Hardware-efficient Softmax Approximation for Self-Attention Networks

Nazim Altar Koca*†, Anh Tuan Do* and Chip-Hong Chang†
* Institute of Microelectronics, A*STAR (Agency for Science, Technology and Research), Singapore
† School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore
Email: koca0001@e.ntu.edu.sg, doat@ime.a-star.edu.sg, echchang@ntu.edu.sg

2023 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-5109-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISCAS46773.2023.10181465

Abstract—**Self-attention networks such as Transformer have become state-of-the-art models for natural language processing (NLP) problems. Softmax function, which serves as a normalizer to produce attention scores, turns out to be a severe throughput and latency bottleneck of a Transformer network. Softmax datapath consists of data-dependent sequential nonlinear exponentiation and division operations, which are not amenable to pipelining and parallelism, nor can they be directly linearized for pretrained models without substantial accuracy drop. In this paper, we proposed a hardware efficient Softmax approximation which can be used as a direct plug-in substitution into pretrained transformer network to accelerate NLP tasks without compromising its accuracy. Experiment results on FPGA implementation show that our design outperforms vanilla Softmax designed using Xilinx IPs with 15x less LUTs, 55x less registers and 23x lower latency at similar clock frequency and less than 1% accuracy drop on main language benchmark tasks. We also propose a pruning method to reduce the input entropy of Softmax for NLP problems with high number of inputs. It was validated on CoLA task to achieve a further 25% reduction of latency.**

I. Introduction

Transformer was introduced in 2017 for machine translation tasks [1]. The notion of self-attention captures the global context relation within the input sequence, causing it to overtake the recurrent neural networks in natural language processing (NLP). Pre-trained transformer networks such as BERT [2], RoBERTa [3] and GPT-3 [4] further reshaped the field by breaking benchmark scores (GLUE [5], SQuAD [6], etc.) and became a model of choice for solving NLP problems.

A computation layer of Transformer consists of two subblocks called self-attention and feed forward network. Their underlying operations are a mix of linear matrix multiplication and non-linear operations like Softmax, layer normalization (LN) and GeLU. These mixed computations and large model size (e.g., 336 million parameters for BERT-large) make Transformer inefficient for deployment on edge devices. Recent hardware accelerators for Transformer aim to improve matrix multiplication operations [7] [8]. However, the actual run-time bottleneck is the Softmax operation when the number of inputs gets larger [9]. Softmax takes almost 30% or more of runtime for recent transformer models with 1024 or larger input length due to its non-linearity and data dependency. Unlike CNN, Softmax is used not only in the last layer of attention network. Hence, its approximation can lead to error escalation that has significant impact on the accuracy for NLP applications. There are two main challenges in Softmax hardware implementation. Firstly, exponentiation and division operations are expensive and they cannot be aggressively approximated without compromising the accuracy of Transformer. Secondly, it has three data-dependent loops which cannot be easily pipelined or parallelized to reduce latency.

In this paper, we proposed an efficient hardware design to approximate Softmax with minimal accuracy loss without retraining for NLP tasks. Our main contributions are: 1) our proposed design has eliminated exponentiation and division operations. The datapath consists of only simple adders and shifters; 2) it can be used as a drop-in replacement to further enhance the efficiency of pre-trained transformer based networks with negligible accuracy drop on main NLP benchmark tasks; 3) the proposed Softmax design outperforms state-of-the-art Softmax designs on FPGA implementation; 4) an input pruning method is also proposed at the expense of hardware resources to further reduce the latency of Softmax for NLP problems that have high number of inputs.

II. Softmax for CNN and Self-attention Networks: Motivation and Related Works

The core concept of Transformer that differentiates it from CNN is self-attention. In self-attention flow, three input matrices, Query (Q), Key (K) and Value (V), are obtained from the same input by applying linear transformation with different parameters. Q and K are used to obtain attention scores that indicate which part of the input is more relevant to the others. These scores are multiplied with V to obtain the output of self-attention units. The dot product of Q and K is scaled to avoid overflow and the input values of Softmax are masked out to prevent leftward information flow in the decoder before it is normalized to obtain the attention scores. A Softmax function is defined as follows:

$$\sigma(z_i) = \frac{e^{z_i - z_{max}}}{\sum_{j=1}^{N} e^{z_j - z_{max}}} \quad for\ i = 1, 2, \ldots, N \qquad (1)$$

It can be computed by Algorithm 1, where the iterative exponentiation and division calculations in Steps 7 and 10 are most hardware-intensive. There are three iterative computations in Softmax. The first loop calculates the maximum value of the input vector. This maximum value is required to scale down the inputs to avoid overflow errors in training. The second loop calculates the divisor of (1) and the last loop completes the normalization. Since each loop is dependent on the output of the previous loop, Softmax has limited pipelinability and parallelism. Besides, the latency and computational resources of each loop increases proportionally with the total number of inputs, N.

