Opportunities and Limitations of in-Memory Multiply-and-Accumulate Arrays

Ruoyu Zhi, Ryan Jurasek, Wolfgang Hokenmaier, Don Labrecque, Jacob Bucci, Bret Dale,
Nibir Islam, Dave Kinney, Angela Johnson
Green Mountain Semiconductor Inc.
whokenmaier@greenmountainsemi.com

***Abstract*—In-memory computing is a promising solution to solve the memory bottleneck problem which becomes increasingly unfavorable in modern machine learning systems. In this paper, we introduce an architecture of random access memory (RAM) incorporating deep learning inference abilities. Due to the digital nature of this design, the architecture can be applied to a variety of commercially available volatile and non-volatile memory technologies. We also introduce a multi-chip architecture to accommodate for varying network sizes and to maximize parallel computing ability. Moreover, we discuss the opportunities and limitations of in-memory computing as future neural networks scale, in terms of power, latency and performance. To do so, we applied this architecture to various prevalent neural networks, e.g. Artificial Neural Network (ANN), Convolutional Neural Network (CNN) and Transformer Network and compared the results.**

***Index Terms*—In-memory computing, deep neural network, deep learning, DRAM, Transformer, memory bottleneck**

I. Introduction

Artificial neural networks have grown tremendously in both size and complexity, but the development and application have always been hindered by the memory bottleneck as data has to be sent back and forth between the memory and processor in the von Neumann machines [1]. Data movement is very expensive in terms of power and time compared to the actual computation that happens inside the processor. Recent analysis shows that the computing power demanded for AI tasks has increased by a factor of 300,000 since 2012 [2], whereas Moore's law is slowing down significantly as we are approaching the physical limit to further scale the CMOS devices.

New architectures such as ASICs (e.g. Google's TPU) and FPGAs (e.g. Xilinx Zynq Series and Intel HARPv2) that utilize weight stationary concepts are developed to accelerate AI tasks. However, they still rely on on-die SRAM which is very demanding in terms of die size. There are various attempts to solve this problem using Dynamic Access Memory (DRAM), which can be manufactured in much larger densities. The High Bandwidth Memory (HBM) standard utilizes an ultra-wide memory data bus on silicon interposers, and 3D stacking of DRAM dies for a very high memory density and bandwidth, as shown in Fig. 1. However, 3D structure is less resilient to manufacturing errors and 3D packaging is comparatively less cost-efficient.

A departure from classic processor-centric von Neumann architecture through in-memory neuromorphic computing has been proposed to solve the increasing processor-memory performance gap of the current artificial neural network accelerators [3]. This may include the implementation of logic cores inside the memory chip (Fig. 2). The current research toward this direction has for some time focused on analogue properties of the memory cells. Especially, resistive memory elements have received particular interest in the literature [4]. However, high power of ADCs/DACs, non-linear conductance response, limited dynamic range of the conductance and its variability do not make these devices ideal candidates [5].

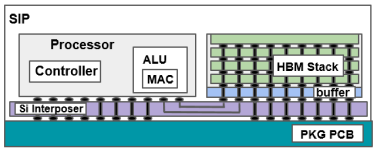

Fig. 1. HBM

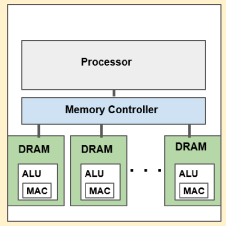

Fig. 2. Weight stationary in-memory

Herein, we present a digital approach of in-memory neuromorphic computing architecture that is consistent with a wide variety of memory technologies. The paper is organized as follows: In Section II, we first provide an introduction to artificial neural networks and a mathematical operation that accelerates neural networks inference in hardware. Then, we briefly review the essential components of a multi-layer in-memory artificial neural network accelerator based on a fully self-contained multi-layer capable edge computing prototype. In Section III, we present a highly parallel/high bandwidth architecture for high density dynamic random access memory (DRAM) products, suitable for high performance AI inference operations. We also demonstrate the commands and workflow of the presented hardware. At last, a methodology to utilize multiple chips for massive parallel computation is illustrated. In Section IV, we first discuss the development trends of artificial neural networks and the influence on in-memory computing. Then, we present an analysis of the performance and power consumption of in-memory computing as future neural networks scale. Section V gives our projection on future work and our conclusions for in-memory neuromorphic computing.

