2022 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-8485-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCAS48785.2022.9937659

An Efficient Hardware Accelerator for Sparse Transformer Neural Networks

Chao Fang[1], Shouliang Guo[1], Wei Wu[1], Jun Lin[1], Zhongfeng Wang[1], Ming Kai Hsu[2], Lingzhi Liu[2]
[1]School of Electronic Science and Engineering, Nanjing University, Nanjing, China
[2]Kuaishou Technology, Palo Alto, CA, USA
Email: fantasysee@smail.nju.edu.cn, zfwang@nju.edu.cn

***Abstract*—Transformers have been an indispensable staple in deep learning. However, it is challenging to realize efficient deployment for Transformer-based model due to their substantial computation and memory demands. To address this issue, we present an efficient sparse Transformer accelerator on FPGA, namely STA, by exploiting N:M fine-grained structured sparsity. Our design features not only a unified computing engine capable of performing both sparse and dense matrix multiplications with high computational efficiency, but also a scalable softmax module eliminating the latency from intermediate off-chip data communication. Experimental results show that our implementation achieves the lowest latency compared to CPU, GPU, and prior FPGA-based accelerators. Moreover, compared with the state-of-the-art FPGA-based accelerators, it can achieve up to 12.28× and 51.00× improvement on energy efficiency and MAC efficiency, respectively.**

I. Introduction

Transformer-based networks are a formidable force in deep learning [1]. Tremendous impact in many fields, such as neural machine translation [2], language understanding [3], and image processing [4], has been made since the invention of Transformers. Nevertheless, the impressive performance of Transformers comes with heavy computing and memory costs, which become a significant barrier to the efficient deployment of Transformer-based applications.

Previous work in [5] proposed a dense systolic array accelerator on FPGA along with a partitioning scheme to realize low-latency inference of Transformers. Moreover, the approaches in [6], [7] exploited block-circulant matrix-based weight representations for Transformer acceleration. However, all prior FPGA-based accelerators are implemented on high-end FPGAs with overwhelming use of hardware resources and high power consumption, which can hardly meet the demands of efficient inference of Transformers.

In this paper, we present an efficient sparse Transformer accelerator, namely STA, on FPGA by fully utilizing the N:M sparsity pattern [8], [9]. It reduces number of operations and size of memories for Transformers, relieving the burden of computing and storage. Hence, STA realizes efficient Transformer inference while retaining high prediction accuracy.

In summary, the main contributions of this paper are:

- We propose a dedicated hardware architecture, namely STA, for efficient inference of Transformers by exploiting N:M fine-grained structured sparsity.

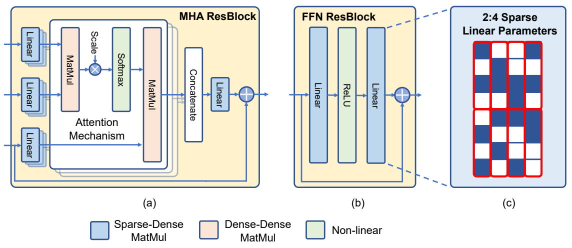

Fig. 1. The operations in (a) the MHA ResBlock and (b) the FFN ResBlock under N:M sparsity pattern. Both Resblocks are the key structures of the Transformer. (c) An illustration of 2:4 sparse parameters in a linear layer.

- We design an efficient diverse matrix multiplication engine (DMME) that unifies both dense and sparse matrix multiplication (MatMul) operations, and a scalable softmax module to avoid frequent off-chip memory accesses.
- We implement STA and evaluate its performance on the Arria 10 SX660 platform. Experimental results show that STA achieves not only the lowest latency, but also 1.31∼1.92× power reduction, 1.27∼12.28× energy efficiency improvement, and 3.45∼51.00× MAC efficiency gain, respectively, compared to prior FPGA solutions.