Unfortunately, efforts to accelerate attention based networks [8] [10] [11] focus mainly on the MAC operations and fully connected layers by using approximated-pruned hardware architectures and model reduction. The CPU and GPU inefficient softmax function that requires dedicated hardware optimization has been omitted at large. A few recently proposed

Algorithm 1: Softmax Function

```
input : $In(1), In(2), ..., In(N)$
output: $Out(1), Out(2), ..., Out(N)$
$In_{max} \leftarrow -\infty$
for $i \leftarrow 1$ to $N$ do
    $In_{max} \leftarrow max(In(i), In_{max}))$
end
$sum \leftarrow 0$
for $i \leftarrow 1$ to $N$ do
    $sum \leftarrow sum + e^{In(i)-In_{max}}$
end
for $i \leftarrow 1$ to $N$ do
    $Out(i) \leftarrow \frac{e^{In(i)-In_{max}}}{sum}$
end
```

quantized Transformers for lower precision computation [12] [13] [14] show the possibility of reducing certain non-linear operations to fixed point arithmetic without unduly hurting its performance.

Early hardware simplification of Softmax was based on LUT implementations [15]. Despite its ability to preserve the accuracy, it is still costly. More efficient hardware implementations for Softmax approximation have been proposed later. Majority of them were validated on CNNs and Multi-Layer Perceptron (MLP) networks for computer vision (CV) tasks, where Softmax is required only in the last layer of the classifier. In [16] and [17], substantial hardware resources have been saved by a smooth approximation of normalizing N vectors of logarithmic probabilities without under- or over-flow using the log-sum-exp operation [18]. In [19], a symmetric mapping (SM)-LUT implementation of base 2 Softmax is proposed. The symmetric power and log functions are exploited for efficient ASIC and FPGA implementations. However, log-sum-exp method escalates accuracy loss due to the cumulation of quantization errors in the attention layer [20]. A more aggressive Softmax approximation that leverages the error tolerance of the last layer of DNN was proposed in [21] to achieve good accuracy for CV tasks at very high frequency. However, when we applied the same Softmax approximation to the attention model, the network fails to distinguish the attention scores with greater than 40% accuracy drop on SST2 task. In [9], an ASIC implementation of base 2 Softmax in Transformer for NLP is proposed. While the latency has been reduced by hardware friendly max normalization algorithm, piece-wise linear reciprocal unit and expensive multiplier are required for the division operation. Furthermore, since, base 2 instead of base e operation is implemented, the network needs to be fine-tuned with the new loss function to minimize accuracy drop. Multipliers and divisor are eliminated in [20] by a memory-centric approximate computing approach. Two LUT-based methods are proposed in [20]. One is based on normalization of reciprocal exponentiation, and another by substituting division with 2D-LUT, but no physical implementation results were reported.

III. Softmax Approximation and Its Hardware Implementation

Quantizing the floating-point arithmetic to 8-bit integer precision for efficient CPU/GPU calculation of Softmax will escalate the accuracy drop of DNNs with attention-mechanism. Instead of 8-bit integer precision, our resource-accuracy trade-off study suggests that 16-bit fixed-point arithmetic with 5-bit integer and 11-bit fractional precision provides similar accuracy as full-precision high-performance Transformer model for the numbers of inputs encountered in commonly used NLP benchmarks, such as GLUE and SQuAD.

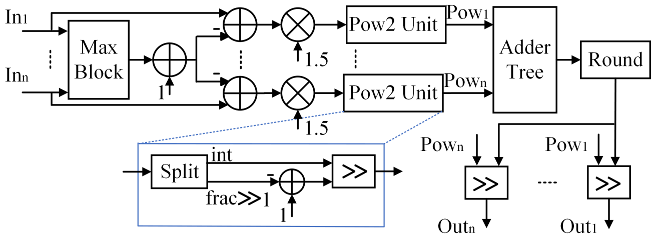


Fig. 1. Proposed Approximate Softmax Architecture.