II. Overview

A. Software Background

An artificial neural network (ANN), or more specifically known as Multi-Layer Perceptron (MLP) model, is made up of interconnected neurons and contains an input layer, an output layer and one or more hidden layers. Except for those in the input layer, each node is calculated by adding a weighted sum of all the input nodes in the previous layer along with a single bias to offset the threshold for activation purposes. The sum is then fed into an activation function. The output of the activation function will be read out or passed as input to the next layer. This process is illustrated in Fig. 3.

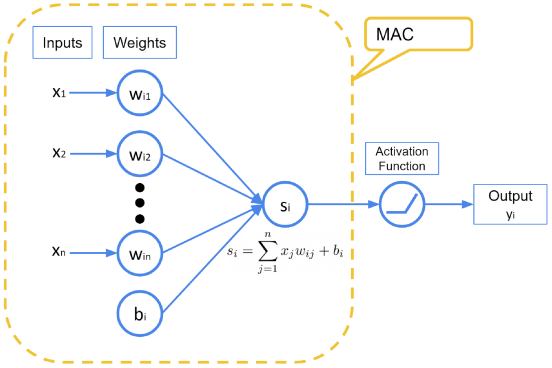

Fig. 3. Computation of one neuron

Neural networks have been the focus of artificial intelligence for the past decade and various architectures have been developed, e.g. convoluted neural networks, recurrent neural networks, Transformer networks, etc. Each Architecture is different and has its own variations, but they all share the same fundamental multiply-and-accumulate (MAC) operations. Therefore, a dedicated MAC accelerator is both universal and crucial to overcome hardware limitations for neural networks, and we show that an in-memory approach would be beneficial to all neural network models.

Quantization is another critical aspect of neural network hardware implementation. Although typical neural networks are trained with float32 precision, due to the neural network's robustness to noise because of its architecture, the precision during inference can be reduced without losing too much accuracy. The process of mapping a high precision floating point range to a low precision integer range is called quantization. A number of edge AI devices have adopted int8 for speed concerns, for example, Edge TPU by Google. Research [6] has shown that with proper quantization from float32 to int8, neural network inference can be twice as fast with acceptable losses across multiple benchmark datasets. There are even extreme cases where binary quantization is exercised with a few limitations [7]. As a general MAC accelerator, we adopt a mixed-precision 8b sequential multiplier and 32b accumulator for the most use cases and efficiency.

III. System Architecture

A. Hardware Architecture

The proposed architecture is based on a commodity DRAM design. Its major difference is that it allows multiple word line activations so that all the sense amplifiers can be utilized simultaneously, whereas in a commodity DRAM, only ˜10% of the sense amplifiers are leveraged.

It is different from an earlier design based on Phase-Change Memory (PCM) [8] in that it features a distributed MAC architecture as is illustrated in Fig. 4. This DRAM consists of 16 half-banks, and each half-bank comprises an array of 12x16 MAC units. This change vastly increases the parallelism of MAC operations compared to its PCM predecessor, with a few limitations, which will be further discussed later.

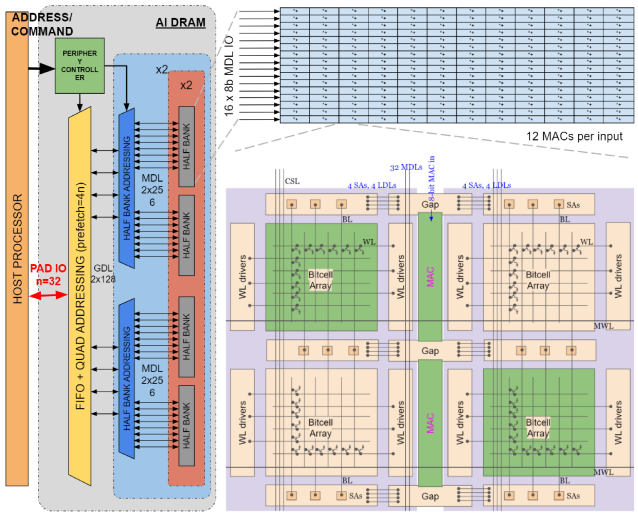

Fig. 4. System Architecture

*1) **Multiply and Accumulate (MAC):*** The MAC unit is responsible for multiplying two 8-bit values (weight and input) and accumulating the product and a bias. Its function is given by (1):

