AttentionRC: A Novel Approach to Improve Locality Sensitive Hashing Attention on Dual-Addressing Memory

Chun-Lin Chu, *Graduate Student Member, IEEE*, Yun-Chih Chen, *Member, IEEE*, Wei Cheng, Ing-Chao Lin, *Senior Member, IEEE*, and Yuan-Hao Chang, *Fellow, IEEE*

***Abstract*—Attention is a crucial component of the Transformer architecture and a key factor in its success. However, it suffers from quadratic growth in time and space complexity as input sequence length increases. One popular approach to address this issue is the Reformer model, which uses locality-sensitive hashing (LSH) attention to reduce computational complexity. LSH attention hashes similar tokens in the input sequence to the same bucket and attends tokens only within the same bucket. Meanwhile, a new emerging nonvolatile memory (NVM) architecture, row column NVM (RC-NVM), has been proposed to support row- and column-oriented addressing (i.e., dual addressing). In this work, we present AttentionRC, which takes advantage of RC-NVM to further improve the efficiency of LSH attention. We first propose an LSH-friendly data mapping strategy that improves memory write and read cycles by 60.9% and 4.9%, respectively. Then, we propose a sort-free RC-aware bucket access and a swap strategy that utilizes dual-addressing to reduce 38% of the data access cycles in attention. Finally, by taking advantage of dual-addressing, we propose transpose-free attention to eliminate the transpose operations that were previously required by the attention, resulting in a 51% reduction in the matrix multiplication time.**

***Index Terms*—Attention, dual-addressing memory, locality sensitive Hashing attention, row-column nonvolatile memory (RC-NVM), reformer.**

I. Introduction

THE ATTENTION models have demonstrated remarkable performance across various domains [1], [2], [3]. Their key innovation is the ability to capture long-range dependencies and effectively manage input sequences of varying lengths. However, this requires significant matrix multiplication between embedding matrices, which contributes heavily to the computational load in Transformer models.

Nonvolatile memory (NVM) is a promising alternative to DRAM due to its key advantages, including larger memory capacity, lower costs, near-DRAM access latencies, and minimal dynamic power consumption. These features make NVM ideal for applications requiring efficient memory access and storage. Additionally, advances in NVM manufacturing technology have led to more powerful storage and memory devices. In summary, NVM offers greater memory capacity, lower costs, and similar access speeds to DRAM, making it a preferred choice for handling large data volumes and reducing power consumption.

At the algorithm level, the attention mechanism suffers from a limitation, known as the *quadratic growth* [4], [5], [6], [7], [8], [9], in both time and space complexity. Assuming that the length of the input sequence equals N, the computational complexity for attending every token in the sequence grows at $O(N^2)$. The quadratic growth issue limits its capacity to handle long sequences since the required computation becomes prohibitively expensive as the length of the input sequence grows.

To address the quadratic growth issue in the attention mechanism, the Reformer [10] model has emerged as a promising solution. It outperforms other attention-like models [4], [6], [9] through two key factors: 1) its efficiency in handling long sequences and 2) its memory-saving strategy based on the reversible residual network [11]. The Reformer model demonstrates excellent results on various natural language processing tasks, showcasing its capability to process longer sequences with faster computation and less memory usage.

As shown in Fig. 1, Reformer employs LSH attention, which groups tokens with similar characteristics into distinct buckets by using the hash functions. Subsequently, the attention mechanism is applied exclusively within each bucket, significantly reducing the required computations. This approach yields a substantial improvement in computational efficiency, allowing it to handle longer sequences with less computation.

With the advancement of memory architecture, a new NVM device, called row-column NVM (RC-NVM) [12], was created. Unlike conventional memory technologies that support

Manuscript received 12 August 2024; accepted 12 August 2024. This work was supported in part by the National Science and Technology Council, Taiwan (R.O.C) under Grant NSTC 109-2628-E-006-012-MY3, Grant 110-2221-E-006-084-MY3, Grant 113-2923-E-006-009, Grant 113-2221-E-006-215, Grant 113-2223-E-001-001, Grant 111-2221-E-001-013-MY3, Grant 113-2927-I-001-502, and Grant 111-2923-E-002-014-MY3; and in part by the Academia Sinica under Grant AS-IA-111-M01. This article was presented at the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding authors: Ing-Chao Lin; Yuan-Hao Chang.)*

Chun-Lin Chu and Ing-Chao Lin are with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan (e-mail: iclin@mail.ncku.edu.tw).

Yun-Chih Chen is with the Chair of Design Automation for Embedded Systems, Technical University of Dortmund, 44227 Dortmund, Germany (e-mail: yunchih.chen@tu-dortmund.de).

Wei Cheng is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: wei.cheng@duke.edu).

Yuan-Hao Chang is with the Institute of Information Science, Academia Sinica, Taipei 115, Taiwan (e-mail: johnson@iis.sinica.edu.tw).

Digital Object Identifier 10.1109/TCAD.2024.3447217

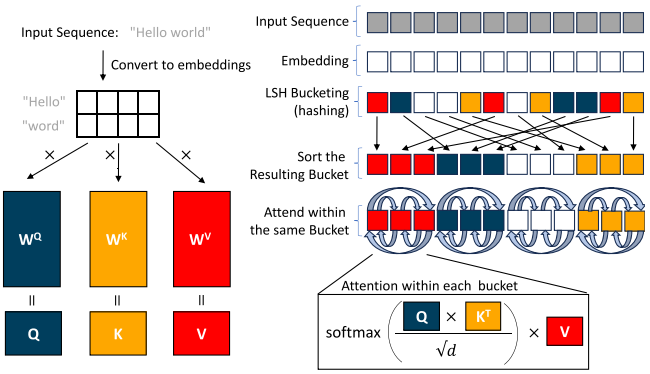

Fig. 1. Attention mechanism (left). The LSH attention mechanism (right).

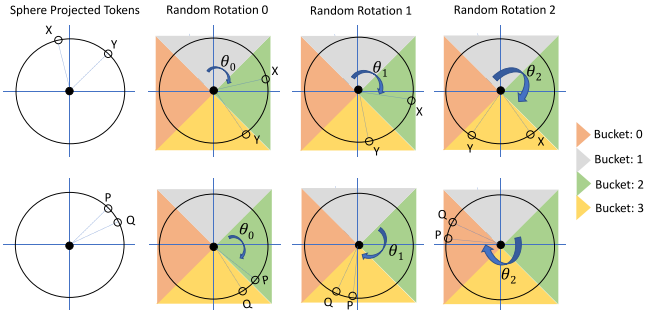

Fig. 2. Random rotation of hash functions.

only row or column access, RC-NVM allows efficient memory access to both rows and columns concurrently. This characteristic not only allows more flexible and efficient memory access patterns but also enables new computing paradigms for various application scenarios [13].

As mentioned earlier, LSH attention is the core component of the Reformer model [10] and is the main reason that the Reformer model performs so well in long sequence tasks. However, there are still computational challenges that need to be addressed.

Challenge 1: Despite considerably reducing the complexity of the attention mechanism, LSH attention is still subject to heavy penalties when dealing with long input sequences and high-dimensional embeddings. A significant amount of overhead is introduced in terms of data reading and writing, thus necessitating careful consideration and solution.

Challenge 2: LSH attention employs multiple rounds of hashing to mitigate the issue of similar tokens being hashed to different buckets. While this improves the overall model accuracy, it comes at the cost of increased timing and memory access overhead. Balancing the tradeoff between accuracy and computational efficiency is the primary consideration.

Challenge 3: LSH attention groups tokens with similar characteristics into distinct buckets and applies the attention mechanism within each bucket. Despite the efforts of grouping tokens into buckets, the computation of the attention mechanism within each bucket still requires a large amount of matrix multiplication and transpose operations.

In this work, we present AttentionRC, a novel approach that leverages the dual-addressing ability of RC-NVM to tackle the challenges posed by the LSH attention mechanism. Our contributions can be summarized as follows.

