THESIS

SILICON PHOTONIC HARDWARE ACCELERATORS FOR TRANSFORMERS AND GRAPH NEURAL NETWORKS

Submitted by

Salma Afifi

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2023

Master's Committee:

Advisor: Sudeep Pasricha

Mahdi Nikdast
Yashwant Malaiya

ABSTRACT

SILICON PHOTONIC HARDWARE ACCELERATORS FOR TRANSFORMERS AND GRAPH NEURAL NETWORKS

The rapid growth of artificial intelligence (AI) applications has revolutionized the way we process data, make decisions, and interact with machines. Specifically, artificial neural networks (ANNs) have significantly evolved and now encompass various advanced neural networks such as transformers and graph neural networks (GNNs). This has enabled the development of innovative AI applications that can transform several industries, including healthcare, recommendation systems, and robotics. Transformer and transformer-based neural networks have outperformed multiple ANNs, such as convolution neural networks (CNNs) and recurrent neural networks (RNNs), across many natural language processing (NLP) tasks. Moreover, transformers are currently being integrated into vision tasks through using the vision transformer model (ViT). Similarly, GNNs have witnessed a surge of advancements over the past few years and have established their proficiency in dealing with graph-structured data.

Nevertheless, each of these neural networks imposes unique challenges, hindering their inference and usage in resource-constrained systems. For instance, the transformer model's size, number of parameters, and complexity of operations lead to long inference times, large memory footprint, and low computation-to-memory ratio. On the other hand, GNNs inference challenges are due to their dense and very sparse computations. Additionally, the wide variety of possible input graphs structure and algorithms dictate the need for a system capable of efficiently adapting their execution and operations to the specific graph structure and effectively scaling to

extremely large graphs. Accordingly, conventional computing processors and ANN accelerators are not tailored to cater for such challenges, and using them to accelerate transformers and GNN execution can be highly inefficient.

Furthermore, the utilization of traditional electronic accelerators entails a number of limitations, including escalating fabrication costs due to low yields and diminishing performance improvements, associated with semiconductor-technology scaling. This has led researchers to start investigating other technologies for ANN acceleration such as silicon photonics which enables performing complex operations in the optical domain with low energy consumption and at very high throughput. While several hardware accelerators leveraging silicon photonics have been presented for networks such as CNNs, none have been customized for emerging complex neural networks such as transformers and GNNs. Due to the various challenges associated with each of these networks, designing reliable and efficient inference hardware accelerators for transformers and GNNs is a non-trivial problem.

This thesis introduces two novel silicon-photonic-based hardware architectures for energy efficient and high throughput inference acceleration. As our first contribution, we propose a non-coherent silicon photonic hardware accelerator for transformer neural networks, called TRON. We demonstrate how TRON is able to accommodate a wide range of transformer and transformer-based neural networks while surpassing GPU, CPU, TPU, and several state-of-the-art transformer hardware accelerators. For GNN inference acceleration, we propose GHOST, a hardware accelerator that integrates various device-, circuit- and architecture-level optimizations which enable it to efficiently process a broad family of GNNs and real-world graph structures and sizes. When compared to multiple state-of-the-art GNN hardware accelerators, GPUs,

CPUs, and TPUs, our experiments showcase how GHOST exhibits significantly better performance and energy efficiency.

ACKNOWLEDGEMENTS

I would like to thank all the individuals whose encouragement and support have made the completion of this thesis possible.

I would like to express my sincere gratitude to my advisor, Dr. Sudeep Pasricha, who has guided me through every step of my research during my Master's Thesis. His professional expertise and constant encouragement have been instrumental in shaping my academic and personal growth. I am deeply grateful for the opportunities and challenges that he has provided me with, which have motivated me to embark on new academic and personal endeavors. Throughout my research journey, he has been a constant source of inspiration, providing me with invaluable advice and feedback that has allowed me to navigate the complexities of graduate school life with confidence. I had the indispensable opportunity to learn a great deal from him about computer architecture, machine learning, silicon photonics, and embedded systems. Moreover, I am extremely thankful for all his patience and mentorship which have enabled me to develop new skills and explore challenging problems in these domains.

I would like to take this opportunity to thank the respected members of my Master's Thesis committee, Dr. Mahdi Nikdast, and Dr. Yashwant Malaiya for their valuable time and feedback. I am also profoundly grateful for Dr. Nikdast's expert guidance, and invaluable help on the research projects in this thesis.

Furthermore, I would like to thank my research partner, Febin Sunny, for his collaboration, expertise, and support throughout the course of our research projects. I am especially grateful for his willingness to engage in intellectually stimulating conversations that have pushed me to think

more deeply about our research questions and have boosted my morale when faced with complex problems. This list cannot be complete without mentioning the company and the help from my current lab mates in Dr. Pasricha's EPIC lab as well.

I am grateful for my wonderful family – my father Sameh Afifi, my mother Somaya Soliman, and my siblings Ghada, Ahmed, and Sarah Afifi – who have supported me throughout my years. As exemplary role models, they ignited my interest in the research field and provided me with the encouragement I need to pursue my graduate studies.

