An RRAM Based Computing-In-Memory Architecture and Its Application in Accelerating Transformer Inference

Zhaojun Lu *Member, IEEE*, Xueyan Wang *Member, IEEE*, Md Tanvir Arafin, Haoxiang Yang, Zhenglin Liu, Jiliang Zhang *Senior Member, IEEE*, and Gang Qu *Fellow, IEEE*

***Abstract*—Deep neural network-based Transformer models have demonstrated remarkable performance in natural language processing (NLP) applications. Unfortunately, the unique scaled dot-product attention mechanism and intensive memory access pose a significant challenge during inference on power-constrained edge devices. One emerging solution to this challenge is computing-in-memory (CIM), which uses memory cells for logic computation to reduce data movement and overcome the memory wall. However, existing CIM designs do not support high-precision computations, such as floating-point operations, that are essential for NLP applications. Furthermore, CIM architectures require complex control modules and costly peripheral circuits to harness the full potential of in-memory computation.**

Hence, this paper proposes a scalable RRAM-based in-memory floating-point computation architecture (RIME) that uses single-cycle NOR, NAND, and Minority logic to implement in-memory floating-point operations. RIME features efficient parallel and pipeline capabilities with a centralized control module and a simplified peripheral circuit to eliminate data movement during computation. Furthermore, the paper proposes pipelined implementations of matrix-matrix multiplication and softmax functions, enabling the construction of a Transformer accelerator based on RIME. Extensive experimental results show that, compared with GPU-based implementation, the RIME-based Transformer accelerator improves timing efficiency by 2.3× and energy efficiency by 1.7× without compromising inference accuracy.

***Index Terms*—Computing-In-Memory, RRAM, Transformer, Accelerator, Scalability, Energy Efficiency.**

I. Introduction

Natural language processing (NLP) techniques have experienced tremendous performance improvements in recent years, thanks to advances in deep neural network (DNN) algorithms, such as the long short-term memory (LSTM) [1], recurrent neural network (RNN) [2], gated recurrent unit (GRU) [3], and Transformer [4]. A Transformer is based on a self-attention mechanism that significantly reduces the path length between long-range dependencies. It is the state-of-the-art model for the sequence-based NLP tasks. Unfortunately, the Transformer models require considerable computational resources and energy. For instance, the original Transformer model has 65 million parameters. Although some structural optimization schemes [5–7] reduce the model complexity, they incur either the loss of intrinsic word dependencies or a more expensive training process [8].

Most of the energy consumed in computing is due to data movement between the compute unit and off-chip memory, which is orders of magnitude greater than that of floating-point operations [9, 10]. Consequently, the memory wall in the conventional von Neumann architecture severely limits efficiency and scalability for large-scale Transformer models. Computing-in-memory (CIM) is a promising solution to break the memory-wall bottleneck and, therefore, is considered a promising way to enhance the performance of the Transformer inference. Research on existing main memory components (*i.e.*, dynamic random-access memory (DRAM)) has demonstrated the high efficiency of in-memory logic and arithmetic computations. Additionally, emerging non-volatile memory solutions represented by resistive random access memory (RRAM) have been widely exploited to support in-memory operations [11–13]. Due to the high density, fast access, and low leakage of resistive memory, RRAM-based CIM architectures exhibit dramatic improvements in accelerating basic logic operations.

However, RRAM-based CIM designs face several challenges when accelerating The Transformer model. First, it is difficult for RRAM-based CIM designs to provide accurate floating-point computation. The resolution of the multi-level RRAM cells limits the precision of the vector-matrix multiplication (VMM) operations [8]. Second, costly peripheral circuits and cumbersome external computing units [14] are necessary for data processing in CIM architectures [15]. Third, the latency for 32/64-bit arithmetic computation will be sharply increased if using a single bit-wise operation (such as NAND or NOR) on a single bipolar RRAM cell [16, 17]. This issue is exacerbated by the heavy use of softmax functions in the Transformer model [7]. Therefore, fundamental design optimization is required for developing accurate and efficient Transformer accelerators based on the CIM architecture.

In this paper, we propose a scalable RRAM-based in-memory floating-point computation architecture (RIME) to accelerate the Transformer model and address the aforementioned issues. RIME was first introduced in our conference paper [17] for 32-bit floating-point multiplication, and the

Corresponding author is Xueyan Wang, E-mail: wangxueyan@buaa.edu.cn

Zhaojun Lu, and Zhenglin Liu are with the School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China. Xueyan Wang is with the Department of Integrated Circuit Science and Engineering, Beihang University, Beijing 100084, China. Md Tanvir Arafin is with the Cyber Security Engineering Department at George Mason University, Fairfax, VA 22030, USA. Haoxiang Yang is with Warsaw University of Technology, Warsaw, Poland. Jiliang Zhang is with College of Semiconductors, Hunan University, Changsha 410082, China. Gang Qu is with the Electrical and Computer Engineering Department at University of Maryland, College Park, MD 20742, USA.

primary contributions of our work are as follows:

First, we present RRAM-compatible single-cycle computation techniques for NOR and 3-input Minority (Min_3) logic functions, which are the fundamental building blocks for the RIME architecture. We also design supporting operations for the 1-bit full adder using NOR and 3-input Minority (Min_3), which are integrated into the RRAM crossbar to reduce the overall complexity of the CIM subsystem.

Next, we provide a detailed implementation of a 32-bit floating-point multiplication design in the RRAM crossbar. We demonstrate that the floating-point multiplication in RIME can achieve three degrees of parallelism: (1) multiple full adders can operate in parallel in the same row of RRAM cells, (2) the main operations in floating-point multiplication, including the sign bit XORing, the exponent bit addition, and the fixed-point multiplication for mantissa bits, can be executed in parallel, and (3) multiple multiplications can be computed in parallel in multiple rows of the RRAM crossbar since a centralized control module controls each row of RRAM with the same instructions.

Furthermore, we design and implement the control module and the peripheral circuits for the RIME architecture. We compare a RIME-based 32-bit floating-point multiplication with state-of-the-art designs [14, 16, 18, 19] in terms of latency, area, and power efficiency. Our results show that a RIME multiplication significantly outperforms current CIM-based 32-bit floating-point multiplications.

Finally, we reimplement the Transformer model in hardware by decomposing the matrix-matrix multiplication (MatMul) and softmax functions into pipeline operations. Based on these results, we propose a RIME-based Transformer inference engine that is scalable, efficient, and accurate. We evaluate the timing and energy efficiency of our design against a GPU platform (NVIDIA RTX 3080). Our experimental results demonstrate a 2.3× timing-efficiency increase and a 1.7× energy-efficiency increase for our design without affecting the inference accuracy.

II. Preliminaries and Related Works

A. The Transformer Model

Transformer models have significantly improved the performance of various NLP tasks due to three major features: (1) less computational complexity per layer, (2) a large amount of operations that can be parallelized, and (3) short path length between long-range dependencies in the network [4]. Thus, multi-head self-attention of a Transformer can connect the input and output sequences of length n with $O(1)$ operations, while the traditional sequential models (*e.g.*, LSTM, RNN, *etc.*) require $O(n)$ operations to complete the same function [8].

In order to compare the performance of different CIM-based methods in accelerating the Transformer inference, we choose the original model proposed in [4] for RIME implementation. As illustrated in Fig. 1(a), the Transformer model has an encoder-decoder structure. The encoder comprises a stack of six identical layers, and each layer has two sub-layers, as shown in Fig. 1(b). The first sublayer is a multi-head self-attention layer, and the second one is a simple, position-wise, fully connected feed-forward network. The Transformer model employs a residual connection around each of the two sub-layers, followed by layer normalization. The decoder, which is also composed of six identical stacked layers, as shown in Fig. 1(c), includes three sub-layers: two sub-layers, similar to the ones in each encoder layer, and a third sub-layer that performs multi-head attention over the output of the encoder stack [4]. Similar to the encoder, residual connections are employed around each of the decoder sub-layers, followed by layer normalization. To prevent positions from attending to subsequent positions, the self-attention sub-layer in the decoder stack is modified.