For Challenge 1: In AttentionRC, we introduce a new data mapping strategy that utilizes the subarray-level parallelism (SALP) [14] in RC-NVM. SALP enables parallel access to multiple subarrays within a memory bank, ideal for data with fixed access patterns. This parallelism natively supports the data parallelism in the LSH attention algorithm. Thus, we propose an LSH-friendly data mapping strategy that distributes high-dimensional embedding data across subarrays, accelerating LSH computations. This strategy boosts memory access efficiency by up to 60.9% for write operations and 4.9% for read operations, showing its effectiveness in various embedding-based computational scenarios beyond LSH attention.

For Challenge 2: As the number of hash rounds increases, more embeddings are read from memory during LSH attention. Leveraging RC-NVM's dual-addressing, we propose a *sort-free RC-aware bucket access strategy* and a *swap strategy* to eliminate sorting time in the LSH process, thereby improving bucket memory access efficiency. Our RC-aware bucket access strategy accelerates LSH, reduces sorting time, and achieves a 38% decrease in bucket access time. Additionally, this strategy is adaptable to other hash-based computations beyond LSH.

For Challenge 3: Leveraging RC-NVM's dual-addressing capability, we introduce *transpose-free attention* to eliminate transpose operations in LSH attention. Traditional memory systems require the matrix transpositions for proper alignment during matrix multiplications. In contrast, RC-NVM accesses both row and column addresses directly, removing this overhead. This dual-addressing also speeds up matrix multiplications, leading to a 51% improvement in LSH attention computation with AttentionRC, making it highly applicable to models using attention mechanisms.

II. Background and Motivation

A. Locality-Sensitive Hashing Attention

Reformer [10] adopted an LSH [15] attention that significantly reduces computational complexity by grouping input tokens into distinct buckets.

First, input sequences are converted into tokens (i.e., X and Y) and projected onto the embedding vector space, as shown in the upper half of Fig. 2. Second, projected embeddings are hashed independently using a random rotation hashing function. The random rotation hashing function is implemented by a random matrix (θ_0, θ_1, and θ_2, respectively, in Fig. 2) that preserves distances and angles between embeddings, making them suitable for the rotation operation. Then, the input embeddings are transformed by multiplying with these matrices. This process enhances the discriminative power of the hash functions by increasing the probability of mapping similar embeddings to the same buckets.

For instance, tokens X and Y are two relatively distant vectors, as shown in the upper half of Fig. 2, and are hashed into different buckets by using the random rotation matrices θ_0 and θ_1. Occasionally, they are hashed into the same bucket

by the random rotation matrix θ_2. After multiple rounds of hashing operations, embeddings that are hashed into the same bucket are believed to be highly correlated since they are adjacent to each other in the embedding vector space. Finally, embeddings that are hashed to the same buckets will be sorted in ascending order to facilitate the memory retrieval process when attention is performed among different buckets.

As shown in the left part of Fig. 1, attending all the query and key embedding pairs in the attention mechanism is computationally expensive; therefore, the LSH attention in the right part of Fig. 1 maps embeddings to corresponding hash buckets and subsequently performs attention among them. By doing so, the attention computation can be reduced by simply attending to the hashed buckets. LSH attention greatly simplifies the computation of the attention mechanism; however, it might result in worse model accuracy if strongly correlated embeddings are hashed into different buckets, thus reducing the quality of the similarity matrix among the hashed buckets. In order to retain the model accuracy in Reformer [10], multiple hash functions are presented such that two strongly correlated embeddings are more likely to be hashed into the same buckets. As shown in the lower half of Fig. 2, tokens P and Q are close in the projected vector space, ideally meaning they should be hashed into the same bucket. However, in the initial hash round, P goes to bucket 2 and Q to bucket 3. In the second and third rounds, both end up in the same bucket. The results from each round are combined by taking their union, so a token in different buckets across rounds will be in all those buckets when combined. This ensures tokens with similar characteristics, which may have been split in a single round, have more chances to be juxtaposed in later rounds. This mechanism highlights the importance of multiple hash rounds as they increase the probability of achieving the desired outcome of similar tokens being mapped to the same bucket.

Even though LSH adopts a highly efficient hashing mechanism, multiple hash functions require a larger number of hash tables stored in memory to keep track of hashed buckets. The tradeoff between the number of hash functions and memory usage is discussed in Reformer reproducibility [16] and is crucial to the LSH attention performance.

B. RC-NVM

A novel memory technology called RC-NVM [12] has been proposed in order to overcome the constraints of the traditional memory devices. RC-NVM uses NVM cells based on highly reliable TaOx ReRAM [17]. This ReRAM shows excellent durability, maintaining stable resistance over multiple switching cycles and data retention, with stability in both high- and low-resistance states for over 3000 h at 150 °C and a predicted retention period of more than ten years at 85 °C. Additionally, memory wear is minimal, with resistance remaining almost constant over prolonged periods at high temperatures. Fig. 3 shows the architecture of RC-NVM. It utilizes a crossbar architecture [18], [19] to organize its memory hierarchy into channels, ranks, chips, banks, subarrays, and mats. One of the key advantages of RC-NVM is its dual-addressing ability, which is enabled by the placement of extra

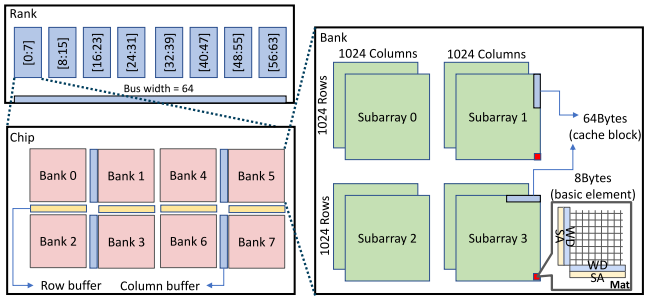

Fig. 3. RC-NVM architecture.

elements at different hardware levels. Specifically, the write driver (WD) and sense amplifier (SA) are placed on both sides of an RC-NVM Mat, allowing the memory cell to be driven or sensed from either row or column direction. This enables data to be accessed from both row-oriented and column-oriented address spaces. The use of RC-NVM offers more flexible and versatile data access capabilities and better power efficiency, compared to the traditional DRAM.

The RC-NVM supports SALP [14], which is a technique that allows for parallelization across different subarrays, enabling more efficient memory access and management. Specifically, SALP can improve the read and write performance of RC-NVM by enabling simultaneous activation of multiple subarrays, allowing for higher throughput and lower latency. Overall, the use of SALP in the RC-NVM [12] represents an important step toward improving the efficiency and scalability of NVM-based systems.

To improve the efficiency of data transfer between memory and processor, RC-NVM employs a cache mechanism with a block size of 64 bytes, which is equivalent to 8 Mats. When a cache miss occurs, the corresponding cache block will be transferred from memory to the corresponding buffer, either row buffer or column buffer, depending on the type of address space used to access the data. Subsequent accesses to the same cache block can then be serviced from the buffer, reducing the latency of accessing data from memory.

C. Motivation of LSH Attention With RC-NVM

The motivation to combine LSH attention and RC-NVM can be summarized as follows.

1) LSH Memory Overhead: The LSH design aims to address the growing time consumption associated with increasing sequence length, enabling attention-like models to handle longer sequences. However, as illustrated in Table I. The rise in embedding read-write time with longer sequences highlights the challenge of excessive access time. This observation underscores the need for an innovative data mapping strategy specifically tailored for LSH. While applying LSH directly on RC-NVM is a consideration, it proves insufficient to mitigate the challenges posed by larger sequence lengths. Consequently, an additional data mapping strategy is essential to leverage the hardware characteristics of RC-NVM effectively and enhance LSH, particularly in terms of access efficiency.

