Squaremax: A Hardware-Friendly Replacement for Softmax and Its Efficient VLSI Design and Implementation

Meng-Hsun Hsieh, Xuan-Hong Li, Yu-Hsiang Huang, Pei-Hsuan Kuo, and *Juinn-Dar Huang*

Department of Electronics and Electrical Engineering & Institute of Electronics
National Yang Ming Chiao Tung University, Hsinchu, Taiwan
{mhhsieh.ee11, alanlee.ee09, ethan0707.ee10, psh09018.ee10, jdhuang}@nycu.edu.tw

Abstract— **The Softmax function holds an essential role in most machine learning algorithms. Conventional realization of Softmax necessitates computationally intensive exponential operations and divisions, thereby posing formidable challenges in developing low-cost hardware implementations. This paper presents a promising hardware-friendly alternative, Squaremax, which gets rid of complex exponential operations. The function definition is extremely simple and can thus be efficiently implemented in both software and hardware. Experimental results show that Squaremax consistently attains comparable or superior accuracy over several popular models. Besides, this paper also proposes an efficient hardware architecture design of Squaremax. It requires no functional units for exponential and logarithmic operations, and is even lookup table (LUT) free. It adopts a flexible 16-bit fixed-point Q format for I/O to better preserve the output precision, which leads to higher model accuracy. Moreover, it yields substantial improvements in speed, area, and power, as well as achieves remarkable area and power efficiency of 664 G/mm² and 1396 G/W in a 40nm process. Therefore, hardware-friendly Squaremax is a very promising alternative to complex Softmax in both software and hardware for deep learning applications, and the proposed hardware architecture design and efficient LUT-free implementation do achieve a notable improvement in speed, area, and power.**

Keywords—hardware-friendly activation function design, Softmax, efficient VLSI implementation.

I. Introduction and Previous Work

Activation functions serve as fundamental building blocks within neural networks, enabling them to transform raw input data into meaningful and actionable insights. Among these crucial functions, the Softmax activation function stands out as a linchpin in the realm of deep learning.

Softmax plays a pivotal role in converting raw neural network outputs into a probability distribution. This process is essential for multiclass classification problems, where models must assign probabilities to various classes. By exponentiating and normalizing the input values, Softmax produces a probability distribution that reflects the likelihood of each class. It is the transformation that allows neural networks to make informed decisions and generate accurate predictions.

Therefore, Softmax is extensively adopted across a multitude of domains, including computer vision [1], natural language processing (NLP) [2]–[3], and beyond, underscoring its versatile applicability across a wide range of tasks. Despite its ubiquity and indispensable role, Softmax also presents a significant computational complexity, especially within the context of hardware design. The intricate mathematical operations involved, including exponentiation and division, which demands innovative strategies to achieve efficient hardware implementations. Consequently, several previous approaches have emerged to tackle the challenges posed by the hardware realization of Softmax.

The most straightforward and also widely adopted method for implementing a computationally intensive exponential operation within the Softmax function is through the use of a lookup table (LUT) [4]–[6]. In this method, precomputed exponential values for a range of possible input values are stored in a RAM-based table. When performing Softmax with a given input, the input value is considered an index to retrieve the corresponding output exponential value from the table. The method eliminates the need of computationally expensive exponential calculations, resulting in notable computation efficiency improvement. Nevertheless, it is worth noting that this method demands substantial memory resources as the lookup table, and the output accuracy is highly dependent on the input bit-width m (table height: 2^m) and the output bit-width n (table width: n); that is, the higher the output precision is desired, the larger the lookup table is demanded. Moreover, if the input value cannot be limited within a small range, it is extremely hard to preserve the output precision since the range of output values stored in the table gets wider exponentially.

Alternatively, mathematical transformations are applied to the exponential function in some studies. Using the Log-Sum-Exp trick, as adopted in [7]–[11], this technique involves taking the logarithm of the sum of exponentials and then exponentiating the result. It ensures numerical stability even when dealing with an unbounded input range. Nevertheless, this method requires additional operations such as logarithm computations, which inevitably leads to a larger and slower implementation. Hence, a base-2 alternative to the exponential function is employed in [12]–[14], where the exponential e^x is substituted by 2^x. Though this approach is favored for its computation efficiency, most of studies only demonstrated its use in CNN-based models, where merely single Softmax layer is used. It is unclear if it is still effective in a Transformer-based model, which includes dozens of Softmax layers.