The architecture of the Transformer model also utilizes scaled dot-product attention mechanisms. In a scaled dot-product attention layer, as illustrated in Fig. 1(e), the input is comprised of queries and keys of dimension d_k and values of dimension d_v. The layer computes the dot products of the query with all keys first and then divides each by d_k. Finally, it applies a softmax function to obtain the weights on the values [4]. Consequently, the layers perform the attention function on a set of queries $\boldsymbol{Q}$ with the keys and values $\boldsymbol{K}$ and $\boldsymbol{V}$. The matrix of outputs is computed as [4]:

$$\boldsymbol{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \boldsymbol{Softmax}(\frac{\boldsymbol{Q}\cdot\boldsymbol{K^T}}{\sqrt{d_k}})\cdot\boldsymbol{V} \quad (1)$$

The attention functions are performed in parallel, yielding dv-dimensional output values that are concatenated and once again projected, resulting in the final values, as depicted in Fig. 1(d).

$$\boldsymbol{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \boldsymbol{Concat}(head_1,...,head_h)\cdot\boldsymbol{W^O} \quad (2)$$

$$head_i = \boldsymbol{Attention}(\boldsymbol{Q}\cdot\boldsymbol{W_i^Q},\boldsymbol{K}\cdot\boldsymbol{W_i^K},\boldsymbol{Q}\cdot\boldsymbol{V_i^V}) \quad (3)$$

The projections are parameter matrices $\boldsymbol{W_i^Q},\boldsymbol{W_i^K} \in \mathbb{R}^{d_{model}\times d_k}$, $\boldsymbol{W_i^V} \in \mathbb{R}^{d_{model}\times d_v}$ and $\boldsymbol{W^O} \in \mathbb{R}^{hd_v\times d_{model}}$.

In addition to the attention sub-layers, each layer in our encoder and decoder architectures features a fully connected feed-forward network. This network is applied to each position independently and identically. The feed-forward network is comprised of two linear transformations with a ReLU activation function sandwiched between them.

$$\boldsymbol{FFN}(\boldsymbol{X}) = \boldsymbol{max}(0,\boldsymbol{X}\cdot\boldsymbol{W_1}+b_1)\cdot\boldsymbol{W_2}+b_2 \quad (4)$$

, where $\boldsymbol{W_1} \in \mathbb{R}^{d_{model}\times d_{ff}}$, $\boldsymbol{W_2} \in \mathbb{R}^{d_{ff}\times d_{model}}$, and the result $\boldsymbol{FFN(x)} \in \mathbb{R}^{d_{model}}$. Here d_{ff} is the dimension of the hidden layer [4].

There exist some quantization methods in the state-of-the-art utilizing 8-bit and even 4-bit integer for computation, however, for quantified transformer, properly expressing the outlier features and high dynamic activation range at low-fixed points is challenging, failing to do this can lead to model accuracy decrease [20]. With model size increases, outlier features exist in each layer of the transformer, leading to degradation of the model's accuracy to a greater extent [21]. Nevertheless, the future development trend of transformers is to employ larger

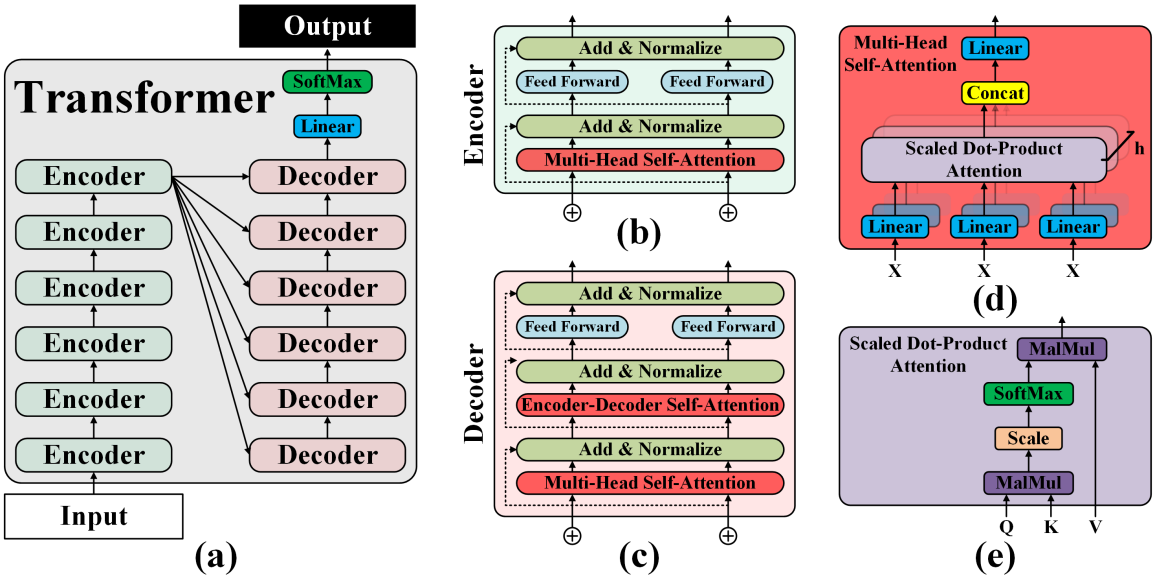


Fig. 1. (a) Transformer model. (b) Structure of encoder. (c) Structure of decoder. (d) Multi-head attention consists of several attention layers running in parallel. (e) Scaled dot-product attention.

and more complex models. As a result, adopting accurate floating-point representations is essential, and our work in this paper proposes to accelerate high precision floating-point operations in transformer.

B. Computing-In-RRAM

The existing multi-bit RRAMs are not suitable for high-precision Transformer model due to the non-ideal factors, including the resistance non-linear problem, the resistance variations, and the quantization errors from the analog and digital interfaces [22]. Therefore, we focus on the single-level-cell RRAM to accelerate the Transformer model without accuracy degradation.

1) IMPLY-based CIM: In 2010, a single-level-cell RRAM-based logic gate, named Material Implication (IMPLY), is presented by Borghetti *et al.* [23]. As illustrated in Fig. 2 (b), IMPLY leverages material implication operations within an RRAM crossbar to perform logic operations. By applying sequential voltage activation at different locations in the crossbar, IMPLY stores the result in one of the input RRAM cells instead of a dedicated output RRAM cell. In 2022, Fatemieh *et al.* [24] proposed a novel algorithm for serial IMPLY-based adders to implement an approximate full-adder, which achieved up to 40% improvement without introducing unacceptable error. Through the material implication operation within an RRAM crossbar, IMPLY applied sequential voltage activation at different locations to perform logic operations in the crossbar. Finally, the result would be stored in one of the input RRAM cells instead of a dedicated output RRAM cell. The main disadvantages of IMPLY-based designs are the costly circuit components, and the complex control modules that are impractical for efficient and high-precision CIM architecture [25].

2) MAGIC-based CIM: In 2014, Kvatinsky *et al.* proposed RRAM-aided logic (MAGIC) [26] that utilized multiple RRAM cells for previously stored data as input and an additional RRAM cell for output as illustrated in Fig. 2 (c). Imani et al. [14] presented FloatPIM that applied MAGIC-based NOR gates to support floating-point representation and enable fast communication between neighboring memory blocks to reduce internal data movement of the CIM computation architecture. Alam *et al.* v[27] demonstrated how to convert numbers between binary and stochastic domains and how to perform multiplications using in-memory computations by MAGIC, in which the multiplication was extended to i-input multiplication by performing i-input MAGIC NOR on i bitstream operands. In 2022, MAGIC-based in-memory designs were provided in [28] for the AND operation and the OR operation on unary bit-streams. AND was realized by first inverting the bit-streams through NOT and then performing bit-wise NOR on the inverted bit-streams in a total of three cycles, while OR was achieved in two cycles by first bit-wise NOR on the input bit-streams and then NOT on the outputs of the NOR operations.

III. NOR, MIN_3, AND NAND IN RIME

In this section, the basic logic operations in RIME-based architecture are elaborated. In addition to utilizing the NOR operation used in other CIM designs, we propose novel structures for the Min3 and NAND operations, which are then incorporated into the RIME architecture. To evaluate the performance of the proposed RIME architecture, we utilize the VTEAM RRAM cell model proposed in [29, 30]. The model parameters are set as follows: $Roff = 10\ M\Omega$, $R_{on} = 10\ K\Omega$, $V_{off} = 0.3\ V$, and $V_{on} = -1.5\ V$.

A. NOR logic

The switch between the high resistance state (R_{OFF}) and the low resistance state (R_{ON}) is shown in Fig. 2(a). In Fig.