TABLE OF CONTENTS

LIST OF RESEARCH PUBLICATIONS

JOURNAL PUBLICATIONS:

S. Afifi, F. Sunny, A. Shafie, M. Nikdast, and S. Pasricha, "GHOST: A Graph Neural Network Accelerator using Silicon Photonics,", *IEEE/ACM CASES (ESWEEK), 2023.* [Under review]

CONFERENCE PUBLICATIONS:

S. Afifi, F. Sunny, M. Nikdast, and S. Pasricha, “TRON: Transformer Neural Network Acceleration with Non-Coherent Silicon Photonics”, to appear, *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2023.

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

This chapter presents an overview on Artificial Intelligence (AI) and outlines the challenges faced when designing application-specific AI hardware accelerators. We discuss the fundamental limitations in existing computing platforms and AI hardware accelerators pertaining to the acceleration of advanced artificial neural networks (ANNs). Moreover, the challenges and benefits associated with leveraging silicon photonics for ANN acceleration are thoroughly outlined. This chapter also presents a general overview of the contributions of this thesis.

1.1 MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

1.1.1 OVERVIEW ON DEEP LEARNING

AI is a transformative technology that has the potential to revolutionize the way we live and work. AI refers to the ability of machines to mimic human intelligence and perform tasks that would otherwise require human interference, such as image processing, speech recognition, decision-making, and natural language processing (NLP). Machine learning (ML) algorithms, specifically, deep learning, have contributed massively to the success of AI. ML is a subset of AI algorithms that leverages statistical techniques to learn patterns and relationships in data [3]. On the other hand, deep learning is a subset of ML that uses neural networks to extract and learn more complex features from different types of datasets.

Through employing deep neural networks (DNNs), deep learning has proven its proficiency as the most prominent ML technique, enabling most of today's advancements in AI [4]. While DNNs have gained a lot of attention in recent years, the idea of deep learning is not

new. In fact, the concept of making machines as intelligent as humans emerged in 1837, pioneered by Charles

Babbage [5]. The field of AI started to surface later when Warren McCulloch and Walter Pitts developed the first computational model for neural networks and neuron operation in 1943 [5]. Frank Rosenblatt, a psychologist, then introduced the perceptron algorithm in 1957 [6], which paved the way for ANNs and DNNs. By utilizing this algorithm, Rosenblatt was able to create the first electronic computational device that adhered to the biological principles behind the human brain's functionality, known as the single-layer perceptron. In general, deep learning aims to mimic the structure of the human brain with its billions of interconnected neurons acting as computational units. The human brain operates in a hierarchical manner, beginning with basic ideas and gradually building up to more complex concepts. This approach to learning is mirrored in deep learning models, which deconstruct input information into features and then combine them to accomplish specific objectives, such as identification or categorization. Following the training phase, during which the model learns important features, DNNs can be used without any human involvement to complete similar tasks [7]. Figure 1 shows how the artificial neuron aims to mimic the brain neuron.

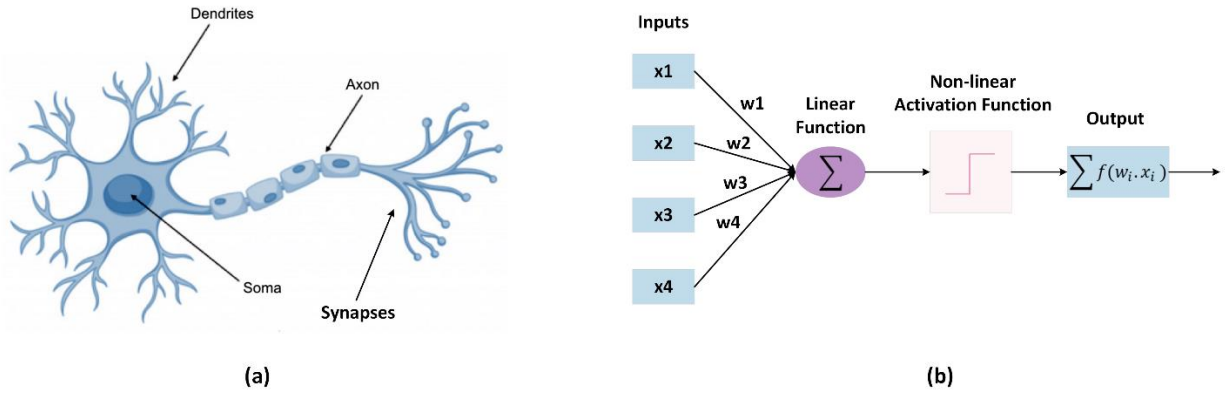


Figure 1: (a) A human brain neuron; (b) An artificial neuron.

Many enabling drivers are contributing to the success of deep learning. In general, AI relies on big data as a fundamental requirement, and it serves as a central element that advances the ability of AI to enhance recognition rates and precision [1]. With the vast development and the wide application of the Internet of Things (IoT), the amount and dimensionality of data available have significantly increased [2]. It is also important to note that such data can differ greatly in terms of type and structure. As deep learning strives to learn intricate characteristics from different data, the quantity and quality of the available data are empirical to the efficient operation of DNNs. Accordingly, with such elevated availability and amount of data, deep learning has been able to reach exceptional milestones. Moreover, various data generation, sampling, and pre-processing techniques have significantly improved the quality of data used for training DNNs [8].

