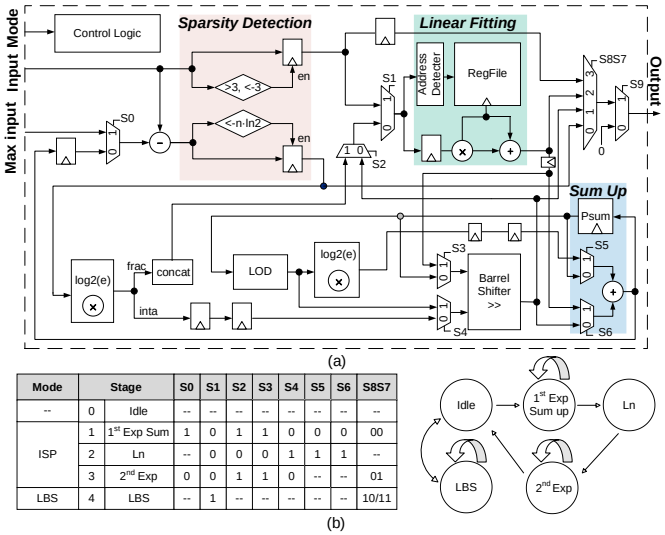
The calculation flows for the exponential function and logarithmic function in *ISP* are shown in Algorithm 2 and Algorithm 3, respectively. In Algorithm 2, a constant multiplication is initially performed on x_i to drive n_i and f_i. Then $exp_{p0}(x_i)$ is computed based on (12). In Algorithm 3, a leading one detector (LOD) is firstly used to find out the highest 1-bit position, which is v in (14). Then, u is obtained through a shifting operation and $ln_{p1}(S)$ is computed based on (15). The proposed *ISP* improves precision by refining the coefficients of linear fitting, without incurring additional computational overhead. These algorithms can be implemented by reconfiguring modules and data paths in hardware to save area cost. For example, the different linear fitting operations of the exp_{p0}, ln_{p1}, and 8-segment *LBS* in Algorithm 1 can be computed by reconfiguring the linear fitting module. By leveraging these characteristics, we propose a novel hardware architecture that is compact and highly reconfigurable in Section IV.

IV. Hardware Architecture

In this section, we design a compact and reconfigurable *SoftAct* architecture to support the efficient execution of the proposed improved softmax with penalties (*ISP*), stage-wise full zero detection (*SFZD*), and low-bit width Swish (*LBS*) algorithms. An overview of the *SoftAct* architecture is provided, followed by explanations of the reconfigurable data paths. To address the mismatching issues during different modes, we propose a symmetric detection and storage structure for a linear fitting module. Furthermore, the detailed structures of other modules are provided.

A. The Overall Architecture

According to the unified computation framework, the overall hardware architecture for *SoftAct* is designed and shown in Fig. 4. The design mainly consists of a sparsity detection module, a linear fitting module, a LOD module, and an adaptive shifter. The sparsity detection module is responsible for implementing the proposed stage-wise full zero detection (*SFZD*) for *ISP* and range detection for *LBS*. The linear fitting module is used for computing nonlinear functions based on PLF, which is a core unit in this architecture. The adaptive shifter module is used for the shifting operations.



Mode	Stage		S0	S1	S2	S3	S4	S5	S6	S8S7
--	0	Idle	--	--	--	--	--	--	--	--
ISP	1	1st Exp Sum	1	0	1	1	0	0	0	00
	2	Ln	--	0	0	0	1	1	1	--
	3	2nd Exp	0	0	1	1	0	--	--	01
LBS	4	LBS	--	1	--	--	--	--	--	10/11

Fig. 4. (a) The overall architecture for *SoftAct*. (b) Stages and configuration.

To achieve a high frequency and reconfigurable architecture, there are five stages in the computation flow, as shown in Fig. 4(b). Stage1 to Stage4 in Fig. 4(b) correspond to Stage1 to Stage4 in Algorithm 1, respectively. Each stage in Fig. 4(b) represents a different function and employs a different data flow. Stage0, called Idle, is responsible for receiving the linear fitting coefficients from external sources and storing them in the linear fitting module. When operating in the *ISP* mode, the *SoftAct* architecture executes from Stage1 to Stage3. Stage1 with a four-stage pipeline is illustrated in Fig. 5(a). In the pipeline, the input is processed by the sparsity detection module in the first stage. This is followed by the constant multiplication and linear fitting modules in the second and third stages, respectively. In the final stage, the data is processed by the shifter module and sum up module to complete Algorithm 2 and the accumulation. The linear fitting module is used to calculate $2_{p_0}^{-f_i}$. If the sparsity detection module detects that the data meet the conditions specified in (19), subsequent operations are bypassed and *Output* is set to 0. The three-stage pipeline of Stage2 is illustrated in Fig. 5(b). The data stored in the register of the sum up module is utilized as the input data. This data, after passing through the LOD module and the shifter module in the first stage, is processed in the linear fitting module to perform $log_2(u)_{p_1}$ in Algorithm 3. Then, the adder unit in the sum up module is reused to complete the logarithmic function. Stage3 is the last stage in the *ISP* mode, as shown in Fig. 5(c) with a four-stage pipeline. The data flow of Stage3 is similar to Stage1 with few differences: the input and output paths are modified, the sparsity detection module achieves (20), and the sum up module is not used. There is one stage to implement the 8-segment *LBS*, which is Stage4 as shown in Fig. 5(d) with a three-stage pipeline. *Input* is fed into the sparsity detection module to avoid redundant computations. The linear fitting module is reconfigured to implement the nonlinear function in (21). Therefore, a significant portion of modules in the *SoftAct*