2) Additional Preprocessing Before Attention: Increasing hashing rounds boosts LSH attention performance but also

TABLE I
CYCLE TIME OF EMBEDDING ACCESS

Sequence Length	4096	8192	16384	32768	65536
Embedding Read Cycle Count (x1M)	5.51	11.02	22.05	44.09	88.18
Embedding Write Cycle Count (x1M)	13.43	26.87	53.73	107.47	214.93

TABLE II
CYCLE TIME OF SORTING IN DIFFERENT HASH ROUNDS

Hash Round	1	2	4	8	16
Cycle count (x1M)	3.11	6.18	12.34	23.18	45.1

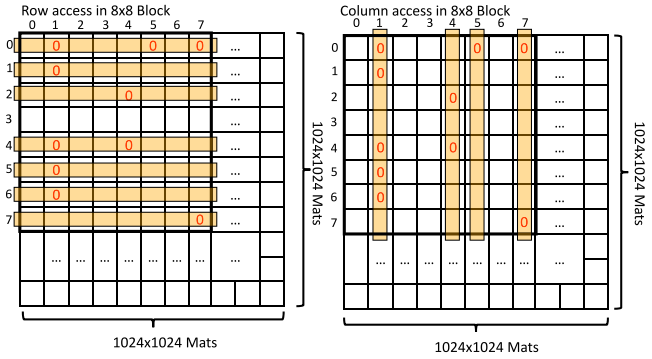

Fig. 4. Access pattern in RC-NVM.

raises memory and sorting costs. Unlike the original attention mechanism, sorting the buckets becomes necessary with LSH attention, and this sorting overhead increases with the number of hash rounds. With eight hash rounds, LSH attention approaches full attention [10], necessitating sorting the buckets 8×. Table II presents time consumption experiments at different hash rounds, demonstrating a linear relationship between the hash rounds and sorting time. In order to mitigate the computational burden associated with more hash rounds, we explore leveraging RC-NVM's dual-addressing capability to reduce operations.

3) Token Access in Bucket: The Reformer employs LSH to hash input sequences into buckets, enabling the attention mechanism to focus on the tokens within the same bucket. While RC-NVM enhances LSH operations, the potential challenge lies in the extended token access time, especially when scanning for the tokens in the same bucket. For instance, when the LSH operation reads all tokens labeled as bucket 0 for attention computation in Fig. 4, 64 tokens are stored in an RC-NVM subarray across 8 × 8 Mats, each storing a bucket ID. Retrieving tokens with a specific bucket ID requires a full memory scan. For instance, fetching tokens labeled as bucket 0 involves accessing rows 0–2 and 4–7, skipping only row 3. However, RC-NVM allows for retrieving these tokens with access to just columns 1, 4, 5, and 7. This flexibility reduces access time compared to the traditional DRAM. Simple operations like swapping may further optimize bucket access during memory retrieval. Thus, we aim to enhance LSH for bucket data retrieval in this context.

4) Attention Operation: After retrieving tokens hashed to the same bucket, we still need to perform time-consuming

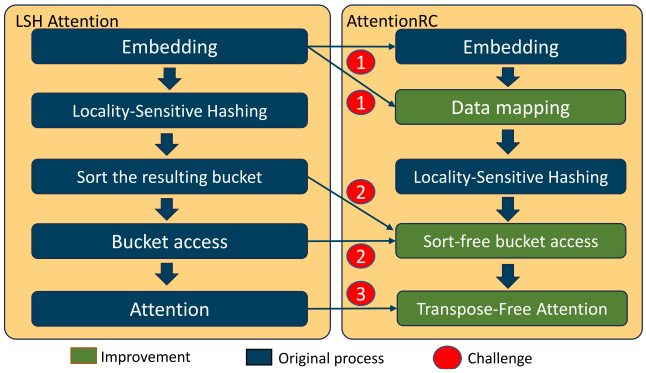

Fig. 5. AttentionRC overview.

attention operations, which include transpose operations and matrix multiplications. Both operations can be efficiently accelerated by utilizing RC-NVM's dual-addressing capability. Traditionally, attention involves time-consuming transpose operations on the key embeddings to get K^T and then computes "Q multiplied by K^T" as shown in Fig. 1. Although RC-NVM's dual-addressing capability inherently suits the matrix multiplications, we still find the need for transpose operations in handling key embeddings. We view this as an opportunity for further improvement in our approach.

III. ATTENTIONRC

In this section, we present AttentionRC, which leverages RC-NVM to address the challenges in LSH attention. Illustrated in Fig. 5, we propose 1) an LSH-friendly data mapping strategy to address Challenge 1 and propose 2) a sort-free RC-aware bucket access strategy to address Challenge 2. For Challenge 3, we present 3) a transpose-free strategy to accelerate attention computation. Throughout this section, we will further discuss the technical details and benefits of our approach.

A. LSH-Friendly Data Mapping in AttentionRC

To efficiently access large embeddings in memory, we propose an LSH-friendly data mapping strategy (for Challenge 1) that consists of two steps. First, each token is stored in an RC-NVM Mat, which is (1/8) size of a cache block. Since the cache block size equals 8 Mats, we use an 8 × 8 block as the minimum unit for token storage, as shown in Fig. 4. As the input sequence length increases, the number of Mats allocated within a subarray also increases. For example, if the input sequence length is 4096, we expand it to several 64 × 64 Mats.

The second step is the mapping of dimensions across subarrays. Each subarray stores the data for one dimension of the tokens. In RC-NVM, there are eight banks in a rank, and each bank contains eight subarrays. This means the maximum size of subarrays in an RC-NVM rank is 64, so we use 64 as the baseline.

For instance, if the dimension exceeds 64, e.g., 128, we split it into two blocks of 64 each. We further describe our mapping strategy in Fig. 6. Specifically, we take the 4096 × 64

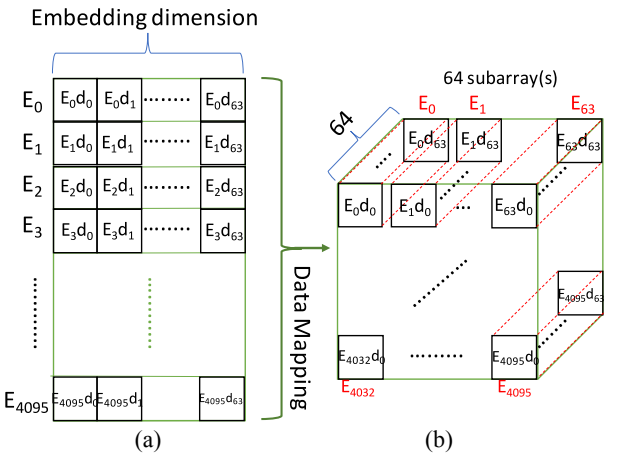

Fig. 6. (a) Embedding. (b) Data mapping in AttentionRC.

embeddings (i.e., 4096 embeddings, each with 64 dimensions) as an example. We store the 4096 tokens of the input sequence in RC-NVM subarrays as a 64 × 64 block and then distribute the 64-D data to 64 subarrays. In other words, we store the 4096 × 64 embeddings using 64 subarrays, where each subarray uses a 64 × 64 block to store the data.

We observed that SALP [14] is ideal for applications with uniform data access patterns across subarrays. In LSH Attention, each input token is mapped to the same dimensional space, requiring all dimensions to be read from memory for each token. This setup leverages the SALP's parallelism by reading 64 dimensions per access, thereby reducing the data retrieval and writing time.

Our LSH-friendly data mapping strategy offers several benefits. Increasing hash rounds to improve accuracy typically raises memory costs, as the LSH operation reads tokens from memory based on the hash results, accessing only certain dimensions. Our strategy, which enables the parallel data access, is expected to lower these costs even with multiple hash rounds.

Moreover, LSH operation usually sorts buckets to store tokens contiguously for easy retrieval. However, with dual-addressing memory, sorting is unnecessary, saving pre-processing time before the attention operation. Additionally, we can efficiently retrieve tokens from the same bucket simultaneously, utilizing RC-NVM's characteristics.