$$s_i = \sum_{j=1}^{n} x_j w_{ij} + b_j \tag{1}$$

The MAC accepts two inputs, an 8-bit parallel weight value $W[7:0]$ and a serial input value X. Each MAC unit consists of two parts, the sequential multiplier and the accumulator. The multiplier generates a 16-bit output word, which is fed into a 32-bit accumulator. The accumulator is able to accumulate up to at least $2^{16} = 65536$ inputs and a bias without overflow, which is more than enough for most neural networks today. After all of the input layer is traversed, we get the MAC result for one output neuron in each MAC unit.

*2) **Activation Function:*** The activation function is a crucial component in all neural networks. The result of the MAC operation is passed into the activation function to decide if this neuron is "activated" or not. It also adds non-linearity to the system which is essential to all modern AI models. The activation function, originally, was the Sigmoid function, but

due to the vanishing gradient problem while training a neural network, it has been substituted by the rectified linear units (ReLU) function, which is defined by (2).

$$ReLU(x) = max(0, x) \tag{2}$$

ReLU is the prevalent activation function in state-of-the-art deep learning research since it is robust and faster to implement in algorithms and adds strong non-linearity.

ReLU is also hardware-friendly. Traditionally, non-linear activation functions like Sigmoid are realized with FPGAs [9], or approximated using a stochastic approach [10]. For ReLU, output is simply the input with the negative values being clipped off, which greatly simplifies the design of a hardware implementation. Nevertheless, a bit shift function is added to the simple clip function. The advantage of this modification is twofold. First, since the accumulator comprises 32 shift registers, the activation function can be implemented right within the accumulator with merely a few extra inverters, which minimizes the footprint of the activation function. Secondly, the ReLU function causes positive mean shift, which means that the neuron values would drastically increase as the network gets wider and deeper. This is not a huge concern on float32 precision processors, but can lead to severe data overflow in the 8-bit architecture. Therefore, a right shifter is needed to scale the activation values back to [0, 255] range. This activation function is given by (3):

$$f(x) = \begin{cases} max(0, floor(\frac{x}{2^n})) & \text{if x} < 256\times2^{\text{n}} \\ 255 & \text{if x} \geq 256\times2^{\text{n}} \end{cases} \tag{3}$$

where n is the number of bits shifted. We call this function clipped staircase ReLU.

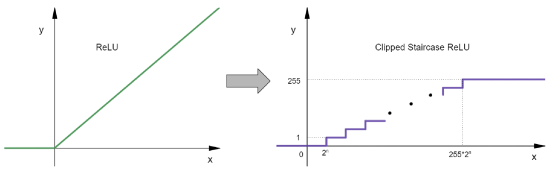

Fig. 5. Activation function transition

A software simulation is conducted to evaluate the feasibility of this approach. First we trained several multi-layer perceptron models to solve the classic mnist problem and achieved an average accuracy of 96.83%. Each model consists of 3 hidden layers and the layer sizes are different from each other. The output layer has 10 neurons, each corresponding to one category. We noticed that the distributions of weight, bias and MAC result (input to activation function) across all four layers are all normal distributions. Therefore, as long as the staircase region of the clipped staircase ReLU covers the majority of the positive region of MAC results, we can minimize precision loss.

Next, we took the trained model and replaced the activation function with our clipped staircase ReLU. The inference result shows that the clipped staircase ReLU is considerably effective on this problem. The accuracy dropped a negligible amount of 0.7% to 96.14%. This proves that the clipped staircase ReLU would work on categorical ANN problems.