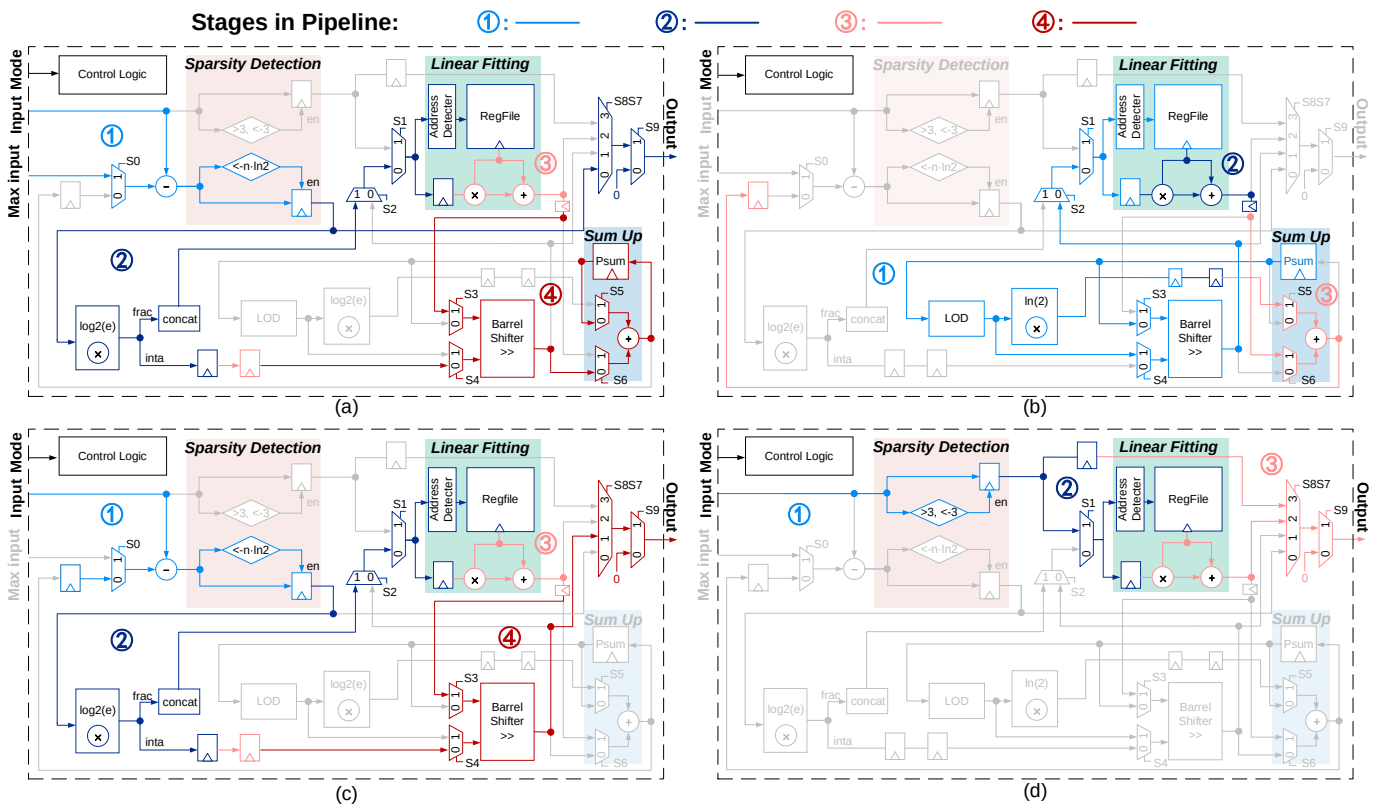


Fig. 5. Configured data paths and detailed pipelines of major functions. (a) The 1^{st} exponential calculation and sum up. (b) Logarithmic calculation. (c) The 2^{nd} exponential calculation. (d) The 8-segment *LBS* operation.

architecture are reused multiple times to achieve various functions with minimal hardware cost. Thanks to the fixed and adaptive penalty algorithm in Section III-A, *SFZD* in Section III-B, and the symmetric detection and storage structure in Section IV-B, there is no need to return to Stage0 to update the linear fitting coefficients in the *ISP* mode. In contrast, the approach proposed in [21] requires updating the LUT of the exponential function for different stages, causing extra power consumption and system delay.