B. Sort-Free Bucket Access in AttentionRC

As for Challenge 2, AttentionRC utilizes RC-NVM to reduce sorting time in LSH attention while enhancing the bucket access efficiency. However, this approach has some challenges. Without sorting the resulting buckets, the memory does not know the exact positions of tokens within the same bucket. For example, with a sequence length of 4096 and tokens hashed into 64 buckets, our LSH-friendly data mapping strategy stores the sequence in 64 × 64 blocks. We need to maintain a 64-entry list in memory to track the row and column of tokens in these blocks. This list allows us to access the tokens within the same bucket according to their positions.

Algorithm 1 RC-Aware Bucket Access

Input: bucket_id
Input: pos = $\{(row_i, col_i), \ldots, (row_j, col_j)\}$ ← tokens' positions in bucket_id
Output: bucket_access

```
1:  procedure BUCKET_ACCESS(bucket_id, positions)
2:      Initialize accessed_tokens ← not accessed
3:      Initialize bucket_access
4:      for pos in positions do
5:          if pos is not accessed then
6:              num_row ← pos.row_access()
7:              row_index ← accessed tokens indices
8:              num_col ← pos.column_access()
9:              col_index ← accessed tokens indices
10:             if num_row ≥ num_col then
11:                 Append bucket_access ← "RR position"
12:                 accessed_tokens[row_index] ← accessed
13:             else
14:                 Append bucket_access ← "CR position"
15:                 accessed_tokens[col_index] ← accessed
16:             end if
17:         end if
18:     end for
19: end procedure
```

However, the tokens within the same bucket may still be scattered, making access time consuming due to discontinuity. This is where the dual-addressing feature of RC-NVM comes into play.

We present a sort-free RC-aware strategy for the bucket access that utilizes the dual-addressing ability of RC-NVM to reduce the memory access time to tokens that are hashed into the same bucket. This strategy assumes that our proposed mapping strategy presented in the previous section is adopted.

As shown in Algorithm 1, the RC-aware bucket access strategy takes two inputs: 1) the *bucket_id*, which determines the specific bucket to be processed and 2) the *positions* of all tokens within that bucket. In lines 2 and 3, this algorithm begins with two initial lists: 1) "accessed_tokens" and 2) "bucket_access." The former is used to store whether a token in the bucket has been accessed or not, while the latter is used to store the access way to access the bucket. In lines 4 and 5, the algorithm then iterates through the token positions in the bucket that have not been accessed yet. From lines 6 to 9, the algorithm calculates the number of tokens belonging to *bucket_id* that can be accessed in terms of row and column, while simultaneously recording the indices of the accessed tokens in *row_index* and *col_index* within the position list. From lines 10 to 15, the algorithm determines which access pattern (row or column) is more efficient based on which pattern accesses more tokens and then adds the access way and the position to the bucket_access list. Finally, the algorithm updates the accessed_tokens list for the accessed tokens to prevent them from being accessed again in future iterations.

The concept of a Sort-free RC-aware bucket access strategy is further illustrated in Fig. 7, which depicts an 8 × 8 matrix

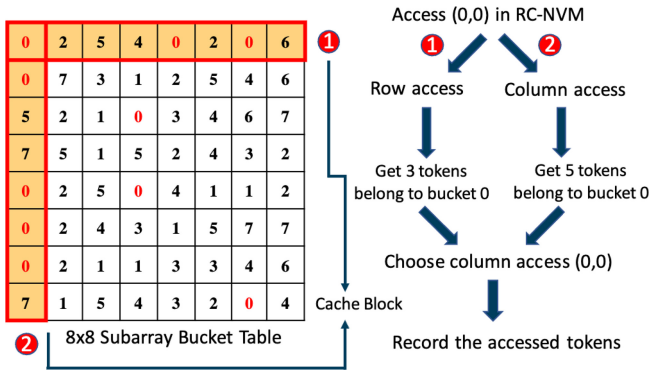

Fig. 7. RC-aware bucket access.

(i.e., eight rows and eight columns in total) representing a total of 64 tokens. Each token is hashed into a specific bucket; for instance, when we want to access all the tokens in bucket 0, there are ten tokens we need to access. According to our Algorithm 1, (0, 0), which means the position of the token is in the zeroth row and the zeroth column, will be the first position in the procedure. As we store the data in RC-NVM, we have two patterns (i.e., row and column accesses) to access the data. Suppose that we use row read (0, 0), which will fetch eight values (i.e., the 64 bytes of data from the zeroth column to the seventh column into the cache).

Conversely, column read (0, 0) fetches data from the zeroth row to the seventh row. Obviously, row read (0, 0) only accesses three tokens that belong to bucket 0, whereas column read accesses five tokens. This example illustrates the main idea of our sort-free RC-aware data access strategy, which is, to find the most efficient memory access pattern (row or column direction) based on the buckets being selected.

C. Further Improvement of Bucket Access in AttentionRC

While we have successfully accelerated bucket access by leveraging the advantages of dual addressing, as depicted in Fig. 7, certain challenges persist. Specifically, when opting for column access at the position (0, 0), we observe the retrieval of eight tokens into the cache block. However, upon closer examination, only five of these tokens belong to our target bucket (bucket 0), leaving the remaining three tokens unwanted. This phenomenon prompts us to consider further optimization opportunities.

We introduce a swap strategy aimed at further optimizing the performance of bucket access by exchanging a small subset of tokens.

As shown in Algorithm 2, the process begins with an 8×8 block of the RCNVM mat. The algorithm takes two inputs: bucket_access, determined by Algorithm 1, representing the most efficient memory access pattern, and the positions of all tokens within that bucket. In line 2, a list called swap_list is initialized to store tokens in this 8×8 block that need swapping. Line 3 filters out singular tokens within the same bucket, as they cannot benefit from swapping. From lines 4 to 5 compute the access ratio for each bucket in the block and identify the least efficient one. The bucket with the minimum ratio, calculated by dividing the total number of tokens by the total accesses required, indicates the need for improvement. In line 6, the algorithm determines two variables based on the least efficient bucket. First, it identifies the swap_token, which represents the token to be swapped within this bucket, and second, it determines the target_token, another token within the same bucket chosen as the target for swapping. This implies that the token designated for swapping will be moved to the access block associated with this target_token. In line 7, the algorithm determines the tokens present in the access block of the target_token. Given the row- and column-oriented access pattern in the RCNVM, the algorithm collects the bucket_id of tokens encountered target_token, stores them in the rc_swap_temp, and treats them as candidates for swapping. From lines 8 to 12, the algorithm validates each potential swap. In line 9, valid swaps are added to swap_candidate, while invalid ones are filtered out. The conditions and reasons for a swap being considered invalid are then outlined.

Invalid Swap Condition 1: This candidate is already in `swap_list`. Clearly, if this candidate is already in the `swap_list`, it indicates that it is already scheduled for swapping, and swapping it again provides no performance benefit.

Invalid Swap Condition 2: This candidate is not present in `bucket_access`. If this candidate does not appear in `bucket_access`, it indicates that it does not need additional access, as it has already been captured by another access. Swapping it may introduce unnecessary access, resulting in a negative impact on performance.

Algorithm 2 Swap Strategy

Input: 8x8 block mat of bucket_access
Input: pos $= \{(row_i, col_i), \ldots, (row_j, col_j)\} \leftarrow$ bucket's positions
Output: new_bucket_access

```
1: procedure SWAP(block, pos)
2:     Initialize swap_list ← tokens to be swapped
3:     Filter buckets which only have one token
4:     Calculate the ratio of every bucket access in the block
5:     Find the smallest ratio
6:     swap_token, target_token ← smallest ratio bucket
7:     rc_swap_temp ← bucket_id
8:     for bucket_id in rc_swap_temp do
9:         if valid_swap then
10:            Append swap_candidate ← pos of bucket_id
11:        end if
12:    end for
13:    if There are multiple candidates in swap_candidate then
14:        Decide which swap is the best and remove others
15:        Append swap_list ← swap_candidate
16:    else
17:        Append swap_list ← swap_candidate
18:    end if
19: end procedure
20: Perform the swap operation
21: new_bucket_access ← Algorithm 1
```

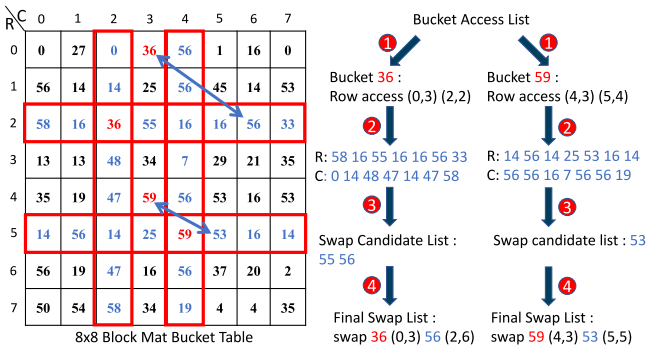

Fig. 8. Process of swap strategy.