Logarithmic transformation has been widely used to replace the slow and costly division and exponential operations by table lookup methods. However, it cannot be used directly to simplify quantized Softmax layer with high dependencies in the middle of self-attention models without severe accuracy loss. The division operation in Softmax limits the input values to between 0 and 1. While this normalization term can be skipped for the scoring function in classification problems [22], it is indispensable in attention as the outputs of Softmax are used as weighting constants for the matrix V. Division can be realized with binary shifters if the dividend is a fixed-point number and the divisor is a power-of-two integer. To eliminate the complex division operation, we round the sum of e^x values in the denominator of (1) to the nearest power-of-two integer value. Rounding is preferred to ceiling or flooring to minimize quantization error.

The domain and range of an exponential function is the set of all real numbers and positive real numbers, respectively. This makes it challenging to simplify its implementation by linear approximation. To overcome this problem, we re-express the power function as a binary exponentiation:

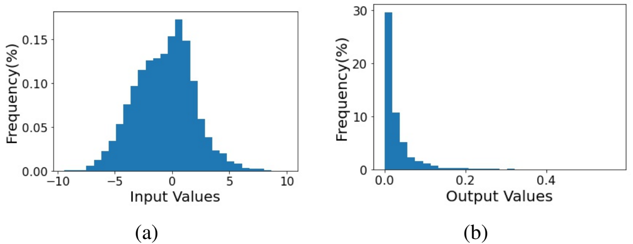
$$x^y = 2^{y \cdot log_2 x} = 2^u \cdot 2^v \tag{2}$$

where u and v represent the integer and fractional parts, respectively of $y \cdot log_2 x$.

In the original Softmax equation (1), the base x of the power function in (2) is the Euler number e. To obtain v and u, the input y needs to be multiplied by the constant $log_2 e$. This multiplication can be eliminated by replacing the base e of (1) with base 2 [9]. However, this Softmax approximation for a pre-trained model will result in nontrivial accuracy drops, e.g., the accuracy drop exceeds 7% on SQuAD problem. To restore the accuracy, the model needs to be retrained with the new base value and the corresponding loss function. As this scaling is needed to amplify the difference between inputs, $log_2 e$ can be rounded to 1.5 to preserve the model accuracy. Consequently, only an adder and hard-wired shifter are needed to produce the scaled input y (as $1.5 \times y = y + y/2$) without the need to fine-tune the pretrained model. Since v is a fractional value, 2^v in (2) can be linearized as its output lies between 1 and 2. The piece-wise linear approximation of the secant to 2^v is then scaled by a factor of 2^u. After the max normalization, u and v values are negative. For $v \in [-1, 0]$, 2^v can be approximated by a linear function $v/2 + 1$, which can be implemented with only an adder as $v/2$ can be obtained by a hardwired right shift

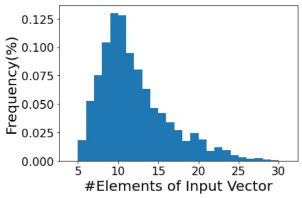
Algorithm 2: Proposed Approximate Softmax

```
   input : $In(1), In(2), ..., In(N)$
   output: $Out(1), Out(2), ..., Out(N)$
1  $In_{max} \leftarrow max(In(1), In(2), ..., In(N))$
2  $In_{max} \leftarrow In_{max} + 1$
3  $sum \leftarrow 0$
4  for $i \leftarrow 1$ to $N$ do
5      $sub(i) \leftarrow In(i) - In_{max}$
6      $mul(i) \leftarrow sub(i) \cdot 1.5$
7      $pow(i) \leftarrow (1 - frac(mul(i)) >> 1) >> int(mul(i))$
8      $sum \leftarrow sum + pow(i)$
9  end
10 $norm \leftarrow round(sum)$
11 for $i \leftarrow 1$ to $N$ do
12     $Out(i) \leftarrow pow(i) >> norm$
13 end
```



(a) (b)

Fig. 2. Distribution of (a) SST-2 Input Values, (b) SST-2 Output Values.