Providing ReLU allows multiple layers of operations to stay within memory, and returns another order of magnitude reduction in memory IO. More complex activation functions are still possible on this architecture by reading out 32-bit accumulations directly, but forego the degree of in-memory advantage The host can then perform any desired activation, filtering, dropout, etc. to generate outputs to be written back into DRAM for the next layer.

*3) **Memory Array**:* The memory array consists of 16 half-banks, and each half-bank is made up of 12x32 bitcell sub-arrays. In the proposed architecture, we insert a column of MAC units (MAC stack) in between two groups of word line drivers between two adjacent bitcell arrays. Every two bitcell arrays share a MAC unit, so there are $16 * 12 * 16 = 3072$ MAC units in total. We also modified the row address decoder so that multiple word lines can be opened up simultaneously. This structure minimizes the distance between the processing unit and storage and greatly increases the parallel throughput. Each MAC accepts weight data from the adjacent bitcell array, as is shown in Fig. 4, data from the green array is transferred to the green MAC unit next to it. The input value X, on the other hand, is broadcasted to all the MACs in a MAC stack via master data lines (MDL). Each MAC stack can be fed with different input values. Thus, we are able to execute thousands of MAC operations simultaneously with one command.

B. Inference Workflow

The workflow of a complete ANN inference can be summarized as follows:

1) Upon receiving a MAC command, a normal read is performed to read out 8-bit input data from the array and store the data temporarily in a buffer off-array.
2) To get the weight data, we activate the word lines and transfer the data from the DRAM cell to the primary sense amplifiers. Then, for each MAC unit, the column select signal (CSL) selects 4 bits from the top SAs and 4 bits from the bottom SAs, and the 8 bits are transferred into the gap area over the local data line (LDL) and presented on the W input of the MAC units parallelly.
3) The off-array buffer that stores 8-bit input data starts to broadcast the input bit by bit to all the MACs across the chip. After 8 clock cycles, a latch accumulator command is issued to accumulate the 16-bit product in the accumulator.
4) CSL selects the next 8-bit weight data from the SAs and Step 3) is repeated until all the input neurons are processed. A counter monitors how many weights have been processed. If all the SAs have been accessed, then new word lines are opened to update the values in the SAs.
5) If the current input layer has been traversed, the activation function is then applied to the 32-bit output of the

accumulator. Then we write the 8-bit output from the activation function back to the array.

6) A counter checks if all the output neurons have been calculated, and if not, Steps 1 through 5 are repeated.

Layer information and network connectivity will be compiled into data that stores in the array. An on-chip controller will extract the required information at runtime and issue commands accordingly to inference the neural network following the above workflow.

C. Multiple Chips

The architecture also has the potential to scale with multiple chips. As illustrated in Fig. 6, where two chips are utilized, the weight data is divided into two chips while the input data is duplicated across the chips. In this case, Chip 1 calculates half of the output neurons and Chip 2 calculates the other half. As long as the weights are allocated properly to each of the chips, they can work simultaneously to increase throughput. At last, the output from each chip can be merged as a whole layer of neurons.

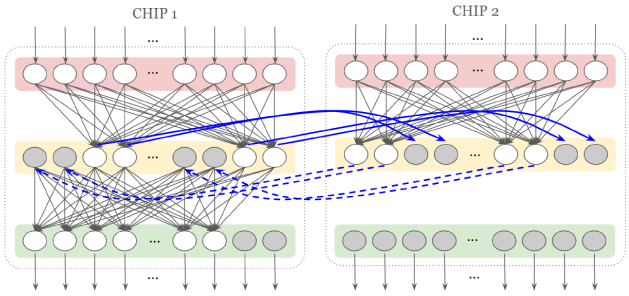

Fig. 6. Multiple chips

D. Summary

A distributed memory array with MAC units working together with a central controller, a fully functional, self contained MAC accelerator has been constructed. It has significant parallel computing ability and very low latency. Nevertheless, it comes with some limitations. First, the row address decoder is modified to support multiple word line activation, which might increase peak current consumption compared to normal DRAMs. Second, it suffers from performance degradation when the activation function is complex. In this situation, a separate activation function needs to be implemented off-array to process the MAC results. At last, embedding the MAC stack and activation function inside the array would increase the chip size by around 12%.