II. Background and Motivation

The key structures of a Transformer [10] are characterized by a multi-head attention (MHA) residual block (ResBlock) and a position-wise feed-forward network (FFN) ResBlock. Fig. 1 (a) and (b) illustrate the inner structures of MHA and FFN ResBlocks, respectively. The inputs and outputs of the FFN ResBlock are connected by a residual connector. Two linear transformation modules along with a ReLU activation function are also inside the FFN ResBlock. The structure of an MHA ResBlock is more complicated. The inputs of MHA ResBlock are split into multiple parallel heads with corresponding linear projections applied to them. Their results are fed as inputs into the attention mechanism in parallel, and finally, the results of the attention heads are concatenated together and passed into a linear layer to obtain the output linear projection. Note that the attention mechanism is totally different from the linear layer, performing parameter-free MatMuls. The residual connector of the MHA ResBlock is organized in the same way as for the FFN ResBlock.

Using N:M sparsity pattern can compress many network models without impacting their output accuracy [8], [9]. Under N:M sparsity pattern, the parameter matrix is divided into multiple groups. Each group has M elements and contains N nonzero elements at most. An example of the parameter matrix with 2:4 sparsity pattern is presented in Fig. 1. Only the 2 nonzero values in each group of 4 values are required to be stored and computed. In this paper, we focus on how to design an efficient Transformer hardware accelerator with a high computational efficiency using the N:M sparsity pattern. As shown in Fig. 1, there are both sparse-dense and dense-dense MatMuls in the key structures of a Transformer with N:M sparsity pattern. Thus, it is crucial to design an efficient computing engine to support diverse types of MatMuls. Moreover, the softmax function is embedded in the MHA ResBlock of Transformers. To avoid frequent off-chip memory accesses of intermediate results for softmax operators, it is essential to implement a scalable softmax module on the accelerator.

III. Hardware Architecture

A. Overall Architecture

The system-level view of STA is shown in Fig. 2. The whole system is an ARM+FPGA heterogeneous architecture, and can be divided into three parts: the software program on ARM, the external memory on the FPGA board, and the custom accelerator on the FPGA chip.

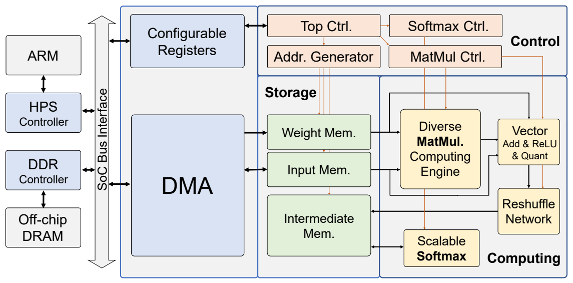

Fig. 2. Overall Architecture of STA.

The software part communicates with the FPGA via the Avalon bus, passing control signals and data to be calculated. At the initialization stage, the software program on ARM sends configurable parameters of the Transformer structure to the FPGA. The inputs and weights are loaded in DRAM in advance of computation on the FPGA accelerator. The software receives corresponding results in DRAM when the hardware accelerator on FPGA turns to ready. The external memory together with the FPGA chip stores all the model parameters and input samples.

The custom accelerator is made up of three major function blocks, including computing, storage, and control. The computing blocks consist of a diverse MatMul engine, a scalable softmax module, a vector unit, and a data reshuffle network. As shown in Fig. 1, DMME supports diverse matrix computations with high computational efficiency. The scalable softmax module performs the softmax operation in MHA sublayers of the Transformer. The vector unit takes charge of operations with low computational density including bias addition, residual addition, and the ReLU activation. The reshuffle network, which is used after every MatMul computing processes and before data concatenation, reorders the temporary results before writing them back to the intermediate on-chip memory. As for on-chip storage, it can be partitioned into three parts, including the weight memory, the input memory, and the intermediate memory. The weight and input memory store the parameters and input data of Transformers from the off-chip DRAM, respectively. The results are also written back to the input memory, and passed to the DRAM. All the temporary results in a ResBlock will be stored in the intermediate memory with no communication to the external memory.

B. Diverse Matrix Multiplication Engine (DMME)

DMME unifies both dense-dense and sparse-dense MatMuls in N:M sparse Transformers with a high computational efficiency. When it performs sparse-dense MatMuls, it only loads nonzero weight parameters and selects corresponding activations to compute, thereby improving the computational efficiency.