Invalid Swap Condition 3: The candidate is present in `bucket_access`, but the corresponding access has captured two or more tokens from the same bucket. If this candidate exists in bucket_access, but the corresponding access includes two or more tokens from the same bucket, swapping is not allowed. This is because swapping in this scenario could lead not only to Invalid Swap Condition 2 but also potentially impact subsequent tokens from the same bucket accessed later, resulting in a degradation of access performance for that bucket.

After the algorithm determines the swap_candidate. In lines 13 to 18, if there are two or more candidates in swap_candidate, a final arbitration is conducted to decide which candidate to swap based on the associated benefits. In line 20, the algorithm decides which tokens to swap, and the actual exchange operation is performed. In line 21, after the exchange, Algorithm 1 is executed once again to obtain the new and further improved new_bucket_access, completing the entire algorithm.

The concept of a swap strategy is further illustrated in Fig. 8, which depicts an 8 × 8 block mat, same as in Fig. 7, representing a total of 64 tokens. When seeking further improvement through swapping, the initialization involves running Algorithm 1 to obtain the bucket access list for the given block. Subsequently, the first step in Fig. 8 corresponds to lines 4 to 6 of Algorithm 2, the algorithm calculates access efficiency to identify buckets that require further improvement. In the example, buckets 36 and 59 are identified. The second step corresponds to line 7 of Algorithm 2, where the algorithm identifies tokens in both the row and column directions that can be swapped. The third step corresponds to lines 8 to 12 of Algorithm 2, where the algorithm filters out tokens that cannot be swapped based on the invalid swap conditions, resulting in the swap candidate list. Finally, the fourth step corresponds to lines 13 to 18 of Algorithm 2. For bucket 36 in the example, the swap candidate list contains tokens that belong to buckets 55 and 56. However, considering that swapping the latter might also improve the access efficiency of bucket 56, the decision is made to swap the token that belongs to 56. In contrast, for bucket 59 in the example, as there is only one candidate in the swap candidate list, no further decision making is necessary.

D. Transpose-Free Attention

For Challenge 3, AttentionRC aims to completely remove transpose operations and accelerate the matrix multiplication.

Algorithm 3 Attention Operation in AttentionRC

Input: bucket_id
Input: Q, K, V $\in \mathbb{R}^{seq\_length \times d}$ ← tokens in bucket_id
Input: d ▷ Embedding dimension
Output: A ▷ Attention matrix
1: **procedure** BUCKET_ATTENTION(bucket_id, Q, K, V, d)
2: QK ← Q × K ▷ Directly perform dot-product
3: S_QK ← QK × $\frac{1}{\sqrt{d}}$
4: softmax_QK ← S_OK.softmax()
5: A ← softmax_QK × V ▷ Column access V matrix
6: **end procedure**

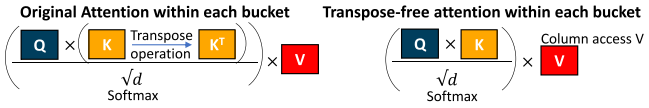

Fig. 9. Transpose-free attention.

After the bucket access pattern is identified, AttentionRC performs attention among tokens hashed into the same bucket. One operation in attention is the matrix multiplication, and we believe that RC-NVM can be beneficial in this process. Aside from the previously mentioned advantages, RC-NVM is particularly suited for attention because it often involves computing the product of a query matrix Q and a transposed key matrix K to obtain attention scores. Both the matrix multiplication and transpose operations require column direction memory access.

The transpose operation can be a bottleneck for either row-major or column-major memory layouts, as it requires extra memory access to swap the rows and columns. However, with the dual-addressing feature of RC-NVM, we can avoid the transpose operation and directly multiply Q and K using any memory layout we prefer. Therefore, RC-NVM can not only accelerate matrix multiplication but also reduce the memory accesses required for the matrix transpose in attention, resulting in more efficient and effective attention in AttentionRC than in LSH attention.

As shown in Algorithm 3, the input of this algorithm consists of three parts. First, we have *bucket_id*, which indicates the specific bucket where the attention operation is being performed. Second, we have the matrices query Q, key K, and value V. The size of these matrices is determined by the number of tokens included in the bucket multiplied by the dimension of each token. Finally, we have the dimension of embedding in *bucket_id*. In line 2, the algorithm directly performs dot-product between Q and K without the need for a transpose operation on K due to the dual-addressing feature of RC-NVM. In lines 3 and 4, the algorithm performs the scaled and softmax operation in the same way as the original attention. In line 5, the algorithm performs the final step of the attention operation, which involves multiplying the softmax of QK with V. With the column access capability of RC-NVM, we can utilize the column access in the V matrix, resulting in reduced matrix multiplication time, compared to the traditional memory. Finally, the algorithm returns the attention matrix A.

As shown in Fig. 9, the upper part is the original attention and the lower part is the transpose-free attention. We

can observe two key distinctions in the modified attention mechanism compared to the original one. The initial attention mechanism necessitates a transpose operation for the key matrix, as the query and key matrices share the same dimensions. In contrast, by leveraging the advantages of RC-NVM, we can circumvent the requirement for the transpose operation and directly perform the matrix multiplication. Furthermore, the inherent suitability of RC-NVM for the matrix multiplication amplifies this enhancement, enabling us to exploit the strengths of RC-NVM's optimized characteristics.

Moreover, our strategy enables us to optimize the storage and retrieval of the value matrix through the column access methodology, thereby contributing to further acceleration. Consequently, our approach not only streamlines the traditional attention mechanism but also leverages the distinctive attributes of RC-NVM to enhance the overall efficiency of the process. This innovative fusion of the attention mechanism and RC-NVM optimization holds the potential to significantly amplify the performance of the entire model. This is particularly valuable in scenarios involving large-scale computations and memory-intensive operations.

E. AttentionRC Discussion

In this section, we discuss the overhead associated with AttentionRC. While RC-NVM can enhance LSH attention performance and streamline its steps, some overhead may arise. By removing sorting steps, we propose an RC-aware bucket access strategy and a swap strategy to minimize the bucket access time, which requires additional time and memory for executing Algorithms 1 and 2, unlike LSH attention. We conducted experiments to determine if this additional time is tolerable. The results will be presented in the next section.

IV. Experimental Evaluation

In this section, we first discuss the experimental setup, including the experimental procedures and datasets used in our experiments. Then, we will evaluate the experimental results.