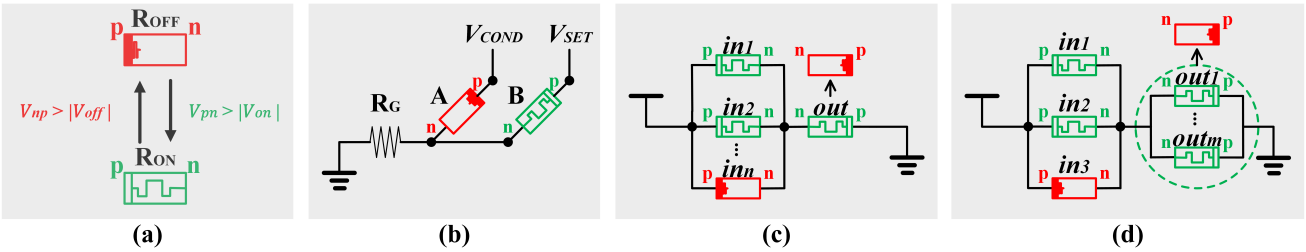


Fig. 2. (a) Switch between high resistance state ($Roff$) and low resistance state (Ron). (b) IMPLY is performed by two simultaneous voltage pulses, VCOND and VSET, applied to RRAM A and B, respectively, to execute conditional toggling on RRAM B depending on the state of RRAM A. (c) MAGIC-based two-input NOR gate is performed with two input RRAMs and an output RRAM, a voltage V_0 is applied at the p terminals of the input RRAMs. (d) RIME-based m-output Min_3 gate is performed with three input RRAMs and m-output RRAMs, a voltage V_0 is applied at the p terminals of the input RRAMs.

2(b), IMPLY is performed by two simultaneous voltage pulses, V_{COND} and V_{SET}, applied to RRAM A and B, respectively, to execute conditional toggling on RRAM B depending on the state of RRAM A. Fig. 2(c) shows the row-wise operation of the NOR logic in the crossbar, where an execution voltage V_0 should be applied to the p terminals of the input RRAM cells, and GND should be connected to the p terminal of the output RRAM cell. It is worth noting that NOT logic can be regarded as a special case of NOR logic, where $n = 1$. The output RRAM cell will be RESET from logic 1 (R_{ON}) to logic 0 (R_{OFF}) as long as there exists at least one input RRAM cell in logic 1 (R_{ON}).

For example, to implement the n-input NOR logic in a row, all n input RRAM cells are initialized to logic 0 (R_{OFF} state), and V_{np} of the output RRAM cell is set lower than V_{off}. For all other input combinations, V_{np} of the output RRAM cell should be higher than V_{off} to ensure that the circuit functions as a NOR gate. To prevent any input RRAM cell from being destructively affected, V_{pn} of all input RRAM cells should be lower than $|V_{on}|$. Considering that a RRAM cell has the characteristic $R_{OFF} >> R_{ON}$, Equation (5) presents the constraint for V_0 in the n-input NOR logic:

$$2 \cdot v_{off} < V_0 \leq |v_{on}| \tag{5}$$

B. Min_3 logic

The minority (Min) gate outputs logic 1 if and only if less than half of the inputs are logic 1. To this end, a m-output Min3 with three input RRAM cells in a row of RRAM crossbar is proposed, as depicted in Fig. 2(d). To operate Min3, the output RRAM cells should be initially SET to logic 1 (RON state). Subsequently, a bias voltage V_0 should be applied to the p terminals of the input RRAM cells, and the p terminals of the output RRAM cells should be connected to GND. If more than one input RRAM cell is in logic 1, the m output RRAM cells will be RESET from logic 1 to logic 0 ($ROFF$ state). However, if no input RRAM cell or only one input RRAM cell is in logic 1, V_{np} of the output RRAM cells will be lower than V_{off}, which is not high enough to RESET the m output RRAM cells to logic 0. To prevent any input RRAM cell from being destructively affected, V_{pn} of all the input RRAM cells should be lower than $|V_{on}|$. Therefore, Equation (6) presents the constraint for V_0 in the m-output Min_3:

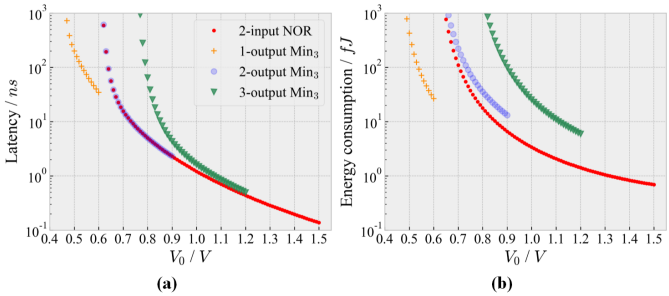


Fig. 3. (a) Latency and (b) energy consumption of basic logic gates in RIME.

$$(\frac{m}{2} + 1) \cdot v_{off} < V_0 \leq min\{(m+1) \cdot v_{off},\ |v_{on}|\} \tag{6}$$

For example, if $m = 1$, $0.45\ V < V_0 \leq 0.6\ V$, and for $m = 2$, $0.6\ V < V_0 \leq 0.9\ V$.

Fig. 3 illustrates the operation latency and energy consumption of the m-output Min_3 logic ($m \in 1, 2, 3$) for different bias voltages V_0. Based on the simulation results, it is observed that the 1-output Min_3 proposed in [31] suffers from two fundamental drawbacks. Firstly, the execution voltages for NOR logic and the 1-output Min_3 logic are dissimilar. Consequently, to integrate both of these logics into the same RRAM crossbar, two distinct voltage sources are required, which significantly increases the complexity of the control module and the peripheral circuits [17]. Secondly, the minimum operation latency and energy consumption of the 1-output Min_3 logic are significantly higher than the NOR logic at their common operating points [17], which negatively impacts the overall performance.

To address these issues, a multiple-output Min_3 logic has been designed for RRAM crossbar to greatly improve efficiency at the acceptable cost of extra RRAM cells. For instance, when $m = 2$, the execution voltage of the Min_3 logic can be selected as $V_0 = 0.9\ V$. The operation latency and energy consumption can remain on the same scale, namely ($1.29\ ns$, $13.22\ fJ$) and ($1.27\ ns$, $6.59\ fJ$) for a 2-output Min_3 logic and a NOR logic, respectively. Consequently, both lower complexity and higher efficiency can be achieved.

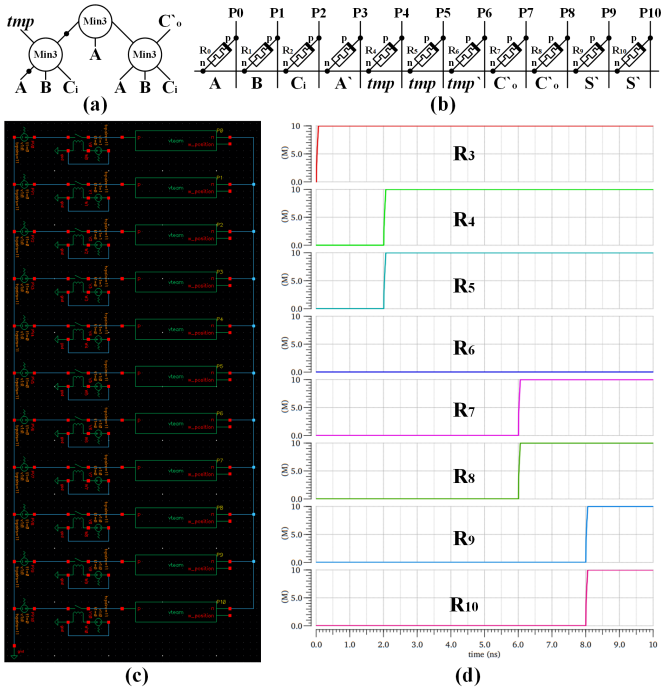


Fig. 4. (a) 1-bit full adder using Min_3 and NOR. (b) Implementation in the RRAM crossbar. (c) Circuits of 1-bit full adder in Cadence Virtuoso. (d) The change of resistance under 2.5 GHz.

C. NAND logic

For the implementation of NAND logic in a row of RRAM crossbar, it is imperative that the output RRAM cells are initially set to logic 1 (R_{ON} state). Subsequently, an execution voltage V_0 must be applied at the p terminals of the input RRAM cells while the p terminals of the output RRAM cells are connected to ground. If all n input RRAM cells are initially in logic 1, the resulting V_{np} of the output RRAM cells will be higher than V_{off}, causing them to be RESET from logic 1 to logic 0 (R_{OFF} state). To prevent any input RRAM cells from being adversely affected, it is necessary to ensure that V_{pn} of all the input RRAM cells is lower than $|V_{on}|$. Therefore, the constraint on V_0 for n-input/m-output NAND logic is expressed by Equation (7).

$$(\frac{m}{n}+1)\cdot v_{off} < V_0 \leq min\{(\frac{m}{n-1}+1)\cdot v_{off},\ |v_{on}|\} \quad (7)$$

Achieving compatibility between Min_3 logic and NAND logic is a particularly challenging task. As illustrated in Fig. 3 and described by Equation (7), ensuring that the NAND logic meets the requirements for hardware implementation in terms of latency and energy consumption necessitates that m must be greater than $n-1$.