Knowing these limitations, we examined multiple classic neural networks and evaluated different architectures to highlight how performing these functions less efficiently still guarantees growing orders of magnitude savings overall as a function of AI's development.

IV. Neural Network Scaling

The first neural network to achieve parity with state-of-the-art algorithms was a Convolutional Neural Network built in 1998 known as LeNet-5, which featured approximately 60k parameters that were trained via unsupervised backpropagation to identify handwritten decimal numbers [15]. Generally speaking, CNNs convolve small kernels of parameters across an image to identify patterns, and use downsampling functions like max pooling to discern higher order information about the image. In the process of convolution and striding, each weight value is reused in tens or hundreds of computations to make the most out of memory operations. Since then, CNNs have grown in size and complexity alongside the computational efficiency of the underlying silicon, which hit an inflection point in 2012 when AlexNet set the standard of deploying GPUs to train over 60M parameters to break ImageNet classification records. This newfound parallelism then made way for architectures like VGG and ResNet to scale with blocks composed of multiple layer operations, which are then stacked to scale performance with over 500M parameters in some designs.

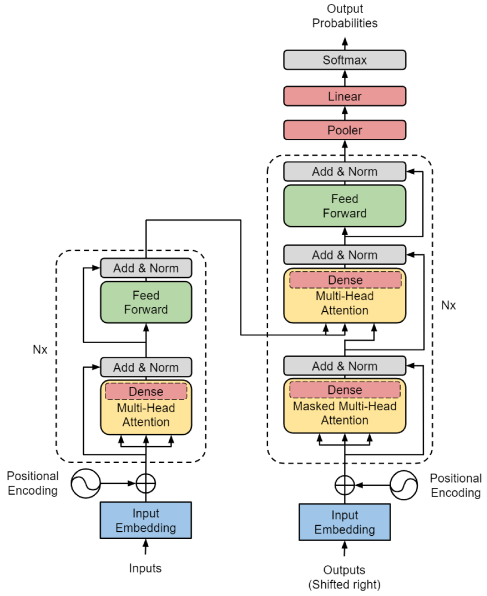

Fig. 7. Transformer

The Transformer network (Fig. 7) emerged in 2017 and quickly gained popularity in the machine learning research field for natural language processing [12]. Recent research has also discovered its potential in image processing tasks with minor modifications [13]. Transformers are closer in nature to Recursive Neural Networks, which operate on sequences of data to make inferences based on previous states and location within the sequence. Unlike CNNs which are easily contained to SRAM and batched over multiple inputs, transformers become prohibitively expensive for longer sequence lengths because each step requires memory IO to perform operations with a significant percentage of all parameters. While the original publication had a base model of 65M parameters, it has not taken long to scale to over 1 trillion parameters in the case of the switch transformer, which is pretrained for translations between 101 different languages [14]. There is

a greater diversity of operations for transformers such as IO embeddings and key-value-query attention mechanisms, but by volume most of the parameters are used for classical fully-connected feedforward MAC and/or ReLU. We pick five most representative Transformer architectures from [14] and show a distribution of weights as Transformer scales in Fig. 8. The following discussion on scaling will be based on these five architectures. Noticeably, it is these classical layers that most readily scale transformer performance.

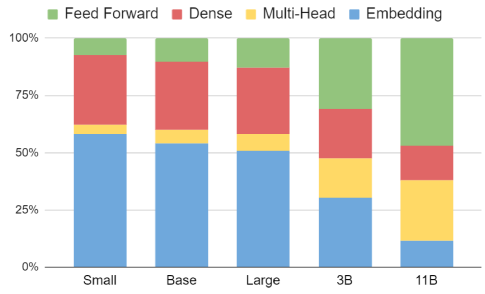

Fig. 8. Weight distribution as Transformer scales

To understand how in-memory computing lends itself to the scaling properties of AI, we develop a model to project performance metrics for architectures of interest for single batch processing:

1) **Baseline von Neumann (von Neumann)** - Network parameters read from memory for all computations in logic
2) **In-memory MAC (IMMAC)** - Layer inputs written to memory for MAC operations, accumulation is read by logic for activations, pooling, softmax, etc.
3) **In-memory MAC+AF (IMnet)** - Network inputs written to memory, final outputs quantized/processed through the ReLU function before readout

The computational path is broken into 6 sections that are common to all architectures. The toggle count for each section is calculated as a function of a layer's hyperparameters. Each section receives an "energy-per-toggle" value derived from SPICE simulations of our mixed-precision MAC/shift/ReLU, which operates with randomized parameters and inputs to achieve an average "per-bitcell-toggle" approximation. To understand how performance scales with memory interface, we use published energy and throughput numbers for a variety of standards. For scaling by logic technology, we apply methods from Stillmaker et al. [11], which similarly use measurements from an FO4 inverter to achieve such projections. Table I shows the result.

Next, we obtain the memory IO statistics for each of the five aforementioned Transformers by breaking down the models block by block and counting the parameters in each block. To emphasize the relative relationship between IO and computation amount, the X axis is the total number of MAC operations. The Y axis is the number of bits on the memory IO. Both axes are logarithmic.

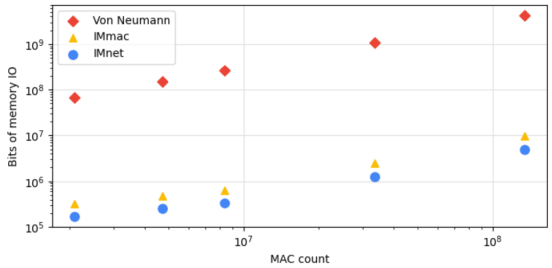

Fig. 9. IO comparison

For each Transformer surveyed, in-memory architectures display 2-3 orders of magnitude less IO than traditional von Neumann architecture, and this difference remains as networks scale. Since this is a loglog plot, it means the advantage of in-memory computing will grow exponentially as a function of AI's development.

We then calculate the total number of operations and translate it into the number of bits toggled according to our architecture for each of the network, for example, bit counts for the "Small" Transformer architecture are shown in Table II. Multiplying the number of bits toggled in each block with its corresponding power data in Table I, we obtain the power consumption for each of the networks, as is shown in Fig. 10.

TABLE I
ENERGY PER BITCELL TOGGLE

Architecture	**IO**		**Compute**			
	input	**weight**	**multiply**	**accumulate**	**AF**	**control signals**
von Neumann	4.00	0.16	0.00037	0.00055	0.00022	0.019
IMMAC		0.35	0.0033	0.005		0.124
IMnet					0.002	0.125

Unit: Picojoules

TABLE II
NUMBER OF BITS TOGGLED (SMALL TRANSFORMER)

Architecture	**IO**		**Compute**			
	input	**weight**	**multiply**	**accumulate**	**AF**	**control signals**
von Neumann	8,388,608	8,388,608	134,217,728	33,554,432	311,296	8,388,608
IMMAC	32,768					18,891,776
IMnet	20,480					18,889,728

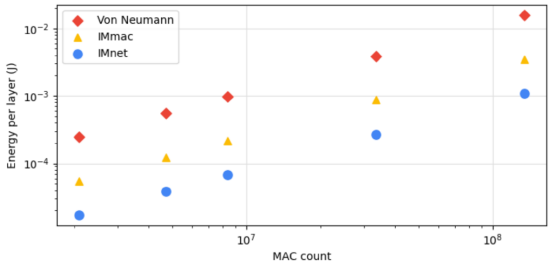

Fig. 10. Power comparison

It demonstrates that in-memory computation of these layers is almost 5x more efficient for real time performance without batching. Again as networks continue to scale in volume, advantages are retained by the overall reduction in IO between logic and memory.

At last, we project the performance of in-memory computing as the manufacturing process advances. Again a "fanout of 4" methodology is outlined to project performance scaling. The result exhibits ongoing potential in terms of energy efficiency for in-memory computing.