Secondly, the evolution of various deep learning algorithms and techniques is a key factor in the success of deep learning. For instance, the training algorithm used plays a critical role in the performance of DNNs. The training process involves iteratively updating the parameters of the model to minimize the loss function, which measures the difference between the model's predictions and the actual target values. Several training algorithms and optimization techniques have been developed to improve the accuracy and efficiency of the training. For example, stochastic gradient descent (SGD) is the most used optimization algorithm in deep learning, as it updates the parameters of the model based on the gradient of the loss function calculated on a subset of the training data. As a result, SGD significantly improves the computational efficiency of DNNs, especially with large datasets. Other training and optimization techniques that have

enabled the triumph of deep learning include adaptive learning rate methods, batch normalization, and learning rate scheduling. [9]

A third and crucial driver for deep learning is the underlying hardware running the algorithm. The choice and efficiency of the hardware depend on the specific deep learning technique used as well as the associated data. Accordingly, requirements can differ greatly in terms of memory and computational capabilities. In the early days of deep learning, the computational demands of training DNNs were a major bottleneck. Training a single model could take days, weeks, or even months, and requires specialized hardware such as graphics processing units (GPUs). However, as the available hardware evolved, so did the ability to train and process more complex models on larger datasets and in shorter amounts of time. The development of specialized hardware such as tensor processing units (TPUs) and field-programmable gate arrays (FPGAs) has further accelerated the speed of training and inference, enabling deep learning models to be trained and deployed in real time. Furthermore, as the complexity and computational requirements of deep learning algorithms keep increasing, the demand for further advancements in DNN hardware platforms is increasing as well.

A wide range of fields has already benefitted substantially from AI and deep learning, including the automotive industry [10], healthcare [11], indoor localization [12], and network security [13]. For instance, an example of the phenomenal advancements in the automotive industry can be observed in autonomous vehicles. Autonomous driving involves the use of various sensors, including lidar, to gather data on road conditions. Furthermore, cameras are used to input images such as street signs, other cars, and pedestrians. By utilizing advanced deep learning algorithms, this data is analyzed and used to optimize the best route and control plan for vehicles on the road [10]. The healthcare industry also witnessed significant progress since the

emergence of AI. Deep learning algorithms are used to provide medical assistance, detect cancer, and develop new drugs [11].

1.1.2 ARTIFICIAL NEURAL NETWORKS

ANNs are computational models utilized in ML and have surpassed the performance of conventional regression and statistical models [14]. As mentioned, they are inspired by the structure and function of the human brain. The primary processing unit in the brain is the biological neuron (Figure 1(a)), composed of four main parts: the soma, dendrites, axon, and synapse. The soma, or cell body, contains the nucleus and other components necessary for cell function. Chemical processing occurs within the soma. Dendrites, which extend from the neuron soma, receive input signals into the neuron. The axon is the tail of the neuron and transmits signals away from the soma. Axons can split into branches, creating a vast network of interconnectivity between neurons, and this network can contain up to 100,000 fan-out connections, a number that cannot be achieved with CMOS logic gates. Finally, neurons connect through synapses, which are contact points located at their extremities. Neural signals are sent between neurons via electrical impulses along these connections formed by dendrites, axons, and synapses. This phenomenon is known as synaptic plasticity and is thought to play a critical role in encoding memories in the brain [15].

On the other hand, ANNs attempt to mirror the biological neurons and their operations by having layers of interconnected nodes, or artificial neurons (Figure 1(b)), that process information through complex mathematical functions [16]. A basic neural network architecture consists of three layers: an input layer, a hidden layer, and an output layer (as shown in Figure 2(a)). The input layer receives data from an external source, the hidden layer performs computations, and the output layer provides the results of the neural network. Non-linear

activation functions such as rectified linear unit (ReLU), sigmoid, or tanh are also used to learn very complex non-linear relationships between input features. This consequently led to the success of ANNs and their ability to learn from data and generalize to new situations.