#Inputs After Pruning	Percentage In Total(%)	#Cycles Before Pruning	#Cycles After Pruning	Cycles Reduction (%)
0-8	15	12	7	42
8-16	68	12	9	25
16-24	15	12	11	8
24-32	2	12	13	-8

(a) (b)

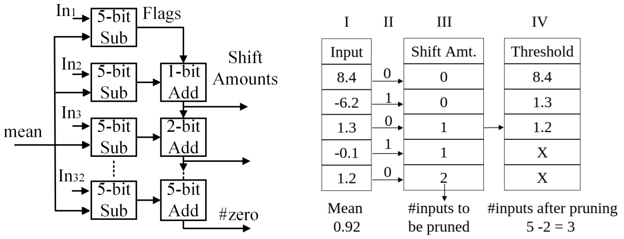
Fig. 3. (a) Distribution of the Number of Inputs of CoLA After Prunning Insignificant Inputs (b) Results of Input Prunning on CoLA.

at no logic cost. Its multiplication with 2^u can be realized with only a right shifter.

The approximation algorithm is depicted in Algorithm 2. The hardware architecture for Algorithm 2 is shown in Fig. 1, where $n \leq N$ is the number of inputs that can be processed parallelly. The inputs are first quantized into 16-bit fixed point numbers. The maximum value between the inputs is found by the max block (Line 1 of Algorithm 2), which is implemented with a 5-bit comparator tree. Only the integer parts of the inputs are compared to find the maximum value and a one is added to it to round it up (Line 2). An extra comparator is used to compare the max values of different batches. These comparators are realized with subtractors. The normalized values after max subtraction (Line 5) are first scaled by 1.5 (Line 6) and then separated into integer and fractional parts. The fractional part is used for linear approximation of the secant. The result is shifted right by the integer part (Line 7). An adder tree is used to calculate the sum of the outputs of the current batch of exp values (Line 8). An accumulator is added after the adder tree to obtain the final sum of all inputs of the batch. This accumulator output is rounded to the nearest power-of-two value to obtain the normalization factor (Line 10). Only 5 most significant bits (excluding the sign bit) are used for rounding since all the outputs of the adder are positive and 1 most significant fraction bit is sufficient to round the number correctly. The rounding circuit is implemented using multiplexers. Finally, each output of the power function is shifted right by the normalization factor in Line 12.

Modern NLP models like GPT-3 [4] usually have a large number of inputs, N. The latency of Softmax calculation increases proportionally with N as the number of inputs that can be processed at any one time n is limited by the hardware cost and energy budget [23]. It is noted that while the input of Softmax exhibits an approximately normal distribution, the output has a right-skew distribution with a long tail of near zero values, as shown in Fig. 2(a) and 2(b), respectively on SST2 task. There are two main reasons for this phenomenon. Firstly, inputs with smaller number of tokens are padded with zeros to make the input length constant. Secondly, human language has high redundancy as reported in [11] [24]. By eliminating the redundant and unimportant inputs, the latency of Softmax can be reduced.

Since Softmax is a monotonic function, the near zero output values in Fig. 2(b) comes from those smallest input values. These small inputs can be pruned by keeping only the $K < N$ biggest elements or elements larger than an empirically determined threshold value. The former method can filter out more trivial inputs without compromising accuracy at the expense of high computational overheads in searching for the K highest values. To reduce the hardware complexity, we choose the latter method using the mean input value as the threshold. As an illustration, we apply this to the Corpus of Linguistic Acceptability (CoLA) task from the GLUE benchmarks. We fine-tune the Transformer model with 32 input tokens by applying padding and truncation. Our baseline Softmax implementation takes $n = 8$ inputs at a time, which requires 12 cycles (4 for max, 4 for divisor and 4 for final result) to calculate all the 32 outputs. The distribution of the inputs after pruning below-mean values is shown in Fig. 3(a). 15% of the inputs that have 0-8 tokens require only 7 cycles (4 for max, 1 for mean, 1 for divisor, and 1 for final result) to calculate, which amounts to 42% latency reduction, and 68% of the inputs that have 8-16 tokens require 9 cycles (4 for max, 1 for mean, 2 for divisor, and 2 for final result), which amounts to 25% latency reduction. Since only 2% of the inputs have 24-32 tokens, the latency overhead due to mean thresholding is negligible compared with at least 25% saving by 82% of the inputs, as summarized in Fig. 3(b). To obtain the mean, the inputs are accumulated in parallel as the max value is calculated. There is no extra hardware required as the idle adder tree in Fig. 1 can be reused in this step. In the next cycle, the mean value can be obtained by shifting the sum value right by $log_2 N = 5$ bits. Inputs lower than the mean value are discarded. The input filtering circuit is shown in Fig. 4(a). To reduce the hardware cost, only the integer parts of the input value and the mean value are compared using 5-bit subtractors. The subtractor outputs 1 (negative) if the input is lower than the mean and 0 (non-negative) otherwise. To preserve the input order, the inputs are orderly shifted to replace the pruned inputs. This can be accomplished by adding the outputs of the subtractors in an adder chain. Each input is then shifted by an amount equal to the output of the adder corresponding to its position. This input pruning can be completed in one cycle. Fig. 4(b) illustrates the operation with a numerical example. The mean value of the 5 inputs