In addition to previously mentioned methods, in [15] the exponential computation is replaced by Maclaurin series and division is replaced with shift by rounding the divisor to the nearest power-of-two. It achieves a notable area reduction at the cost of relatively large precision loss. It is worth mentioning that most previous studies on the implementation of complex Softmax function focused on approximating or substituting exponential computation and eliminating division in order to reduce hardware complexity.

In this paper, we present a new hardware-friendly function, Squaremax, as an alternative to Softmax. Its simple definition includes no exponential operations, which thus guarantees that hardware implementations can easily be efficient. In addition, no approximations are required since hardware implements exactly the same definition as software does. According to our experiments, Squaremax can achieve comparable or even higher accuracy in several well-known Transformer-based models including a dozen of Softmax layers inside.

The rest of this paper is organized as follows. Section II introduces the proposed function, Squaremax. Section III presents the details of hardware design and implementation. The experimental results and comparisons in terms of model accuracy and hardware performance are given in Section IV. Finally, the concluding remarks are given in Section V.

II. Function Definition of Squaremax

A. Original Softmax Function

The definition of Softmax function is shown below:

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

For a given input vector $x = [x_1, x_2, ..., x_n]$, the Softmax function produces an output vector $y = [y_1, y_2, ..., y_n]$, where y_i = Softmax(x_i). Softmax can be divided into two major steps. First, the Softmax function calculates the exponential value of each element x_i in the input vector. This is to amplify the differences between two positive elements; i.e., large positive values become even larger after exponentiation. Next, each individual exponential value of x_i in the input vector is divided by the sum of all exponential values. This step ensures that each output value y_i is less than 1 and the sum of all output values exactly equals 1, which makes the output vector y represent a probability distribution. That is, Softmax utilizes the exponential function for nonlinear weighting to intensely enlarge the difference between two positive input values. The key properties of Softmax can be further summarized below:

1. Do nonlinear weighting through the exponential function.
2. Summation of all values in the output vector is equal to 1.
3. Every output value y_i is always positive.
4. Weighting function is a strictly increasing function.
5. Weighting function is differentiable.

It is a great challenge to implement the Softmax function exactly in hardware: both exponential and division operations are computationally expensive, which inevitably leads to a large and slow implementation. As a result, an alternative that can be easily and efficiently implemented in both software and hardware without the need of approximation is desired.

B. Proposed Squaremax Function

To make the alternative simple, let us think outside the box: how about replacing the expensive exponential function with another weighting function that can be easily implemented in both software and hardware? Our solution is thus given in (2):

$$f(x_i) = \frac{ReLU(x_i)^p}{\sum_{j=1}^{N} ReLU(x_j)^p} \quad (2)$$

Moreover, we propose the Squaremax function, which is defined in (2) as p is set to 2. ReLU is used to clamp negative inputs to zero, preventing the function curve from bending upwards or downwards when x_i is negative. Similarly, the key properties of Squaremax can be summarized as follows:

1. Do nonlinear weighting through a polynomial function.
2. Summation of all values in the output vector is equal to 1.
3. Every output value y_i is always nonnegative.
4. Weighting function is a strictly nondecreasing function.
5. Weighting function is differentiable except for input $x = 0$.

It should be crystal clear that Softmax and Squaremax hold the same or very similar key properties, whereas Squaremax is much easier in both software and hardware implementations.