B. The Symmetric Detection and Storage

The linear fitting module is the core component in *SoftAct* to achieve nonlinear function calculations based on PLF. The linear fitting module mainly consists of an address detector, a Register File (RegFile), a penalty module, a multiplier, and an adder, as illustrated in Fig. 6(a). The address detector module generates the address to the RegFile module based on the input data. The RegFile module is used for storing and retrieving the linear fitting coefficients, which are then passed to the multiplier and the adder. The penalty module represents P_{ln} in (15). The *NeedP* signal refers to the control signal for the adaptive algorithm (17), which can be generated by checking the output of the LOD module. Since $th = 3$ is predetermined, the penalty module is considered a constant addition.

We use 4-segment PLF to perform the calculation of exponential and logarithmic functions in *ISP*, and 8-segment PLF for *LBS*. This results in mismatching issues between the segmentation detection of input data and the loading of the PLF coefficients during different modes. For example, during *ISP*, the address detector module needs to detect four segments for *exp* and *ln*, whose inputs are in the range of [0, 1) and [1, 2), respectively. They are corresponding to two sets of four PLF coefficients. For *LBS*, eight segments need to be detected within the range [-3, 3] with eight PLF coefficients. To address this issue, we propose a symmetric detection and storage structure for the linear fitting module, which can achieve the aforementioned functions with minimal hardware cost, as shown in Fig. 6.

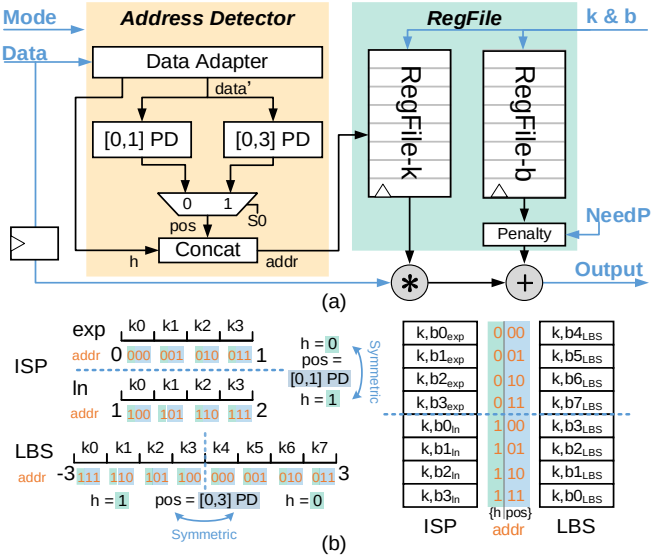


Fig. 6. (a) The architecture of the linear fitting module (PD: position detector). (b) The mechanism of the symmetric detection and storage design.

The data adapter first generates *h* based on the input signals *Mode* and *Stage*. When *Mode* is *ISP*, *h=0* for *exp* and *h=1* for *ln*. Note that only the fractional data are needed for the 4-segment detection within [0, 1) and [1, 2). Therefore, the data adapter extracts the fractional part of the *Data* to obtain *data'*, which is passed to the [0, 1] position detector (PD) for both [0, 1) and [1, 2) detection to obtain the position signal *pos*. When *Mode* is *LBS*, the data adapter obtains *h* based on the sign bit of *Data* and takes its absolute value to get *data'*, which is in the range of [0, 3]. Consequently, only one 4-segment detector [0, 3] PD is required and generates the *pos* signal for 8-segment detection. Finally, *h* and *pos* are concatenated to get the address information *addr*, which is then passed to the RegFile module. The RegFile stores the corresponding *k* and *b* symmetrically, as shown in Fig. 6(b), to accomplish all the required functions.

C. The LOD Module and Adaptive Shifter Module

The LOD module is used to detect the position of the highest 1-bit position in the input data. We optimize this module based on the binary search algorithm, as illustrated in Fig. 7 by using an example with 8-bit input *In* and a 3-bit output *Out*. Compared to the conventional LOD structure, the proposed binary search structure reduces the computational complexity from *O(n)* to *O(log n)*, thus effectively reducing the latency.

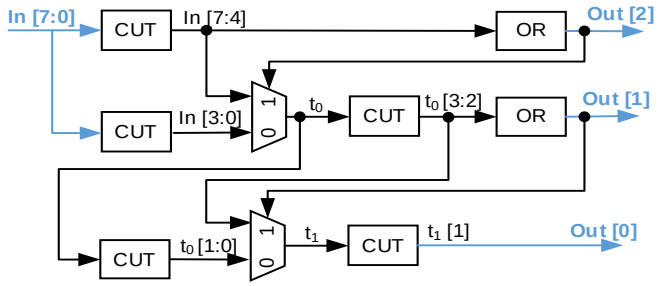


Fig. 7. The architecture of the LOD based on the binary search algorithm.