In light of the aforementioned analysis, we have determined that the RIME architecture will incorporate the n-input NOR logic, the 2-output Min_3 logic, and the 2-input/2-output NAND logic.

IV. Floating-Point Computation in RIME

In Equation (8), the IEEE-754 format for a 32-bit floating-point number is presented, whereby $Sign$ represents a single bit, $Fraction$ is 23 bits, and $Exponent$ is 8 bits. To facilitate the floating-point multiplication of two 32-bit operands, we utilize fixed-point multiplication and addition. The fundamental logic gates developed in Section III, namely NOR, Min_3, and NAND, are leveraged to design the sub-components of the RIME architecture, which include: (1) a full adder, (2) a fixed-point multiplication module, (3) a floating-point multiplication module, and (4) a control module. These modules will be integrated into the RRAM crossbar to facilitate the implementation of RIME-based floating-point computation units.

TABLE I
The operations of 1-bit full adder using NOR and Min_3

Cycle #	Logic Operation
0	$R_0 = A, R_1 = B, R_2 = C_i$
1	$R_3 = NOR(R_0) = A'$
2	$\{R_4, R_5\} = Min_3(R_1, R_2, R_3) = temp$
3	$R_6 = NOR(R_5) = temp'$
4	$\{R_7, R_8\} = Min_3(R_0, R_1, R_2) = C'_o$
5	$\{R_9, R_{10}\} = Min_3(R_0, R_6, R_8) = S'$

$$X = (-1)^{Sign} \times (1.Fraction)_2 \times 2^{(Exponent-127)} \quad (8)$$

A. Full Adder

Fig. 4(a) illustrates that the carry-bit C_o in a 1-bit full adder can be represented by Equation (9), utilizing a Min_3 logic and a NOR gate. Similarly, the sum S can be expressed in Equation (10) using three Min_3 logic and three NOR gates.

$$C_o = A\cdot B + B\cdot C_i + A\cdot C_i = Min'_3(A, B, C_i) \quad (9)$$

$$S = A \oplus B \oplus C_i = Min'_3(Min'_3(A', B, C_i), A, Min_3(A, B, C_i)) \quad (10)$$

We make the assumption that the inputs, specifically A, B, and C_i, are first loaded into the RRAM cells before computation. Fig. 4(b) depicts the process of mapping the 1-bit full adder into the RRAM crossbar to obtain C'_o and S'. As illustrated in Table I and Fig. 4(b), a RIME-based full adder requires just 11 RRAM cells and five clock cycles. This significantly outperforms the MAGIC-based 1-bit full adder [32], which utilizes only NOR logic gates and requires 15 RRAM cells and 12 clock cycles to execute a 1-bit full adder. As shown in Fig. 4(c), we simulate the Min_3 gate and the full adder logic in Cadence Virtuoso and obtain the results under 2.5 GHz in Fig. 4(d).

B. Fixed-Point Multiplication

The shift-and-add and Wallace-tree algorithms are two commonly used methods for fixed-point multiplication. While the Wallace-tree algorithm has higher speed, it also consumes more area compared to the shift-and-add algorithm [32]. The proposed RIME-based multiplication adopts the Wallace-tree algorithm and splits the RRAM crossbar for parallel computation [17]. This ensures both time and energy efficiency.

Fig. 5 shows the flow diagram for a 4-bit fixed-point multiplication [33], which involves two 4-bit operands, $(a_3a_2a_1a_0)$

and $(b_3b_2b_1b_0)$, with $p_{ij} = a_ib_j$ representing the partial product. Three full adders operate in parallel from S1 to S3, while only one full adder operates from S4 to S6. This selective operation is achieved by two switches that divide a row of RRAM cells into three full adders, thereby preventing interference between the three full adders when the switches are turned off.

To execute each stage of the N-bit fixed-point multiplication, the partial products are initially written into the corresponding RRAM cells of each full adder sequentially, which is achieved using the NOR logic, $p_{ij} = NOR(a'_i, b'_j)$. Subsequently, all $N-1$ full adders operate in parallel to compute the C'_os and S's. Finally, the C_os are written into the target RRAM cells of local full adders in parallel, and the Ss are written into the target RRAM cells of the neighboring full adders sequentially for the next stage. According to the simulation method in Fig. 4, We have simulated N-bit fixed-point multiplication using Cadence Virtuoso, where N takes values from 4 to 8. The simulation results verify the correctness of the design and the latency for the N-bit fixed-point multiplication is $2 \times N^2 + 16 \times N - 19$ clock cycles.

C. 32-bit Floating-Point Multiplication

The process of 32-bit floating-point multiplication entails a combination of XOR operations, addition, and fixed-point multiplication. Specifically, in the case of 24-bit fixed-point multiplication, 23 full adders are necessary for the first 23 stages, while only one full adder is required in the final 23 stages. In the last 23 stages, two full adders can function in parallel for both the 1-bit XOR operation and the 8-bit $Exponent$ addition. As a result, the overall latency of a 32-bit floating-point multiplication equals that of a 24-bit fixed-point multiplication. To execute a RIME-based 32-bit floating-point multiplication, it is necessary to store two operands using $2 \times 33 = 66$ RRAM cells, while an additional $1+10+48 = 59$ RRAM cells are required to store the result. Additionally, $23 \times 11 = 253$ RRAM cells are needed for the 23 full adders. Consequently, a row of 378 RRAM cells is necessary for a RIME-based 32-bit floating-point multiplication.

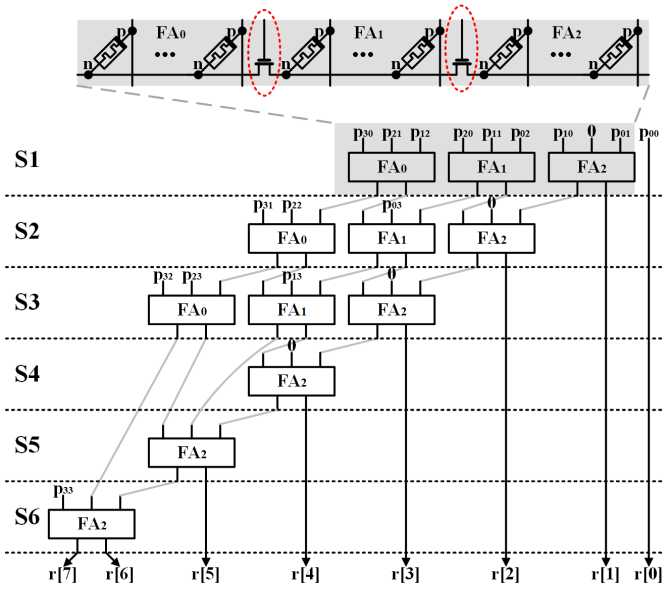


Fig. 5. Implementation of a 4-bit Wallace-tree multiplication in RIME.

V. Transformer Inference Accelerator in RIME

In order to enhance the efficiency of The Transformer model and ensure the accuracy of the inference phase, we propose the RIME architecture, which is designed to accelerate applications similar to The Transformer that utilize the self-attention mechanism. This section provides an overview of the RIME architecture and outlines the pipeline of matrix-matrix multiplication (MatMul) and softmax in RIME.

A. Overview

Fig. 6(a) presents an overview of the proposed RIME architecture, which is composed of the central controller, the CIM controller, the external I/O interface, the CMOS computing unit, and the RRAM tiles.

Central Controller. The central controller is responsible for managing the entire RIME architecture to accelerate the Transformer inference, including switching between different function modes, preprocessing and transferring data, communicating with external units, etc.

CMOS Computing Units. The CMOS computing units perform the calculations of e^x and division that are not demanding on parallelism and are too complex to be implemented in an RRAM crossbar.

Tile. As illustrated in Fig. 6(b), all the tiles are controlled by a CIM controller for parallel multiplications, and the input/output data are stored in the I/O buffers. Each tile has 4×4 RRAM crossbars, and a data transfer controller is deployed in each RRAM crossbar as shown in Fig. 6(c). The data transfer controller ensures the exchange efficiency of operands and intermediate data in the crossbar.

CIM Controller. As illustrated in Fig. 6(d), all the titles are controlled by the CIM controller to facilitate the implementation of 32-bit floating-point addition and multiplication. The decoders and sense amplifiers are designed using proven analog circuit technology. The major steps for parallel computation are provided in Fig. 6(e).