C. Division-to-Multiplication Conversion

Since the division is also expensive in hardware, it is also crucial to eliminate its use. Hence, we propose a division-to-multiplication conversion process, given in (3), which only requires multiplication and shift operations:

$$\frac{x}{D} = \frac{x}{m \times 2^n} = x \times \frac{1}{m} \times \frac{1}{2^n} \quad (3)$$

In (3), the denominator D is further expressed as $m \times 2^n$, where $1 \le m < 2$. First, multiplying by 2^{-n} can be easily done by a shift operation. Next, m is approximated as 1.*abc*, where *abc* are the most significant three bits next to the binary point, as shown in (4). Since m only has 8 possible values after approximation, merely an extremely low-cost 8-entry decoder with a 3-bit input (i.e., *abc*) is required to generate the reciprocals of those 8 possible values of m, as shown in (5). We will show later that considering only most significant 3 bits (i.e., *abc*) is enough to achieve a high model accuracy.

$$D = m \times 2^n \approx 1.abc \times 2^n \quad (4)$$

$$\frac{x}{D} = \frac{x}{m \times 2^n} \approx x \times \frac{1}{1.abc} \times \frac{1}{2^n} \quad (5)$$

Hence, the proposed division-to-multiplication conversion process requires neither an expensive divider nor a LUT-based reciprocal unit. It simply demands a multiplier, a decoder, and a shifter, which leads to a very efficient implementation. Moreover, there will be no approximation errors between software model and hardware implementation if both of them comply with the proposed conversion process.

III. Hardware Architecture and Implementation

The proposed implementation adopts the 16-bit fixed-point Q number format, where the exact input and output formats are set to Q16.0 and Q1.15, respectively.

A. Algorithm

The overall computation flow of the proposed Squaremax function is detailed in Algorithm 1. N is the length of the given input/output vector, and can be up to 8192 in the proposed hardware implementation.

Algorithm 1: Squaremax Function

Input: $x(1), x(2), \dots, x(N)$

Output: $Squaremax(1), Squaremax(2), \dots, Squaremax(N)$

```
1   acc ← 0
2   for i ← 1 to N do    // Step 1
3       Mult(i) ← ReLU(x(i)) × ReLU(x(i))
4       acc ← acc + Mult(i)
5       RSQR(i), dynamic_shift(i) ← LOD(Mult(i))
6   end
7   abc, static_shift ← LOD(acc)
8   for i ← 1 to N do    // Step 2
9       Mult(i) ← Decoder(abc) × RSQR(i)
10      Shift(i) ← static_shift − dynamic_shift(i)
11      Squaremax(i) ← R_shift(Mult(i), Shift(i))
12  end
```

In Step 1, ReLU is first applied to each input x_i and the outcome is further squared for each input x_i (Line 3). Note that $(\text{ReLU}(x))^2$ is a strictly nondecreasing function. Next, adding up $(\text{ReLU}(x_i))^2$ into 40-bit *acc* to get the denominator D in (2) (Line 4). A 15-bit unsigned multiplier is used for the square operation because the output of ReLU is always nonnegative. The output of a 15-bit multiplier, *Mult*(i), is 30-bit wide. Nevertheless, the squared value is used as the multiplier input in Step 2, which limits the maximum bit-width to 15. To minimize the precision loss in this 30-bit to 15-bit conversion, a dynamic scaling approach is developed. A leading one detector (LOD) is used to determine 30-bit *Mult*(i) should be right-shifted by s = *dynamic_shift*(i) bits, where $0 \le s \le 15$. Then, right-shift *Mult*(i) by s bits and take the rightmost 15-bit result as *RSQR*(i) (Line 5). Similarly, an LOD can be used to find out the values of *abc* and n in (4) (Line 7). At last, the reciprocal of m in (5), S_{abc}, can be generated via a decoder.

In Step 2, *RSQR*(i) is first multiplied by S_{abc} to get the fraction part (Line 9), which needs to be further right-shifted to get the final *Squaremax*(i) in Q1.15 format. The correct right-shift amount, *Shift*(i), is equal to the difference between *static_shift* and *dynamic_shift*(i) (Line 10). Finally, a proper right shift is applied to get every *Squaremax*(i) (Line 11).

B. Hardware Architecture

We also propose a hardware architecture design, shown in Fig. 1, which can realize the algorithm previously mentioned in Section III-A. The proposed design can process 8 input elements simultaneously to boost the design throughput. Since it is a 2-step algorithm, those blue lines in Fig. 1 indicate data flows in Step 1, while red ones indicate data flows in Step 2.

