ORIGINAL RESEARCH
published: 11 April 2022
doi: 10.3389/felec.2022.847069

Hardware-Software Co-Design of an In-Memory Transformer Network Accelerator

Ann Franchesca Laguna[1]*, *Mohammed Mehdi Sharifi*[1], *Arman Kazemi*[1], *Xunzhao Yin*[2]*, *Michael Niemier*[1] *and X. Sharon Hu*[1]

[1]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, United States, [2]College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China

Transformer networks have outperformed recurrent and convolutional neural networks in terms of accuracy in various sequential tasks. However, memory and compute bottlenecks prevent transformer networks from scaling to long sequences due to their high execution time and energy consumption. Different neural attention mechanisms have been proposed to lower computational load but still suffer from the memory bandwidth bottleneck. In-memory processing can help alleviate memory bottlenecks by reducing the transfer overhead between the memory and compute units, thus allowing transformer networks to scale to longer sequences. We propose an in-memory transformer network accelerator (iMTransformer) that uses a combination of crossbars and content-addressable memories to accelerate transformer networks. We accelerate transformer networks by (1) computing in-memory, thus minimizing the memory transfer overhead, (2) caching reusable parameters to reduce the number of operations, and (3) exploiting the available parallelism in the attention mechanism computation. To reduce energy consumption, the following techniques are introduced: (1) a configurable attention selector is used to choose different sparse attention patterns, (2) a content-addressable memory aided locality sensitive hashing helps to filter the number of sequence elements by their importance, and (3) FeFET-based crossbars are used to store projection weights while CMOS-based crossbars are used as an attentional cache to store attention scores for later reuse. Using a CMOS-FeFET hybrid iMTransformer introduced a significant energy improvement compared to the CMOS-only iMTransformer. The CMOS-FeFET hybrid iMTransformer achieved an 8.96× delay improvement and 12.57× energy improvement for the Vanilla transformers compared to the GPU baseline at a sequence length of 512. Implementing BERT using CMOS-FeFET hybrid iMTransformer achieves 13.71× delay improvement and 8.95× delay improvement compared to the GPU baseline at sequence length of 512. The hybrid iMTransformer also achieves a throughput of 2.23 K samples/sec and 124.8 samples/s/W using the MLPerf benchmark using BERT-large and SQuAD 1.1 dataset, an 11× speedup and 7.92× energy improvement compared to the GPU baseline.

Keywords: transformer network, processing-in-memory, accelerator, sparsity, crossbars, CAMs, FeFET, CMOS

OPEN ACCESS

Edited by:
Ram Krishnamurthy,
Intel, United States

Reviewed by:
Jae-Sun Seo,
Arizona State University, United States
Priyadarshini Panda,
Yale University, United States

****Correspondence:***
Ann Franchesca Laguna
alaguna@nd.edu
Xunzhao Yin
xzyin1@zju.edu.cn

Specialty section:
This article was submitted to Integrated Circuits and VLSI, a section of the journal Frontiers in Electronics

Received: *01 January 2022*
Accepted: *14 March 2022*
Published: *11 April 2022*

Citation:
Laguna AF, Sharifi MM, Kazemi A, Yin X, Niemier M and Hu XS (2022) Hardware-Software Co-Design of an In-Memory Transformer Network Accelerator. Front. Electron. 3:847069. doi: 10.3389/felec.2022.847069

1 INTRODUCTION

Transformer networks (or simply transformers) have continually risen in popularity because of their capability to outperform recurrent neural networks (RNNs) and convolutional neural networks (CNNs), particularly for sequence-based tasks. Different transformer network variants, such as BERT (Devlin et al., 2018), ALBERT (Lan et al., 2019), Megatron (Shoeybi et al., 2019), GPT3 (Brown et al., 2020), and XLNet (Yang et al., 2019) currently hold the best performance in various natural language processing (NLP) applications such as machine translation, named entity recognition, and question answering. Transformer networks are also applicable to other sequential tasks such as audio (Boes and Van hamme, 2019; Yang S.-W. et al., 2020; Wei et al., 2020) and video (Boes and Van hamme, 2019; Wei et al., 2020; Li et al., 2020b) applications. Transformer networks have also recently been used in computer vision applications (Dosovitskiy et al., 2020; Liu et al., 2021). The transformer's superior performance is attributed to the scaled dot-product attention (SDPA) mechanisms that determine the correlation between sequence elements. The attention mechanism in transformer networks can achieve O (1) complexity when completely parallelized and can better model long-range dependencies, making them superior to CNNs and RNNs.

Because of the transformer network's capability to learn long-range dependencies, the transformer network can better analyze longer sequence lengths compared to CNNs and RNNs. This leads to the increase in sequence lengths in NLP datasets (Rae et al., 2019; Sharir et al., 2020). For example, in language modeling, the Penn Treebank (Marcus et al., 1993) and WikiText-103 (Merity et al., 2016) datasets, which are obtained from news and Wikipedia articles, have an average sequence length of 355 and 3.6 K words, respectively. On the other hand, PG-19 (Rae et al., 2019), a newer dataset for language modeling which is obtained from Project Gutenberg books, has an average sequence length of 69 K words. The use of transformer networks in image and video applications can also contribute to the sequence length explosion. As transformer networks improve and are used in more complex applications, the number of parameters also continues to increase (Sharir et al., 2020). The vanilla (original) transformer (Vaswani et al., 2017) began with millions of parameters. Later, transformer network models, e.g., Megatron (Shoeybi et al., 2019) and GPT-3 (Brown et al., 2020), contain billions of parameters. Recently, *switch transformers* (Fedus et al., 2021) used trillions of parameters to account for long-range dependencies in the language model of the PG-19 dataset.

Transformer networks are commonly implemented using general-purpose graphical processing units (GPUs) to exploit the parallelism inherent in the attention mechanism. However, the complexity of implementing the attention mechanism in the GPU is limited to $O\ (dn^2/c)$, where n is the sequence length, d is the number of feature embedding dimensions, and c is the number of parallel cores. Increasing the sequence length and the number of parameters greatly increases the computation latency, memory bandwidth, and energy requirements of transformer networks (Vaswani et al., 2017) because of the quadratic time and space complexity with respect to the sequence length. Transformer networks with linear time complexity have been proposed (Beltagy et al., 2020; Zaheer et al., 2020), but incur the cost of additional space complexity, causing increased memory demand. Moreover, large transformer networks are severely limited by the memory bandwidth. For example, Megatron (Shoeybi et al., 2019), one of the largest transformer networks to date, only achieves 30% of the theoretical peak FLOPS of a GPU because of the memory bandwidth bottleneck.

Different techniques have been proposed to alleviate problems associated with the explosion in memory and time complexity. These techniques include model parallelism using multi-GPUs (Shoeybi et al., 2019), caching attention weights (Beltagy et al., 2020), cross-layer parameter sharing (Lan et al., 2019), model compression (Zafrir et al., 2019; Li et al., 2020c), and sparsification (Child et al., 2019; Kitaev et al., 2020; Fedus et al., 2021). Model parallelism (Shoeybi et al., 2019) further exacerbates the memory bandwidth bottleneck because of sparse random memory accesses and communication between different GPUs. Transformer network model compression is implemented *via* cross-layer parameter sharing (Lan et al., 2019), quantization (Zafrir et al., 2019), and pruning (Li et al., 2020c). Transformer network sparsity can either be (temporal) locality-based (Child et al., 2019; Beltagy et al., 2020) or content-based (Kitaev et al., 2020; Roy et al., 2021). An example of content-based sparsity is using locality-sensitive hashing (LSH), an approximate nearest neighbor search algorithm that hashes nearby points to the same hash signature. However, these techniques do not solve the memory bandwidth bottleneck problem.

Processing-in-memory (PIM) (Mutlu et al., 2020; Sebastian et al., 2020) has been proposed to solve the memory bandwidth bottleneck by eliminating the communication overhead between the compute unit and the memory. PIM has been used in various applications such as few-shot learning (Ni et al., 2019; Ranjan et al., 2019; Challapalle et al., 2020; Reis et al., 2021), DNA assembly (Kaplan et al., 2018; Huangfu et al., 2018; Laguna et al., 2020), and security (Reis et al., 2020b). In particular, PIM-based attention mechanisms have been proposed using content-addressable memories (CAMs) (Laguna et al., 2019a,b), crossbar arrays (Ranjan et al., 2019; Challapalle et al., 2020), and general-purpose computing-in-memory arrays (GP-CiM) (Reis et al., 2020a). CAMs can perform fast parallel searches in a single cycle, while crossbar arrays can perform matrix-vector multiplications in a single cycle. GP-CiM can perform bitwise and arithmetic operations in memory. However, PIM-based attention mechanisms have primarily focused on recurrent and memory augmented neural networks (MANNs). Transformer networks that use SDPA have more parallelization opportunities than recurrent and MANN-based attention mechanisms. The SDPA used in transformer networks can be efficiently implemented using crossbar arrays to perform matrix-vector multiplications. CAM arrays can be used to implement content-based sparse attention *via* LSH.

PIM architectures are either based on complementary metal-oxide-semiconductor (CMOS) memories or emerging technologies (Jeloka et al., 2016; Kang et al., 2017; Zhang et al., 2017; Reis et al., 2018; Ranjan et al., 2019; Yin et al., 2019). While improvements in CMOS technology due to transistor scaling have continuously reduced the cost of on-chip and off-chip memories, PIM devices implemented in CMOS technology have low density and high leakage power and require periodic data refreshing. This makes it difficult to apply CMOS solutions to large, data-centric workloads. Alternatively, non-volatile memories (NVM) based on emerging technologies such as ferroelectric field-effect transistors (FeFET), resistive memories (ReRAM), and phase change memories (PCM) have high density, consume low power, and are non-volatile. However, NVMs require higher write times and energy than CMOS technology, making them less ideal for high-write scenarios. FeFETs are CMOS compatible and have been co-integrated in CMOS platforms by GlobalFoundries (Beyer et al., 2020).

In this paper, we present iMTransformer, an in-memory computing-based accelerator for transformer network inference. iMTransformer employs a combination of crossbars and CAMs. We also use algorithm-based techniques to improve the latency and energy consumption of iMTransformer. iMTransformer reduces the execution time by (1) mitigating the memory-bandwidth bottleneck with processing-in-memory-based hardware, (2) reducing the computational requirements *via* data reuse using attention caches, and (3) maximizing the parallelism that can be achieved with different types of attention mechanisms. iMTransformer further improves energy efficiency by (1) employing an attention selector that can implement masked attention and locality-based sparse attention, (2) using CAMs to implement content-based sparsity through LSH, and (3) using non-volatile FeFET-based crossbars for high-read sublayers and write-efficient CMOS-based crossbars for high-write sublayers.

The standard CMOS implementation of iMTransformer achieves a delay improvement of 7.7× and an energy improvement of 7.81× compared to the GPU baseline for a sequence with length 512 by using PIM. After including model parallelization, sparsity, and using CMOS-FeFET hybrid implementation, iMTransformer achieves 8.96× delay improvement and 12.58× energy improvement compared to the GPU baseline. Furthermore, implementing BERT achieves a delay improvement of 4.68× for the standard implementation and 13.71× after including model parallelization, sparsity, and CMOS-FeFET hybrid iMTransformer implementation. The BERT energy improvement is 4.78× for the standard implementation and 8.95× for the implementation with model parallelization, sparsity, and using CMOS-FeFET hybrid iMTransformer implementation. The hybrid iMTransformer can process 2.23 K samples/s and 125 samples/s/W of the SQuAD 1.1 dataset using BERT-large and achieves an end-to-end improvement of 11× for the delay and 7.92× for the energy compared to the GPU baseline.

2 BACKGROUND

Transformer networks (discussed in **Section 2.1**) currently hold the state of the art accuracy in NLP, computer vision, and various fields and have the capacity to model long-range dependencies (Tay et al., 2020b). That said, the impressive results achieved by transformer networks come with high computational and memory costs as the sequence length increases. Algorithm-based solutions that aim to reduce the space and the computational complexity of transformer networks are presented in **Section 2.2**. These algorithm-based solutions, however, do not solve the memory-bandwidth bottleneck problem. We propose using a PIM-based solution to remove the need for massive data transfers. The PIM-based computing kernels are presented in **Section 2.3**.

2.1 Transformer Networks

Transformer networks have outperformed CNNs and RNNs in various tasks because of their ability to model long-range dependencies through attention mechanisms. Because of this, the transformer networks have been used in various applications such as machine translation, text generation, and language modeling. These different applications have led to different transformer designs. **Section 2.1.1** discusses a key component of transformer networks: the attention mechanism, particularly scaled-dot product attention (SDPA) and multi-head attention (MHA). **Section 2.1.2** then discusses the different types of transformer networks, while **Section 2.1.3** considers the execution time distribution of transformer networks.