B. RRAM Crossbar

The RRAM crossbars in RIME are capable of supporting massively parallel computation. For instance, a 512×512 RRAM block, as depicted in Fig. 6(d), can support up to 512 multiplications being executed in parallel. As illustrated in Fig. 6(c), several RRAM blocks are cascaded together. In each tile, the CIM controller oversees the column decoders to enable a large number of additions and multiplications to be executed simultaneously. The data transfer controller governs the row decoders independently, allowing for fine-tuned block activation and efficient data transfer between different RRAM blocks in a row-parallel manner. As a result, MatMul and softmax can be substantially accelerated in RIME by executing the additions and multiplications in parallel.

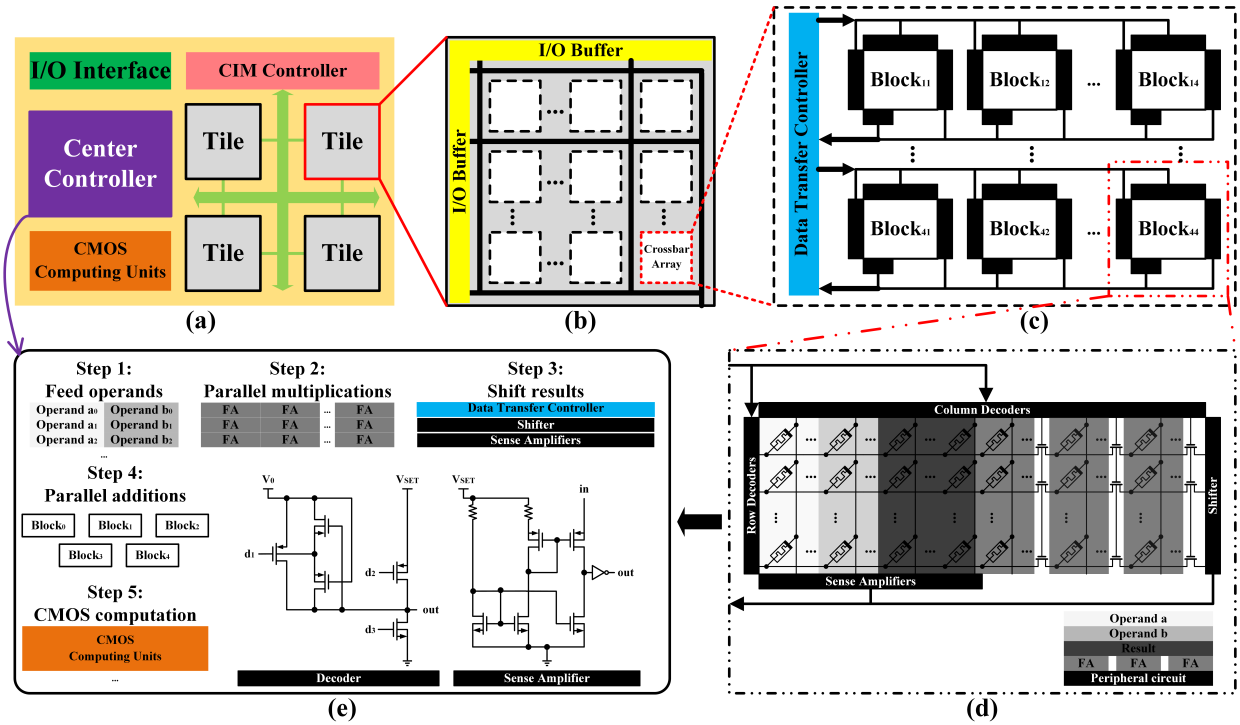


Fig. 6. (a) Overview of RIME architecture. (b) Structure of a tire. (c) A crossbar. (d) A RRAM block. (e) Major steps of CIM controller.

C. Function Modes

The center controller in RIME incorporates four modes that offer essential functionalities for the Transformer model, as elaborated below.

Writing Mode. The center controller manages the decoders in the writing mode. It facilitates the writing of two floating-point numbers into a row in a column-parallel manner within a single clock-cycle, or the writing of distinct floating-point numbers into RRAM blocks within multiple clock-cycles in a row-parallel manner.

Reading Mode. The center controller manages the decoders in the reading mode. In a column-parallel mode, the switches at the right end of the RRAM blocks (Fig. 6(d)) are activated, and the column decoders read the values of the selected column of RRAM cells within a single clock cycle. In a row-parallel mode, the row decoders read the operands and the result in the selected row within one clock cycle.

Transferring Mode. The center controller manages the data transfer controller in the transferring mode. It transfers data in a row-parallel manner by reading the values of the selected column of the RRAM block and writing them into the target column either in a local block or another block.

Computing Mode. The center controller manages the CIM controller and the CMOS computing unit in the computing mode. The CIM controller provides digital signals to the column decoders and the row decoders to perform selected floating-point additions or multiplications in parallel. The CMOS computing unit instantiates several modules to execute e^x and transmits the results to the data transfer controllers based on instructions from the center controller.

Irrespective of the number of rows in the RRAM block, the latency of data writing, reading, and transferring depends solely on the bit-width of the floating-point number. Therefore, RIME-based RRAM crossbars are equipped with massively parallel and scalable computing units. Additionally, for data-intensive applications, RIME offers fundamental functionalities, including MatMul and softmax, with 32-bit floating-point precision and highly efficient communication between neighboring RRAM crossbars.

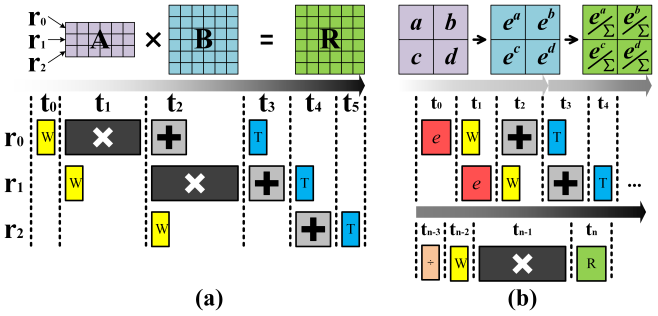


Fig. 7. (a) Pipeline of RIME-based MatMul. (b) Pipeline of RIME-based softmax.

D. Matrix-matrix Multiplication

Conventional von Neuman architecture results in significant time and energy consumption due to the need to transfer intermediate results between processing units and memory, particularly during MatMul operations, which make up a considerable portion of the processing required for the Transformer model. To address this issue, the proposed RIME architecture optimizes MatMul operations by utilizing matrix

decomposition and a pipeline structure, as illustrated in Fig. 7(a). During the initialization phase, each element in matrix $\boldsymbol{B}$ is assumed to be written into the RRAM crossbar.

At time t_0, RIME writes the first row of matrix $\boldsymbol{A}$ into the RRAM crossbar. At t_1, RIME performs 32-bit floating-point multiplication and writes the second row of matrix $\boldsymbol{A}$ into the crossbar. At t_2, RIME performs additions and multiplications and writes the third row of matrix $\boldsymbol{A}$ into the crossbar. At t_3, the results are transferred to another crossbar and additions and multiplications are performed. At t_4, the results are again transferred to another crossbar and additions are performed. Finally, at t_5, the results are transferred to another crossbar for the next MatMul operation. m is the number of elements in a row of matrix $\boldsymbol{A}$ and N is the number of elements in matrix $\boldsymbol{B}$. In the RIME architecture, N 32-bit floating-point multiplications can be performed simultaneously with operators writing, data transfer, and $m/2$ additions, resulting in a 11.7% performance improvement in MatMul.

E. Softmax

In the context of the Scaled Dot-Product Attention, the softmax function plays a critical role but can also result in significant time and energy consumption. To address this, the proposed RIME architecture optimizes the softmax function using a similar approach to MatMul as shwon in Fig. 7(b). However, given that performing 32-bit floating-point e^x and division operations in RRAM is not efficient, the CMOS computing unit is used to instantiate multiple e^x modules for parallel operation and a division module for $1/\Sigma$.

During the initialization phase, each element in the matrix $\boldsymbol{A}$ is assumed to be written into the RRAM crossbar. The CMOS computing unit performs e^x and writes the results to the RRAM crossbar in the first few clock cycles (t_0~t_1). Then, RIME performs the additions and transfers the intermediate values in the next few clock cycles (t_2~t_4). Finally, the results are transferred to another crossbar for the following operations in the remaining clock cycles (t_{n-3}~t_n).

The pipeline structure of RIME enables the CMOS computing unit to improve the efficiency of the softmax function by 4.5% using only one division module and several e^x modules.

VI. Evaluation

The RIME architecture was evaluated in our experiment from three perspectives to demonstrate its scalability and high efficiency. Firstly, we compared the 32-bit floating-point multiplication in RIME with state-of-the-art architectures [14, 16, 18, 19] in terms of latency, area, and energy consumption. Then, we compared the time and energy consumption of MatMul and softmax in the RIME architecture with those in the GPU platform. Finally, we compared the overall performance of The Transformer model in the RIME architecture with that in the GPU platform.