In Step 1, a 16-bit signed input x is fed into a ReLU unit. The ReLU output is determined by the sign bit of x; however, the result is always a 15-bit unsigned value. After squaring ReLU(x_i) via a 15-bit multiplier, an LOD is used to process 30-bit multiplication output and then produces 15-bit *RSQR*(i) and its associated 4-bit *dynamic_shift*(i). A 40-bit accumulator is used to sum up all $(\text{ReLU}(x_i))^2$, and the succeeding LOD and the decoder are in charge of producing the values of S_{abc} and *static_shift* required in Step 2 later. Note that the input vector length N can be up to 8192, which is large enough to support recent large language models such as GPT-4-8k [16].

In Step 2, the same 15-bit multiplier is used to multiply *RSQR*(i) and S_{abc}, which significantly reduces the area cost via resource sharing. Then, the right-shift amount is calculated by subtracting *dynamic_shift*(i) from *static_shift*. At the end, a shifter is utilized to get the *Squaremax*(i) in Q1.15 format.

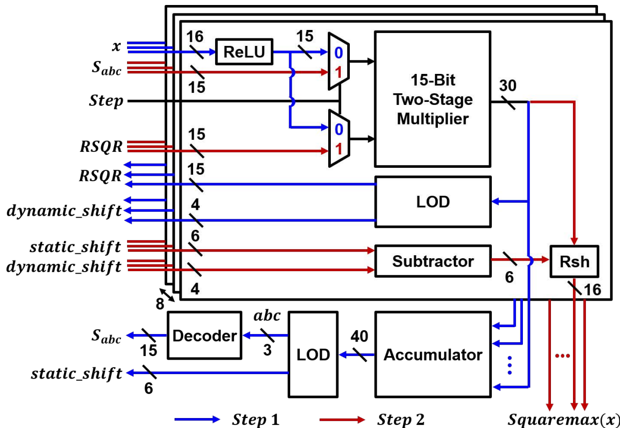

Fig. 1. Proposed hardware architecture design for Squaremax.

IV. Experimental Results and Comparisons

This section includes a set of experiment results, analyses, discussions, and comparisons between our work and previous studies. The first half focuses on the model accuracy, while the second half discusses the hardware performance.

A. Model Accuracy

To verify whether Squaremax is a good replacement to Softmax, two famous Transformer models, DeiT [17] from Meta (Facebook) and Swin [18] from Microsoft, are selected for performance evaluation since they both include rich Softmax layers inside. All experiments have been conducted in the PyTorch framework. The model accuracy is evaluated using the well-known ImageNet-1K dataset.

TABLE I reports the Top-1 accuracy of DeiT-Tiny using the original Softmax and the proposed polynomial-based functions in (2) with p = 1~4 after 32-epoch post-training. It is evident that the proposed polynomial functions achieve the same level of accuracy as compared with original Softmax. Among these four versions, the cubic one achieves the highest model accuracy. Nevertheless, the cubic version (p = 3) requires more multiplications in hardware implementation, which either increases the area cost or reduces the throughput. In contrast, the square version (p = 2) merely requires one multiplication, and its accuracy is virtually the same as that of the cubic one. On the other hand, the accuracy of the linear version (p = 1) is significantly lower though it demands no multiplication. In our opinion, the square version achieves the best balance between hardware cost and model accuracy. Therefore, we name the square version "Squaremax" and select it as an alternative to original Softmax.

TABLE II presents the model accuracy over a set of Squaremax variants. First, two baseline models, DeiT-Tiny and Swin-Tiny, originally adopting Softmax are initially trained for 300 epochs. Next, we replace all Softmax layers with Squaremax ones in DeiT-Tiny and Swin-Tiny, and train the modified models from scratch for 300 epochs as well. The results in TABLE II clearly show that Squaremax outperforms Softmax: the model accuracy is raised by 1.12% and 0.01% in DeiT-Tiny and Swin-Tiny, respectively. Note that both models consist of 12 Transformer layers; that is, the above comparisons are not made just at the last layer in most CNN models, which most previous studies did. Hence, it is evident that Squaremax is indeed a promising alternative to Softmax.