2.1.1 Attention

Attention, a crucial component of human intelligence, allows humans to determine the most relevant parts of a sequence (i.e., text) or object and pay less attention to less relevant parts. Neural attention mechanisms work similarly to the human attention mechanism, where more important regions are given more attentional weights than less important ones. Transformer networks rely on SDPA (**Figure 1A**) to determine the relationship between sequence elements. SDPA uses a key-value-based retrieval where each key corresponds to a value. The query vector q is compared to a set of n key vectors $\mathbf{K} = \{k_1, k_2, k_3, \ldots, k_n\}$ to retrieve similar values in the set of n value vectors $\mathbf{V} = \{v_1, v_2, v_3, \ldots, v_n\}$. The SDPA is then calculated as a linear combination of value vectors weighted by the scaled probability distribution of the similarity between q and each k_j in **K.** To keep the variance equal to one, the dot-product attention is scaled by the number of dimensions of the key vectors d_k.

$$attention(q, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{q\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{1}$$

Transformer networks also introduced the concept of MHA (**Figure 1B**) that allows the network to look at the input in different subspaces. MHA allows transformer networks to analyze the relationships among sequence elements in a highly parallelizable manner. In MHA, the feature embedding is projected into different subspaces (one subspace per head) where the sequence elements can be attended in parallel. Each head (or subspace) can reveal different information regarding the input. The i-th head projects the query vector q', the set of key vectors $\mathbf{K}'$, and the set of value vectors $\mathbf{V}'$ by using projection

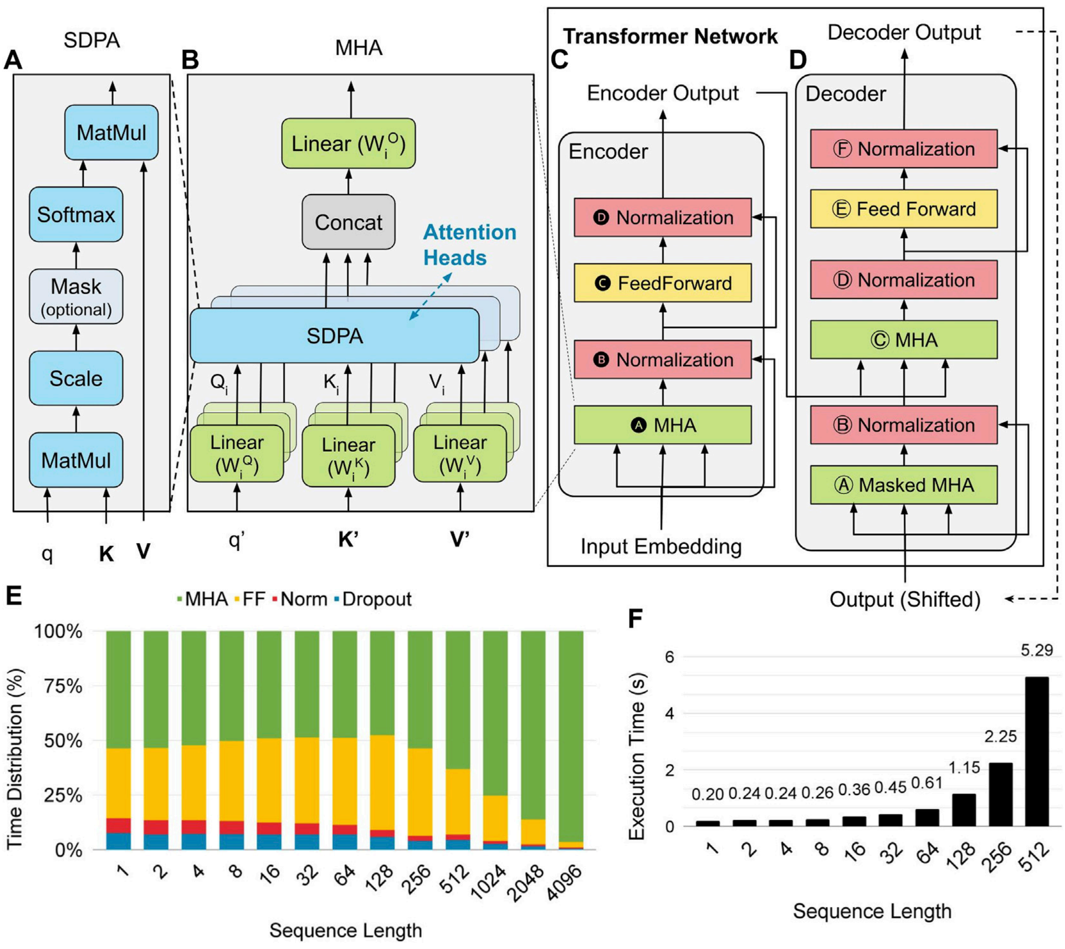

FIGURE 1 | Transformers use scaled dot product attention **(A)** and multi-head attention **(B)** to model long-range dependencies and use an encoder-decoder architecture (Vaswani et al., 2017). The encoder **(C)** and decoder **(D)** layers are composed of multihead attention, feedforward and normalization sublayers. The execution time distribution **(E)** shows the transformer networks becomes more dominated by the multi-head attention as sequence length increases. The execution time **(F)** of transformers increases as the sequence length increases.

matrices $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V$ before calculating the SDPA. The output of the SDPA for each attention head is then concatenated and projected by a linear layer.

2.1.2 Encoder and Decoder Layers of Transformer Networks

The (original) vanilla transformer (Vaswani et al., 2017) is a neural network model that uses an encoder-decoder architecture (**Figures 1C,D**). The encoder layer (**Figure 1C**) accepts a variable-length input and transforms it to a fixed-length feature vector. The decoder layer (**Figure 1D**) accepts this fixed-length feature vector and then transforms it into a variable-length feature vector. This allows the network to accept inputs of varying lengths. Sequence-to-sequence models (Vaswani et al., 2017; Junczys-Dowmunt et al., 2018; Lewis et al., 2019; Raffel et al., 2019) follow the encoder-decoder structure of the vanilla transformer (Vaswani et al., 2017) and are commonly used in machine translation tasks, question answering, and summarization. Other applications, however, require different topologies. Some applications (embedding to sequence), such as text generation, only require decoder layers. Others, such as language modeling and sentence classification (sequence to embedding), only require encoder layers. Transformer networks hence can be categorized into (1) sequence-to-sequence models, (2) decoder-only (or auto-regressive) models, and (3) encoder-only (auto-encoding) models.

Decoder-only transformer networks are naturally used in text and image generation. Decoder-only (auto-regressive) transformer models, such as GPT-2 (Radford et al., 2019) and Transformer-XL (Dai et al., 2019), use masked MHA where the attention scores of the current input are only based on attention scores of past inputs and not on future inputs. Because of their auto-regressive nature, caching the attention scores (keys and values) can reduce the computational requirements of each time step at the cost of higher storage demands (Dai et al., 2019).

Encoder-only transformer networks are usually used for language modeling and sentence/token classification. Encoder-only (auto-encoding) transformer models, such as BERT (Devlin et al., 2018) and ALBERT (Lan et al., 2019), do not use masking, and each input is influenced by past and future inputs (bidirectional). Due to their bidirectional nature, auto-encoding transformer models can be greatly parallelized *via* data and model parallelization (Shoeybi et al., 2019).

2.1.3 Transformer Network Execution Time

To investigate the execution time needed by different functions in the encoder and decoder layers of transformer networks, we profiled the Vanilla Transformer running on a Titan X GPU. **Figures 1E,F** shows the resulting execution time distribution of the transformer network. The encoder and decoder layers of transformer networks are primarily composed of MHA, feedforward, and normalization sublayers, as shown in **Figures 1C,D**. As the sequence length n increases, the execution time of the transformer network increases quadratically $O(n^2)$ due to the MHA. The feedforward layers only increase linearly $O(n)$. Because of this, the MHA dominates the execution time of the transformer network as the sequence length increases, as shown in **Figure 1E**.

In this work, we focus on accelerating MHA. In particular, different types of transformer networks have different MHA properties, which can be exploited when designing transformer network accelerators. For example, decoder-only transformer models require masked MHA. By not executing the masked operations, the number of computations can be reduced. The encoder-only transformer models use bidirectional MHA. By exploiting model parallelism for bidirectional MHA, transformer networks can be accelerated. We utilize these transformer network properties in designing our transformer network accelerator.

2.2 Algorithm-Based Transformer Network Acceleration

Transformer networks have high computational and space complexity because of the employed attention mechanism. Transformer network GPU implementations are bounded by the memory, particularly with longer sequences, because of the $O(dn + dn^2)$ spatial complexity, where d represents the feature embedding dimension, and n is the sequence length. A transformer can also be computation-limited because of the $O(dn^2)$ serialized time complexity of the MHA. The $O(dn^2)$ complexity comes from each time step (sequence element) attending to every other time step (sequence element). However, an $O(1)$ time complexity can be achieved with adequate parallelism. The following sections review four types of algorithm-based acceleration: quantization (**Section 2.2.1**), attention caching (**Section 2.2.2**), model parallelism (**Section 2.2.3**) and sparse attention (**Section 2.2.4**).

2.2.1 Quantization

Reducing the representation precision by quantization can alleviate the memory demand and reduce the time complexity of transformer networks by reducing the amount of data transfer required between the compute and memory units. FullyQT and Q8BERT studied transformer quantization. FullyQT (Prato et al., 2019) used k-bit uniform quantization. Transformer networks quantized in 8-bits performed better in 21 out of 35 experiments made in FullyQT, and there was minimal degradation on the other experiments. Q8BERT (Zafrir et al., 2019) used quantization-aware training and has shown that it performs better than using dynamic quantization. Both FullyQT and Q8BERT have found that 8-bit quantization is found to have comparable accuracy to transformer networks at full precision.

Non-uniform quantization has also been proposed for transformer networks (Chung et al., 2020) and has shown better compression without sacrificing accuracy. These proposed non-uniform quantization techniques decouple feature vectors into a set of weights and binary vectors. These binary codes are also more hardware-friendly than other quantization methods.

2.2.2 Attention Caching

The decoder layer of the transformer is auto-regressive (i.e., the current output is the next input). The attention keys and values from previous time steps affect the current time step. These keys and values have been computed in previous time steps and can be reused instead of recomputing from scratch as implemented in the vanilla transformer (Vaswani et al., 2017). Transformer-XL (Dai et al., 2019) implemented attention caching and achieved up to 1800× improvement during inference at a sequence length of 3.8 K. *Attention caching* improves the computational complexity of the transformer during inference at the cost of higher space complexity. Furthermore, attention caching allows the modeling of a longer range of dependencies by using in conjunction with sparse attention (Child et al., 2019; Dai et al., 2019). Sparse attention is explained in detail in **Section 2.2.4**. We use attention caching in designing our transformer network accelerator.

2.2.3 Model Parallelism

Model parallelism aims to distribute a neural network model into multiple compute units to reduce time complexity and improve compute unit utilization. Megatron (Shoeybi et al., 2019) used model parallelism to split different attention heads into multiple GPUs. By implementing model parallelism. Megatron improved the GPU utilization from 30% theoretical peak FLOPS for a single GPU to 52% theoretical peak FLOPS on 512 GPUs (Shoeybi et al., 2019). Model parallelism is particularly beneficial in accelerating encoder layers of the transformer network because of its bidirectional properties. We utilize model parallelism in accelerating the encoder layers of the transformer network accelerator.

2.2.4 Sparse Attention

The SDPA mechanism described in **Section 2.1.1** uses the full attention mechanism where the query vector is compared to all key and value vectors. The full attention mechanism is the most accurate among different attention patterns and is ideal for shorter sequences. However, because of the quadratic computational complexity of the full attention pattern, it may be necessary to use an attention pattern with less computational complexity at higher sequence lengths. Introducing sparsity in the attention layers has been one of the prominent algorithm optimizations to reduce the computational complexity of the full attention mechanism. Sparse attention only attends to the keys and values relevant to the query. Two major types of sparse attention have been proposed: locality-based sparsity and content-based sparsity. We use both types of sparsity in designing our transformer network accelerator.

Locality-based sparsity uses fixed/random patterns based on the query's relative position to the current time step. The type of data determines the choice of attention pattern, and each attention pattern has its strengths and weaknesses. Longformer (Beltagy et al., 2020) proposed the sliding window and dilated sliding window attention patterns and achieved state-of-the-art results for character modeling tasks. The *sliding window attention*, obtained by focusing on sequence elements t_{sld} time steps away, is ideal for data with high spatial locality. The sliding window can also be dilated where it only pays attention to every other t_{sld} time step. Sparse transformers (Child et al., 2019) proposed *strided attention* and *strided + sliding window attention* patterns and achieved equal or better accuracy compared to full attention mechanisms while reducing the number of operations. Sparse attention is used for music and image generation and machine translations of text. *Strided attention* is obtained by skipping t_{str} time steps and performs well in repeating or oscillating sequences such as audio or video data. A combination of *strided global attention with sliding window attention* is recommended for long documents (Beltagy et al., 2020) where there may be a correlation in texts that are farther away.

Content-based sparsity is another type of sparsity and is based on the similarity of key-value attention pairs with the query. *Content-based sparsity* uses the similarity of the current time step with the previous time step to determine the sparsity. The outing transformer (Roy et al., 2021) uses k-means clustering while Sinkhorn network (Tay et al., 2020a) uses sorting to determine sparsity. Reformer (Kitaev et al., 2020) uses angular locality-sensitive hashing (LSH) to accelerate the transformer for long sequences. Angular LSH uses random hyperplanes that pass through the origin, and the angle of a point is determined by its position with respect to the different hashing hyperplanes. By using LSH to hash, the complexity of the SDP attention is reduced from $O\ (n^2)$ to $O\ (n \log\ n)$. We use angular LSH to introduce content-based sparsity in iMTransformer.

LSH is an approximate nearest neighbor search (NNS) approach for alleviating the curse of dimensionality when searching a large number of data points, thus, allowing for a fast NNS. This is accomplished by hashing similar items with the same binary signature. To perform an NNS, the binary signature of a query is compared to the binary signatures of the keys using a Hamming distance. LSH attention has been used in multiple attention-based neural network architectures (Kaiser et al., 2016; Kitaev et al., 2020).

2.3 In-Memory Computing Based Hardware Kernels for Acceleration

To accelerate transformer networks, iMTransformer leverages crossbars to implement the SDPA and CAMs to realize content-based sparsity using LSH. Crossbars and CAMs are described in detail in **Section 2.3.1**. Crossbars and CAMs can be implemented in CMOS or non-volatile devices based on emerging technologies such as FeFET. These devices are discussed in **Section 2.3.2**. Finally, we review attention and transformer network accelerators in **Section 2.3.3** based on crossbars and CAMs using CMOS and FeFET devices.

2.3.1 Circuits

The transformer network attention mechanism uses linear layers and SDPA, requiring matrix-vector multiplications. Crossbars can accelerate matrix-vector multiplications, and have been used in a variety of DNN applications such as CNNs (Shafiee et al., 2016; Chen P.-Y. et al., 2018) and MANN (Ranjan et al., 2019). A *crossbar* (Gokmen and Vlasov, 2016) is an array-like circuit structure where each input is connected to every output, and vice versa as shown in **Figure 2A**. To perform matrix-vector multiplications, the matrix must be encoded as conductances stored in the crossbar crosspoints $g_{i,j}$, and the vectors as input voltages V_i. The output of the matrix-vector multiplication is read as currents I_j at the columns using analog-to-digital converters (ADCs). The ADCs, typically, consume the majority of the energy (58%) and area (81%) of crossbar arrays (Roy et al., 2020). This energy and area consumption also increases exponentially with increased precision (Shafiee et al., 2016). NeuroSim, a DNN simulator (Chen P.-Y. et al., 2018), uses crossbars to benchmark convolutional and multilayer perceptron-based neural networks.