The adaptive shifter module is reused multiple times when implementing the softmax function. The module has two input ports: *data* and *num*, realizing the function of *data* >> *num*, where *num* is a variable. We implement this unit based on the barrel shifter structure, as shown in Fig. 8 with an example of 3-bit *num*.

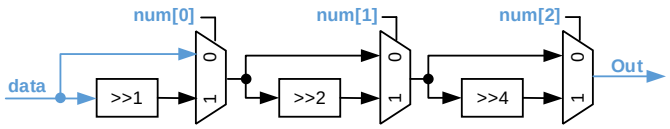


Fig. 8. The architecture of the adaptive shifter.

V. Experimental Results

In this section, the proposed *SoftAct* is evaluated at both the software and hardware levels. In the software implementation, we use PyTorch to realize the proposed algorithms, perform analysis, and compare *ISP* and *SFZD* with the state of the art in terms of algorithm precision, network accuracy, and sparsity utilization rate. Then we evaluate and compare the area and power consumption of the proposed *SoftAct* architecture with the state of the art.

A. Analysis of ISP Precision and Network Accuracy

For the existing method [21], several sets of random values with different ranges following a uniform distribution are chosen as the input data of softmax. However, the results of this method are unstable due to insufficient randomness within each set. For instance, the algorithm may perform well under certain random conditions, but poorly under others. To address this issue, we propose a more rigorous testing method, namely grouped set random test, to eliminate the instability. We use 50 groups of four different sets as the input data. Each set contains 5000 random values in different ranges, which are denoted as rand1, rand5, rand10, and rand100. For instance, the rand10 set consists of 5000 randomly selected points in the interval [-10, 10] that adhere to a uniform distribution. To compare the results with other algorithms, we use two metrics: the mean absolute error (*MAE*) and the mean square error (*MSE*).

$$MAE = \frac{1}{g}\sum_{i=1}^{g}\left(\frac{1}{n}\sum_{j=1}^{n}|o_a - o_t|\right), \tag{22}$$

$$MSE = \frac{1}{g}\sum_{i=1}^{g}\left(\frac{1}{n}\sum_{j=1}^{n}(o_a - o_t)^2\right). \tag{23}$$

o_a is the output value of proposed algorithms, o_t is the theoretical value of the softmax function, n is the number of points in each set, and g is the group number. A lower *MAE* indicates consistent precision, while a lower *MSE* emphasizes the proficiency of the method in mitigating large error.

The *MAE* and *MSE* comparisons between *ISP* and the latest three works under 50-group random data are shown in Fig. 9. We reduce 99.29%, 97.87%, and 88.89% *MAE* and 99.76%, 99.66%, and 97.74% *MSE* compared to [19], [20], and [21], respectively, in grouped set random test. The softmax method in [19], which is based on the original softmax, is set as the baseline softmax. Compared with the baseline, the improvement can be attributed to two factors. Firstly, the log-sum-exp format of *ISP* saves the division operation and constrains the data range for PLF, which reduces 66.44% *MAE* and 30.56% *MSE* compared to the baseline. Secondly, the introduced penalty algorithm in *ISP* effectively reduces errors caused by approximate constant multiplication, thereby further reducing 97.87% *MAE* and 99.66% *MSE* compared to only using the log-sum-exp format.

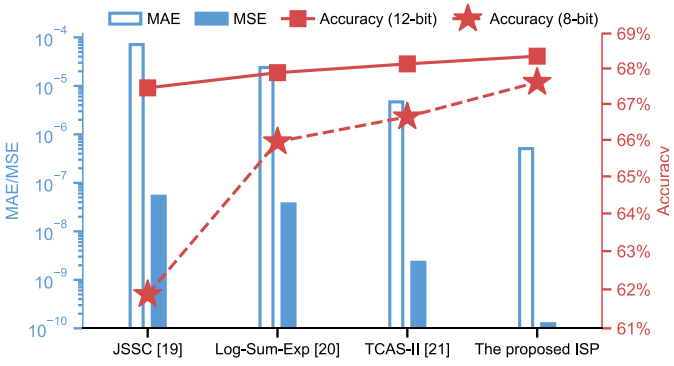


Fig. 9. The *MAE*, *MSE*, and network accuracy comparison of different softmax methods.

To compare the proposed algorithm with the state-of-the-art algorithms on the CAC Transformer network, we implement

them on the MobileViT-xxs for verification. The open-source tool Distiller [31] is used for quantization and retraining MobileViT-xxs to recover the accuracy. The highest network accuracy we can achieve under different methods (12-bit and 8-bit quantization) is shown in Fig. 9.