A. 32-bit Floating-Point Multiplication

1) Experimental Setup: In the performance evaluation, we utilize the VTEAM RRAM model proposed in [29] with the parameters and simulation setup described in [34]. To obtain the latency of different 32-bit floating-point multiplications, the period is set to 2 nm according to the simulation results of the basic gates from Cadence Virtuoso. The voltage at each port is controlled by the CIM controller, which is implemented using Verilog and is separately evaluated for area and energy consumption using Synopsys Design Compiler with the FreePDK 45nm library [34].

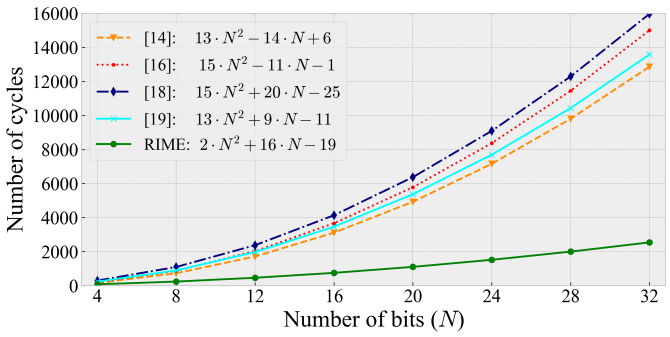


Fig. 8. Latency of N-bit fixed-point multiplication.

2) Comparison with CIM-based architecture: We compare the RIME-based 32-bit floating-point multiplication with the state-of-the-art CIM-based architectures [14, 16, 18, 19] in terms of latency, area, and energy consumption.

Latency. As shown in Fig. 8, we implement the N-bit fixed-point multiplications in [14, 16, 18, 19] and analyzed the latency compared with RIME. We consider only the algorithms that had full precision without skipping the most significant bits. Since RIME executes the 1-bit XOR, the 8-bit addition, and the 24-bit fixed-point multiplication in parallel, the total number of cycles for the 32-bit floating-point multiplication is $2 \times 24^2 + 16 \times 24 - 19 = 1,517$ in RIME. In contrast, it is $6 + (12 \times 8 + 1) + (13 \times 24^2 - 14 \times 24 - 1) = 7,214$ in [14], $6 + (12 \times 8 + 1) + (15 \times 24^2 - 11 \times 24 - 1) = 8,478$ in [16], $6 + (12 \times 8 + 1) + (15 \times 24^2 + 20 \times 24 - 25) = 9,198$ in [18], and $13 \times 24^2 + 9 \times 24 - 11 = 7,693$ in [19]. Therefore, the RIME-based 32-bit floating-point multiplication is significantly faster than the state-of-the-art.

Area. Area. The total area of a 32-bit floating-point multiplication can be divided into three parts: the RRAM crossbar, the peripheral circuit, and the control module. For a single 32-bit floating-point multiplication, the number of RRAM cells required is 523 for [14], 8,450 for [16], 456 for [18], 628 for [19], and 378 for RIME. The peripheral circuit in [16] is the most complicated due to the need to read out data from

TABLE II
AREA / μm^2 & ENERGY CONSUMPTION / pJ FOR A SINGLE 32-BIT FLOATING-POINT MULTIPLICATION

	RRAM crossbar	Peripheral Circuit	Control Module
[14]	173 & 171	631 & 532	8,425 & 7,621
[16]	2,906 & 203	1,326 & 1,041	8,743 & 9,082
[18]	163 & 158	525 & 462	9,243 & 9,854
[19]	195 & 189	734 & 689	7,148 & 6,353
RIME	**122 & 136**	**448 & 321**	**4,361 & 1,090**

TABLE III
EFFECT OF V_0/V DROP ON THE ACCURACY OF 32-BIT FLOATING-POINT MULTIPLICATIONS

	0.70	0.75	0.80	0.85	0.90
FloatPIM	0.71	0.83	0.95	1.00	1.00
RIME	**0.70**	**0.88**	**0.97**	**1.00**	**0.99**

the RRAM crossbar, transfer it to the periphery, process it, and then write it back [35]. The area of the control module is jointly determined by the total number of cycles and the total number of RRAM cells, as the execution voltages at the n and p terminals of each RRAM cell are controlled cycle-by-cycle to implement the multiplication. The results in Table II demonstrate that, for a single 32-bit floating-point multiplication, RIME is more area-efficient than the state-of-the-art.

Energy consumption. The experiments also measured the energy consumption of the RRAM crossbar, the peripheral circuit, and the control modules. The energy consumption of the RRAM crossbar is positively correlated with the total number of underlying logic operations, while the energy consumption of the peripheral circuit is primarily driven by the decoders. The overall energy consumption of the control module is determined by the latency and complexity of the algorithm. The results in Table II demonstrate that, for a single 32-bit floating-point multiplication, RIME is more energy-efficient than the state-of-the-art.

Accuracy. Due to the non-ideal factors of RRAMs, the accuracy of the CIM architecture degrades if the execution voltage fluctuates. We simulate the RIME-based and the FloatPIM-based 32-bit floating-point multiplications with random operands using the VTEAM RRAM cell model. The clock frequency is set to 2.5 GHz and V_0 is set in [0.70 V, 0.90 V]. The average accuracy is shown in Table III, which indicates that low voltage will greatly impact the accuracy of CIM-based computation.

3) Comparison with von Neumann architecture: Although a single 32-bit floating-point multiplication in a von Neumann Application Specific Integrated Circuit (ASIC) implementation has a latency of 3 ns, an area of 3,523 μm^2, and an energy consumption of 7 pJ [36], which are less than the RIME-based implementation, the ASIC-based implementation has two fundamental drawbacks. On the one hand, due to area and power constraints, a limited number of computation units can be executed in parallel on a chip. On the other hand, the movement of data from off-chip memory significantly increases latency and energy consumption. For instance, accessing two 32-bit operands from an on-chip memory results in an energy consumption of about 78 pJ, which is 200 times greater than external DRAM (10 nJ) [37].

In contrast, the proposed RIME architecture facilitates scalable computation, allowing for thousands of 32-bit floating-point multiplications to execute in parallel while eliminating the need for data movement in ASIC implementation. The results depicted in Fig. 3 indicate that the latency of a logic operation is less than 2.5 ns, thereby resulting in a total latency of $1,517 \times 2.5 = 3,793$ ns for a 32-bit floating-point multiplication. Taking into account the latency of data movement [37], the throughput of RIME with 512 32-bit floating-point multiplications is significantly higher than that of ASIC implementation. As illustrated in Fig. 9, the relationship between the number of 32-bit floating-point multiplications executing in parallel and the total area and energy consumption is linear. For data-intensive applications, the RIME architecture is expected to outperform conventional von Neumann designs in terms of throughput and efficiency.

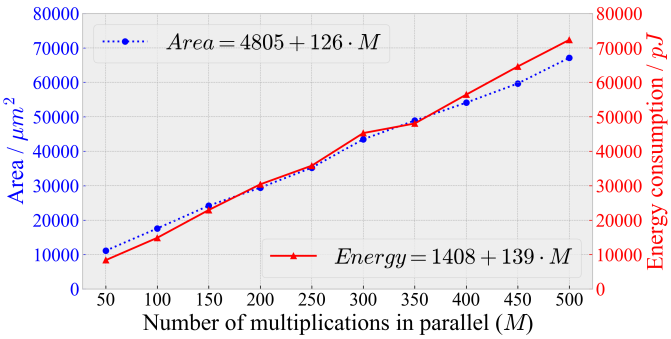


Fig. 9. Area and energy consumption of M 32-bit floating-point multiplications in RIME.

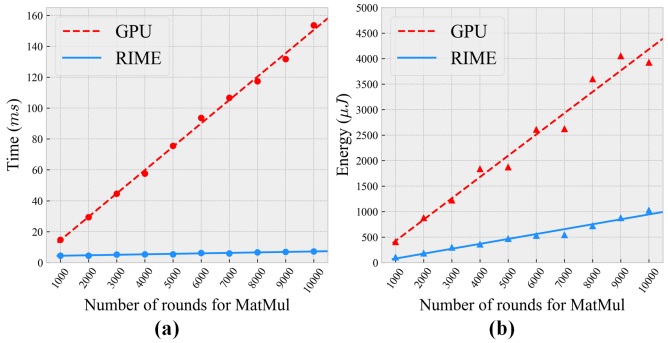


Fig. 10. (a) Time consumption of MatMul in GPU and RIME architecture. (b) Energy consumption of MatMul in GPU and RIME architecture.