Content-addressable memories (CAMs) have been used to accelerate attention mechanisms for MANNs (Ni et al., 2019; Laguna A. F. et al., 2019). CAMs are a special type of memory that can perform fast parallel searches across the entire memory (**Figure 2B**). Different CAM designs have been proposed for accelerating various search operations. *Binary CAMs* (BCAMs) store either a logic "0" or logic "1" in each cell while *Ternary CAMs* (TCAMs) can store an additional don't care value "X," to signify that the bit can match to either a logic "0" or a logic "1." Hamming distance is the most straightforward metric for approximate search in BCAMs/TCAMs. CAMs, which are traditionally used in routers and caches (Karam et al., 2015; Yin et al., 2020), have been gaining popularity in data-intensive applications such as nearest neighbor search (Kohonen, 2012; Kazemi et al., 2020, 2021b), bioinformatics (Laguna et al., 2020), neural networks (Chang, 2009; Wang et al., 2010; Li C. et al., 2020, Li et al., 2021 H.; Kazemi et al., 2021a), etc. CAMs also have been used in implementing LSH-based attention (Ni et al., 2019).

2.3.2 Devices

Crossbars and CAMs can be implemented using CMOS or non-volatile memories (NVMs) based on emerging technologies such as resistive RAMs (ReRAMs) and FeFETS. CMOS-based crossbars and CAMs have low write latency and energy, making them ideal for transformer sublayers requiring many write operations. However, CMOS-based circuits have high leakage power (Jerry et al., 2018) which is detrimental for storing static weights.

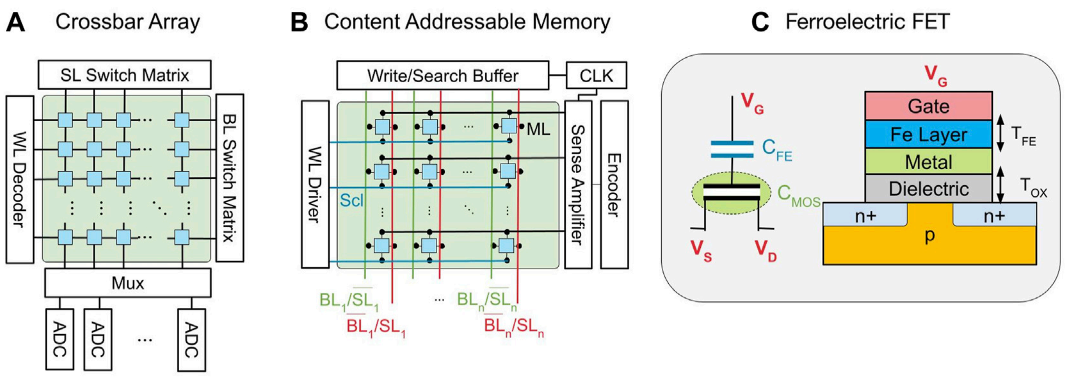

FIGURE 2 | (A) Crossbars are array-like circuit structures where each input is connected to every output typically by a resistor at their crosspoints. **(B)** CAMs can perform fast parallel searches in their memory. **(C)** FeFET devices are CMOS compatible devices that provide non-volatility, high density, and low power consumption.

Unlike CMOS-based memories, NVMs, such as ReRAM and FeFET devices, can store static weights without periodic refreshes. Moreover, NVMs also have a higher density compared to CMOS-based memories. ReRAMs offer good density and fast reads, making them a good candidate for applications requiring a large number of read operations. However, ReRAMs can suffer from cycle-to-cycle (C2C) variation and small $G_{\max}/G_{\min}$ ratios (Jerry et al., 2018). Moreover, ReRAMs have low endurance (10^6) compared to CMOS-based devices (10^{16}) putting them at a disadvantage for write operations (Yu and Chen, 2016).

FeFET based crossbars and CAMs are also great candidates for high-read operations because of their high density and fast read operation. FeFETs have acceptable $G_{\max}/G_{\min}$ and observe lower C2C variations compared to ReRAMs (Jerry et al., 2018). As shown in **Figure 2C**, FeFETs have a similar structure to the metal-oxide-semiconductor field-effect transistors (MOSFETs) in standard CMOS. The only difference between the two is the extra layer of FE oxide deposited in the FeFET's gate stack. Because of this similarity, FeFETs can be integrated into the CMOS fabrication process (Beyer et al., 2020). This enables us to consider a combination of FeFETs and CMOS devices in our architecture to realize better performance. One of the shortcomings of FeFETs (as well as other emerging devices) is device variation. In order to alleviate device variation, we need to use write-verify programming schemes (Sharifi et al., 2021) which increases the write time. FeFET devices also have lower endurance (10^{10}) than CMOS devices (Yu and Chen, 2016). This makes the use of FeFETs challenging for applications that demand a high number of write operations. Transformer networks also may require large-scale memories. Large-scale FeFET memories have been demonstrated (Beyer et al., 2020).

2.3.3 Attention and Transformer Network Accelerators

Different attention-based accelerators have previously been proposed utilizing CAMs (Laguna A. et al., 2019; Challapalle et al., 2020; Laguna A. F. et al., 2019), crossbar arrays (Challapalle et al., 2020; Ranjan et al., 2019), and GP-CIMs (Reis et al., 2020a). These accelerators were used to accelerate the attention mechanism for MANNs and RNNs. However, the attention mechanism for transformer networks has different properties than the ones in MANNs and RNNs. Existing attention accelerators (Laguna et al., 2019a,b; Reis et al., 2020a) use *k*-nearest neighbors, which are not applicable for the transformer network since the attentional weights need to be passed to the next layer. The transformer network also uses MHA, which can be highly parallelized and has not been utilized in existing attention accelerators. An increase in parallelism can also be achieved by exploiting the autoregressive and bidirectional properties of the MHA. These properties have not been considered in existing attention accelerators.

A ReRAM-based transformer (ReTransformer) (Yang X. et al., 2020) has been proposed to accelerate SDPA using ReRAM-based crossbars. ReTransformer uses matrix decomposition to avoid writing the intermediate results. ReRAM has lower endurance compared to CMOS and FeFETs; hence writing in the crossbars must be minimized (Yu and Chen, 2016). However, attention caching (Dai et al., 2019), which necessitates writing in the crossbars, has been shown to improve the execution time of transformer networks for long sequences by reducing the number of operations. The low endurance of ReRAMs makes them undesirable as memory devices for attentional caches. Compared to the GPU approach, ReTransformer achieves a speedup of 23.21× with a 1086× power reduction.

3 METHODOLOGY

To eliminate the limitation from memory bandwidth and exploit transformer networks' high degree of achievable parallelism as sequence length increases, we propose an in-memory computing-based transformer network architecture referred to as iMTransformer. iMTransformer follows a hierarchical design style. **Section 3.1** presents an overview of iMTransformer. Transformer networks have three types of MHA: bidirectional, masked, and encoder-decoder. Each of these MHA types has different characteristics that can be exploited to improve the latency and energy performance of iMTransformer. The mapping of attention mechanisms for different types of MHA is

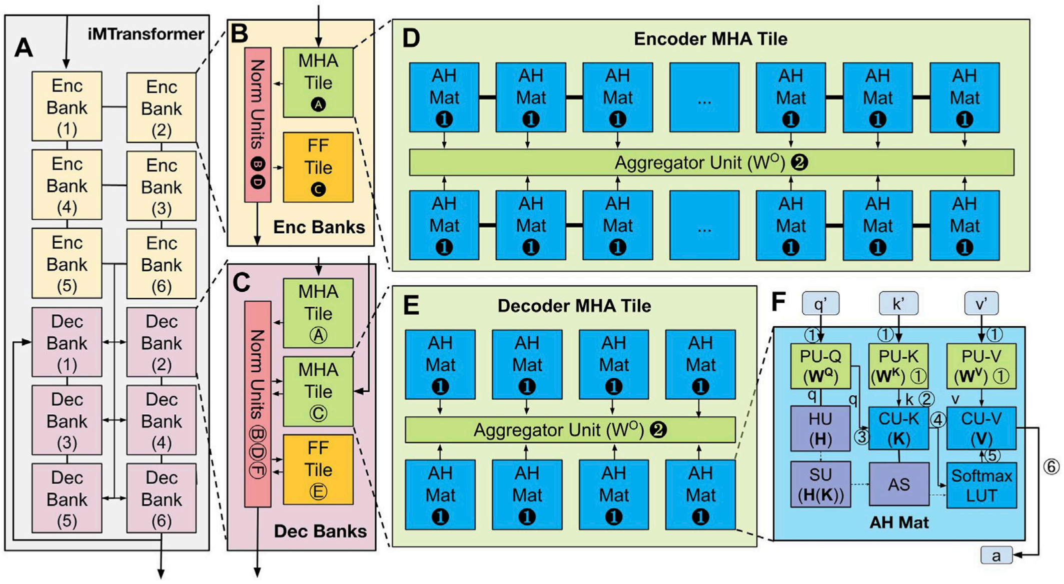

FIGURE 3 | The iMTransformer follows a hierarchical memory structure. The iMTransformer **(A)** is composed of encoder **(B)** and decoder **(C)** banks. The encoder bank is composed of an encoder MHA tile **(D)** and FF tiles and normalization units. The decoder bank is composed of two decoder MHA tiles **(E)**, an FF tile and a normalization unit. The encoder **(D)** and decoder **(E)** MHA tiles are composed of AH Mats **(F)** and an aggregator unit.

expounded in **Section 3.2**. The computational complexity of transformer networks can be reduced by introducing either a content-based or locality-based sparsity. **Section 3.3** aims to reduce energy consumption by utilizing sparsity. Energy can also be improved by utilizing FeFETs for sublayers with high read rates and CMOS for sublayers with high write rates. This technology-level mapping is explained in **Section 3.4**. Finally, we summarize the circuit and device level mapping in **Section 3.5**.

3.1 In-Memory Transformer Network Architecture

iMTransformer is organized in a hierarchical pattern of banks, tiles, and mats as shown in **Figure 3**. This hierarchical pattern follows existing memory hierarchies and the hierarchical pattern of transformer networks as shown in **Figure 1**. The transformer network is composed of encoder layers and/or decoder layers. Each encoder or decoder layer is composed of MHA, FF, and normalization layers. The MHA then can be greatly parallelized into multiple attention heads.

The encoder and decoder layers of transformer networks have different properties which can be exploited to increase parallelism and reduce the number of computations in implementing transformer networks. Hence, iMTransformer uses two types of banks: encoder banks and decoder banks. For an encoder-decoder transformer network, the information first flows through the encoder banks (i.e., Enc Banks 1–6 in **Figure 3A**) before being processed by the decoder banks (i.e., Dec Banks 1–6 in **Figure 3A**). Enc Banks processes data one after another. The output of the final Enc Bank is then passed to all the Dec Banks as the stored key-value pairs of the SDPA. These key-value pairs are then processed in parallel. Because the decoder layers are autoregressive, the input query of Dec Bank 1 is the output of Dec Bank 6.

As shown in **Figures 1C,D**, each encoder and decoder layer of the transformer network consists of multi-head attention, feedforward, and normalization sublayers. The iMTransformer banks (**Figures 3B,C**) are hence composed of a normalization unit (NU), the feedforward tile (FF Tile), and the multi-head attention memory tile (MHA Tile). The NUs execute the layer normalization using additions and shift operations. Since no weights are stored in the NUs, they are shared between sublayers. The FF Tile is composed of two crossbar sub-arrays and performs feedforward layer operations with ReLu activation in between. Encoder banks (**Figure 3B**) have one MHA tile while decoder banks (**Figure 3C**) have two MHA tiles. To follow the information flow in **Figure 1C**, the encoder bank (**Figure 3B**) passes the information to the MHA tile and then to the NU. Then, the NU passes the information to the FF tile and then back to the NU for the output. To map the decoder layer in **Figure 1D** to iMTransformer, the decoder bank's input (**Figure 3E**) is passed to one of its MHA Tiles Ⓐ. The attention vector from the MHA tile is then passed to the NU Ⓑ. The normalized attention vector from the NU is then passed to a different MHA tile Ⓒand back to the NU Ⓓand then to the FF Tile Ⓔ. The final attention vector is then passed to the NU for output Ⓕ.

The MHA (**Figure 1B**) splits the input into queries, keys, and values and passes them to different attention heads. The different attention heads are then concatenated together using a linear function. Since each attention head can be executed in parallel with each other, we decompose each attention head into multiple attention memory mats ❶ (AH Mat), where each mat represents an attention head. The encoder MHA tile (**Figure 3D**) also groups multiple AH mats to take advantage of the bidirectional property of encoder MHA. The details for this are further discussed in **Section 3.2**. Afterward, the aggregator unit (AU) in the MHA Tile (**Figure 3D ❷**) computes a linear combination of the different attention heads.

Each attention head of the MHA is mapped onto an AH Mat. The AH Mat (**Figure 3E**) is composed of three projection units (PU), two caching units (CU), a hashing unit (HU), a sparsity unit (SU), an attention selector (AS), and a softmax lookup table (LUT). As shown in **Figure 1B**, each MHA head is composed of three linear layers that project the query, key, and value into a different subspace. These linear layers are implemented by PU-Q, PU-K, and PU-V, which store the projection matrices of the linear layers ($\mathbf{W^Q}$, $\mathbf{W^K}$, $\mathbf{W^V}$) to project the input to different feature spaces. These projection matrices have static weights during inference. The projected query $q = \mathbf{W^Q} q_t'$, obtain from PU-Q, is then compared to the present and previous projected keys $\mathbf{K} = \{k_t, k_{t-1}, k_{t-2} \dots \}$. and values $\mathbf{V} = \{v_t, v_{t-1}, v_{t-2} \dots \}$ obtained from PU-K ($k_t = \mathbf{W^K} k_t'$) and PU-V $v_t = \mathbf{W^V} v_t'$ respectively. ① The PUs (shown in green boxes as in **Figure 3F**) process the input in parallel. The output of the PUs represent the input to the SDPA, which are implemented using the CUs and the softmax LUT (shown as dark blue boxes in **Figure 3F** ②–⑥).