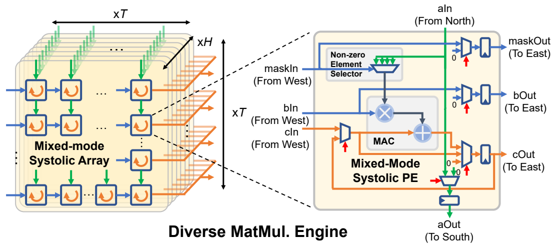

Fig. 3. The architecture of DMME. Mixed-mode PEs capable of handling both dense and sparse operations are arranged in a combinational grid to form a 2D systolic array. Multiple arrays are packed into the 3D DMME utilizing the parallelism from the diverse MatMuls.

The architecture of DMME is illustrated in Fig. 3. It is a two-level hierarchy design with a full exploration of parallelism inside the MatMuls of N:M sparse Transformers. DMME consists of H parallel $T \times T$ systolic arrays, every one of which can efficiently perform both dense-dense and sparse-dense MatMuls in a time-division multiplexing manner. The capability of performing dense-dense and sparse-dense MatMuls comes from the inner mixed-mode systolic PE. It is composed of a MAC, a non-zero element selector, multiple multiplexers, and registers. The MAC accepts two 16-bit inputs to perform multiplication, and then accumulates the result with the local 32-bit output partial sum. The non-zero element selector, only being activated in the sparse-dense MatMul mode, selects the proper activation according to the input bitmask. The multiplexers and registers are used for datapath selection and temporary data storage, respectively.

Fig. 4 illustrates how DMME supports both dense-dense and sparse-dense MatMuls in a computation-efficient manner. It is presented using the 1:2 sparsity pattern for simplicity.

In Fig. 4 (a), the mixed-mode systolic array performs dense-dense MatMuls with the same computational efficiency as the computing array in [5]. When it comes to sparse-dense MatMuls, the redundant pruned computations can be skipped by the mixed-mode systolic array as shown in Fig. 4 (b). There is no redundant computation in DMME when performing sparse-dense MatMul, which improves the MAC utilization and thereby achieves a reduction of execution cycles.

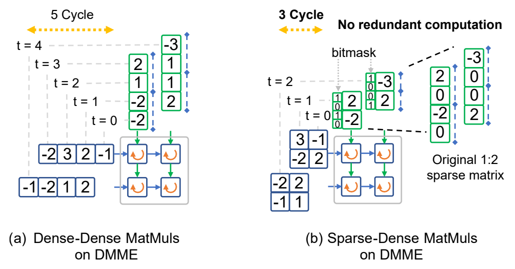
5 Cycle
3 Cycle
No redundant computation
bitmask
Original 1:2 sparse matrix
(a) Dense-Dense MatMuls on DMME
(b) Sparse-Dense MatMuls on DMME

Fig. 4. Computational manner of DMME under different modes: (a) dense-dense MatMuls and (b) sparse-dense MatMuls.

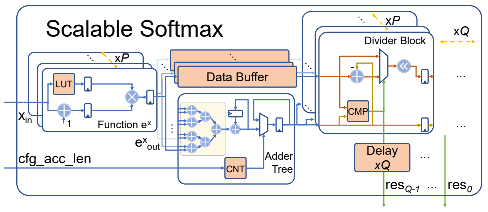

Fig. 5. The scalable softmax architecture.

C. Scalable Softmax Module

The scalable softmax module can perform softmax function of arbitrary length. It keeps all the intermediate results fully local, avoiding off-chip data communication. Fig. 5 presents the details of our proposed scalable softmax architecture. It consists of three major parts: an area-efficient exponential function, a partial sum accumulator, and a scalable divider. The exponential function is approximated using a lookup table combined with a first order Taylor expansion as shown in Fig. 5. An exponential operator can be implemented using only one multiplier and one adder. The configurable partial sum accumulator can adapt to input vectors of various lengths, which improves the flexibility of the hardware. To reduce the latency of the division, we design a highly parallel divider by cascading multiple divider blocks with pipelines, where a divider block is merely composed of subtractors and shifters. The precision of the divider can be adjusted by the pipeline length. The fully pipelined softmax architecture can overlap the division of the previous vector and the exponent operation of the current vector. As shown in Fig. 5, there are two adjustable parameters, P and Q, of the architecture, where P denotes the parallelism of the architecture, and Q represents the pipeline depth, as well as the output precision. The P-parallel inputs are fed into the softmax module. The exponent outputs are not only accumulated using an adder tree, but also temporarily stored in the data buffer. Once the accumulation process is done, the divider module accepts both accumulated results and exponent outputs to perform Q-level pipelined division.