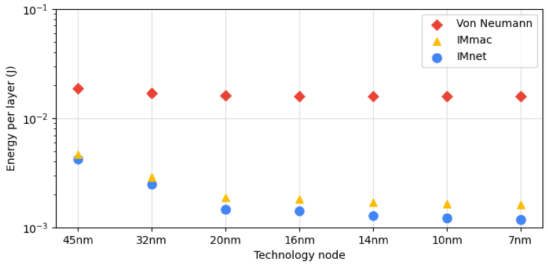

Fig. 11. Technology trend

V. Future Work and Conclusion

There are still endless possibilities that the work in this paper could be extended as increasing attention has been focusing on in-memory approaches for AI accelerating. First, more work needs to be done to carry out the design that meets tight thermo budget and metal area. Second, AI algorithm acceleration is a comprehensive task that requires revolution in the whole technology stack. Further efforts need to be made in the operating system and compiler to fully support the functionality provided by the hardware.

In this work, we introduced a novel approach to implement a fully digital MAC accelerator in a DRAM memory. We presented a mechanism that minimizes the circuit size of MAC units and activation function so that massive parallel MAC in memory is possible. We also proposed a multiple-chip architecture to tackle scaling networks in the future. What's more, an evaluation of the power and performance is conducted to discuss the advantages and limitations of in-memory MAC acceleration. We concluded from the analysis that the proposed architecture will excel in an AI scenario that requires no batching, low power and low latency.

References

[1] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," 03-Jan-2020. [Online]. Available: https://arxiv.org/abs/1907.00235. [Accessed: 15-Apr-2021].

[2] A. Mehonic, A. Sebastian, B. Rajendran, O. Simeone, E. Vasilaki, and A. J. Kenyon, "Memristors—From In-Memory Computing, Deep Learning Acceleration, and Spiking Neural Networks to the Future of Neuromorphic and Bio-Inspired Computing," Advanced Intelligent Systems, vol. 2, no. 11, 2020.

[3] I. Giannopoulos, A. Sebastian, M. Le Gallo, V. P. Jonnalagadda, M. Sousa, M. N. Boon, and E. Eleftheriou, "8-bit Precision In-Memory Multiplication with Projected Phase-Change Memory," 2018 IEEE International Electron Devices Meeting (IEDM), 2018.

[4] W. Hokenmaier, D. Labrecque, R. Jurasek, V. Butler, C. Scoville, A. Willey, S. Loeffler, Y. Li, and S. Sharma, "A 90nm 32-mb phase change memory with flash SPI compatibility," 2014 IEEE 6th International Memory Workshop (IMW), 2014.

[5] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," arXiv preprint arXiv:1705.06963, 2017.

[6] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.

[7] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," The Journal of Machine Learning Research, 18(1), 2017, pp. 6869-6898.

[8] M. Golmohamadi, R. Jurasek, W. Hokenmaier, D. Labrecque, R. Zhi, B. Dale, N. Islam, D. Kinney, and A. Johnson, "Verification and Testing Considerations of an In-Memory AI Chip," 2020 IEEE 29th North Atlantic Test Workshop (NATW), 2020.

[9] I. Tsmots, O. Skorokhoda, and V. Rabyk, "Hardware Implementation of Sigmoid Activation Functions using FPGA," 2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), 2019.

[10] V.-T. Nguyen, T.-K. Luong, H. Le Duc, and V.-P. Hoang, "An Efficient Hardware Implementation of Activation Functions Using Stochastic Computing for Deep Neural Networks," 2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), 2018.

[11] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," Integration, vol. 58, pp. 74–81, 2017.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," arXiv.org, 06-Dec-2017. [Online]. Available: https://arxiv.org/abs/1706.03762v5. [Accessed: 30-Mar-2021].

[13] 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," arXiv.org, 22-Oct-2020. [Online]. Available: https://arxiv.org/abs/2010.11929.

[14] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," arXiv.org, 28-Jul-2020. [Online]. Available: https://arxiv.org/abs/1910.10683.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.