(a) (b)

Fig. 4. (a) Input Pruning Selection Circuit with Comparators and Counters (b) A Numerical Example of Pruning Algorithm.

is calculated to be 0.92 in Step I. The sign bits of the five subtractor outputs are shown in Step II. The outputs of the counters (adders) are shown in Step III. The inputs in Step 1 are shifted upward by the number of positions corresponding to the counter outputs in Step III. The final buffer content are shown in Step IV. The inputs marked X in Step IV are set to '0'. The last counter value in Step III indicates the number of inputs that are smaller than the mean. By subtracting it from the initial number of inputs, the final number of preserved inputs can be calculated.

IV. Results

We test the accuracy of the most widely used Transformer model BERT from Huggingface libraries [25] by substituting the Softmax of its pretrained model with our proposed Softmax approximation on GLUE and SQuAD benchmark problems. Table I shows the results of the original 32-bit floating-point Softmax (Precise), our proposed linearized Softmax without input pruning (Ours I) and with input pruning (Ours II) for different tasks. The results show that our proposed approximation methods I and II introduce less than 1% accuracy drop for all GLUE tasks and SQuAD without fine tuning. Our method actually increases CoLA score by 2%. Introducing an approximation may have helped BERT model to generalize better on the validation set in this case.

To compare with these FPGA-based Softmax modules, we described our proposed approximated Softmax method I in Verilog HDL and implemented it on Xilinx FPGA chip of ZCU102 Evaluation Board. The hardware resources, operation frequency and latency are compared with the FPGA implementation results reported in existing Softmax approximation with different number of inputs (n) and input bitwidth (w) in Table II. All w are fixed point except Xilinx FP $w = 32$ is floating point. LUT and REG refer to the number of LUTs and registers used, respectively. F_{max} is the maximum clock frequency that the design can run. As Softmax cannot be easily pipelined into approximately equal delay stages, increasing F_{max} by pipelining will increase the total latency and register cost. Latency is crucial for NLP tasks like chatbots and translation machines for fluent communication. Although the design of [21] can operate at 588 MHz, the latency is 22.1 ns. Our proposed design can operate at 476 MHz with the shortest latency among those designs that have latency reported. Since none of the existing FPGA implementations of Softmax was tested on pre-trained self-attention networks on GLUE and SQuAD tasks, we also implemented the 32-bit floating-point Softmax using Xilinx Integrated IP libraries (Xilinx FP) on the same FPGA platform for comparison. The accuracies of this full-precision Softmax for NLP tasks are given by the 'precise' row in Table I. Our design requires ∼15x less LUTs, ∼55x less flip flops and no DSP unit (as opposed to 38 DSPs in Xilinx FP) compared to Xilinx FP yet our design computes ∼23x faster in terms of latency at similar maximum clock frequency with less than 1% accuracy drop when it is incorporated into BERT for NLP tasks. Since the results reported for existing Softmax designs are implemented based on different n and w, for a more informative comparison, we have added the following figure of merit in the last column of Table II:

TABLE I

Accuracy results of our softmax approximation methods for BERT on NLP tasks

Task	SQuAD	SST2	QNLI	MNLI	MRPC	QQP	RTE	CoLA
Precise	88.22	92.43	91.54	84.45	84.55	90.90	72.56	53.38
Ours I	88	92.08	91.21	84.06	84.31	90.60	72.20	55.21
Ours II	88.01	92.08	91.01	84.09	83.57	90.61	72.20	55.47

TABLE II

Resource and Performance Comparison of FPGA Implementations of Softmax Accelerators

Paper	w	n	Area (LUT, REG)	F_{max} (MHz)	Latency (ns)	FOM
ICSICT'18[15]	32	1	17870,16400	150	NA	0.140
ISCAS'20[16]	16	1	2229,224	154	NA	1.004
TCAS-II'20[17]	16	8	1580,1800	500	NA	18.934
TCAS-II'22[21]	16	1	128,97	588	22.1	41.813
TCAS-I'22[19]	16	10	1476,698	500	NA	36.798
Xilinx FP	32	8	13254,18664	435	232.3	3.488
Ours I	16	8	909,333	476	**10.5**	**49.056**

$$FOM = \frac{f_{clk}(MHz) \times n \cdot w}{LUT + REG} \tag{3}$$

Our design has the highest *FOM* among all designs in comparison. We also implemented our proposed design II for 32 inputs task. It consumes 6319 LUTs and 1545 registers on the same FPGA. The average latency has been reduced by another 25% over design I for this task.