IV. Experimental Results

A. Accuracy w.r.t. Model Sparsity

Model accuracy is firstly evaluated with various N:M configurations to determine the most efficient hardware implementation. In our experiments, a diverse set of low-resource translation pairs from the TED Talks [11] and the IWSLT'15 [12] corpora are selected as the datasets. We trained on these datasets with the Transformer base model in [10]. Our dense baseline Transformer is trained using the approach in [13]. All the sparse models are finetuned for 40 epochs using the SR-STE method [9] based on the dense baseline for a fair comparison. BLEU [14], the commonly used metric, is applied to evaluate our model accuracy. The higher the BLEU score, the higher the model accuracy. As shown in Table I, the performance of sparse Transformer has merely a negligible loss even with a sparsity up to 75% compared with the dense baseline. It is also observed that if we fix N to 1, we can achieve similar performance to the higher N under the same sparsity. Therefore, to avoid costly hardware overhead, we select 1:4 as the N:M configuration for further hardware performance evaluation.

TABLE I
BLUE scores of Transformer with various sparsities

Sparsity		50.00%		75.00%		87.50%		93.75%	
N:M		1:2	2:4	1:4	2:8	1:8	2:16	1:16	2:32
BLUE (Loss)	En→Vi	28.14 (-0.08)	28.19 (-0.03)	27.70 (-0.52)	27.81 (-0.41)	27.23 (-0.99)	27.37 (-0.85)	25.81 (-2.41)	25.73 (-2.49)
	Es→Pt	32.45 (+0.12)	32.14 (-0.19)	32.29 (-0.04)	32.22 (-0.11)	31.88 (-0.45)	32.18 (-0.15)	31.63 (-0.70)	31.65 (-0.68)
	Pt→En	42.50 (+0.41)	42.31 (+0.22)	41.82 (-0.27)	42.07 (-0.02)	41.15 (-0.94)	41.53 (-0.56)	39.62 (-2.47)	39.59 (-2.50)
	Ru→En	27.06 (-0.26)	27.03 (-0.29)	26.03 (-1.29)	26.30 (-1.02)	24.46 (-2.86)	25.31 (-2.01)	22.43 (-4.89)	21.96 (-5.36)

TABLE II
FPGA Resource Utilization

	ALMs	DSPs	M20Ks
Available	251,680	1,032	2,131
Used	188,072	520	1,549
Utilization	74.73%	30.82%	72.69%

B. Hardware Performance

The overall hardware system is synthesized with Quartus Prime 18.1 using SystemVerilog. The platform of our design is an Intel Arria 10 SX660 SoC device. According to the selected 1:4 sparsity, STA-4 is implemented with the following configurations, which is determined by a cycle-accurate simulator for priori performance evaluation. The (H, T) of DMME is (4, 16), and therefore there are 1024 MACs in STA-4. In the softmax module, P and Q are 8 and 16, respectively. Table II presents the resource utilization of our STA-4, which