The accuracy of the MobileViT-xxs network is 68.42% for FP32 implementation. The network accuracy at 12-bit quantization is acceptable when using the baseline softmax method in [19] with 0.9% accuracy loss, but declines severely at 8-bit quantization with 6.68% accuracy loss. This is because radical network quantization degrades the network accuracy, and the low-precision softmax method in [19] worsens the accuracy degradation caused by radical quantization. The methods in [20] and [21] can ensure accuracy at 12-bit quantization with 0.54% and 0.29%, respectively. However, their accuracies are not acceptable at 8-bit quantization with 2.47% and 1.3% degradation, respectively. The proposed *ISP* ensures 68.35% network accuracy at 12-bit quantization with nearly no degradation, and exhibits only 0.81% accuracy loss at 8-bit quantization, indicating the efficiency of our *ISP* even under aggressive quantization scenarios. Compared with the state-of-the-art works [19], [20], and [21], our *ISP* achieves 5.87%, 1.66%, and 0.97% network accuracy improvements in 8-bit quantization, and 0.9%, 0.47%, and 0.22% network accuracy improvements in 12-bit quantization, respectively.

B. Evaluation of SFZD

Exploiting the sparsity derived from the combination of softmax and quantization can further enhance hardware energy efficiency by eliminating redundant operations. To achieve this, we employ linear quantization to perform fixed-point 12-bit and 8-bit quantization on the MobileViT-xxs network, respectively. To accommodate the proposed *SFZD* algorithm, we fix the min and max values when quantizing the softmax function. Among the existing works, only [19] features the sparsity detection for the softmax function. We compare the proposed *SFZD* with [19] in terms of sparsity utilization, which is the ratio of the sparsity utilized by the algorithm and the sparsity provided by the network quantization. The results are listed in Table I.

TABLE I
SPARSITY UTILIZATION RATIO OF DIFFERENT METHODS

MobileViT-xxs	JSSC 2023 [19]	Ours
12b	42.49%	100%
8b	30.31%	100%

The sparsity extraction method of [19] is based on (19), which fails to fully utilize the sparsity offered by quantized softmax. This is because [19] overlooks the sparsity brought by $\sum_{j=0}^{N} e^{x_j - x_{max}}$, which accounts for a substantial role in the softmax sparsity assessment. The proportion of sparsity generated by $\sum_{j=0}^{N} e^{x_j - x_{max}}$ to the overall sparsity increases as the size of Transformer-based networks decreases. Consequently, the efficiency of the sparsity detection method in [19] deteriorates in small sized networks. Our method takes $ln(S)$ into consideration and improves the sparsity utilization ratio by 2.4× and 3.3× under 12-bit and 8-bit quantization, respectively. The proposed *SFZD* not only eliminates redundant calculations, but also reduces the hardware area cost. *SFZD* constrains the input range and reduces the bit width for Stage1 and Stage3 in Fig. 2, which saves ~18% area in hardware. Furthermore, *SFZD* ensures a fixed and identical input range for Stage1 and Stage3, which allows for the reuse of the same hardware modules and leads to further savings of ~30% area. After evaluation, *SFZD* only occupies 2.5% hardware area in the whole architecture while contributing 19.2% power savings.

C. Hardware Implementation Results

To thoroughly analyze our proposed *SoftAct* architecture, two versions are implemented in Verilog HDL: fix-point 16b and fix-point 32b (the bit width of the partial sum in network processing). The precision of the 16b version is consistent with Tables I and Fig. 9, where the bit width of partial sum is also 16b. Due to the increased bit width, the 32b version has a better precision than the 16b version by observing the *MAE* and *MSE*. We synthesize these designs by Cadence Genus using the TSMC 28-nm CMOS technology, then perform gate-level simulations based on 12b quantized MobileViT-xxs data by using Cadence NC-Sim. The VCD (value change dump) files are dumped for power analysis. The results of our proposed *SoftAct* are listed in Table II. To evaluate the hardware architecture, we use three metrics: area efficiency (*AE*), energy efficiency (*EE*), and overall efficiency (*OE*). The definitions for *Throughput* and *OE* are

$$Throughput = Freq \times NUM. \quad (24)$$

$$OE = Throughput/(Area \times Power). \quad (25)$$

The *Freq* is the running frequency and *NUM* is the number of inputs or outputs in hardware. In the 16b and 32b versions, the hardware performance of the *SoftAct* is evaluated at two operating frequencies: the maximum frequency and a frequency of 500 MHz. The lower the bit width, the higher the maximum frequency and the better the hardware metrics can be achieved. For example, at 500 MHz, the 16b *SoftAct* outperforms the 32b version in *AE*, *EE*, and *OE* by factors of 1.6×, 2.1×, and 3.4×, respectively.