The SDPA is mapped to the CUs and the softmax LUT. Specifically, the keys and values are cached in crossbars (CU-K and CU-V, respectively) for reuse to reduce computation ②. The outputs of PU-K ($k_t = \mathbf{W^K} k_t'$) and PU-V $v_t = \mathbf{W^V} v_t'$ are written to a column of CU-K and a row of CU-V, respectively. During inference, CU-K and CU-V are constantly written to, while PU-Q, PU-K, PU-V have static weights. ③ The output by PU-Q q is used as the input to CU-K. ④ The output of CU-K, $q\mathbf{K}^T$, passes through the softmax LUT. The dot product scaling $(1/\sqrt{dk})$ is implemented by dropping the three least significant bits for a d_k = 64. This is equivalent to dividing 8. [d_k = 64 is used in most transformer network (Vaswani et al., 2017; Devlin et al., 2018)]. ⑤ The output of the softmax unit is then used as the input to CU-V. ⑥ CU-V produces the final output of the AH Mat. To combine the outputs of AH Mats, the AU concatenates and pools the output of each AH Mat in an MHA Tile. The AU stores the weights $\mathbf{W^O}$ which are static during inference, and ❷ outputs the final multi-head attention score.

The iMTransformer stores a pre-trained transformer network model. Hence, the sizes and number of crossbar arrays for the PUs and HU can be set to exactly store the trained weights. However, the sequence length of each input is variable. Hence the size of the crossbars in the CUs and CAMs in the SU cannot be fixed. Since the keys are stored column-wise, increasing the number of keys (increasing the sequence length) will only require crossbar arrays that are implemented in parallel. On the other hand, as the sequence length increases, the values will require more rows. This will also require more crossbar arrays where each crossbar array outputs a partial sum that needs to be aggregated. We limit the sequence length to 512 to prevent a large number of partial sums and use content-based sparsity for sequence length $n > 4096$. The CAMs are also designed to hold up to sequence lengths up to $n = 4096$. Beyond this, a replacement algorithm must be designed to determine which sequence elements can be removed from the memory. We have not explored this replacement algorithm in this research, and it is a topic for future work.

3.2 Multi-Head Attention Operation Mapping

As discussed in **Section 2.1.3**, transformer networks are heavily dominated by MHAs. Hence iMTransformer focuses on accelerating MHA, which relies primarily on matrix-vector multiplications. Transformer networks have three different types of attention: bidirectional MHA, masked MHA, and encoder-decoder MHA. Each type has different properties, which are exploited by iMTransformer to improve the time and energy consumption of transformer networks. The bidirectional MHA can be parallelized during inference as the entire input sequence is available and does not need to be computed. On the other hand, the energy consumption of the masked MHA can be reduced by turning off columns in the CUs. Finally, encoder-decoder MHA can parallelize the computation of the projected keys and values across different layers. In this section, we discuss the operation mapping for the masked MHA (**Section 3.2.1**), bidirectional MHA (**Section 3.2.2**) and the encoder-decoder MHA in **Section 3.2.3**.

3.2.1 Masked Multi-Head Attention

The masked MHA uses a decoder MHA tile and does not use model parallelism (**Figure 4A**). The transformer's decoder layer is autoregressive by nature (i.e., the input is a delayed version of the output) and uses masked MHA. The masking is essential to prevent a backward information flow (i.e., the future affects the past). The masked MHA computes the projection layers and SDPA for each sequence element before computing the next sequence element.

We utilize an AS, a SU, and a HU to implement masking and sparsity (i.e., locality-based sparsity and content-based sparsity). When masking is implemented, the AS disables the rows of the CU-V that represent future time steps when selecting the values for the SDPA. The AS is a configurable circular shift register that allows different types of locality-based sparsity based on the stored pattern in the register. More details regarding locality-based sparsity and the different attention patterns are discussed in **Section 3.3.1**. The HU is a crossbar array that hashes the keys using LSH based on random projection and stores the hash signatures $H(K)$ in the SU. Content-based sparsity mapping is further expounded in **Section 3.3.2**.

3.2.2 Bidirectional Multi-Head Attention

The encoder layer uses bidirectional MHA to attend to each sequence element's previous, current, and future values. The bidirectional MHA can be parallelized by duplicating weights in the PUs. **Figure 4B** shows the operations for each time step. (Step 1) The PU-K and PU-V of all AH Mats in the same tile perform matrix-vector multiplications in parallel. (Step 2) The first AH Mat then broadcasts the computed key-value pair (k_1, v_1) to other AH Mats. (Step 3) Each AH Mat then writes (k_1, v_1) to the CU-K column-wise and CU-V row-wise. Steps 2-3 are

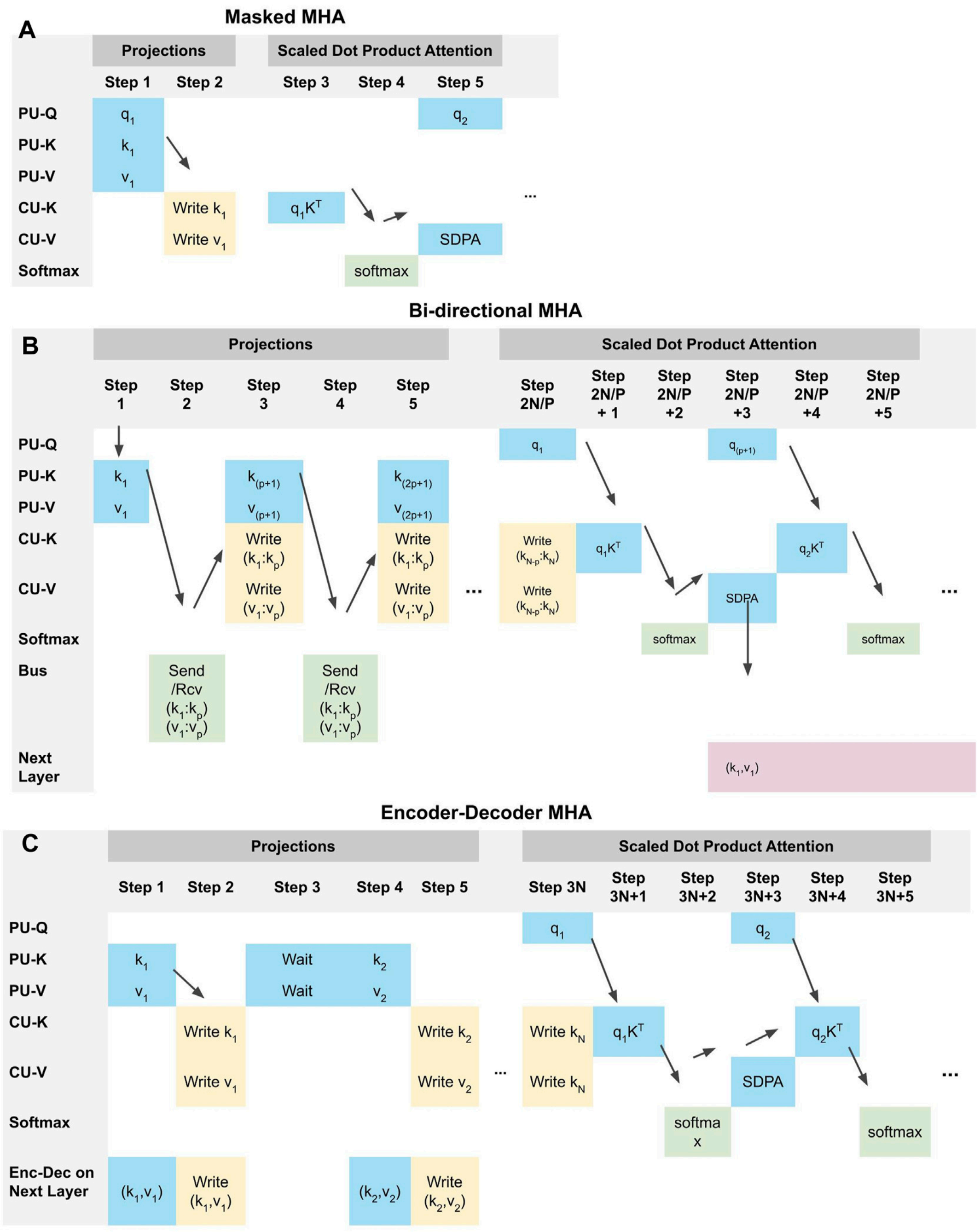

FIGURE 4 | The timeline of implementation for the different types of MHA namely masked MHA **(A)**, bi-drectional MHA **(B)** and encoder-decoder MHA **(C)**. The bi-directional MHA uses model parallelism across multiple AHUs while the encoder-decoder MHA parallelizes the computation of the keys and values across different decoder banks.

repeated for (k_2, v_2) and so on, until the whole sequence is processed. For a sequence length of n and the degree of parallelism p (Step 2n/p to 2n/p+5), each AH Mat performs the SDPA operation ($q\mathbf{K}^T$) for each sequence element. However, the MHA still has an $O(n)$ complexity because of the number of writes. The complexity of the SDPA is reduced to $O(n)$ for $n \leq p$ ($p = 3$ in **Figure 3**). If $n > p$, the time complexity of the SDPA is $O\ (n/p)$. However, the increased parallelism requires more AH Mats and thus a higher peak power. As such, the attention parallelism is limited by the total memory size and/or the thermal design power.

3.2.3 Encoder-Decoder MHA

The encoder-decoder MHA (**Figure 1D**) is implemented by the a decoder MHA tile © in a Dec Bank (**Figure 1C**). The key-value

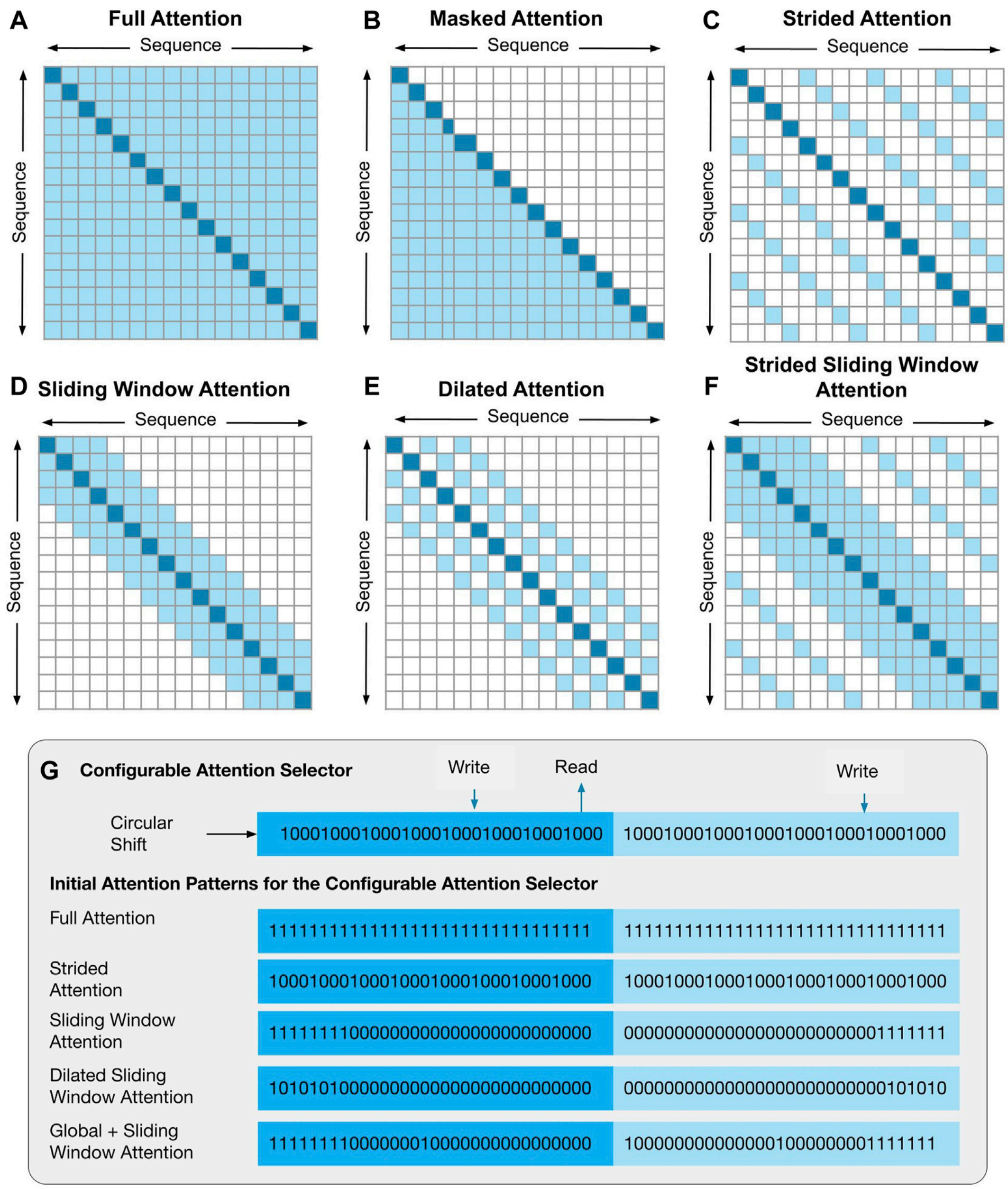

FIGURE 5 | The full attention **(A)** and masked attention pattern **(B)** have high computational complexity. To improve the computational efficiency, different locality-based sparse attention patterns can be used to improve the computational efficiency. Examples of these attention patterns are strided **(C)**, sliding window **(D)**, dilated sliding window **(E)** and strided **(F)** sliding window attention. The configurable attention selector **(G)** uses a circular shift register which contains a pre-defined attention pattern based on the type of attention matrix used.

pair inputs of the MHA tiles ($\mathbf{K}'$, $\mathbf{V}'$) all come from the output of the last encoder bank (Enc Bank 6). Hence the encoder-decoder MHA requires communication between the encoder and the decoder banks. Instead of serializing the computation of the keys and values per decoder layer as in the standard iMTransformer implementation discussed in **Section 3.1**, the computation of the projected key-value pairs ($\mathbf{K}$, $\mathbf{V}$) can be parallelized for all decoder banks (Dec 1–6) in **Figure 3A**. The encoder-decoder MHA processes the decoder sequence query one element at a time as shown in **Figure 4C**. However, the keys and values do not need to be recalculated.