TABLE III
COMPARISON OF STA WITH PREVIOUS WORKS AND COMMERCIAL PRODUCTS

Platform	CPU	GPU		FPGA				
	i9-9900X	Jetson Nano	RTX 2080 Ti	SOCC'20 [5]	ISLPED'20 [6]	ISQED'21 [7]	Our work	
							$N{:}M$=1:4 (STA-4)	$N{:}M$=1:8 (STA-8)
Chip	Skylake	Tegra X1	TU102	XCVU13P	XCVU9P	XCU200	Arria 10 SX660	
Technology	14 nm	20 nm	12 nm	16 nm	16 nm	16 nm	20 nm	
Frequency	3.50 GHz	640 MHz	1.35 GHz	200 MHz	-	-	200 MHz	
# MAC units	-	-	-	4096	$\sim$ 5647	$\sim$ 3368	1024	
Bit Precision	FP-32	FP-32	FP-32	FIX-8	FIX-16	-	FIX-16	
Test Network	Shallow Transformer							
Latency (ms)	2.17	16.24	1.70	0.25	2.94	0.32	0.28	0.21
Batch-1 Throughput (GOP/s)	50.69	6.77	64.71	440.00	75.34	343.75	392.85	523.81
Power (W)	165.00	7.56	250.00	16.70	22.45	-	11.68	12.73
Energy Efficiency (GOP/J)	0.31	0.90	0.26	26.35	3.35	-	33.64	41.15
MAC Efficiency (GOP/s/unit)	-	-	-	0.11	$\sim$ 0.01	$\sim$ 0.10	0.38	0.51

is reported based on the post-implementation summaries from Quartus Prime 18.1. It shows that ALMs and M20Ks of STA-4 dominate the resource consumption. That is the trade-off to keep all the intermediate computation values fully local and eliminate frequent DRAM access for efficient inference.

C. Cross-platform Comparison

For cross-platform comparison, the hardware setup to execute the Transformer inference tasks is as follows. The CPU result is measured using Intel(R) i9-9900X CPU @ 3.50GHz, and the GPU results are measured using a Jeston Nano edge computing device [15], and an NVIDIA RTX2080Ti card, respectively. Both CPU and GPU results are evaluated under the PyTorch framework. The previous FPGA-based work [5] utilizes up to 4096 MACs with a 64 $\times$ 64 systolic array, while STA-4 utilizes merely a quarter of MACs of that to achieve comparable inference latency and much higher computational efficiency. Besides STA-4, we further build STA-8 with an $N{:}M$ configuration of 1:8 for those applications that care about latency and power much more than accuracy.

Fig. 6 shows the idealized performance speedup of different hardware platforms normalized to the CPU. Here we consider single-batch processing time in all the MHA and FFN ResBlocks of these Transformer-based models. The selected models target different applications. TinyBERT [16] is a lightweight BERT model [17] for multiple language tasks. Dino [18], a tiny vision Transformer, can be the backbone for many computer vision tasks. Transformer-base model [10] is the classic one for the neural machine translation task. As shown in Fig. 6, STA-4 can achieve comparable latency with the prior high-end FPGA solution [5]. Moreover, compared to all other solutions in Fig. 6, STA-8 can achieve the lowest inference latency, which is 1.27$\sim$1.49$\times$ much faster than the prior FPGA solution [5].

Table III presents a fair performance comparison without batching on various platforms. The shallow Transformer model in [6], [7] is used for a fair evaluation. Latency, throughput, power, energy efficiency and MAC efficiency are key metrics for applications, and thus used for performance evaluation. The comparison is benchmarked on CPU, GPUs, prior cutting-edge FPGA solutions, and STA. The results show that STA-8 achieves the lowest latency compared to all other solutions in Table III. It has at least 8$\times$ lower latency in comparison with CPU and GPU platforms, especially 77$\times$ faster than

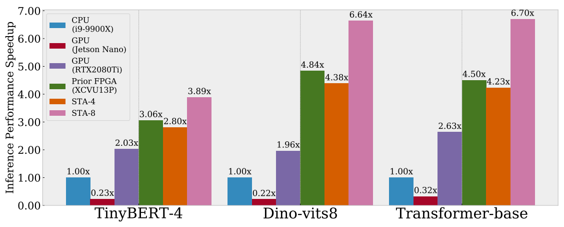

Fig. 6. The speedup of Transformer-based models on various platforms.

the popular edge computing device, Jetson Nano. In terms of throughput, STA-4 and STA-8 can achieve 392.85 and 523.81 GOP/s, respectively. Compared with prior FPGA solutions, STA-4 achieves 1.43$\sim$1.92$\times$ power reduction, 1.27$\sim$10.04$\times$ energy efficiency improvement, 3.45$\sim$38.00$\times$ MAC efficiency gain, respectively. Moreover, STA-8 achieves 1.31$\sim$1.76$\times$ power reduction, 1.56$\sim$12.28$\times$ energy efficiency improvement, 4.64$\sim$51.00$\times$ MAC efficiency gain, respectively. These results demonstrate the superior performance of STA for efficient Transformer acceleration.