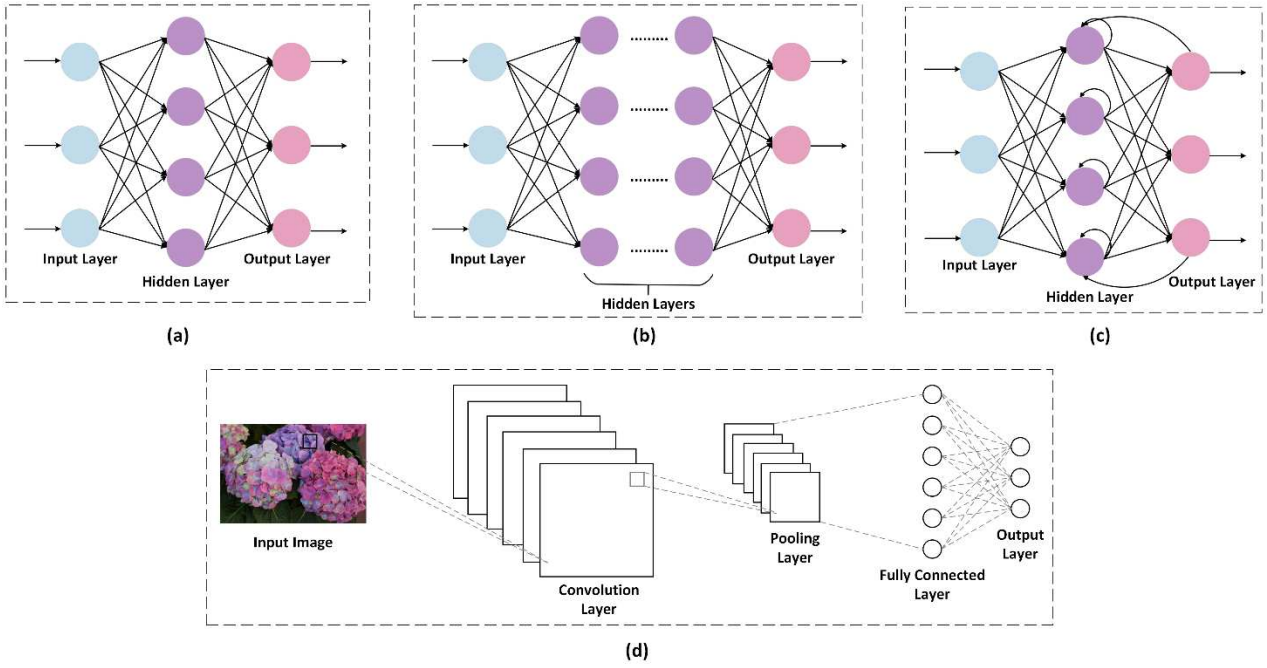


Figure 2: (a) A shallow FF ANN architecture with one hidden layer; (b) A FF DNN architecture with multiple hidden layers; (c) A fully connected RNN, which shows the feedback connections in its hidden layer, and simulates memory or saved states in this architecture; (d) CNN architecture with its various layers: input, convolution, pooling, fully connected, and output.

DNNs are a type of ANN with multiple layers of neurons (hidden layers), enabling them to learn more complex representations of data. DNNs have revolutionized the field of AI, achieving state-of-the-art performance in numerous tasks. Their success is attributed to their ability to learn hierarchical representations of data, which capture increasingly abstract features at each layer of the network. This makes DNNs highly effective at learning patterns and relationships in large datasets, leading to improved accuracy in prediction and classification tasks [17]. Their

application has led to breakthroughs in numerous domains, such as image recognition, where DNNs have been used to achieve unprecedented accuracy on large datasets such as ImageNet [18]. Moreover, DNNs have been used in NLP, where they have been applied to tasks such as machine translation [19], language modeling [20], and speech recognition [16].

1.1.2.1 CONVENTIONAL ANN MODELS

There are several types and models of ANNs that have been developed, each with its own unique architecture and characteristics. Each model usually targets a specific task or handles certain types of data. The simplest type of ANNs is Feedforward (FF) neural networks [21], which are shown in Figures 2(a) and 2(b). They consist of an input layer, one or more hidden layers, and an output layer. They are often used for tasks such as classification and regression. Convolutional neural networks (CNNs) are another widely used prominent ANN model, and their structure is illustrated in Figure 2(d). They have allowed machines to recognize patterns and features in images with unprecedented accuracy. CNNs use a hierarchical structure to extract complex representations of image features, enabling them to recognize objects, faces, and other complex visual patterns. They employ convolution by sliding a small filter (also known as a kernel) over the input data, one position at a time, and computing the dot-product between the filter and the overlapping patch of input data. This produces a single output value, which is placed in a new grid-like structure known as a feature map. These networks have found applications in diverse fields such as healthcare, autonomous driving, and robotics, and continue to drive progress in the field of deep learning. Another famous ANN model is recurrent neural networks (RNNs) [22]. An internal state or memory is included in its hidden layers, as shown in

Figure 2(c). This allows them to have feedback connections, enabling them to process sequences of data such as time series or natural language sentences.

1.1.2.2 TRANSFORMER NEURAL NETWORKS

While all the ANN models discussed have effectively proven their efficiency and built the foundation for various technologies and applications, they can still be inefficient when dealing with very complex tasks and irregular data structures. Over the past few years, more advanced techniques and ANN models have emerged. For instance, the transformer model [23], has garnered a lot of attention from both, research and industry. Not only did it surpass the performance of traditional networks such as RNNs and CNNs, but it also elevated the progress in tasks such as sequence learning and NLP. Moreover, the transformer model structure is actively being inaugurated into vision tasks by employing vision transformer models (ViT) [24]. The success of transformer and transformer-based models can be attributed to their attention mechanism employed. The attention mechanism is a neural network component that allows models to selectively focus on different parts of the input during the learning process [25]. This allows the transformer model to be more interpretable by highlighting the most important features of the input data. The original structure of a transformer model is composed of an encoder and a decoder. The encoder is composed of a self-attention layer, followed by a FF layer, with add and normalization layers, and residual connections in between. This allows the encoder to convert the input sequence into a continuous, abstract representation. The decoder then takes that representation and generates a single output while also receiving the previous outputs as input. The composition of the decoder includes the same layers used in the encoder, but with other self-attention layers that help the decoder focus on relevant parts of the input sentence. Encoder and decoder blocks are usually cascaded a certain number of times.