3.3 Sparse Attention for iMTransformer

A full self-attention mechanism (**Figure 5A**) computes for the global correlation between elements in a sequence. However, computing for the full self-attention mechanism can be costly as the sequence length increases (Child et al., 2019). Introducing sparsity in the attention mechanism can reduce the transformer network's computational complexity without sacrificing accuracy. ADCs in a crossbar array consume most of the energy in performing matrix-vector multiplications (Roy et al., 2020). By introducing sparsity, the ADCs in a crossbar that does not contribute to improving the accuracy can be turned off to reduce energy consumption. As discussed in **Section 2.2.4**, there are two main types of sparse attention: locality-based sparse attention and content-based sparse attention. Locality-based sparse attention (**Section 3.3.1**) focuses on the temporal locality between sequence elements while content-based sparse attention (**Section 3.3.2**) focuses on the similarity between sequence elements.

3.3.1 Locality-Based Sparse Attention

Locality-based sparse attention focuses on the sequence elements based on their position relative to the query. Different attention patterns have been proposed that allow more efficient computation of attention for longer sequences without sacrificing accuracy. Examples of supported attention patterns include: strided attention (**Figure 5C**), sliding window attention (**Figure 5D**), dilated sliding window attention (**Figure 5E**) and strided sliding window attention (**Figure 5F**). It is also possible to combine masking with any of these attention patterns.

iMTransformer uses a configurable AS (as shown in **Figure 5G**), which consists of a circular shift register to store a predefined attention pattern for determining the crossbar columns to be activated. The shift register is configurable to implement different sparsity patterns. We use a 128-bit circular shift register to control a 64-column crossbar (**Figure 5**). The first 64 bits determine the crossbar columns to be activated. The last 64 bits function as a buffer necessary to implement certain attention patterns and masking. The shift register is shifted to the right by 1 bit at every time step. The stored pattern in the AS is different for each attention pattern, as shown in **Figure 5G**. The AS has all 1's in its register for the full attention pattern. The strided attention activates every other *c* column. For the 128-bit circular shift register, *c* must be a factor of 128. The sliding window width can be configured by setting the number of 1's stored in the AS.

3.3.2 Content-Based Sparse Attention

Another type of sparsity is based on the similarity of the query with the keys, or content-based sparsity. Content-based sparsity can be implemented using LSH. LSH reduces the number of attention computations for CU-K and CU-V. We implemented angular LSH using random hyperplanes to represent cosine distance. To implement an LSH-based attention mechanism, the keys need to be hashed to a binary signature, where each bit in the signature is hashed using equation $H(q) = (\text{sign}\ (q \cdot r) + 1)/2$. The logic "0" or "1" determines if the point is in the left or right of the hyperplane, respectively.

In iMTransformer, the hash function $H(q)$ is implemented by the HU using crossbars. The binary signature of the query $H(q)$ is compared to the signature of keys $H(K)$, using a CAM-based SU. The SDPA, implemented by the CU-K and CU-V, is only implemented on the keys with m most similar buckets as $H(q)$. By only focusing on the most similar keys and values with the query, we reduce the required multiplications that are more computationally expensive. The hashing function introduces an additional latency and energy overhead when computing the signature. However, it can reduce the softmax computations and value comparisons as the sequence length increases. Thus, LSH-based sparsity is only beneficial when the computational savings associated with avoiding comparisons with all key-value pairs exceeds the hashing overhead. A study of the effect of the hashing overhead and the effect of increasing sequence length is later explained in **Section 4.3.4**.

3.4 Device-Level Mapping

Different devices have different properties that make them ideal for certain operations. CMOS devices, for example, are ideal when the devices need to perform a large number of write operations because of their low write energy and high endurance. However, CMOS devices also have high leakage power, making them undesirable when weights must be stored for extended periods. Alternatively, FeFETs, are non-volatile and have low leakage power. However, FeFETs have much lower endurance compared to CMOS devices. Thus, CMOS devices are better for operations tasks that require a high number of write operations, while FeFETs are better when there are minimal writes and non-volatility is important.

To determine which devices are good fits for iMTransformer, we examine the usage of IMC kernels for iMTransformer. iMTransformer employs crossbars for two different reasons: (i) to store attentional (in PUs and AU) and feedforward weights (in FF Tiles), and (ii) to serve as attentional caches to store intermediate activation (in CUs). The PUs, AU, FF Tiles, and CUs perform different operations and require different properties. The weights in PU, AU, and FF Tiles, once trained, do not change. Hence, they do not require additional write operations and would benefit from using NVMs such as FeFET devices. Alternatively, CUs require frequent write operations and would benefit from memories with lower write times and energy and higher endurance, such as CMOS devices. Therefore, iMTransformer employs FeFET-based crossbars for attentional and feedforward weights because of the FeFET's non-volatility and CMOS-based crossbars as attentional caches because of the CMOS's low write energy and high endurance. The FF Tiles and AU also use FeFET-based crossbars as they also do not require writes and can benefit from FeFET's non-volatility.

FeFET-based crossbars have been proposed in Chen X. et al. (2018). However, this crossbar design stores binary weights and performs XNOR operations. To utilize FeFET-based crossbars, we use a binary-code-based quantization technique introduced for transformer networks (Chung et al., 2020). The binary-code-based quantization uses non-uniform quantization and decouples a feature vector into a scaling factor and a binary vector. Scaling can be done in the crossbar's ADCs, and only bitwise XNOR operations are necessary.

3.5 Summary of Mapping

To summarize the mapping of the iMTransformer architecture, **Table 1** shows the functional units of iMTransformer and their mapping to circuits and devices. The PUs, HUs, AUs, and FFUs

TABLE 1 | Summary of hardware mapping of Transformer Network to iMTransformer.

Transformer network	iMTransformer	Crossbar	CAM	Shifter	CMOS	FeFET
Linear Unit	PU-Q	✓				✓
Linear Unit	PU-K	✓				✓
Linear Unit	PU-V	✓				✓
Attention Cache	CU-V	✓			✓	
Attention Cache	CU-K	✓			✓	
Hash Function	HU	✓				✓
Hash Table	SU		✓		✓	
Sparsity	AS			✓	✓	
Linear Layer	AU	✓				✓
Feedforward Layer	FFU	✓				✓

are implemented using FeFET-based crossbars, while CUs are implemented using CMOS-based crossbars. On the other hand, the SU is implemented using CMOS-based CAM, and AS is implemented using a CMOS-based shifter.

4 RESULTS AND EVALUATION

This section discusses the evaluation of executing transformer networks using iMTransformer. We follow a bottom-up approach beginning with an array-level evaluation in **Section 4.1** and experiment setup in **Section 4.2**. The latency and energy evaluation of using MHA is then shown in **Section 4.3**. For the evaluation in **Section 4.3**, we only assume CMOS-based crossbars. An end-to-end accuracy, latency, and energy evaluation are then presented in **Section 4.4**. We also show the effect of using CMOS-FeFET hybrid iMTransformer implementations where CMOS-based crossbars are used as attentional cache, and FeFET-based crossbars are used to store static weights in **Section 4.4**. Finally, we compare iMTransformer using the MLPerf Inference benchmark for edge devices in **Section 4.5**.

4.1 Array-Level Evaluation

iMTransformer relies heavily on crossbars and CAMs in implementing transformer networks. We consider crossbars and CAMs implemented in a 14 nm technology node. Both CMOS and FeFET devices are used for crossbars, while only CMOS devices are evaluated for CAMs. We used Neurosim to obtain the energy and delay results for reading and writing in crossbar arrays. For the 64 × 64 crossbar arrays, each synapse has 8-bit precision (8 SRAM cells). Each crossbar has one 8-bit ADC per 8 columns, and their overhead is accounted for in the results. We obtained the CAM results using SPICE simulations based on the model used in Yin et al. (2017). Based on our estimation, the interconnects between the arrays add about 20% overhead to latency and energy of the overall architecture. We calculated the interconnection delay by estimating the length of the longest wire in the interconnections. We used the 22 nm design rules (wire width and pitch) to calculate the capacitance and resistance of the longest wire. Using the calculated numbers, we simulated the RC model for the wires and calculated the delay of the longest wire using SPICE simulations.

TABLE 2 | Latency and energy array-level results for CMOS and FeFET crossbars and CAMs.

		Latency (ns)	Energy (fJ)
CMOS Crossbar	Read (whole array)	17.06	78270
	Write (single row)	1.04	2.5
FeFET Crossbar	Read (whole array)	17.39	1900
	Write (single row)	175	118
CMOS CAM	Read (whole array)	0.181	441
	Write (single row)	0.25	1570

Table 2 shows the latency and energy results for 64 × 64 CMOS-based and FeFET-based crossbars as well as the 64 × 64 CMOS-based CAM array. The write latency and energy shown is a write for a single row, while the read latency and energy are for the whole array. CMOS-based crossbars have lower latencies than FeFET-based crossbars. Alternatively, FeFET-based crossbars have 4.12× lower read energy than CMOS-based crossbars. CMOS-based crossbars have faster writes and lower write energy than FeFET-based crossbars. However, these array-level evaluations do not consider the leakage current necessary to store the weights in a CMOS-based crossbar. FeFET-based crossbars are superior for this figure of merit.

4.2 Experiment Setup

We use the following as our baseline hardware for comparison: an Intel Core i7-10750H CPU (2.60GHz, 2592 Mhz, six cores) with a Titan RTX GPU (672 GB/s memory bandwidth and peak performance of 130 TFLOPS). To measure the latency and energy of the baseline implementation of the vanilla transformer (Vaswani et al., 2017), we use NVIDIA NSight and python line-profiler. We use *nvidia-smi* to collect the operating power while the transformer network runs. The average power is then multiplied by the average latency to obtain the energy. We use 8-bit quantization for both the GPU and iMTransformer using quantization-aware fine-tuning (Zafrir et al., 2019). 8-bit quantization is the lowest quantization for both weights and activations and achieves acceptable accuracies (Zafrir et al., 2019; Li et al., 2020c).

Using the python line-profiler, we obtain the number of operations executed in each stage and map them to the number of operations in iMTransformer. The total number of operations varies depending on the sequence length, the number of layers, the embedding size, and the number of heads in the MHA. We then calculate the latency and energy of iMTransformer from the number of operations and the corresponding array level results.

4.2.1 Transformer Models and Datasets

In our experiments, we use three transformer models: the Vanilla transformer, the BERT-base, and BERT-large. The vanilla transformer has six encoder and six decoder layers. The vanilla transformer also has 512-dimensional embedding and eight attention heads in each MHA. The BERT-base has 12 encoder layers, with each layer having 12 heads in each MHA. In comparison, BERT-large has 24 encoder layers with 16 heads in each MHA. The multihead attention model used in **Section 4.4.1** uses a 512-dimensional embedding (similar to the Vanilla Transformer). We use the Vanilla transformer and BERT-base parameters in **Section 4.4.2**. The maximum sequence length is not restricted. BERT-base is then used to evaluate the GLUE dataset in **Section 4.4.3**. MLPerf inference edge uses BERT-large, hence, it is used when comparing accelerators in **Section 4.5**.

We simulated various sequence lengths by truncating a large passages of text to a specified sequence length to obtain the delay and energy metrics in **Sections 4.3**, **4.4.1**, **4.4.2**.

Section 4.4.3 uses the General Language Understanding Evaluation (GLUE) benchmark, which is a collection of NLP datasets for various NLP tasks. The GLUE dataset is composed of the Stanford Sentiment Treebank (SST-2), the Microsoft Research paraphrase Corpus (MRPC), the Quora Question Pairs (QQP) dataset, the Semantic Textual Similarity benchmark (STSB), Multi-Genre Natural Language Inference (MNLI) Corpus, Question Natural Language Inference (QNLI) dataset, Recognizing Textual Entailment (RTE), Winograd Natural Language Inference (WNLI) Schema Challenge.

To evaluate and compare with other hardware implementations, **Section 4.5** uses the setup of MLPerf, an industry-standard machine learning benchmark to evaluate hardware devices in a myriad of machine learning tasks. MLPerf Inference Edge, in particular, specifically evaluates machine learning systems in edge devices during inference. One of the categories in MLPerf Inference Edge is the language processing task that uses BERT-large and the Stanford Question Answering Dataset (SQuAD) 1.1. The QNLI dataset of the GLUE benchmark is derived from the SQuAD dataset. The SQuAD 1.1 dataset is a reading comprehension dataset consisting of 100k + question and answer pairs where the answers to the questions can be obtained from a set of 500 + Wikipedia articles.

4.3 Multi-Head Attention Evaluation

This section presents the multi-head attention evaluation of iMTransformer compared to the GPU baseline. The input sequence length affects the computational demands of transformers. By using crossbars and attention caches, iMTransformer can improve the execution time of transformers, particularly the MHA, as the sequence length increases (**Section 4.3.1**). Furthermore, bidirectional MHA can be parallelized to achieve higher speedups (**Section 4.3.2**). Energy consumption can be further improved by using locality-based sparsity (**Section 4.3.3**) and content-based sparsity (**Section 4.3.4**). We assume a CMOS-based crossbar in this evaluation.