B. Functions

1) Experimental Setup: The implementation of softmax and MatMul was carried out in Python, while the configurations of the GPU platform are presented in Table IV. To measure the time and energy consumption of the GPU, we utilized the pyJoules package [38], which is a software toolkit that quantifies the energy footprint of a host machine during the execution of Python code [39]. For softmax, we instantiated eight e^x units, as proposed in [40], and one $1/x$ unit (where x represents a 32-bit floating-point number) using the FreePDK 45 nm library [34].

2) MatMul: We conducted $N, 000$ rounds of $\boldsymbol{A} \cdot \boldsymbol{B}$ on both the GPU platform and the RIME architecture, with the size of $\boldsymbol{A}$ being 8×32 and the size of $\boldsymbol{B}$ being 8×32. The time and energy consumption results are presented in Fig. 10(a) and Fig. 10(b), respectively. Considering the existing physical limitations [41], we assumed that each RRAM Block has a size of 512×512, each crossbar comprises 4×4 Blocks, and

TABLE IV
CONFIGURATIONS OF THE GPU PLATFORM

GPU	NVIDIA GeForce RTX 3080
Memory	10 GB
Memory Bandwidth	760 GB/s
CUDA Cores	8,704
CUDA Version	11.0
Power Consumption	320 W

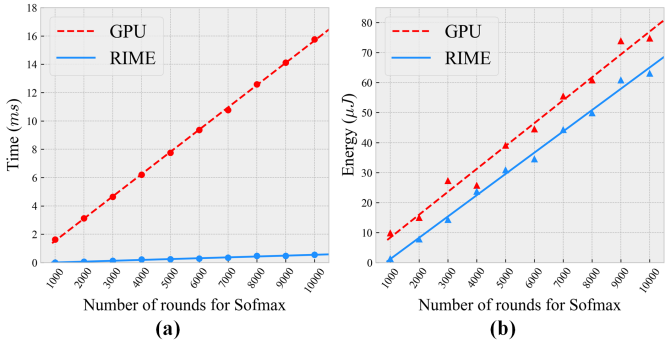


Fig. 11. (a) Time consumption of softmax in GPU and RIME architecture. (b) Energy consumption of softmax in GPU and RIME architecture.

each tire consists of 4×4 crossbars. As a result, the RIME architecture can simultaneously handle $512 \times 16 \times 16 \times 4 = 2^{19}$ 32-bit floating-point multiplications. Additionally, the pipeline structure illustrated in Fig. 7(a) provides the RIME architecture with high efficiency and scalability advantages.

3) Softmax: We performed $N, 000$ rounds of $\boldsymbol{Softmax}(\boldsymbol{C})$ on both the GPU platform and the RIME architecture, with the size of $\boldsymbol{C}$ being 16×16. The time consumption and energy consumption results are presented in Fig. 11(a) and Fig. 11(b), respectively. As the e^x and $1/x$ modules are situated outside of the crossbars, most of the energy in softmax is consumed during data transfer. Nevertheless, the energy consumption of softmax is considerably lower than that of MatMul. Therefore, the data transfer between the CMOS computing unit and the RRAM crossbars does not significantly affect the overall efficiency.

C. Transformer Accelerator

1) Experimental Setup: We implemented the Transformer model [4] using PyTorch and simulated the RIME architecture using NeuroSim [42], with the same configurations as in [14]. To evaluate the center controller proposed for RIME, we utilized Verilog and Synopsys Design Compiler for implementation and analysis.

2) Encoder and Decoder: Based on the results of the experiments depicted in Fig. 12 and Fig. 13, we can draw three conclusions. Firstly, Self-Attention incurs the highest time and energy consumption. Secondly, the overall time consumption of the Encoder in the RIME architecture is 42.5% of that in the GPU platform, and the overall energy consumption of the Encoder in the RIME architecture is 54.7% of that in the GPU platform. Thirdly, the overall time consumption of the Decoder in the RIME architecture is 43.9% of that in the GPU platform,

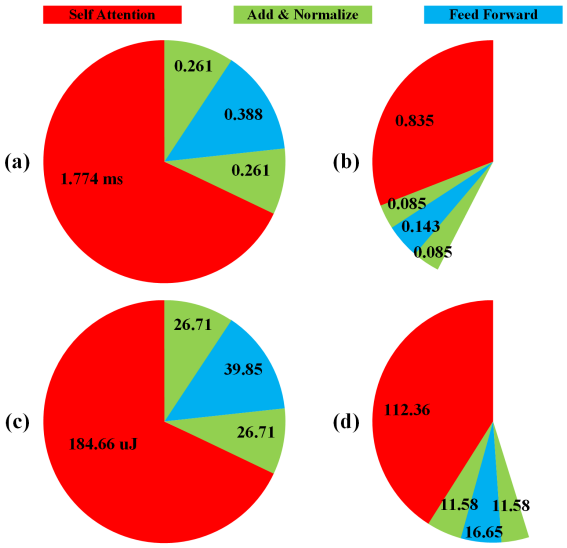


Fig. 12. (a) Time consumption of Encoder in GPU. (b) Time consumption of Encoder in RIME architecture. (c) Energy consumption of Encoder in GPU. (d) Energy consumption of Encoder in RIME architecture.

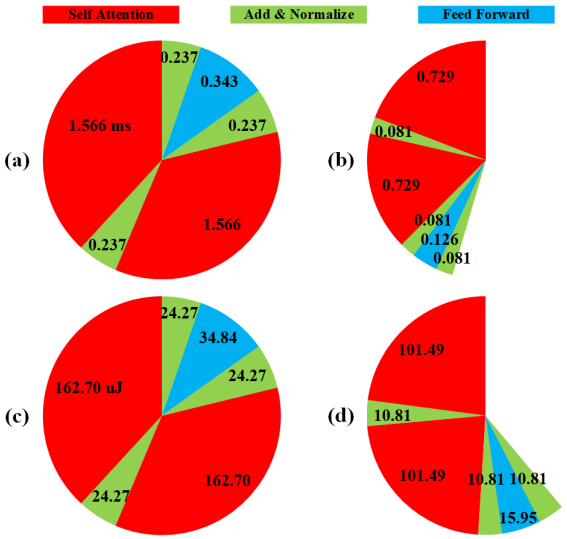


Fig. 13. (a) Time consumption of Decoder in GPU. (b) Time consumption of Decoder in RIME architecture. (c) Energy consumption of Decoder in GPU. (d) Energy consumption of Decoder in RIME architecture.

and the overall energy consumption of the Decoder in the RIME architecture is 58.1% of that in the GPU platform.

D. Future Work

Reliability and efficiency are equally important for computationally intensive systems. The non-ideal factors of RRAMs and the environments, such as fluctuations in temperature and voltage will lead to errors in calculations. Therefore, error correction schemes and fault-tolerance methods are essential to guarantee the reliable outputs of the Transformer model. In future work, we will focus on improving the reliability of the RIME-based architecture against the hash environments.

VII. Conclusion

In this paper, we introduce RIME, an innovative in-memory floating-point computation architecture that leverages resistive random-access memory (RRAM) technology to accelerate The Transformer model. The RIME architecture is designed with single-cycle NOR, NAND, and Minority logic, which effectively reduces the computation latency and energy consumption. Our experimental results demonstrate that RIME achieves remarkable performance improvements compared to the state-of-the-art in 32-bit floating-point multiplication, with a 4.8X speedup, 1.9X area improvement, and 5.4X energy efficiency. Moreover, our proposed RIME-based Transformer accelerator achieves 2.3x speedup and 1.7x energy efficiency improvements over the GPU platform while ensuring inference accuracy. Our study highlights the potential of RRAM-based in-memory computing for high-performance and energy-efficient machine learning applications.

Acknowledgement

Zhaojun Lu's work is supported by the National Natural Science Foundation of China under No. 62202178. Xueyan Wang's work is supported by National Natural Science Foundation of China under No.62004011. Dr. Arafin received support in computing instruments from the ARLIS project at the University of Maryland, Sub Task Order #95109-Z9634201.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, no. 3. Makuhari, 2010, pp. 1045–1048.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[5] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[6] B. Zhang, D. Xiong, and J. Su, "Accelerating neural transformer via an average attention network," *arXiv preprint arXiv:1805.00631*, 2018.

[7] T. Xiao, Y. Li, J. Zhu, Z. Yu, and T. Liu, "Sharing attention weights for fast transformer," *arXiv preprint arXiv:1906.11024*, 2019.

[8] X. Yang, B. Yan, H. Li, and Y. Chen, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[9] X. Wang, J. Yang, Y. Zhao, X. Jia, R. Yin, X. Chen, G. Qu, and W. Zhao, "Triangle counting accelerations: From algorithm to in-memory computing architecture," *IEEE Transactions on Computers*, 2021.