The emergence of transformer neural networks paved the way for various advanced AI applications. The generative pre-trained transformer (GPT) family of models developed by OpenAI represents one of the most significant advancements in NLP in recent years. These models are based on the transformer architecture and have achieved state-of-the-art performance on a wide range of NLP tasks, including language modeling, text completion, and dialogue generation [29].

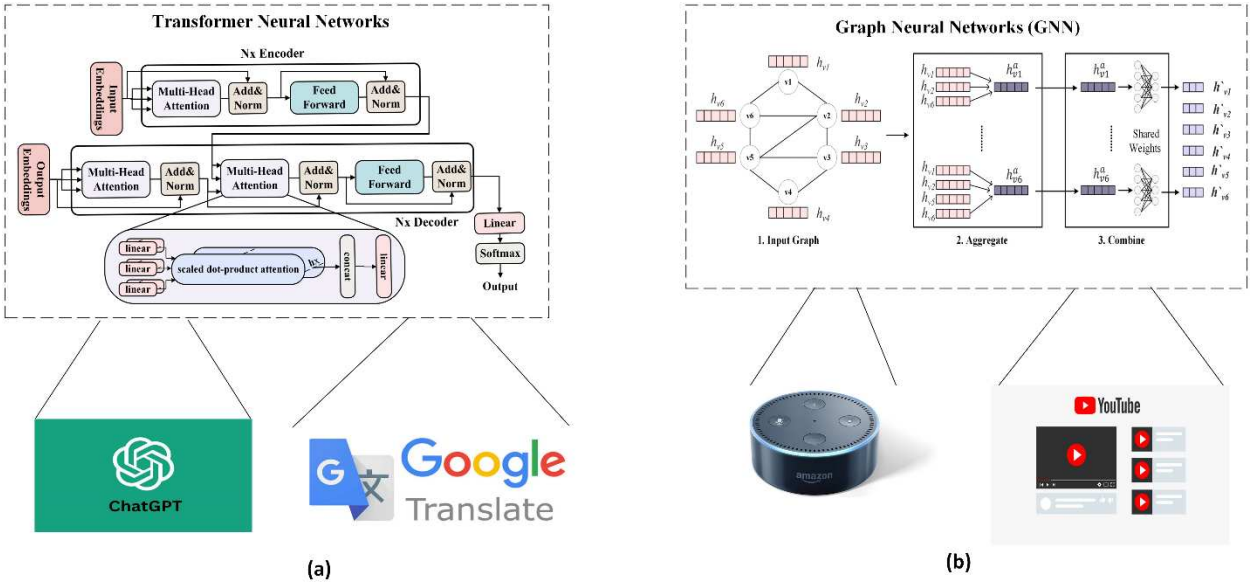


Figure 3: Advanced ANN structures and their prominent applications for (a) Transformer Neural Network used in ChatGPT and Google Translate; (b) GNN used in Amazon Alexa and YouTube recommendation systems.

The first iteration of the GPT model, GPT-1, was introduced in 2018 with 117 million parameters and demonstrated impressive results on language modeling and text completion tasks. Subsequent versions, including GPT-2 and GPT-3, increased the number of parameters to 1.5 billion and 175 billion, respectively, resulting in even more impressive performance [30]. GPT-3, specifically, has been hailed as a major milestone in AI research due to its ability to perform a

wide range of language tasks with high accuracy and fluency. It has demonstrated remarkable capabilities in areas such as language translation, question-answering, and creative writing. The model's ability to generate coherent, contextually appropriate responses to prompts has led to speculation about its potential impact on areas such as chatbots, and virtual assistants [31]

1.1.2.3 GRAPH NEURAL NETWORKS

Graph neural networks (GNNs) are another type of cutting-edge neural networks that have recently transformed the field of AI [23]. They introduced neural networks into the graph processing realm, allowing graph-structured data to be efficiently modeled and learned from. This consequently benefitted a myriad of fields and applications such as recommendation systems, social network analysis, drug discovery, and robotics. A graph is typically represented as a set of matrices describing the features for each node and edge as well as the connections. A GNN is used in one of three main task categories: node-level task, graph-level task, or edge-level task. While several GNN models have recently emerged such as graph convolution neural networks (GCNs) [27] and graph attention networks (GATs) [28], they all follow a similar structure. They are usually composed of several GNN layers, and each layer contains various operations and steps to gather each node and edge’s information, transform the feature data gathered and finally combine and update the graph data.

Furthermore, GNNs are widely used in the field of recommendation systems. They have been shown to be effective in modeling various relationships, allowing for more accurate and personalized recommendations [32]. One notable example of a successful application of GNNs in recommendation systems is the Amazon Alexa personal assistant [33]. Amazon Alexa uses GNNs to model the relationships between users, devices, and contextual features such as location

and time, allowing it to provide personalized recommendations for music, news, and other content. GNNs have also been used in other recommendation systems, such as YouTube's recommendation algorithm, which uses GNNs to model the relationships between videos, users, and other features such as metadata and viewing history [34]. In conclusion, GNNs have become an important tool for building more effective and personalized recommendation systems, and their use in applications such as Amazon Alexa, YouTube, and a lot of other applications, is a testament to their power and versatility.