V. CONCLUSION

In this work, a novel hardware accelerator for sparse Transformer, namely STA, is presented to realize efficient Transformer-based model deployment. STA features a unified 3D computing engine, which achieves a high computational efficiency for both sparse and dense matrix multiplications. It also features a scalable softmax module eliminating off-chip data communication for intermediate results. Experimental results demonstrate that STA significantly outperforms GPUs and the prior FPGA-based accelerators in terms of latency, throughput, power, energy efficiency, and MAC efficiency, showing its promising potential on those applications using Transformer-based models.

ACKNOWLEDGMENT

The authors would like to sincerely thank all reviewers and Kuaishou Technology for their feedback on improving this work. This work was supported in part by the National Natural Science Foundation of China under Grant 62174084, 62104097 and in part by the High-Level Personnel Project of Jiangsu Province under Grant JSSCBS20210034, the Key Research Plan of Jiangsu Province of China under Grant BE2019003-4.

REFERENCES

[1] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, “Efficient Transformers: A Survey,” *arXiv preprint arXiv:2009.06732*, 2020.

[2] K. Song, K. Wang, H. Yu, Y. Zhang, Z. Huang, W. Luo, X. Duan, and M. Zhang, “Alignment-Enhanced Transformer for Constraining NMT with Pre-specified Translations,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 34, no. 05, 2020, pp. 8886–8893.

[3] J. D. M.-W. C. Kenton and L. 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.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” in *International Conference on Learning Representations (ICLR)*, 2021.

[5] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, “Hardware Accelerator for Multi-Head Attention and Position-Wise Feed-Forward in the Transformer,” in *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*, 2020, pp. 84–89.

[6] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, “FTRANS: Energy-Efficient Acceleration of Transformers Using FPGA,” in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2020, pp. 175–180.

[7] H. Peng, S. Huang, T. Geng, A. Li, W. Jiang, H. Liu, S. Wang, and C. Ding, “Accelerating Transformer-based Deep Learning Models on FPGAs using Column Balanced Block Pruning,” in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2021, pp. 142–148.

[8] Z. Yao, S. Cao, W. Xiao, C. Zhang, and L. Nie, “Balanced Sparsity for Efficient DNN Inference on GPU,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 33, no. 01, 2019, pp. 5676–5683.

[9] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, and H. Li, “Learning N:M Fine-grained Structured Sparse Neural Networks from Scratch,” in *International Conference on Learning Representations (ICLR)*, 2021.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention Is All You Need,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, 2017, pp. 6000–6010.

[11] Y. Qi, D. Sachan, M. Felix, S. Padmanabhan, and G. Neubig, “When and Why Are Pre-Trained Word Embeddings Useful for Neural Machine Translation?” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2018, pp. 529–535.

[12] M.-T. Luong and C. D. Manning, “Stanford Neural Machine Translation Systems for Spoken Language Domain,” in *International Workshop on Spoken Language Translation (IWSLT)*, Vietnam, 2015.

[13] T. Q. Nguyen and J. Salazar, “Transformers without Tears: Improving the Normalization of Self-Attention,” *arXiv preprint arXiv:1910.05895*, 2019.

[14] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “BLEU: A Method for Automatic Evaluation of Machine Translation,” in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics (ACL)*, 2002, pp. 311–318.

[15] NVIDIA, “Jetson Nano Developer Kit,” https://developer.nvidia.com/zh-cn/embedded/jetson-nano-developer-kit, 2019, [Online; accessed 14-October-2021].

[16] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, “TinyBERT: Distilling BERT for Natural Language Understanding,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings (EMNLP)*, 2020, pp. 4163–4174.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[18] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, “Emerging Properties in Self-Supervised Vision Transformers,” in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.