A. Experimental Setup

1) Experimental Procedure: Our experimental procedure consists of three parts: 1) LSH attention; 2) memory trace generation; and 3) experimental result evaluation. First, we apply the LSH attention algorithm from Reformer, which is implemented by PyTorch [20] in our experiment. We use eight hash rounds. This choice is based on findings from the Reformer model, which indicate that the accuracy becomes perfect when evaluated with eight hash rounds. Next, we use a custom simulator written in Python to generate memory trace files for the entire execution process. This simulator is capable of simulating the LSH attention computation with or without the support of dual-addressing ability from RC-NVM. Table III demonstrates the row-/column-oriented address mappings in RC-NVM. The only difference between these two access patterns is the order of row and column bits in the 32-bit address. Our custom simulator breaks down LSH attention computation into three stages: 1) embedding access; 2) LSH

TABLE III
Address Mappings for Row-Oriented and Column-Oriented Accesses

	Rank	Subarray	Bank	Channel	Row/Column	Column/Row	Intra Bus
Row-oriented	2-bit	3-bit	3-bit	1-bit	10-bit	10-bit	3-bit
Column-oriented							

TABLE IV
Configuration of Simulated Systems

Processor	4 cores, x86, 2.0 GHz
L1 cache	private, 64B cache line, 8-way associative, 32 KB
L2 cache	private, 64B cache line, 8-way associative, 256 KB
L3 cache	shared, 64B cache line, 8-way associative, 8 MB
Memory controller	32 entry request queues per controller, FR-FCFS
DRAM	DDR3-1333, tCAS: 10, tRCD: 9, tRP: 9, tRAS: 24, Channels: 2, Ranks: 2, Banks: 8, Rows: 65536, Columns: 256, Row buffer size: 2048 B, Capacity: 4 GB, Access time: 14 ns
RC-NVM	LPDDR3-800, tCAS: 6, tRCD: 12, tRP: 1, tRAS: 0, Channels: 2, Ranks 4, Banks: 8, Rows: 8192, Columns: 1024, Row buffer size: 8192 B, Column buffer size: 8192 B, Capacity: 4 GB, Read access time: 29 ns, Write pulse width: 15 ns, four 512 * 512 mats in a subarray

execution; and 3) attention computation. For each stage, if there is a memory access, a memory trace is generated per the address mappings in Table III. In AttentionRC, we consider four types of memory access due to dual addressing: 1) row read; 2) row write; 3) column read; and 4) column write. In contrast, the original LSH attention model includes only row read and row write. These traces are recorded in a trace file, sequentially documenting all the memory traces produced during the simulation. Finally, we evaluate the performance by feeding the trace files to the RC-NVM simulator to obtain the overall execution time.

2) Datasets: We conduct our experiments on five datasets: 1) enwiki8 [21]; 2) BookCorpus [22]; 3) Internet movie database (IMDB) [23]; 4) GutenBerg [24]; and 5) OpenWebText [25]. enwiki8 [21] dataset consists of the first 100 million bytes of text from the english wikipedia. BookCorpus [22] is a large dataset consisting of over 11 000 books from a wide range of genres and topics. IMDB [23] is a collection of movie reviews from the IMDB, containing labeled sentiment (positive/negative) for each review. GutenBerg contains a large collection of books, including classic literature, historical texts, and other public domain works. OpenWebText [25] is a dataset created by scraping and preprocessing publicly available Web text, resulting in a large corpus of diverse Web content. To evaluate the performance of AttentionRC, we feed these five datasets into LSH attention with varying sequence lengths and dimensions as input. By varying these parameters, we aim to investigate the effectiveness and scalability of our proposed strategy on handling different input sizes.

3) System Configuration: The system configuration is listed in Table IV. In the simulated DRAM, we have two channels, four ranks per channel, and eight banks per rank. As for the simulated RC-NVM system, RC-NVM has two channels, four ranks per channel, eight banks per rank, and eight subarrays per bank. Each subarray comprises 1024 rows and 1024 columns, where RC-NVM supports both row-oriented and column-oriented memory accesses. The total capacity of the memory system is 4 GB and the well-known FR-FCFS [26] is used as a basic scheduling policy.

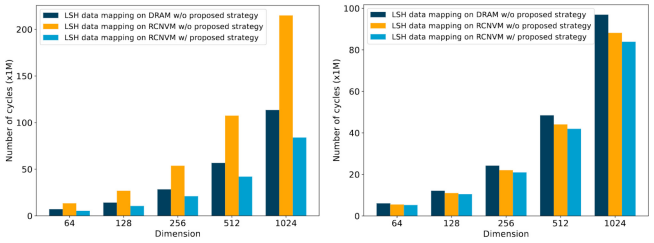

Fig. 10. Cycle number comparison of accessing Embeddings w/ and w/o LSH-friendly data mapping strategy. (a) Embedding write. (b) Embedding read.

4) RC-NVM Latency Overhead: Extra peripheral circuitry needed in RC-NVM also induces latency overhead mainly from wire routing. Since more multiplexing transistors are added to the critical path, the read and write latency increases. The latency overhead of multiplexers, however, is trivial because the majority of access latency comes from the cell access and wiring delay. To quantify the latency overhead, RC-NVM [12] runs SPICE simulations, and the latency overhead for RC-NVM is moderate. When the number of WL/BL is 512 and 1024, the timing overhead is just about 15% and 10%, respectively.

B. Experimental Results

In this section, we use the five investigated datasets and apply LSH attention with varying sequence lengths and dimensions to evaluate performance improvement. The section is divided into four parts.

1) We discuss the improvement offered by the proposed LSH-friendly data mapping strategy in RC-NVM.
2) We analyze the time spent on preprocessing before attention computation and the overhead introduced by our strategy.
3) We evaluate the impact of the proposed RC-aware bucket access strategy.
4) We will analyze the amount of memory access time in attention operation that can be saved by the proposed AttentionRC, compared to LSH attention.

1) LSH-Friendly Data Mapping Strategy: Fig. 10 compares the number of cycles of data accesses on DRAM and on RC-NVM with/without our proposed LSH-friendly data mapping strategy by using a sequence length of 4096 with input embeddings of 64, 128, 256, 512, and 1024 dimensions. Specifically, Fig. 10 compares the read and write performance of memory accesses between our proposed strategy and a conventional strategy that does not consider the SALP properties.

Our proposed strategy disperses the token embeddings into different subarrays based on their dimensions, while the conventional strategy stores all the token embeddings in the same subarray. Our results show that as the dimension increases, the performance improvement in our proposed mapping strategy becomes more significant, with a maximum write improvement of up to 60.9% and a read improvement of 4.9%. This is due to the increased parallelism in SALP as the dimension increases, resulting in fewer cycles needed.

TABLE V
BANK ENERGY CONSUMPTION COMPARISON OF ACCESSING EMBEDDINGS W/ AND W/O LSH-FRIENDLY DATA MAPPING STRATEGY

Configuration		Energy Consumption (nJ)	
Sequence Length	Dimension	w/o	w/
4096	64	2.1368×10^7	2.136816×10^7
4096	128	4.27362×10^7	4.273608×10^7
4096	256	8.54724×10^7	8.54619×10^7
4096	512	17.0945×10^7	17.09431×10^7
4096	1024	34.189×10^7	34.18854×10^7

These findings indicate that our proposed LSH-friendly data mapping strategy is well-suited for NLP tasks, where higher dimensions represent more features per token, leading to better model accuracy.

The improvement in write operations is more significant than in read operations due to the characteristics of NVM [27]. Writing to NVM cells involves applying a voltage or current to modify the cell state, incurring overheads like programming time, making write operations slower. In contrast, read operations are faster and simpler, involving only sensing the charge or resistance levels without modifying the cell states. This discrepancy in performance improvements when using SALP in our strategy stems from the slower nature of write operations compared to the relatively faster read operations.

As shown in Table V, our LSH-friendly data mapping strategy maps embedding data to subarrays across various banks. In our experiments with a sequence length of 4096 and dimensions of 64, 128, 256, 512, and 1024, we observe a subtle improvement in energy consumption as the dimension increases. The energy consumption is primarily determined by execution time and memory accesses. Our strategy leverages SALP, which parallelizes access to multiple subarrays, reducing memory access and execution time. Higher dimensions increase parallelism, further lowering energy consumption. This experiment shows our approach reduces data access time and enhances energy efficiency.

2) Sort-Free RC-Aware Bucket Access: Fig. 11 compares the cycle numbers with and without sort-free RC-aware bucket access under different hash rounds. Five datasets with a sequence length of 4096 and 64 dimensions are used. The number of hash rounds varies between 1, 2, 4, 8, and 16. It can be seen that the cycle number reduction ratio ranges between 84.3% and 93.7%. Even with an increased number of hash rounds, the proposed sort-free RC-aware bucket access still has a high reduction ratio. In our pursuit to eliminate sorting time entirely, the experiments have revealed that, despite the removal of sorting steps, there remains a need for additional time within AttentionRC. This requirement arises from the adoption of the proposed strategy, which seeks to reduce access time for each bucket. Consequently, although certain algorithms deemed unnecessary in LSH attention are omitted, they become indispensable in the context of AttentionRC.