In conclusion, the structures of both transformers and GNNs are much more complex and advanced than conventional ANNs, which enabled them to excel in numerous tasks and outperform many ANN models. The architecture, different models, and operations of transformers and GNNs will be further discussed in detail in chapters 2 and 3.

1.2 CHALLENGES IN AI ACCELERATION

Due to the continuous advancements in deep learning, and the current vast availability of diverse data types, accelerating ANN training and inference processes has become a complex problem. The ANN training phase typically requires much longer durations of time than ANN inference, and they are usually done only once offline or once every specified period. On the other hand, when the training phase is complete, ANNs and deep learning algorithms are now being deployed on resource-constrained IoT, edge, and embedded platforms. Accordingly, there is an increasing demand for solutions for energy efficient and low power ANN inference acceleration. Effort in accelerating ANNs is done either on the software side or the hardware side. While we employ various software optimization techniques in our accelerator hardware architectures for transformer and GNN models, this thesis' primary focus is on hardware

acceleration. Moreover, it is important to note that this thesis tackles the challenge of AI inference acceleration only and not accelerating the training process.

The next sections outline the current effort being conducted in hardware ANN inference acceleration, the limitations in existing hardware platforms pertaining to the setback in providing sufficient support to run AI applications, and finally we discuss the novel technologies that have recently emerged and are currently being integrated into various AI hardware accelerators.

1.2.1 HARDWARE AI ACCELERATION

A hardware platform running an AI application must be able to meet several computational, memory, latency, and energy requirements, all while abiding by strict power constraints. Accordingly, Hardware acceleration of ANN inference has become an important area of research due to the increasing demand for efficient and high-performance computation in ML applications. Currently, GPUs are considered the most popular computing platform for ANN inference due to their highly parallel architecture and efficient memory bandwidth utilization. Compared to CPUs, GPUs have a significantly larger number of cores, enabling them to perform massively parallel operations on multiple data points. This parallelism is a key requirement for many ML applications, where large datasets and models must be processed in a timely manner. Additionally, GPUs have an optimized memory hierarchy, with high-bandwidth memory and cache systems, allowing for efficient data transfer between the processor and memory [35].

However, despite the benefits general-purpose processors such as CPUs and GPUs offer, they are no longer able to keep up with the deep learning advancements and more attention has been recently given to application-specific AI hardware accelerators. Application-specific accelerators are specialized hardware devices that are designed and optimized for specific

applications or workloads. In the context of ANN inference acceleration, application-specific accelerators are designed to accelerate the computations required for ANN processing, such as matrix multiplication, convolution, and pooling operations. There are several types of application-specific accelerators for ANN inference acceleration, including FPGAs, application-specific integrated circuits (ASICs), and digital signal processors (DSPs). These accelerators offer higher performance and energy efficiency when compared to general-purpose processors like CPUs and GPUs, as they are specifically designed for the target workload. They can also reduce the complexity of the required software programs and optimizations since a hardware-software codesign approach is usually adopted when designing an application-specific hardware accelerator, leading to reduced overheads and improved performance [36].

TPUs are one famous example of an ASIC designed specifically for accelerating ML workloads. They are designed by google to optimize the execution of matrix multiplication operations, which are a fundamental building block of ANN models. TPUs offer high performance and energy efficiency compared to traditional CPUs and GPUs, due to their ability to perform computations using reduced precision arithmetic. This approach allows TPUs to process data more quickly while using less power. In addition, TPUs can be used in clusters to scale performance and handle larger workloads, making them a good fit for large-scale distributed systems [37].

1.2.2 NOVEL TECHNOLOGIES FOR AI ACCELERATION

All the electronic platforms discussed in section 1.2.1 including the general-purpose processors such as CPU and GPU, as well as the application-specific hardware accelerators using FPGAs, ASICs, and DSPs, have enabled various advancements in AI and ML applications.

Nevertheless, their limitations are becoming more evident during these past few years. Despite their significant capabilities, relying solely on electronic AI accelerators may prove inadequate in meeting the ever-increasing demands of DNN inference. This is due to their susceptibility to the limits of the post-Moore's law era, where diminishing performance improvements are being observed as the size of transistors approaches the atomic scale [38]. To overcome these limitations, researchers are exploring alternative computing technologies, such as processing in memory (PIM), resistive RAM (ReRAM)-based accelerators, and silicon photonics.