The proposed *SoftAct* architecture can support both softmax and low-bit width Swish functions, which are inseparable due to the high reconfigurability of the architecture. Alternatively, the existing works only support the softmax function. To evaluate the performance of *SoftAct*, we perform comparisons with the state of the art optimized for the softmax function. According to the layout shrinking rules of the foundries (for 65 nm, 40 nm, and 28 nm), the area is reduced by around 5.4× (2.3^2) with the technology scaled from 65 nm to 28 nm and reduced by around 2× (1.4^2) with the technology scaled from 40 nm to 28 nm. The supply voltage is reduced from 1.2 V to 0.9 V with the technology scaled from 65 nm to 28 nm and reduced from 1.1 V to 0.9 V with the technology scaled from 40 nm to 28 nm. With the area and supply voltage scaling, the area efficiency, energy efficiency, and overall efficiency of [13], [32], and [34] can be approximately scaled into 28-nm technology node as shown in Table II, which demonstrates that *SoftAct* achieves the highest frequency and *OE* among the

TABLE II
PERFORMANCES COMPARISON OF THE DIFFERENT HARDWARE ARCHITECTURES

	ICSICT 2018 [34]	GLSVLSI 2019 [32]	TCAS-II 2020 [21]	HPCA 2021 [13]	JSSC 2022 [33]	JSSC# 2023 [19]	**SoftAct**			
Technology	65 nm	65 nm	28 nm	40 nm	28 nm	28 nm	**28 nm**			
Function	Softmax	Softmax	Softmax	Softmax	Softmax	Softmax	**Softmax / Low-bit width Swish**			
Sparsity	NA	NA	NA	NA	NA	Softmax	**Softmax / Low-bit width Swish**			
Voltage [V]	1.2	1.2	0.9	1.1	1.0	1.1	**0.9**			
BW/NUM	32/1	16/1	16/8	FP12/8	16/16	16/1	**16/1**		**32/1**	
Frequency [GHz]	1♦	0.5♦	1.64♦	1♦	0.24♦	0.51♦	**0.5**	**1.85♦**	**0.5**	**1.56♦**
Area [mm^2]	0.445	0.64	18.39 ×10^{-3}*	0.786	0.12	5.297 ×10^{-3}	**2.326 ×10^{-3}**	**2.872 ×10^{-3}**	**3.746 ×10^{-3}**	**4.453 ×10^{-3}**
Power Consumption [mW]	333	0.82	NA	496.6†	1.06†	1.50	**0.80**	**3.46**	**1.66**	**5.76**
Throughput [G/s]	1	0.5	13.12	8	3.84	0.51	**0.5**	**1.85**	**0.5**	**1.56**
Scaled Area Efficiency‡ [G/(s·mm^2)]	12.11	4.21	713.35*	20.78	31.90	96.28	**214.96**	**644.85**	**133.48**	**350.33**
Scaled Energy Efficiency‡ [G/(s·mW)]	2.88×10^{-2}	5.84	NA	4.91×10^{-2}†	3.61†	0.34	**0.63**	**0.54**	**0.301**	**0.27**
Scaled Overall Efficiency‡ [G/(s·mm^2·mW)]	0.35	49.19	NA	0.13†	30.00†	64.14	**270.05**	**186.48**	**80.38**	**60.82**

BW: Bit width. NUM: Number of inputs or outputs. '♦' represents the maximum frequency. '*' represents the cell area without the interconnect area.
'†' means the power consumption is obtained from the simulation of the entire network. If only the softmax data are processed, the results are worse.
'#' represents the reimplemented work. '‡' Scaled to 28-nm technology node, according to "$Energy\ Efficiency \propto 1/(Area \times V_{DD}^2)$" derived in [35].

evaluated works. A 16b fixed-point softmax is implemented in [32], which has lower power consumption than our proposed *SoftAct*. However, our *SoftAct* achieves 51.1× and 5.5× improvement in *AE* and *OE*, respectively, compared to [32]. [21] only considers the cell area without the interconnect area. Meanwhile, there is no power information in [21]. The cell area of our 16b *SoftAct* is 2.157×10^{-3} mm^2 under maximum frequency 1.85 GHz, with corresponding 858.6 G/(s·mm^2) *AE*, which is 1.2× better than [21]. The design in [13] comprises a floating-point 12b softmax unit. Although with a smaller bit width than our *SoftAct*, the area and power consumption of the softmax unit in [13] are still significantly higher due to the floating-point unit. [33] incorporates a 16b fixed-point softmax module operating at a lower frequency than our *SoftAct*. Note that the power consumptions given in both [13] and [33] are derived from the entire network. If these works are analyzed exclusively for the softmax function, the power consumption, *EE*, and *OE* would appear less favorable. This is because the softmax unit is possibly in the idle state for most of the time during full network processing, leading to low power consumption of the softmax unit. Despite these circumstances, with our 16-bit *SoftAct*, we attain a higher operating frequency, with 31× and 6.7× better *AE* as well as 1435× and 9× *OE* compared to [13] and [33], respectively. We have implemented the softmax structure of [19] in Verilog HDL using the same 28-nm CMOS technology and obtained the design metrics of the softmax of [19]. The design in [19] supports softmax and sparsity detection. However, the lack of input data constraint necessitates high-segment linear fitting to ensure precision, leading to high area consumption. Compared to [19], our approach increases *AE* by 2.2×, *EE* by 1.9×, and *OE* by 4.2× with higher softmax precision and network accuracy. [34] proposes a 32b fix-point softmax architecture, whose area and power consumption are unfavorable. Our proposed 32b *SoftAct* architecture can achieve an operating frequency of up to 1.56 GHz, ensuring better area and power consumption, with an improved *OE* of 174× compared to [34].