4.3.1 Effect of Increasing Sequence Length

We first evaluate the unparalleled MHA with respect to the GPU baseline. The unparallelized MHA implementation reduces the memory transfer overhead *via* PIM, reduces the number of computations by caching the keys and values, and increases parallelism by using crossbars. **Figures 6A–C** shows the improvement of implementing MHA in iMTransformer when running a bidirectional transformer for inference. At a sequence length of $n = 64$, the standard speedup (shown as red bars in **Figure 6A**) of running the transformer using crossbars is 10.6×. The GPU fully utilizes the memory and compute units at this sequence length and achieves optimal performance.

As the sequence length n increases, the GPU's memory bandwidth and the compute limitations are reached. At $n = 256$, the execution time of the baseline starts to grow quadratically by approximately $n^2/256$. By caching the keys and values and using crossbars, iMTransformer only grows linearly. Caching the keys and values reduces the required number of matrix-vector multiplications for $\mathbf{W^Q}$, $\mathbf{W^K}$, and $\mathbf{W^V}$ by n, thus reducing the delay by n. Since the PUs contains static weights, their execution time still grows by n, as we need to compute the projection of the q', $\mathbf{K}'$, and $\mathbf{V}'$ for each segment. The number of required columns increases for K while the number of required rows increases for $\mathbf{V}$ as n increases. Since the columns in the crossbars can be treated independently, $\lceil n/64 \rceil$ 64×64 arrays can be used to compute the matrix-vector multiplications for K in parallel. Increasing the number of rows in $\mathbf{V}$ requires adding partial sums from the crossbars, incurring additional latency. This additional latency, however, is still not significant at $n = 4096$. Hence, the complexity of iMTransformer is $O(n)$. Thus, iMTransformer achieves a speedup close to $n/256$ compared to a GPU-based solution as shown by the red bars in **Figure 6A**.

4.3.2 Improvement due to Model Parallelism

For the encoder layers, latency can be improved by introducing more parallelism *via* duplicating the crossbars in each head by p (**Section 3.2.2**). The projection operation is parallelized but is bottlenecked by the write operation. SDPA, on the other hand, results in $p\times$ additional improvement in the execution time. By using crossbars and parallelization ($p = n$), we can reduce the time complexity of the projection to $O(n)$ and SDPA to O (1). Accounting for the projections and SDPA, the latency improvement is shown as yellow bars in **Figure 6A**.

Memory and power, however, are bounded. As an example, we set the memory size of each attention head to 15MB, shown as blue bars in **Figure 6A** as bidirectional MHA BM (Bounded Memory). Once the memory limit of iMTransformer is reached, the execution time returns to O (n/p), where p is the number of AH Mats used to represent a single attention head to achieve attention-level parallelism through duplication. At $n = 64$, this speedup equates to 682× speedup (in which $\approx 10 \times$ speedup is from computing in-memory in

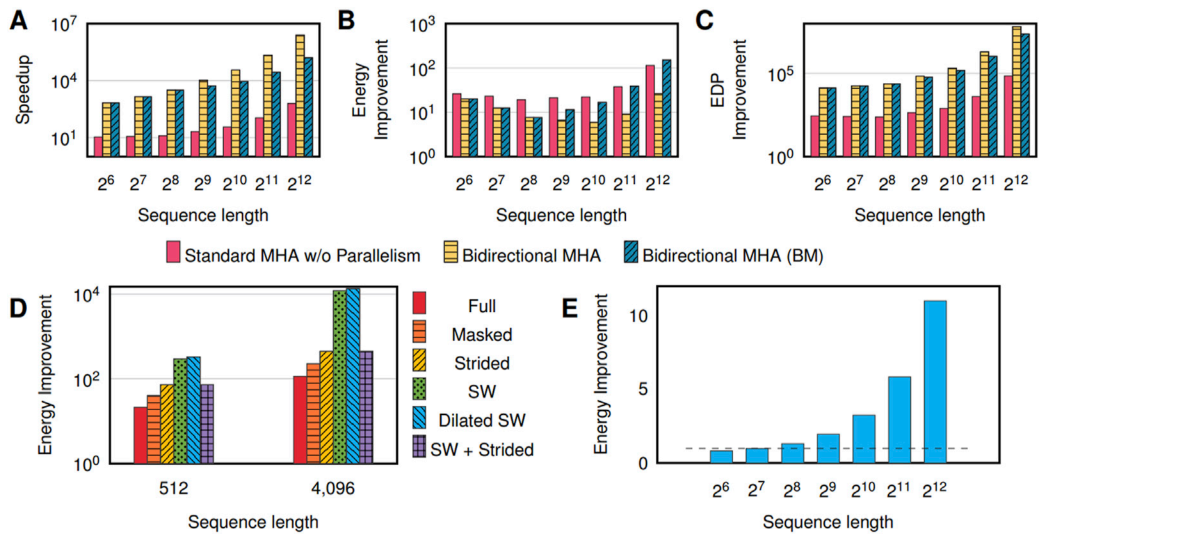

FIGURE 6 | Comparison between CMOS-based iMTransformer and GPU Baseline in terms of the speedup **(A)**, energy **(B)** and EDP **(C)** evaluation results for multi-head attention (with and without parallelism) for varying sequence length. Energy improvements of MHA without parallelism for locality-based **(D)** and content-based **(E)** sparsity.

crossbars, and around ≈ 64 × speedup from parallelization). At n = 4096, a speedup of five orders of magnitude is achieved (×10 from the standard implementation, 100× from duplicating the crossbars, and 100× from attention duplication).

However, duplicating crossbars increases the number of writes and requires communication between AH Mats, resulting in lower energy improvements. In the parallel scenario (**Figure 6B**), increasing attention-level parallelism with respect to sequence length n also requires n× more writes (because $p = n$), resulting in a significant degradation in energy improvement. If $n < p$, only n× more writes are required. If $n \geq p$, there are p× more writes. The memory requirement also increases by a factor of n.

Having more parallelism results in a higher speedup but lower energy gains. However, per **Figure 6C**, increased parallelism translates to better EDP owing to the exponential decrease of delay improvement despite the linear increase in energy improvement.

4.3.3 Improvement due to In-Memory Locality-Based Sparse Attention

Different transformer models have used different attention patterns to reduce the space and computational complexity of the transformer networks. In iMTransformer, the computation of attention scores is highly parallelized. Therefore changing the attention pattern does not reduce latency. However, because of fewer parallel computations, energy consumption is reduced. **Figure 6D** shows the energy improvement of using full, masked, strided, sliding window, dilated, and sliding window + strided attention patterns as compared to the full attention pattern implementation in the GPU.

Compared to the full MHA (red bars in **Figure 6D**), the masked MHA (orange bars in **Figure 6D**) reduces the number of rows in CU-V that needs to be queried by 2× regardless of the sequence length. This is equivalent to turning off the ADCs in CU-K. Masked MHA achieves a speedup of 1.9× and 2× compared to the full SDPA at a sequence length 512 and 4096, respectively. The energy consumption of the strided window MHA (shown as yellow bars in **Figure 6D**) depends on the stride length. (In our example, it is 4). Hence, it can achieve up to 4× energy improvement than the full MHA. For sequence lengths of 512 and 4096, sliding window attention (shown as green bars in **Figure 6D**) achieves energy improvements of 3.43× and 3.91×, respectively. The sliding window MHA achieves higher energy improvement as the sequence length increases.

The sliding window MHA only focuses on w sequence elements for each iteration, where w is the sliding window length. Hence, the sliding window attention increases linearly as the sequence length increases. Compared to the full MHA, sliding window MHA results in an energy improvement of 13.92× at a sequence length equal to 512, and energy improvement of 106× at a sequence length equal to 4096. As some sequence elements are skipped, the dilated sliding window has a small improvement against the sliding window MHA and achieves energy improvements of 15.51× and 118.14× for sequence lengths of 512 and 4096, respectively. Strided MHA consumes more energy than the sliding window MHA and dominates the sliding window + strided MHA. The dilated sliding window attention pattern shows the highest energy improvement, followed by the sliding window attention pattern among the six different patterns shown. However, as stated before, which attention pattern should be used is application-dependent.

4.3.4 Improvement due to In-Memory Content-Based Sparsity

The accuracy of using LSH followed by SDP (LSH + SDP) is dependent on signature length. We have found that comparable accuracies with SDP can be achieved when using a signature length of 1024 bits with a k-nearest neighbor of $k = 16$ on the General Language Understanding Evaluation (GLUE) dataset (Wang et al., 2018). This setup is also similar to using 64 buckets of four hashes each or $1024 = 64 \cdot 2^4$ bits for the angular LSH used in the Reformer network (Kitaev et al., 2020). A pre-trained BERT (Devlin et al., 2018) is used for evaluation. The GLUE dataset is a common benchmark used in transformers (Devlin et al., 2018; Lan et al., 2019).

Since we are only appending the LSH in a pre-trained network, an 11% (1.11×) increase in delay is incurred because of the CAM search. LSH reduces the number of dot-products required as the sequence length increases. Latency overhead still outweighs latency improvement due to fewer attention computations that must be performed at a sequence length of $n = 4096$. However, as CAM searches require fewer numbers of bits and are more energy-efficient than searching *via* dot-product on crossbars, this results in an exponential improvement in energy as the sequence length increases (**Figure 6E**). The bars below the dashed line (speedup = 1) have worse execution times, representing a slowdown. LSH is only advisable for longer sequence lengths (greater than 256).

4.4 End-to-End Evaluation

We evaluate the end-to-end accuracy, delay, and energy of iMTransformer in implementing the transformer network. **Section 4.4.1** considers the end-to-end energy and delay evaluation of iMTransformer. **Section 4.4.2** discusses the improvement in energy consumption by iMTransformer using FeFET-based crossbars for high-read operations, and CMOS-based crossbars for high-write operations. Finally, **Section 4.4.3** reports the effect of device-to-device resistance variation on application-level accuracy.

4.4.1 Energy and Delay Evaluation

Figures 7A,B shows the delay and energy improvement of feedforward and MHA with parallelism and LSH enhancements on the Vanilla and BERT-based transformer at sequence lengths $n = 512$ and $n = 4096$. The standard implementation (without attention-level parallelism) achieves a speedup of 16× and 6.4× for the vanilla transformer and BERT, respectively, when $n = 512$. This increases to 305.6× and 167.2× at $n = 4096$. For the vanilla transformer, only the encoder layers can be parallelized, as opposed to bidirectional transformers such as BERT, where all layers can be parallelized. Therefore, as the sequence length increases, the speedup gained from attention parallelization is smaller for vanilla transformers than bidirectional transformers. LSH slows down the iMTransformer as shown in **Figure 7A** because of the additional hashing operation achieving a 456× speedup for Vanilla transformer and 39.49K× speedup for BERT for $n = 4096$.

The energy improvement of the standard implementation of iMTransformer for the vanilla transformer is 16.84× at $n = 512$. This increases to 56.78× at $n = 4096$ because of fewer number of computations due to caching compared to the GPU implementation. For the standard implementation of BERT, the iMTransformer has an energy improvement of 6.78× at $n = 512$ which increases to 31.16 × $n = 4096$. The parallel implementation is faster than the standard implementation but requires hardware duplication, thus, consuming more energy. LSH improves the energy consumption to 40.18× at $n = 4096$ compared to the GPU implementation.

By including layer normalization as shown in **Figure 7C**, iMTransformer can achieve a delay improvement of 9.01× for Vanilla Transformer at $n = 512$, 13.71× for BERT at $n = 512$, 76× for Vanilla Transformer at $n = 4096$ and 199× for BERT at $n = 4096$. iMTransformer also achieves an energy improvement of 5.83× for Vanilla Transformer at $n = 512$, 1.49× for BERT at $n = 512$, 60.4× for Vanilla Transformer at $n = 4096$ and 33.6× for BERT at $n = 4096$. Algorithmic replacements for layer normalization that are more hardware friendly are needed to improve the iMTransformer accelerator further.

4.4.2 Improvement due to Using Emerging Technology

Based on the data in **Table 2**, CMOS-based crossbars are 1.02× faster than FeFET-based crossbar. On the other hand, FeFET-based crossbars have 4.12× lower read energy than CMOS-based crossbars. CMOS-based crossbars have faster writes and lower write energy than FeFET-based crossbars. As discussed before, CMOS crossbars also have higher endurance than FeFET-based crossbars. Because of this, we utilize CMOS-based crossbars for crossbars that require a high number of write operations and FeFET crossbars for crossbars that do not require write operations. In the CMOS-FeFET hybrid iMTransformer, CMOS is used for attentional caches, while FeFETs are used to store trained weights.

Figure 7D shows the end-to-end latency and energy improvements of iMTransformer using CMOS, FeFET and CMOS + FeFET hybrid iMTransformer implementations when implementing the Vanilla Transformer at $n = 512$. The CMOS-based iMTransformer achieves an end-to-end speedup and energy improvement of 9.01× and 5.83×, respectively. In contrast, the FeFET-based iMtransformer achieves 4.68× end-to-end latency and 12.03× end-to-end energy improvements. The CMOS-based iMTransformer performs 1.93× faster than the FeFET-based iMTransformer. However, the FeFET based iMTransformer has 2.06× lower energy consumption than the CMOS-based transformer.