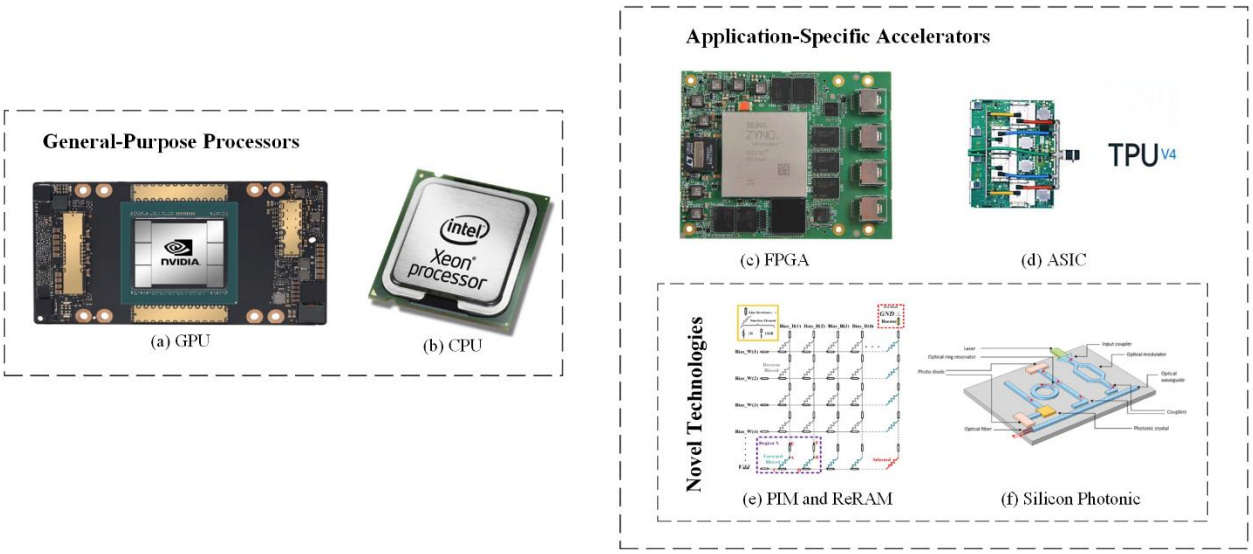


Figure 4: AI Hardware Accelerators categorized as general-purpose processors: (a) GPU; (b) CPU; (c) TPU, electronic-based application-specific accelerators: (d) FPGA; (e) ASIC, and application-specific accelerators based on novel technologies: (f) PIM and ReRAM based; (g) Silicon Photonic based.

One way through which researchers are trying to meet the extensive computational load presented by the rapidly growing ML and AI algorithms is through exploring alternative non-Von Neumann computer architectures such as in-memory and near-memory computing. The concept of in-memory computing (IMC) or PIM involves integrating processing units into the

memory chip to eliminate the separation between memory and processor [39]. Over the last decade, a significant amount of research has focused on developing machines for DNN inference and training using various memory platforms. These include both traditional memory platforms like static and dynamic random access memory (SRAM & DRAM), as well as emerging non-volatile technologies like ReRAM, phase-changing memory (PCM), and magnetic RAMs like spin transfer torque MRAM (STT-MRAM) and spin orbit torque MRAM (SOT-MRAM). Despite the usual need for quantization and down-scaling of data parameters in DNN algorithms, studies have found that a satisfactory level of accuracy can still be achieved [40]. ReRAM-based hardware accelerators in specific, have been gaining a lot of attention recently and are being leveraged in some research work targeting a diverse set of DNNs such as CNNs [41], RNNs [42], transformers [43], and GNNs [44]. Such accelerators usually leverage the crossbar array computing architecture shown in Figure 4(e) which can perform massively parallel analog multiply and accumulate (MAC) operations by leveraging a 2D cell array of analog resistive cells [40].

Another prominent novel technology that is being integrated into AI and ML accelerator designs is silicon photonics. In general, optical communication is widely used in communication networks for long-distance, low-power consumption, and high-bandwidth communication. Silicon photonics, which uses light to transmit data on a CMOS chip, has become popular in commercial offerings for low-cost optical interconnects in data centers. It is now also being considered for connecting multiple processing chips at the board level and even connecting cores within a single computing chip. Wavelength-division multiplexing (WDM) is used to carry multiple wavelengths of light simultaneously, each capable of carrying data at high speed and without interference. Dense wavelength-division multiplexing (DWDM) can increase the WDM

degree to 64 or beyond. To transmit digital data from electronic components over a waveguide with multiple carrier wavelengths, electronic to optical conversion devices like microring resonator (MR) modulators are used. Similarly, optical-to-electronic conversion devices like photodetectors (PDs) are used for receiving the optical signal. Accordingly, silicon-photonics is successfully able to overcome the bandwidth and power bottlenecks of conventional electronic AI computing platforms for acceleration ML algorithms and load [7]. How silicon photonics is used for ANN acceleration is thoroughly discussed in detail in section 1.3.

1.2.3 LIMITATIONS IN EXISTING AI HARDWARE ACCELERATION PLATFORMS

The electronic platforms along with the novel technologies used for accelerating ANNs, discussed in sections 1.2.1 and 1.2.2, are the foundation for all current AI applications and progress witnessed in the past years. Nevertheless, each computing platform is equipped with its limitations and challenges. Such limitations are becoming more evident, especially with the ever-growing vast advancements in ANN algorithms. For instance, while GPUs can significantly outperform CPUs, they still have several limitations when used for ANN inference acceleration. One of the main limitations of GPU-based ANN acceleration is the memory bandwidth bottleneck. Since ANNs require large amounts of data to be processed simultaneously, GPUs can become memory-bound, where the data transfer rate between the memory and processing units becomes a limiting factor. This can result in a slowdown of ANN performance [45]. Moreover, although GPUs are highly parallel devices, the parallelism of ANN computations is limited by the dependencies between different layers and neurons and is very dependent on the specific ANN structure. This can lead to the underutilization of GPU resources and reduced performance [46]. While electronic application-specific hardware accelerators alleviate some of GPUs'

limitations since they are usually tailored for specific ANN architectures, they are still bound by several constraints. As the dimensions of transistors decrease to the atomic scale, the reduction in size does not lead to a corresponding reduction in energy consumption since the voltage required to drive these transistors has not decreased proportionally. This results in a higher power density, lower energy efficiency, and reduced performance improvements. Moreover, the reduced performance of electronic systems is compounded by bottlenecks in data movement due to the slow interconnects on metallic chips [47].