As shown in Fig. 10, under the maximum frequency, the proposed 16-bit *SoftAct* architecture improves the area efficiency by 31×, 53.2×, 20.2×, 153.2×, and 6.7×, compared with [13], [34], [33], [32], and [19], respectively. Meanwhile, to evaluate the performance in terms of area and power consumption, the overall efficiency is compared. The proposed 16-bit *SoftAct* architecture improves the overall efficiency by 1435×, 532.8×, 6.2×, 3.8×, and 2.9×, compared with [13], [34], [33], [32], and [19], respectively.

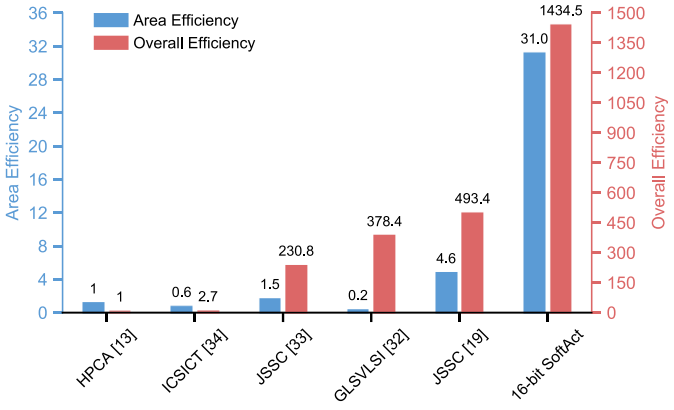


Fig. 10. The area efficiency and overall efficiency improvement compared with the state of the art.

VI. Conclusion

In this paper, an algorithm-hardware co-designed *SoftAct* architecture is proposed to achieve various nonlinear functions in convolution and attention co-designed Transformer-based networks. To improve the softmax precision in hardware implementation, an improved softmax algorithm with penalties is proposed for log-sum-exp softmax. A stage-wise full zero detection method is introduced to skip the redundant computation in log-sum-exp softmax. A compact and reconfigurable hardware architecture with a symmetric detection and storage structured linear fitting module is developed to achieve high precision and multiple nonlinear functions. Benchmarked with the MobileViT-xxs network classifying the ImageNet-1k dataset, the proposed improved softmax with penalties algorithm achieves up to 99.29% *MAE* and 99.77% *MSE* reductions in grouped set random test, and 5.87% network accuracy improvement compared with the evaluated works. The proposed *SoftAct* architecture is implemented in the TSMC 28-nm CMOS technology, achieving up to 153.2× the area efficiency and 1435× overall efficiency improvements among the state of the art, thereby providing an attractive option for Transformer-based network accelerators.

References

[1] A. Vaswani *et al*., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst*., vol. 30, 2017, pp. 1–11.

[2] A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020.

[3] W. Wang, X. Yang and J. Tang, "Vision transformer with hybrid shifted windows for gastrointestinal endoscopy image classification," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 9, pp. 4452-4461, Sept. 2023.

[4] H. Li, J. Xiao, M. Sun, E. G. Lim and Y. Zhao, "Transformer-Based language-person search with multiple region slicing," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 3, pp. 1624-1633, March 2022.

[5] Z. Liu *et al*., "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 10012–10022.

[6] R. Ji, J. Li, L. Zhang, J. Liu and Y. Wu, "Dual transformer with multi-grained assembly for fine-grained visual classification," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 9, pp. 5009-5021, Sept. 2023.

[7] Q. Han *et al*., "On the connection between local attention and dynamic depth-wise convolution," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[8] X. Zhang, J. Wang, T. Wang and R. Jiang, "Hierarchical feature fusion with mixed convolution attention for single image dehazing," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 2, pp. 510-522, Feb. 2022.

[9] A. Gulati et al., "Conformer: Convolution-augmented transformer for speech recognition," in Interspeech, 2020.