For the CMOS + FeFET hybrid iMTransformer implementation, we can achieve an 8.96× speedup, which is close to the latency improvement of the CMOS iMTransformer implementation (9.01×). The CMOS + FeFET hybrid iMTransformer implementation achieves a 12.57× energy improvement, higher than either the CMOS or the FeFET iMTransformer implementation. This is because we utilize CMOS crossbars (which are more energy efficient for write operations) for attentional caches and FeFET crossbars (which are more energy efficient for read operations) for storing trained

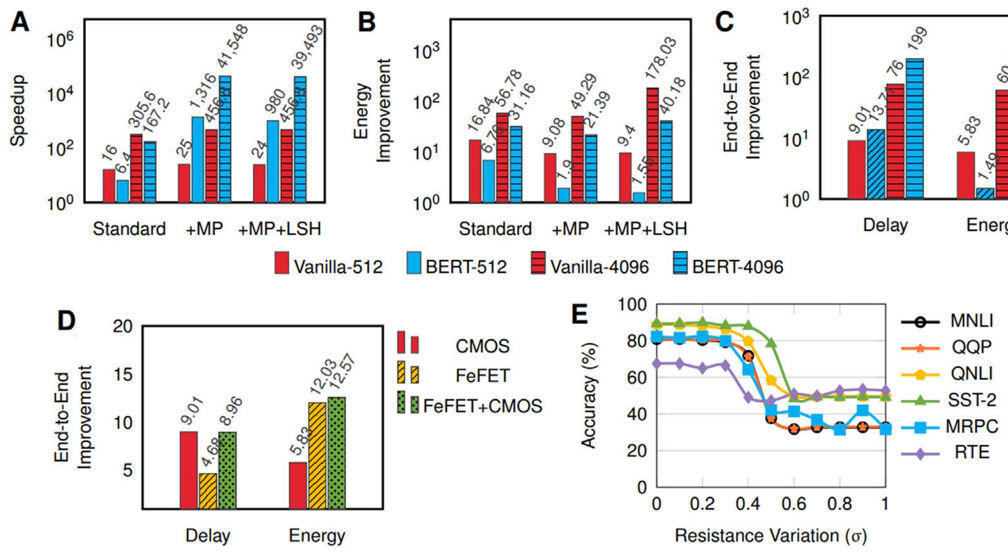

FIGURE 7 | Latency **(A)** and energy **(B)** improvements of evaluating the MHA with and without model parallelism (MP) and sparsity and Feedforward units of Vanilla transformers and BERT with sequence length of 512 and 4096 using CMOS-based iMTransformer compared to GPU Baseline. End-to-end improvement **(C)** of CMOS-based iMTransformer for Vanilla transformers and BERT with sequence length of 512 and 4096 compared to GPU baseline **(D)** End-to-end improvement of iMTransformer for Vanilla transformers with sequence length of 512 using CMOS, FeFET and CMOS + FeFET iMTransformer implementation compared to the GPU baseline. **(E)** Effects of crossbar resistance variation on the accuracy the GLUE dataset.

weights. Thus the CMOS + FeFET iMTransformer implementation exhibits a much better energy-delay-product than using a CMOS or FeFET iMTransformer implementation alone.

4.4.3 Accuracy Evaluation

Device programming methods for writing specific values to FeFETs can result in different stochastic variations of the stored resistance in crossbars. We evaluate the effect of the resistance variation in the crossbar array on the accuracy of the GLUE dataset using the BERT-base model. Crossbar resistance variation is usually modelled as a log-normal distribution (Li P. et al., 2021; Lastras-Montaño et al., 2021) $R_{d2d} = R_{ideal}e^{\theta}$ where $\theta = \mathcal{N}(0, \sigma^2)$. The resistance variation of crossbars can affect the accuracy of implementing the transformer network in iMTransformer. **Figure 7E** shows the effect of the resistance variation on the GLUE benchmark. At $\sigma < 0.3$ (equivalent to 12.7% either below or above the mean), we achieve iso-accuracy for almost all the datasets in the GLUE benchmark. The accuracy then continues to drop until $\sigma > 0.5$ or at 19.15% either below or above the mean. This means that the maximum allowable resistance variation in the crossbar must be below ±12.7%.

4.5 Comparison With Other Accelerators

To provide a comparison with GPU baselines and PIM-based accelerators for transformer networks, we use the MLPerf Inference Single Stream setup. The MLPerf uses BERT-Large and the SQuAD 1.1 dataset. The SQuAD 1.1 dataset has a maximum sequence length of 384, with most of the questions/answers in the dataset in the 100–200 sequence length range. **Table 3** shows the comparison between results from the best MLPerf baseline, the GPU baseline ReTransformer, and the CMOS-based and CMOS-FeFET hybrid Transformer baselines. The MLPerf Inference Edge results represent the best *Single Stream* (batch size of one) results[1] as of the time of this writing. It uses Intel® Xeon® Platinum 8358, NVIDIA A100-SXM-80GB using TensorRT 8.0.1 and CUDA 11.3. The Titan X GPU (the GPU used in this paper as the main baseline) throughput is obtained by averaging all dataset samples. The ReTransformer results are obtained from the best latency performance (81.85 GOps/*s*), and energy efficiency (467.68/*s*/*W*) reported in Yang X. et al. (2020). The iMTransformer results are obtained by the hardware implementation for BERT-Large model parallelization and attention caching. iMTransformer did not use sparsity because the SQuAD 1.1 dataset has short sentences (low sequence length); hence, sparsity will not improve the results significantly.

As shown in **Table 3**, the MLPerf Inference Edge achieves an average throughput of 649.35 samples/sec. The Titan X GPU baseline has a slower GPU and slower memory bandwidth leading to a 204.84 samples/sec throughput. The ReTransformer also has smaller operations/second than MLPerf GPU (A100) and Titan X GPU hence the smaller throughput of 2.73 samples/sec. However, this implementation for ReTransformer is not optimized for BERT and does not utilize the available parallelism of Transformers. The iMTransformer uses model parallelization and attention caching which are not present in the ReTransformer. The iMTransformer achieves a throughput of 2.25 K for the CMOS iMTransformer

[1]Results taken 15 February 2022

TABLE 3 | Comparison of the leading MLPerf Inference - Edge results, the Titan X baseline, ReTransformer and the CMOS-based and Hybrid iMTransformer using MLPerf setting. MLPerf uses BERT-large and the SQuAD 1.1 dataset with a maximum sequence length of 384.

Accelerator	Throughput (Samples/*s*)	Throughput/W (Samples/s/*W*)	Throughput/J (Samples/s/*J*)
MLPerf (Best delay)	649.35	Not available	Not available
Titan X (Baseline)	204.84	15.76	3.23 K
ReTransformer	2.73	15.60	42.59
iMTransformer-CMOS	2.25 K	23.48	52.83 K
iMTransformer-Hybrid	2.23 K	124.8	278 K

implementation and 2.23 K for the CMOS-FeFET hybrid iMTransformer implementation, which are about 11× better than Titan X and 3.43× improvement over the current MLPerf state of the art results.

The energy results are summarized in **Table 3**. The energy results are not available for the MLPerf Inference Edge baseline and hence are not reported in **Table 3**. The Titan X does not achieve its peak compute utilization and uses an average dynamic power of 13 W (peaking at 35 W) to run the transformer network, giving a throughput/W of 15.76 samples/s/W and throughput/J of 3.23 K samples/s/J as shown in **Table 3** iMTransformer reduces the memory transfer overhead by using in-memory computing compared to Titan X and reduces the computation *via* attention caching compared to ReTransformer. iMTransformer achieved around 1.5× energy improvement compared to Titan X and ReTransformer. The CMOS-FeFET hybrid iMTransformer achieves an additional 5.31× energy improvement leading to 125 samples/s/W. Given in-memory computing, model parallelization, attention caching, and using a CMOS-FeFET hybrid iMTransformer implementation, we achieve 278K throughput/J, which is 5.26× better than the CMOS iMTransformer and 86× better than the Titan X (the baseline GPU).

5 DISCUSSION

In this work, we introduced iMTransformer, an in-memory computing-based transformer network accelerator (iMTransformer) that uses FeFET-based and CMOS-based crossbars and CAMs to accelerate transformer networks, specifically the multi-head attention computations.

iMTransformer achieves latency and energy improvements by (1) reducing the memory bottleneck *via* computing-in-memory, (2) reducing the number of computations by storing reusable data in crossbars, and (3) maximizing the parallelism for bidirectional MHA and encoder-decoder MHA. Computing-in-memory alleviates the memory transfer bottleneck by reducing the need to move data from the memory to the compute unit and vice versa. By using crossbars as attentional caches, we can keep data computed from previous time steps to be reused for succeeding time steps. Finally, different types of MHA have parallelism characteristics that can be exploited, such as the bi-directionality of encoder MHAs and parallelizing the computation of encoder-decoder keys and values across different layers.

Furthermore, iMTransformer improves energy efficiency by (1) using an attention selector, (2) exploiting content-based sparsity using CAMs, and (3) using CMOS and FeFET devices. The attention selector allows masking and locality-based sparse attention. The attention selector reduces the number of computations and activation of unnecessary ADCs based on the attention patterns. CAMs are employed to implement LSH for realizing content-based sparsity. Though the LSH computations incur latency overhead, they significantly reduce energy consumption as the sequence length increases. Finally, non-volatile FeFET-based crossbars are used for crossbars that involve highly frequent read operations in the feedforward tile and processing unit. Because of their high write requirements, CMOS-based crossbars are used for the attentional caching units.

For the Vanilla Transformer at sequence length of 512, **Figure 8A** shows a delay improvement of 7.7× for the standard implementation compared to the GPU baseline. Including model parallelization further improves the end-to-end delay by 1.17× or an end-to-end improvement of 9.01× compared to the GPU baseline. Introducing sparsity does not improve the delay while using the CMOS-FeFET hybrid iMTransformer reduces the end-to-end improvement to 8.96×. The end-to-end energy improvement of the standard implementation is 7.81× compared to the GPU implementation. Because of the increase in the write operations, the energy is reduced to 5.71× and slightly improves with sparsity at 5.83× compared to the GPU baseline. The standard implementation achieves a 60.17× EDP improvement while adding all the enhancements improves the EDP by 112.66× compared to the GPU.

Figure 8B shows the delay, energy, and EDP improvement of BERT for a sequence length of 512. The standard implementation of iMTransformer achieves a 4.68× energy improvement compared to the GPU implementation. Because of its bi-directional nature, BERT greatly benefits from model parallelism and achieves 13.76× energy improvement (a 2.94× increase) compared to the GPU baseline. Introducing sparsity and using the hybrid iMTransformer slightly reduces the improvement to 13.71×. The standard implementation has an energy improvement of 4.78×. Because of the additional energy due to writes and greater utilization of model parallelism, the

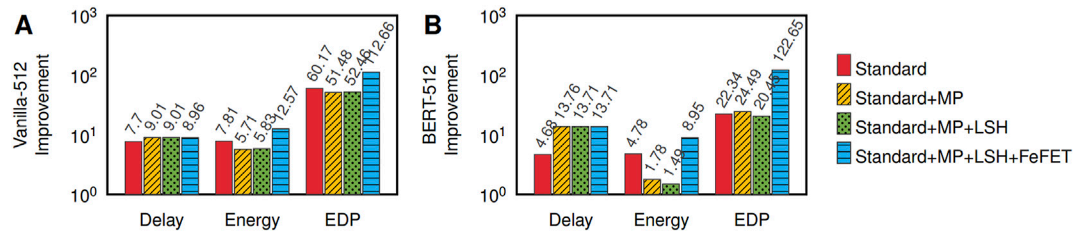

FIGURE 8 | Delay, Energy and EDP improvement for the Vanilla Transformer **(A)** and BERT **(B)** with sequence length of 512 with respect to additional enhancement: Model Parallelism (MP), Locality Sensitive Hashing (LSH) and using FeFET devices.

implementation with model parallelism achieves a 1.78× improvement while adding sparsity further reduces the implementation to 1.49×. However, introducing the use of hybrid transformer improves the energy to 8.95× when compared to the GPU. The standard implementation achieves a 22.34× EDP improvement while adding all the enhancements improves the EDP to 122.65× compared to the GPU baseline.

For both Vanilla and BERT, introducing model parallelization improves the delay. However, because BERT is an encoder-type transformer, it can benefit from model parallelization more than the Vanilla transformer. However, model parallelization comes at the cost of more energy usage. Using FeFET devices for static weights slightly slows down the iMTransformer but greatly improves the energy. The results in **Figure 8** are only for a sequence length of 512. Greater improvements can be achieved at longer sequence lengths. Using the MLPerf benchmark, the hybrid iMTransformer can query 2.23 k samples/sec at 125 samples/s/W as shown in **Table 3**.

In this paper, we have introduced iMTransformer, an in-Memory computing Transformer Network accelerator. We have been able to greatly improve the delay and energy consumption of the transformer's attention mechanism and feedforward layers. Transformer networks are evolving, and different variants for various applications have been developed in the past few years. Transformer network accelerators must adapt to different types of attention mechanisms and transformer network configuration. This work is catered to the original transformer network, where the input is a sequence. This makes transformer networks ideal for NLP applications. Because of the transformer's ability to model long-range dependencies, the sequence length can continue to increase and require more sophisticated algorithms and accelerators.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: https://gluebenchmark.com/, https://rajpurkar.github.io/SQuAD-explorer/.

AUTHOR CONTRIBUTIONS

AL performed the application level and architectural simulations and was the primary proponent of the conception of the idea. MS performed the simulations for the peripherals and interconnects. AK performed the crossbar simulations. XY performed the CAM simulations. MN and XH supervised all experiments. AL wrote the manuscript with input from all authors. All authors reviewed the document.

FUNDING

This work was supported in part by ASCENT, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The funder was not involved in the study, design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

REFERENCES

Beltagy, I., Peters, M. E., and Cohan, A. (2020). *Longformer: The Long-Document Transformer*.

Beyer, S., Dunkel, S., Trentzsch, M., Muller, J., Hellmich, A., Utess, D., et al. (2020). *FeFET: A Versatile CMOS Compatible Device with Game-Changing Potential*.

Boes, W., and Van hamme, H. (2019). "Audiovisual Transformer Architectures for Large-Scale Classification and Synchronization of Weakly Labeled Audio Events," in *Proceedings of the 27th ACM International Conference on Multimedia*. Editors L. Amsaleg, B. Huet, M. A. Larson, G. Gravier, H. Hung, C.-W. Ngo, et al. (New York, NY, USA: Association for Computing Machinery), 1961–1969. doi:10.1145/3343031.3350873

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). *Language Models Are Few-Shot Learners*.

Challapalle, N., Rampalli, S., Jao, N., Ramanathan, A., Sampson, J., and Narayanan, V. (2020). FARM: A Flexible Accelerator for Recurrent and Memory Augmented Neural Networks. *J. Sign Process. Syst.* 92, 1247–1261. doi:10.1007/s11265-020-01555-w

Chen, P.-Y., Peng, X., and Yu, S. (2018a). NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 37, 3067–3080. doi:10.1109/tcad.2018.2789723

Chen, X., Yin, X., Niemier, M., and Hu, X. S. (2018b). "Design and Optimization of FeFET-Based Crossbars for Binary Convolution Neural Networks," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)* (IEEE), 1205–1210. doi:10.23919/date.2018.8342199

Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). *Generating Long Sequences with Sparse Transformers*.