Fig. 12 compares the cycle number in the five datasets with and without the sort-free RC-aware bucket access under different sequence lengths. Specifically, it compares the time required to extract all tokens from all buckets using sequence lengths of 4096, 8192, 16 384, 32 768, and 65 536 with 64 dimensions when tokens are hashed to 64 buckets. We choose

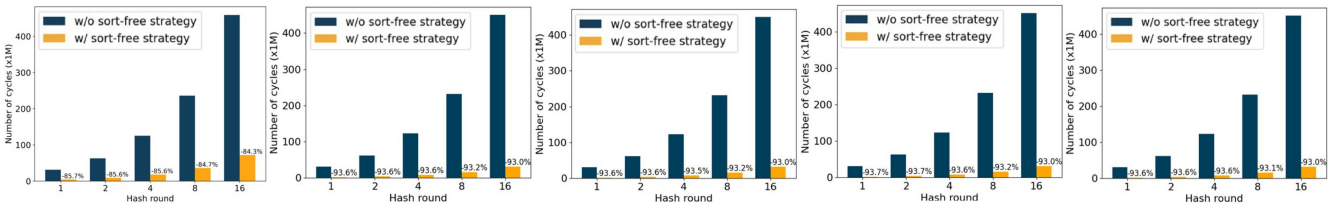

Fig. 11. Cycle Comparison w/ and w/o Sort-free RC-aware Bucket access strategy under different hash rounds. (a) Enwiki8. (b) BookCorpus. (c) IMDB. (d) GutenBerg. (e) OpenWebText.

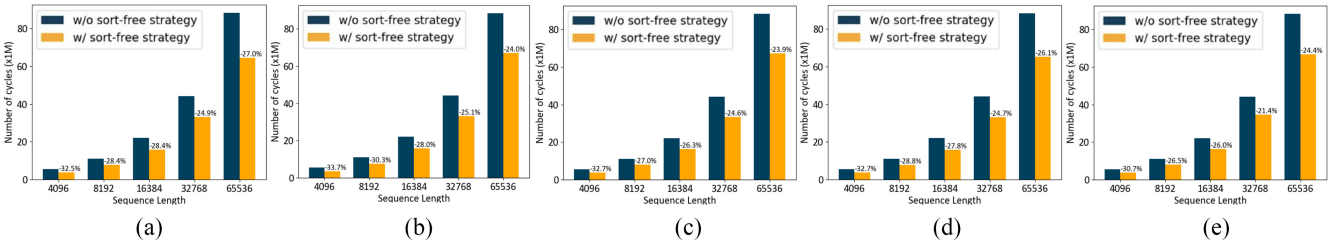

Fig. 12. Cycle Comparison w/ and w/o Sort-free RC-aware bucket access strategy under different sequence lengths. (a) Enwiki8. (b) BookCorpus. (c) IMDB. (d) GutenBerg. (e) OpenWebText.

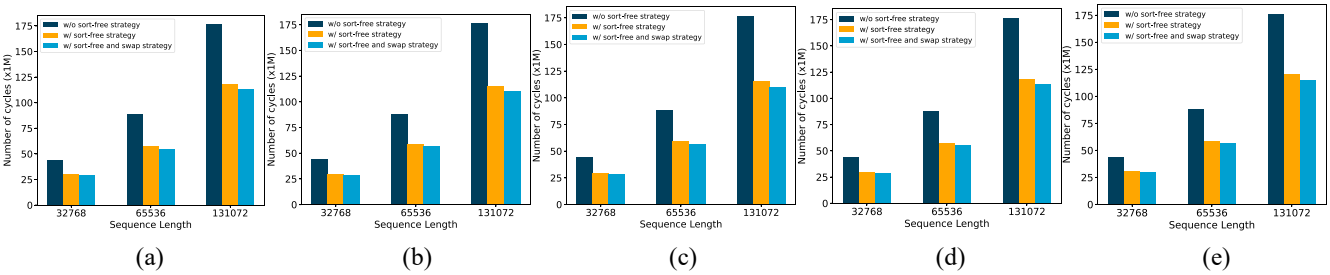

Fig. 13. Cycle Comparison w/ and w/o swap strategy under different sequence lengths. (a) Enwiki8. (b) BookCorpus. (c) IMDB. (d) GutenBerg. (e) OpenWebText.

to alter the sequence length rather than the dimension because LSH attention was originally proposed to handle larger sequence lengths with nonquadratic complexity. Therefore, it is more meaningful to perform comparisons with different sequence lengths. It can be seen that our sort-free strategy achieves significant cycle number reductions up to around 33% for different sequence lengths.

3) Further Improvement of Bucket Access: Fig. 13 conducts with sequence lengths of 32 768, 65 536, and 131 072. In the context of setting the sequence length, an additional sequence length 131 072 was introduced. This decision was motivated by the observed trend in modern language models, where the sequence length is progressively increasing.

As the sequence length increases, the reduction in access count becomes more pronounced. This phenomenon can be attributed to the approach taken in the swap strategy, where the bucket table is divided into individual blocks of 8×8 matrices for improvement through token swapping. Consequently, as the sequence length increases, resulting in more blocks being separated and more token exchanges, the magnitude of improvement also increases accordingly.

Fig. 13 not only compares the results with the previous outcomes but also includes the improvements achieved through our applied swap strategy. The effectiveness of our swap strategy becomes more pronounced as the sequence length grows. Specifically, for the sequence length of 131 072, the application of the swap strategy results in an approximate enhancement of 5%. The reduction in cycle numbers exhibits a more significant impact compared to the scenario with a sequence length of 65 536. This observation indicates the suitability of our approach for contemporary language models with larger sequence lengths.

4) Overhead of Further Improvement: Fig. 14 illustrates the additional time required for improving the bucket access with the swap strategy at different sequence lengths. The x-axis represents the five datasets, while the y-axis denotes the cycle numbers expended. The observed increase in overhead after applying the swap strategy is due to the extra execution of Algorithm 2. However, the notable point is that even with this additional time, the overhead remains significantly smaller than the original sorting time. Fig. 14 clearly demonstrates that our approach leads to a considerable enhancement in bucket access efficiency, incurring only a modest increase in required time. Importantly, this overhead is far less than the time consumed by the sorting operation, reinforcing the effectiveness of our strategy.

5) Transpose-Free Attention: Table VI compares the number of cycles required in the five datasets with and without

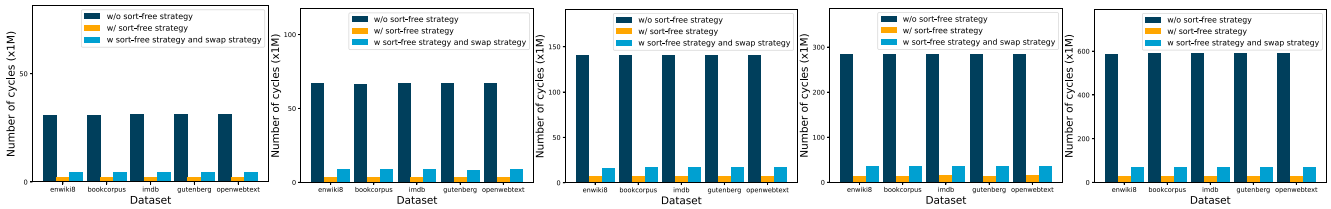

Fig. 14. Cycle Comparison of Overhead w/ and w/o swap strategy under different sequence lengths. (a) 4096. (b) 8192. (c) 16 384. (d) 32 768. (e) 65 536.