In the proposed hardware implementation, the division in Squaremax is converted to a combination of multiplication, decoding, and shifting operations. To check the effectiveness of the conversion, the division in Squaremax is replaced, and the modified models are further post-trained for another 32 epochs. Moreover, a 3-bit index *abc* is sent to the decoder in Section II-C. A set of experiments have been conducted to see how the length of index (1~4 bits) affects the model accuracy. In general, a shorter index leads to a smaller design but incurs a larger accuracy loss potentially. The experiment results are also listed in TABLE II. It is apparent that an index length of 3 or 4 bits is quite enough to make the conversion a success. Hence, a 3-bit index is selected in our implementation. More surprisingly, models incorporated with such conversion even achieve a higher accuracy than their original counterparts.

Therefore, it is convincing that the proposed Squaremax is a very competitive alternative against Softmax since it not only can achieve the same level of model accuracy but also enables extremely efficient hardware implementations.

TABLE I. Accuracy of Softmax and various weighting polynomials.

	Top-1 Accuracy DeiT-Tiny
Softmax (pretrained, 300 epochs)	74.48
Softmax (post-training)	**74.96**
ReLU(x)	74.11
ReLU(x)^2	74.72
ReLU(x)^3	**74.74**
ReLU(x)^4	74.73

TABLE II. Comparisons between Squaremax and Softmax.

	Top-1 Accuracy	
	DeiT-Tiny	**Swin-Tiny**
Softmax (pretrained, 300 epochs)	74.48	81.15
Squaremax (train-from-scratch)	**75.60**	**81.16**
Squaremax (4-bit, post-training)	**75.62**	81.28
Squaremax (3-bit, post-training)	75.61	**81.32**
Squaremax (2-bit, post-training)	**75.62**	81.21
Squaremax (1-bit, post-training)	75.37	81.17

B. Hardware Implementation Results and Comparisons

The proposed Squaremax design has been implemented in Verilog and synthesized using Synopsys Design Compiler with a TSMC 40nm cell library. The synthesized netlist and the signal activity log are fed into Synopsys PrimeTime-PX for more accurate power estimation. TABLE III gives our implementation results and comparisons against previous arts. The I/O data format in the proposed design is 16-bit fixed-point. Our design can process 8 inputs in parallel and can handle an input vector of up to 8192 elements, which can be easily extended if necessary. Its operating frequency can be up to 1.67 GHz, which implies the peak throughput is 13.36 G/s. The area and power consumption is 20119 μm^2 and 9.57 mW, which suggests that our design exhibits an area efficiency of 664.049 G/mm^2 and a power efficiency of 1396.029 G/W.

To evaluate the performance of our design, three existing designs [6], [8] and [9] are selected for comparisons. They try to mimic the original Softmax function and use LUT-based methods to approximate exponential or logarithmic operations. In addition, all of them do not make extensive and aggressive approximations, such as rounding to the nearest power-of-two in [15] or discarding the use of the log unit in [4]. Though such approximations can effectively lower the hardware cost, they often result in a significant undesired accuracy loss.

From TABLE III, it is obvious that the proposed design outperforms the previous arts virtually in all aspects. First, the designs in [6] and [9] cannot process multiple input elements in parallel, which leads to a notably low throughput. Next, our design is well pipelined (e.g., two-stage multipliers are in use) and thus owns the fastest operating frequency. It is even faster than the design in [8] utilizing a 28nm technology. Besides, reported area efficiency (G/mm^2) and power efficiency (G/W) of every design are normalized (i.e., scaled) with respect to a 40nm technology to make comparisons fair among different designs. Again, the normalized area efficiency and power efficiency of our design is 664.049 G/mm^2 and 1396.029 G/W respectively, which are the highest values in comparisons. Moreover, the improvement in area efficiency against [6] and [9] is 112× and 322×, respectively. It suggests that our LUT-free design drastically boosts the area and power efficiency and can thus outperform those LUT-based designs easily.

V. CONCLUSION

In this paper, we propose a new activation function, named Squaremax, to be an alternative to widely-used Softmax. From the algorithm perspective, Squaremax and Softmax share a set of identical/similar key properties. From the implementation perspective, Squaremax is highly hardware-friendly since it requires no exponential operations and can thus be efficiently implemented in both software and hardware without the need of LUT-based approximations. In this paper, we also present a well-crafted LUT-free hardware design and implementation. Multiple inputs can be processed in parallel, computing resources are shared for design size reduction, a dynamic scaling approach is adopted for precision loss minimization, and a division-to-multiplication conversion is utilized to eliminate the need of divisions entirely.