[10] Y. Wang, Y. Qiu, P. Cheng and J. Zhang, "Hybrid CNN-Transformer features for visual place recognition," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 3, pp. 1109-1122, March 2023.

[11] H. Wu *et al*., "CvT: Introducing convolutions to Vision Transformers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 22-31.

[12] S. Mehta and M. Rastegari, "MobileViT: light-weight, general-purpose, and mobile-friendly Vision Transformer," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[13] H. Wang, Z. Zhang, and S. Han, "SpAtten: Efficient sparse attention architecture with cascade token and head pruning," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 97–110.

[14] C. Gao, S. Braun, I. Kiselev, J. Anumula, T. Delbruck, and S.-C. Liu, "Real-time speech recognition for IoT purpose using a delta recurrent neural network accelerator," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[15] F. Spagnolo, S. Perri, F. Frustaci, and P. Corsonello, "Energy-efficient architecture for CNNs inference on heterogeneous FPGA," *J. Low Power Electron. Appl*., vol. 10, no. 1, p. 1, Dec. 2019.

[16] T. J. Ham *et al*., "A3: Accelerating attention mechanisms in neural networks with approximation," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 328–341.

[17] R. Rizk, D. Rizk, F. Rizk, A. Kumar, and M. Bayoumi, "A resource-saving energy-efficient reconfigurable hardware accelerator for BERT-based deep neural network language models using FFT multiplication," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2022, pp. 1675-1679.

[18] B. Li *et al*., "Ftrans: energy-efficient acceleration of transformers using FPGA," in *Proc. IEEE/ACM Int. Symp. Low Power Electron Design (ISLPED)*, 2020, pp. 175-180.

[19] Y. Wang *et al*., "An energy-efficient Transformer processor exploiting dynamic weak relevances in global attention," *IEEE J. Solid-State Circuits*, vol. 58, no. 1, pp. 227-242, Jan. 2023.

[20] B. Yuan, "Efficient hardware architecture of softmax layer in deep neural network," in *Proc. IEEE Int. System-on-Chip Conf. (SOCC)*, Sep. 2016, pp. 323–326.

[21] D. Zhu, S. Lu, M. Wang, J. Lin, and Z. Wang, "Efficient precision-adjustable architecture for softmax function in deep learning," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 12, pp. 3382–3386, Dec. 2020.

[22] A. Kagalkar and S. Raghuram, "CORDIC based implementation of the softmax activation function," in *Proc. Int. Symp. VLSI Design Test (VDAT)*, Jul. 2020, pp. 1–4.

[23] F. Spagnolo, S. Perri, and P. Corsonello, "Aggressive approximation of the softmax function for power-efficient hardware implementations," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 1652–1656, Mar. 2022.

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

[25] Y. Zhang *et al*., "Base-2 softmax function: suitability for training and efficient hardware implementation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, pp. 3605–3618, Sep. 2022.

[26] S. R. Chiluveru, Gyanendra, S. Chunarkar, M. Tripathy, and B. K. Kaushik, "Efficient hardware implementation of DNN-based speech enhancement algorithm with precise sigmoid activation function," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 11, pp. 3461–3465, Nov. 2021.

[27] M. Barthel, J. Rust, J. Gustafson, and S. Paul, "Improving the precision of SORN arithmetic by introducing fused operations," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2022, pp. 258-262.

[28] H. Sun *et al*., "A universal method of linear approximation with controllable error for the efficient implementation of transcendental functions," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 1, pp. 177–188, Jan. 2020.

[29] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-BERT: Integer-only BERT quantization", in *Proc. Int. Conf. Mach. Learn. (ICML)*, pp. 5506-5518, Jul. 2021.

[30] A. Howard *et al*., "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314-1324.

[31] Distiller. Accessed: Sep. 8, 2023. [Online]. Available: https://github.com/IntelLabs/distiller.

[32] G. Du, C. Tian, Z. Li, D. Zhang, Y. Yin, and Y. Ouyang, "Efficient softmax hardware architecture for deep neural networks," in *Proc. Great Lakes Symp. VLSI*, May 2019, pp. 75–80.

[33] *F. Tu et al.*, "TranCIM: Full-digital bitline-transpose CIM-based sparse Transformer accelerator with pipeline/parallel reconfigurable modes," *IEEE J. Solid-State Circuits*, vol. 58, no. 6, pp. 1798-1809, June 2023.

[34] Q. Sun *et al.*, "A high speed softmax VLSI architecture based on basicsplit," in *Proc. IEEE Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, 2018, pp. 1–3.

[35] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "How to understand and evaluate deep learning processors: TOPS/W (alone) considered harmful," *IEEE Solid State Circuits Mag.*, vol. 12, no. 3, pp. 28-41, Aug. 2020.