V. Conclusion

In this paper, we proposed a new approximate Softmax implementation to accelerate attention networks with minimal accuracy degradation. The proposed design can be implemented on hardware without expensive dividers, multipliers and LUTs. By replacing complex non-linear operations and multiple table lookups with reduced and simpler datapath elements, the latency has been reduced significantly. The FPGA implementation of our proposed Softmax design achieves the highest throughput per FPGA resource among existing FPGA implementations of Softmax. More importantly, we have validated its substitution into pretrained transformer model incurs no more than 1% accuracy drops on popular NLP tasks. Our input pruning can further squeeze another 25% speedup for CoLA with 32 input tokens.

VI. Acknowledgement

This research is partially supported by Programmatic grant no. A18A6b0057 (SpOT-Lite), Singapore RIE2020 Advanced Manufacturing and Engineering domain.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[3] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available: http://arxiv.org/abs/1907.11692

[4] 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. M. 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," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[5] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," *CoRR*, vol. abs/1804.07461, 2018. [Online]. Available: http://arxiv.org/abs/1804.07461

[6] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," *CoRR*, vol. abs/1606.05250, 2016. [Online]. Available: http://arxiv.org/abs/1606.05250

[7] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee, and D.-K. Jeong, "Aˆ3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 328–341.

[8] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 97–110.

[9] 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.

[10] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 692–705.

[11] H. Peng, S. Huang, S. Chen, B. Li, T. Geng, A. Li, W. Jiang, W. Wen, J. Bi, H. Liu, and C. Ding, "A length adaptive algorithm-hardware co-design of transformer on fpga through sparse attention and dynamic pipelining," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1135–1140. [Online]. Available: https://doi.org/10.1145/3489517.3530585

[12] Y. Lin, Y. Li, T. Liu, T. Xiao, T. Liu, and J. Zhu, "Towards fully 8-bit integer inference for the transformer model," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, ser. IJCAI'20, 2021.

[13] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," *International Conference on Machine Learning*, 2021.

[14] T. Yang, D. Li, Z. Song, Y. Zhao, F. Liu, Z. Wang, Z. He, and L. Jiang, "Dtqatten: Leveraging dynamic token-based quantization for efficient attention architecture," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 700–705.

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

[16] Y. Gao, W. Liu, and F. Lombardi, "Design and implementation of an approximate softmax layer for deep neural networks," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[17] D. Zhu, S. Lu, M. Wang, J. Lin, and Z. Wang, "Efficient precision-adjustable architecture for softmax function in deep learning," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3382–3386, 2020.

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

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

[20] I. Vasyltsov and W. Chang, "Efficient softmax approximation for deep neural networks with attention mechanism," *CoRR*, vol. abs/2111.10770, 2021. [Online]. Available: https://arxiv.org/abs/2111.10770

[21] F. Spagnolo, S. Perri, and P. Corsonello, "Aggressive approximation of the softmax function for power-efficient hardware implementations," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1652–1656, 2022.

[22] R. S, "Reduced softmax unit for deep neural network accelerators," *CoRR*, vol. abs/2201.04562, 2022. [Online]. Available: https://arxiv.org/abs/2201.04562

[23] Z. Wei, A. Arora, P. Patel, and L. John, "Design space exploration for softmax implementations," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2020, pp. 45–52.

[24] X. He, I. Keivanloo, Y. Xu, X. He, B. Zeng, S. Rajagopalan, and T. Chilimbi, "Magic pyramid: Accelerating inference with early exiting and token pruning," *CoRR*, vol. abs/2111.00230, 2021. [Online]. Available: https://arxiv.org/abs/2111.00230

[25] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *CoRR*, vol. abs/1910.03771, 2019. [Online]. Available: http://arxiv.org/abs/1910.03771