Chua-Chin Wang, C., Chia-Hao Hsu, C., Chi-Chun Huang, C., and Jun-Han Wu, J. (2010). A Self-Disabled Sensing Technique for Content-Addressable Memories. *IEEE Trans. Circuits Syst.* 57, 31–35. doi:10.1109/tcsii.2009.2037995

Chung, I., Kim, B., Choi, Y., Kwon, S. J., Jeon, Y., Park, B., et al. (2020). "Extremely Low Bit Transformer Quantization for On-Device Neural Machine Translation," in *Findings of the Association for Computational Linguistics: EMNLP 2020* (Online: Association for Computational Linguistics), 4812–4826. doi:10.18653/v1/2020.findings-emnlp.433

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-XL: Attentive Language Models beyond a Fixed-Length Context.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2020). An Image Is worth 16x16 Words: Transformers for Image Recognition at Scale

Fedus, W., Zoph, B., and Shazeer, N. (2021). *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*.

Gokmen, T., and Vlasov, Y. (2016). Acceleration of Deep Neural Network Training with Resistive Cross-Point Devices: Design Considerations. *Front. Neurosci.* 10, 333. doi:10.3389/fnins.2016.00333

Huangfu, W., Li, S., Hu, X., and Xie, Y. (2018). "RADAR: A 3D-ReRAM Based DNA Alignment Accelerator Architecture," in Proceedings of the 55th Annual Design Automation Conference (New York, NY, USA: Association for Computing Machinery), 1–6. Article 59 in DAC '18. doi:10.1109/dac.2018.8465882

Jeloka, S., Akesh, N. B., Sylvester, D., and Blaauw, D. (2016). A 28 Nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T Bit Cell Enabling Logic-In-Memory. *IEEE J. Solid-state Circuits* 51, 1009–1021.

Jerry, M., Dutta, S., Kazemi, A., Ni, K., Zhang, J., Chen, P.-Y., et al. (2018). A Ferroelectric Field Effect Transistor Based Synaptic Weight Cell. *J. Phys. D Appl. Phys.* 51, 434001.

Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, T., et al. (2018). *Marian: Fast Neural Machine Translation in C++*.

Kaiser, L., Nachum, O., Roy, A., and Bengio, S. (2016). "Learning to Remember Rare Events," in 5th International Conference on Learning Representations.

Kang, W., Wang, H., Wang, Z., Zhang, Y., and Zhao, W. (2017). In-Memory Processing Paradigm for Bitwise Logic Operations in STT-MRAM. *IEEE Trans. Magn.* 53, 1–4. doi:10.1109/tmag.2017.2703863

Kaplan, R., Yavits, L., and Ginosar, R. (2018). *RASSA: Resistive Pre-alignment Accelerator for Approximate DNA Long Read Mapping*, 44–54.

Karam, R., Puri, R., Ghosh, S., and Bhunia, S. (2015). Emerging Trends in Design and Applications of Memory-Based Computing and Content-Addressable Memories. *Proc. IEEE* 103, 1311–1330. doi:10.1109/jproc.2015.2434888

Kazemi, A., Sahay, S., Saxena, A., Sharifi, M. M., Niemier, M., and Sharon Hu, X. (2021a). A Flash-Based Multi-Bit Content-Addressable Memory with Euclidean Squared Distance.

Kazemi, A., Sharifi, M. M., Laguna, A. F., Müller, F., Rajaei, R., Olivo, R., et al. (2020). *Memory Nearest Neighbor Search with FeFET Multi-Bit Content-Addressable Memories*.

Kazemi, A., Sharifi, M. M., Zou, Z., Niemier, M., Hu, X. S., and Imani, M. (2021b). "MIMHD: Accurate and Efficient Hyperdimensional Inference Using Multi-Bit In-Memory Computing," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 1–6. doi:10.1109/islped52811.2021.9502498

Kitaev, N., Kaiser, L., and Levskaya, A. (2020). "Reformer: The Efficient Transformer," in *8th International Conference on Learning Representations*.

Kohonen, T. (2012). *Associative Memory: A System-Theoretical Approach*, 17. Springer Science & Business Media.

Laguna, A. F., Gamaarachchi, H., Yin, X., Niemier, M., Parameswaran, S., and Hu, X. S. (2020). "Seed-and-Vote Based In-Memory Accelerator for DNA Read Mapping," in *2020 IEEE/ACM International Conference on Computer Aided Design*, 1–9. doi:10.1145/3400302.3415651*(ICCAD)*

Laguna, A. F., Yin, X., Reis, D., Niemier, M., and Hu, X. S. (2019b). "Ferroelectric FET Based In-Memory Computing for Few-Shot Learning," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. Editors H. Homayoun, B. Taskin, T. Mohsenin, and W. Zhao (New York, NY, USA: Association for Computing Machinery), 373–378. GLSVLSI '19. doi:10.1145/3299874.3319450

Laguna, A., Niemier, M., and Hu, X. S. (2019a). "Design of Hardware-Friendly Memory Enhanced Neural Networks," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. Editors J. Teich and F. Fummi, 1583–1586. ieeexplore.ieee.org. doi:10.23919/date.2019.8715198

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). "ALBERT: A Lite BERT for Self-Supervised Learning of Language Representations," in *ICLR*.

Lastras-Montaño, M. A., Del Pozo-Zamudio, O., Glebsky, L., Zhao, M., Wu, H., and Cheng, K.-T. (2021). Ratio-based Multi-Level Resistive Memory Cells. *Sci. Rep.* 11, 1351. doi:10.1038/s41598-020-80121-7

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., et al. (2019). *BART: Denoising Sequence-To-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*.

Li, C., Graves, C. E., Sheng, X., Miller, D., Foltin, M., Pedretti, G., et al. (2020a). Analog Content-Addressable Memories with Memristors. *Nat. Commun.* 11, 1638. doi:10.1038/s41467-020-15254-4

Li, H., Chen, W.-C., Levy, A., Wang, C.-H., Wang, H., Chen, P.-H., et al. (2021a). SAPIENS: A 64-kb RRAM-Based Non-volatile Associative Memory for One-Shot Learning and Inference at the Edge. *IEEE Trans. Electron. Devices* 68, 6637–6643. doi:10.1109/ted.2021.3110464

Li, P., Song, G., Cai, K., and Yu, Q. (2021b). Across-Array Coding for Resistive Memories with Sneak-Path Interference and Lognormal Distributed Resistance Variations. *IEEE Commun. Lett.* 25, 3458–3462. doi:10.1109/lcomm.2021.3111218

Li, Z., Li, Z., Zhang, J., Feng, Y., Niu, C., and Zhou, J. (2020b). *Bridging Text and Video: A Universal Multimodal Transformer for Video-Audio Scene-Aware Dialog*.

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., et al. (2020c). "Train Big, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers," In *Proceedings of the 37th International Conference on Machine Learning*. Editors H. D. Iii and A. Singh, 5958–5968. (PMLR), vol. 119 of Proceedings of Machine Learning Research.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., et al. (2021). *Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows*.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). *Building a Large Annotated Corpus of English: The Penn Treebank*.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel Mixture Models

Mutlu, O., Ghose, S., Gómez-Luna, J., and Ausavarungnirun, R. (2020). *A Modern Primer on Processing in Memory*.

Ni, K., Yin, X., Laguna, A. F., Joshi, S., Dünkel, S., Trentzsch, M., et al. (2019). Ferroelectric Ternary Content-Addressable Memory for One-Shot Learning. *Nat. Electron.* 2, 521–529. doi:10.1038/s41928-019-0321-3

Prato, G., Charlaix, E., and Rezagholizadeh, M. (2019). *Fully Quantized Transformer for Machine Translation*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language Models Are Unsupervised Multitask Learners. *OpenAI blog* 1, 9.

Rae, J. W., Potapenko, A., Jayakumar, S. M., and Lillicrap, T. P. (2019). *Compressive Transformers for Long-Range Sequence Modelling*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., et al. (2019). *Exploring the Limits of Transfer Learning with a Unified Text-To-Text Transformer*, 10683. arXiv preprint arXiv:1910.

Ranjan, A., Jain, S., Stevens, J. R., Das, D., Kaul, B., and Raghunathan, A. (2019). "X-MANN: A Crossbar Based Architecture for Memory Augmented Neural Networks," in *Proceedings of the 56th Annual Design Automation Conference 2019* (New York, NY, USA: Association for Computing Machinery), 1–6. Article 130 in DAC '19.

Reis, D., Laguna, A. F., Niemier, M., and Hu, X. S. (2020a). "A Fast and Energy Efficient Computing-In-Memory Architecture for Few-Shot Learning Applications," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 127–132. ieeexplore.ieee.org. doi:10.23919/date48585.2020.9116292

Reis, D., Laguna, A. F., Niemier, M., and Hu, X. S. (2021).Attention-in-Memory for Few-Shot Learning with Configurable Ferroelectric FET Arrays. In Proceedings of the 26th Asia and South Pacific Design Automation Conference. New York, NY, USA: Association for Computing Machinery, 49–54. ASPDAC '21. doi:10.1145/3394885.3431526

Reis, D., Niemier, M., and Hu, X. S. (2018). "Computing in Memory with FeFETs," in *Proceedings of the International Symposium on Low Power Electronics and Design* (New York, NY, USA: ACM), 1–6. Article 24 in ISLPED '18. doi:10.1145/3218603.3218640

Reis, D., Takeshita, J., Jung, T., Niemier, M., and Hu, X. S. (2020b). Computing-in-Memory for Performance and Energy-Efficient Homomorphic Encryption. *IEEE Trans. VLSI Syst.* 28, 2300–2313. doi:10.1109/tvlsi.2020.3017595

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. (2021). Efficient Content-Based Sparse Attention with Routing Transformers. *Trans. Assoc. Comput. Linguistics* 9, 53–68. doi:10.1162/tacl_a_00353

Roy, K., Chakraborty, I., Ali, M., Ankit, A., and Agrawal, A. (2020). "In-Memory Computing in Emerging Memory Technologies for Machine Learning: An Overview," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 1–6. doi:10.1109/dac18072.2020.9218505

Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R., and Eleftheriou, E. (2020). Memory Devices and Applications for In-Memory Computing. *Nat. Nanotechnol.* 15, 529–544. doi:10.1038/s41565-020-0655-z

Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J. P., Hu, M., et al. (2016).Isaac. *SIGARCH Comput. Archit. News* 44, 14–26. doi:10.1145/3007787.3001139

Sharifi, M. M., Pentecost, L., Rajaei, R., Kazemi, A., Lou, Q., Wei, G.-Y., et al. (2021). Application-driven Design Exploration for Dense Ferroelectric Embedded Non-volatile Memories.

Sharir, O., Peleg, B., and Shoham, Y. (2020). *The Cost of Training NLP Models: A Concise Overview*.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2019). Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism

Tay, Y., Bahri, D., Yang, L., Metzler, D., and Juan, D.-C. (2020a). "Sparse Sinkhorn Attention," in *International Conference on Machine Learning*, 9438–9447.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., et al. (2020b). *Long Range arena: A Benchmark for Efficient Transformers*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). "Attention Is All You Need,". In *Advances in Neural Information Processing Systems*. Editors I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, et al. (Long Beach, California: Curran Associates, Inc.), 30, 5998–6008.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2018). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP* (Brussels, Belgium: Association for Computational Linguistics), 353–355. doi:10.18653/v1/w18-5446

Wei, L., Zhang, J., Hou, J., and Dai, L. (2020). Attentive Fusion Enhanced Audio-Visual Encoding for Transformer Based Robust Speech Recognition.

Yang, S.-W., Liu, A. T., and Lee, H.-Y. (2020a). Understanding Self-Attention of Self-Supervised Audio Transformers.

Yang, X., Yan, B., Li, H., and Chen, Y. (2020b). "ReTransformer: ReRAM-Based Processing-In-Memory Architecture for Transformer Acceleration," in Proceedings of the 39th International Conference on Computer-Aided Design (New York, NY, USA: Association for Computing Machinery), 1–9. Article 92 in ICCAD '20.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding*Adv. Neural Inf. Process. Syst.* (Vancouver, Canada: Curran Associates, Inc.) 32, 5753–5763.

Yen-Jen Chang, Y. (2009). A High-Performance and Energy-Efficient TCAM Design for IP-Address Lookup. *IEEE Trans. Circuits Syst.* 56, 479–483. doi:10.1109/tcsii.2009.2020935

Yin, X., Li, C., Huang, Q., Zhang, L., Niemier, M., Hu, X. S., et al. (2020). FeCAM: A Universal Compact Digital and Analog Content Addressable Memory Using Ferroelectric. *IEEE Trans. Electron. Devices* 67, 2785–2792. doi:10.1109/ted.2020.2994896

Yin, X., Ni, K., Reis, D., Datta, S., Niemier, M., and Hu, X. S. (2019). An Ultra-dense 2FeFET TCAM Design Based on a Multi-Domain FeFET Model. *IEEE Trans. Circuits Syst.* 66, 1577–1581. doi:10.1109/tcsii.2018.2889225

Yin, X., Niemier, M., and Hu, X. S. (2017). "Design and Benchmarking of Ferroelectric FET Based TCAM," in *Design, Automation Test in Europe Conference Exhibition (DATE)* (Lausanne, Switzerland), 1444–1449. doi:10.23919/date.2017.7927219

Yu, S., and Chen, P.-Y. (2016). Emerging Memory Technologies: Recent Trends and Prospects. *IEEE Solid-state Circuits Mag.* 8, 43–56. doi:10.1109/mssc.2016.2546199

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. (2019). *Q8BERT: Quantized 8bit BERT*.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., et al. (2020). "Big Bird: Transformers for Longer Sequences," in *NeurIPS*.

Zhang, J., Wang, Z., and Verma, N. (2017). In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array. *IEEE J. Solid-state Circuits* 52, 915–924. doi:10.1109/jssc.2016.2642198

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.