Experimental results indicate that Squaremax achieves the same or even better accuracy than Softmax in a few popular Transformer-based models. Moreover, the proposed hardware implementation outperforms a set of previous designs in terms of maximum operating frequency, area efficiency, and power efficiency. Therefore, it is conclusive that Squaremax is indeed a competitive and promising alternative to Softmax in both software and hardware for deep learning applications.

TABLE III. Comparisons between the proposed design and previous studies.

	[6]	[8]	[9]	**Ours**
Technology	65nm	28nm	65nm	**40nm**
Data Width (w)	32	16	16	**16**
Parallelism (n)	1	8	1	**8**
Number of Input Elements (N)	N/A	N/A	Up to 4096	**Up to 8192**
Frequency (GHz)	1.0	1.64	0.5	**1.67**
Latency (cycles)	N/A	N/A	**7**	**6**
Throughput (G/s)	1.0	13.12	0.5	**13.36**
Area (μm^2)	444858	15926	640000	**20119**
Area Efficiency (G/mm^2)	2.248	823.810	0.78125	**664.049**
Normalized Area Efficiency (G/mm^2)	5.936	403.667	2.063	**664.049**
Power (mW)	333	N/A	0.82	**9.57**
Power Efficiency (G/W)	3.003	N/A	609.756	**1396.029**
Normalized Power Efficiency (G/W)	4.880	N/A	990.854	**1396.029**

REFERENCES

[1] L. Peng, S. Zheng, P. Li, Y. Wang, and Q. Zhong, "A Comprehensive Detection System for Track Geometry Using Fused Vision and Inertia," in *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–15, 2021.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019, pp. 4171–4186.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[4] I. Kouretas, and V. Paliouras, "Hardware Implementation of a Softmax-Like Function for Deep Learning," *Technologies*, vol. 8, no. 3:46, 2020.

[5] X. Dong, X. Zhu, and D. Ma, "Hardware Implementation of Softmax Function Based on Piecewise LUT," in *IEEE International Workshop on Future Computing (IWOFC)*, 2019, pp. 1–3.

[6] Q. Sun, Z. Di, Z. Lv, F. Song, Q. Xiang, Q. Feng, Y. Fan, X. Yu, and W. Wang, "A High Speed SoftMax VLSI Architecture Based on Basic-Split," in *IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2018, pp. 1–3.

[7] B. Yuan, "Efficient Hardware Architecture of Softmax Layer in Deep Neural Network," in *IEEE International System-on-Chip Conference (SOCC)*, 2016, pp. 323–326.

[8] D. Zhu, S. Lu, M. Wang, J. Lin, and Z. Wang, "Efficient Precision-Adjustable Architecture for Softmax Function in Deep Learning, " in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3382–3386, 2020.

[9] G. Du, C. Tian, Z. Li, D. Zhang, Y. Yin, and Y. Ouyang, "Efficient Softmax Hardware Architecture for Deep Neural Networks," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, pp. 75–80.

[10] N. A. Koca, A. T. Do, and C.- H. Chang, "Hardware-efficient Softmax Approximation for Self-Attention Networks," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2023, pp. 1–5.

[11] M. Wang, S. Lu, D. Zhu, J. Lin and Z. Wang, "A High-Speed and Low-Complexity Architecture for Softmax Function in Deep Learning," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2018, pp. 223–226.

[12] Y. Zhang, Y. Zhang, L. Peng, L. Quan, S. Zheng, Z. Lu, and H. Chen, "Base-2 Softmax Function: Suitability for Training and Efficient Hardware Implementation," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 9, pp. 3605–3618, 2022.

[13] 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)*, 2021, pp. 469–474.

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

[15] F. Spagnolo, S. Perri, and P. Corsonello, "Aggressive Approximation of the SoftMax Function for Power-Efficient Hardware Implementations," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1652–1656, 2022.

[16] OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, 2023.

[17] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[18] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10012–10022, 2021.