TABLE VI
CYCLE COMPARISON BETWEEN TRANSPOSE-FREE ATTENTION AND THE ORIGINAL ATTENTION

Datasets	Sequence length	Original Attention			Transpose-free attention			Reduction(%)
		Transpose Operation(x1M)	Matrix Multiplication(x1M)	Total(x1M)	Transpose Operation(x1M)	Matrix Multiplication(x1M)	Total(x1M)	
Enwiki8	4096	3.15	13.81	16.96	0	10.66	10.66	37
	16384	19	61.91	80.91	0	42.92	42.92	47
	65536	59.36	231.37	290.73	0	162.96	162.96	44
BookCorpus	4096	3.5	14.19	17.69	0	10.11	10.11	43
	16384	18.82	61.73	80.55	0	40.7	40.7	49
	65536	58.97	230.97	289.94	0	162.96	162.96	44
IMDB	4096	3.25	13.9	17.15	0	10.11	10.11	41
	16384	19.11	62.02	81.13	0	40.7	40.7	50
	65536	62.78	234.79	297.57	0	162.96	162.96	45
GutenBerg	4096	3.32	13.96	17.28	0	10.11	10.11	41
	16384	18.86	61.78	80.64	0	40.7	40.7	50
	65536	59.03	231.03	290.06	0	162.94	162.94	44
OpenWebText	4096	3.32	13.99	17.31	0	10.11	10.11	42
	16384	18.21	61.15	79.36	0	40.7	40.7	49
	65536	60.72	232.73	293.45	0	162.96	162.96	44

TABLE VII
TOTAL CYCLE COMPARISON

Datasets	Total cycle of original LSH attention	Total cycle of AttentionRC	Reduction(%)
Enwiki8	699999430	345013125	50.71
BookCorpus	699990375	344859655	50.73
IMDB	700063900	344916080	50.73
GutenBerg	699937400	344922120	50.72
OpenWebText	700033145	345038970	50.71

transpose-free attention for performing attention on 64 buckets with sequence lengths of 4096, 8192, 16 384, 32 768, and 65 536.

The results indicate that we can achieve up to a 51% reduction in attention time with varying sequence lengths. This is primarily due to two key factors. First, as outlined in Section III-A, the dual-addressing capability of RC-NVM eliminates the need for transpose operations during the attention process. In contrast, traditional memory would require numerous transpose operations proportional to the number of buckets, especially as sequence length—and consequently, the number of tokens per bucket—increases. Second, this dual-addressing ability also enhances the matrix multiplication efficiency, particularly for operations involving extensive column access, further improving performance. These factors enable AttentionRC to outperform the original LSH attention by reducing the time needed for attention operations.

6) AttentionRC on Transformer-Based Models: As shown in Table VII, we compute cycles between the original LSH attention and AttentionRC within the five datasets, which involves a sequence length of 4096, 64 dimensions, and eight hash rounds. Reformer [10] was introduced to reduce the computational burden of the Transformer using LSH attention. According to Reformer, with eight hash rounds, it achieves about 40% less computation time for attention compared to the standard Transformer. Our experiments show that AttentionRC further reduces computation time by 50% compared to LSH attention. As attention mechanisms account for much of the computation time in Transformer models during both training and inference. By optimizing the attention mechanism, AttentionRC significantly enhances Transformer performance and is applicable to other Transformer-based models. This improvement underscores the potential of AttentionRC to set a new standard for efficient and scalable Transformer architectures.

V. CONCLUSION

We present a novel approach called AttentionRC to improve various aspects of LSH attention, including an LSH-friendly data mapping strategy, bucket access, and matrix multiplication. First, our LSH-friendly data mapping strategy leverages SALP for parallel data read/write, significantly reducing memory operation time. This strategy extends beyond LSH attention and is applicable to other models involving embeddings and optimizing memory access across various scenarios. Second, we propose an RC-aware bucket access strategy combined with a swap strategy that uses the dual-addressing feature of RC-NVM for faster token access in the same bucket than conventional memory. This strategy benefits situations where the data is preprocessed through hashing or bucketing, showcasing its versatility beyond LSH-based computations. Third, we propose transpose-free attention to eliminate transpose operations in attention computation, achieving substantial efficiency improvement in the matrix multiplication. This method is well-suited for integration into a broad range of models involving attention computations In conclusion, AttentionRC shows promising results in enhancing LSH attention performance and demonstrates the potential advantages of utilizing RC-NVM across various models. However, our approach is currently evaluated in a simulated environment, as RC-NVM technology is still in the simulation phase. Real-world applications may present challenges and costs. Future research will explore the feasibility and complexity of deploying our approach in real systems, addressing issues, such as manufacturing costs, hardware design complexity, temperature stability, and durability. This aims to facilitate the practical adoption of our approach.

REFERENCES

[1] A. Vaswani et al., "Attention is all you need," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, 2017, pp. 1–11.
[2] A. Galassi, M. Lippi, and P. Torroni, "Attention in natural language processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 10, pp. 4291–4308, Oct. 2021.

[3] K. Han et al., "A survey on vision transformer," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 1, pp. 87–110, Jan. 2023.
[4] Z. Dai et al., "Transformer-XL: Attentive language models beyond a fixed-length context," 2019, *arXiv:1901.02860*.
[5] R. Child et al., "Generating long sequences with sparse transformers," 2019, *arXiv:1904.10509*.
[6] J. Qiu et al., "Blockwise self-attention for long document understanding," 2019, *arXiv:1911.02972*.
[7] S. Wang et al., "Linformer: Self-attention with linear complexity," 2020, *arXiv:2006.04768*.
[8] M. Zaheer et al., "Big bird: Transformers for longer sequences," 2021, *arXiv:2007.14062*.
[9] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," 2020, *arXiv:2004.05150*.
[10] N. Kitaev et al., "Reformer: The efficient transformer," 2020, *arXiv:2001.04451*.
[11] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," 2017, *arXiv:1707.04585*.
[12] S. Li et al., "RC-NVM: Dual-addressing non-volatile memory architecture supporting both row and column memory accesses," *IEEE Trans. Comput.*, vol. 68, no. 2, pp. 239–254, Feb. 2019.
[13] W. Cheng et al., "GraphRC: Accelerating graph processing on dual-addressing memory with vertex merging," in *Proc. ICCAD*, 2022, pp. 1–9.
[14] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," in *Proc. 39th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2012, pp. 368–379.
[15] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, "Practical and optimal LSH for angular distance," in *Proc. 29th Annu. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1–21.
[16] A. Arutiunia et al., "Reproducibility challenge: Reformer," in *Proc. 33rd Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2021, pp. 1–10.
[17] Z. Wei et al., "Highly reliable TaOx ReRAM and direct evidence of redox reaction mechanism," in *Proc. IEEE Int. Electron Devices Meeting*, 2008, pp. 1–4.
[18] H.-Y. Cheng et al., "Future computing platform design: A cross-layer design approach," in *Proc. Design, Automa. Test Europe Conf. Exhib. (DATE)*, 2021, pp. 312–317.
[19] Y.-W. Kang, C.-F. Wu, Y.-H. Chang, T.-W. Kuo, and S.-Y. Ho, "On minimizing analog variation errors to resolve the scalability issue of ReRAM-based crossbar accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3856–3867, Nov. 2020.
[20] A. Paszk et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. 33rd Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 8026–8037.
[21] M. Mahoney. "Large text compression benchmark," 2011. [Online]. Available: https://www.mattmahoney.net/dc/text.html
[22] Y. Zhu et al., "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2015, pp. 19–27.
[23] A. L. Maas et al., "Learning word vectors for sentiment analysis," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguist., Hum. Lang. Technol.*, 2011, pp. 142–150.
[24] M. Gerlach and F. Font-Clos, "A standardized project Gutenberg corpus for statistical analysis of natural language and quantitative linguistics," 2018, *arXiv:1812.08092*.
[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI*, vol. 1, no. 8, pp. 1–24, 2018.
[26] S. Rixner, "Memory controller optimizations for web servers," in *Proc. 37th Int. Symp. Microarchit. (MICRO)*, 2004, pp. 355–366.
[27] A. Chen, "A review of emerging non-volatile memory (NVM) technologies and applications," *Solid-State Electron.*, vol. 125, pp. 25–38, Nov. 2016.