[10] J. Park, Y. Jeong, J. Kim, S. Lee, J. Y. Kwak, J.-K. Park, and I. Kim, "High dynamic range digital neuron core with time-embedded floating-point arithmetic," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2022.

[11] T.-H. Kim, B. Song, I.-J. Jung, and S.-O. Jung, "A sneak current compensation scheme with offset cancellation sensing circuit for reram-based cross-point memory array," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 4, pp. 1583–1594, 2021.

[12] F. Karimzadeh, J.-H. Yoon, and A. Raychowdhury, "Bits-net: Bit-sparse deep neural network for energy-efficient rram-based compute-in-memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 5, pp. 1952–1961, 2022.

[13] X. Wang, J. Yang, Y. Zhao, X. Jia, G. Qu, and W. Zhao, "Hardware security in spin-based computing-in-memory: Analysis, exploits, and mitigation techniques," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 4, pp. 1–18, 2020.

[14] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 802–815.

[15] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 367–382.

[16] M. Imani, S. Gupta, and T. Rosing, "Ultra-efficient processing in-memory for data intensive applications," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[17] Z. Lu, M. T. Arafin, and G. Qu, "Rime: A scalable and energy-efficient processing-in-memory architecture for floating-point operations," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2021, pp. 120–125.

[18] J. Xu, Y. Zhan, Y. Li, J. Wu, X. Ji, G. Yu, W. Jiang, R. Zhao, and C. Wang, "In situ aging-aware error monitoring scheme for imply-based memristive computing-in-memory systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 1, pp. 309–321, 2021.

[19] S. E. Fatemieh, M. R. Reshadinezhad, and N. TaheriNejad, "Fast and compact serial imply-based approximate full adders applied in image processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 13, no. 1, pp. 175–188, 2023.

[20] Y. Bondarenko, M. Nagel, and T. Blankevoort, "Understanding and overcoming the challenges of efficient transformer quantization," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Nov. 2021, pp. 7947–7969.

[21] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3.int8(): 8-bit matrix multiplication for transformers at scale," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 30 318–30 332.

[22] Z. Zhu, H. Sun, Y. Lin, G. Dai, L. Xia, S. Han, Y. Wang, and H. Yang, "A configurable multi-precision cnn computing framework based on single bit rram," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[23] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'memristive'switches enable 'stateful'logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[24] S. E. Fatemieh, M. R. Reshadinezhad, and N. TaheriNejad, "Approximate in-memory computing using memristive imply logic and its application to image processing," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 3115–3119.

[25] O. Leitersdorf, R. Ronen, and S. Kvatinsky, "Multpim: Fast stateful multiplication for processing-in-memory," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1647–1651, 2021.

[26] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[27] M. R. Alam, M. H. Najafi, and N. TaheriNejad, "Exact stochastic computing multiplication in memristive memory," *IEEE Design & Test*, vol. 38, no. 6, pp. 36–43, 2021.

[28] M. R. Alam, M. H. Najafi, and N. Taherinejad, "Sorting in memristive memory," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 4, pp. 1–21, 2022.

[29] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[30] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, no. 1, pp. 13–24, 2013.

[31] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–7.

[32] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (magic)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.

[33] G. G. Kumar and S. K. Sahoo, "Implementation of a high speed multiplier for high-performance and low power applications," in *2015 19th International Symposium on VLSI Design and Test*. IEEE, 2015, pp. 1–4.

[34] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 171–178.

[35] A. Haj-Ali, R. Ben-Hur, N. Wald, and S. Kvatinsky, "Efficient algorithms for in-memory fixed point multiplication using magic," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[36] P. Anuhya and R. Dhanabal, "Asic implementation of efficient floating point multiplier," in *2018 4th International Conference on Electrical Energy Systems (ICEES)*. IEEE, 2018, pp. 138–141.

[37] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-cost inter-linked subarrays (lisa): Enabling fast inter-subarray data movement in dram," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 568–580.

[38] M. Colmant, P. Felber, R. Rouvoy, and L. Seinturier, "Wattskit: Software-defined power monitoring of distributed systems," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 514–523.

[39] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier, "Powerapi: A software library to monitor the energy consumed at the process-level," *ERCIM News*, vol. 2013, no. 92, 2013.

[40] M. Heidarpour, A. Ahmadi, and R. Rashidzadeh, "A cordic based digital hardware for adaptive exponential integrate and fire neuron," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 11, pp. 1986–1996, 2016.

[41] S. Rafiq, J. Hazra, M. Liehr, K. Beckmann, M. Abedin, J. S. Pannu, S. K. Jha, and N. C. Cady, "Investigation of reram variability on flow-based edge detection computing using hfo 2-based reram arrays," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 7, pp. 2900–2910, 2021.

[42] P.-Y. Chen, X. Peng, and S. Yu, "Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3067–3080, 2018.

Zhaojun Lu received his Ph.D. degree in microelectronic and solid-state electronics from Huazhong University of Science and Technology, Wuhan, China, in 2018. He is currently an Assistant Professor at the School of Cyber Science and Engineering at Huazhong University of Science and Technology, Wuhan, China. His research interests include embedded system security, Very Large Scale Integration (VLSI) design, and Artificial Intelligence (AI) security and privacy.

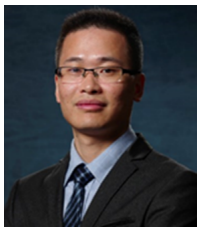
Xueyan Wang received the BS degree in computer science and technology from Shandong University, Jinan, China, in 2013, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2018. From 2015 to 2016, she was a visiting scholar at the University of Maryland, College Park, MD, USA. She is currently an Assistant Professor with the School of Integrated Circuit Science and Engineering at Beihang University, Beijing, China. Her current research interests include processing-in-memory architectures, AI chips, and hardware security.

Md Tanvir Arafin (S'09) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA. He is currently an Assistant Professor with the Department of Cyber Security Engineering at George Mason University, VA, USA. His current research interests include semiconductor physics, integrated circuits, and embedded security of microelectronic devices.

Haoxiang Yang Majors in Computer Science at Warsaw University of Technology, Warsaw, Poland. He is a junior college student and developed a series of favorable science studying methods. He is familiar with data structure, C++, operating systems, PyTorch, deep learning.

Zhenglin Liu received his Ph.D. degree from the Department of Electronic Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2001.

He is currently a Professor at the School of Optical and Electronic Information, Huazhong University of Science and Technology. His main research interests include embedded system security and very-large-scale integration (VLSI) design.



Jiliang Zhang received the Ph.D. degree in Computer Science and Technology from Hunan University, Changsha, China in 2015. From 2013 to 2014, he worked as a Research Scholar at the Maryland Embedded Systems and Hardware Security Lab, University of Maryland, College Park. From 2015 to 2017, he was an Associate Professor with Northeastern University, China. He is currently a Full Professor at Hunan University. He is Vice Dean of the College of Semiconductors (College of Integrated Circuits) at Hunan University, the Director of Chip Security Institute of Hunan University, and the Secretary-General of CCF Fault-Tolerant Computing Professional Committee. His current research interests include Hardware Security, Integrated Circuit Design and Intelligent System. He has authored more than 60 technical papers in leading journals and conferences. He was the recipient of CCF Integrated Circuit Early Career Award. He is serving as a steering member for Hardware Security Forum of China and a Guest Editor of the IEEE Transactions on Circuits and Systems II: Express Briefs. He is a senior member of IEEE.

Gang Qu (Fellow, IEEE) received the B.S. and M.S. degrees in mathematics from the University of Science and Technology of China, in 1992 and 1994, respectively, and the Ph.D. degree in computer science from the University of California, Los Angeles, in 2000. Upon graduation, he joined the University of Maryland at College Park, where he is currently a professor in the Department of Electrical and Computer Engineering and Institute for Systems Research. Dr. Qu is the director of Maryland Embedded Systems and Hardware Security Lab and the Wireless Sensors Laboratory. His primary research interests are in the area of embedded systems and VLSI CAD with focus on low power system design and hardware related security and trust. He is an associate editor for the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Emerging Topics in Computing, ACM Transactions on Design Automation of Electronic Systems, Journal of Hardware and System Security, Journal of Computer Science and Technology, and Integration, the VLSI Journal. He has served 18 times as the general or program chair/co-chair for conferences, symposiums and workshops. He is the co-founder of IEEE Asian Hardware Oriented Security and Trust Symposium, Hot Picks in Hardware and System Security Workshop, and the IEEE CEDA Hardware Security and Trust Technical Committee.