Novel technologies that have recently emerged and are currently being integrated into ANN hardware accelerators such as ReRAM cells and silicon-photonics, as discussed, also face their own unique challenges. First, operation in the analog domain brings several challenges such as noise, non-zero wire resistance, nonlinear I-V characteristics, and I/O stage resistance. ReRAM-based accelerators in specific, face several reliability challenges due to their high defect rate, limited write endurance, resistance drift, and susceptibility to process variation. Moreover, ReRAM cells have high write energy/latency which leads to increasing the overall power consumption [48]. Using silicon photonics also requires overcoming some challenges, given the novelty of leveraging optical computation in AI hardware accelerators. For example, silicon-photonic devices are susceptible to inevitable variations in the fabrication process as well as runtime temperature fluctuations. Signal loss and crosstalk reliability are also some of the main challenges when designing silicon-photonics-based accelerators [47]. Nevertheless, various efforts, including the work presented in this thesis, are proposing several ways to mitigate such challenges related to using silicon photonics, as will be discussed later.

1.3 SILICON PHOTONICS FOR AI ACCELERATION

1.3.1 MOTIVATION FOR USING SILICON PHOTONICS

As the demands of AI applications leveraging DNNs keep escalating and becoming more complex, hardware architectures must scale up in terms of processing capabilities, memory capacity, and communication capabilities. As discussed in section 1.2.3, current computing platforms which are typically being used for DNN acceleration, such as GPUs, are becoming less efficient and unable to match today's stringent DNN acceleration requirements. Moreover, due to their susceptibility to the limits of the post-Moore's law era, relying solely on electronic AI accelerators may prove inadequate in meeting the ever-increasing demands of DNN inference, and technologies such as ReRAM's various persistent challenges as discussed.

On the other hand, silicon photonics has the capacity to bring about significant changes in the field of AI by offering rapid data transfers and processing, while using considerably less energy than conventional electronic computing. Optical computing systems are analog in nature and can perform complex operations with low energy consumption and very high speeds. Moreover, silicon photonics allows for the integration of optical and electronic components on a single chip, enabling the development of highly integrated and compact systems. This is particularly important for AI applications that require large-scale matrix multiplication, convolution, and other common AI operations. The ability to integrate both optical and electronic components on a single chip reduces the distance that data should travel, minimizing the power consumption and latency associated with data movement. Another advantage of silicon photonics is its ability to perform complex operations, such as MAC, summation, and non-linear activation functions with low energy consumption and very high speeds.

In addition to various research work presented for accelerating specific types of ANNs using optical computations [47], several companies in the industry have already started integrating silicon photonics in their AI hardware accelerators. Lightmatter [49], for example, works on several hardware devices for AI acceleration. In 2020, they presented their device, MARS, which is designed to accelerate AI workloads. The MARS photonics core supports 8k operations per cycle, 8-bit signed operands, and a 1 GHz vector rate. The photonics core has an overall area of only 150mm^2 [50]. In addition to Lightmatter, several other companies have also already drifted towards leveraging silicon-photonics for AI acceleration such as Intel [51], Ayar Labs [52], and Hewlett Packard Enterprise (HPE) [53].

1.3.2 SILICON PHOTONIC HARDWARE AI ACCELERATORS

AI hardware accelerator architectures leveraging silicon photonics can be broadly categorized into coherent and non-coherent AI accelerators. Coherent photonic AI accelerators use Mach-Zehnder interferometers (MZIs) and a single wavelength to manipulate the phase of an optical signal for parameter imprinting and multiplication. Previous research has used cascaded MZIs arranged in a mesh to achieve large-scale and high-throughput matrix operations. There is growing interest in using coherent architectures to reduce the computational complexity of DNN acceleration at the server level. However, to achieve this, the entire DNN model must be deployed, which leads to some disadvantages such as large MZI mesh sizes, increased error propagation, and large-area requirements [52].

On the other hand, non-coherent photonic AI accelerators use multiple wavelengths simultaneously to perform several matrix-vector operations concurrently. To imprint parameters, these accelerators use MRs with wavelength-selective filtering and modulation capabilities. MRs

perform amplitude modulation on different wavelengths using a tuning circuit to imprint the parameters. Multiplication of two parameters is accomplished by deploying them on MRs with the same resonant wavelength, where the first MR modulates the signal to represent the first parameter and the parameter-specific amplitude modulation with the second MR results in the multiplication operation. This technique, known as broadcast and weight, allows non-coherent AI accelerator architectures to achieve vector-based operations, and it can be combined with careful dataflow orchestration to achieve matrix-level operations. This thesis focuses on non-coherent architectures and implementing such an accelerator architecture requires various devices and circuits that are discussed in the following section.

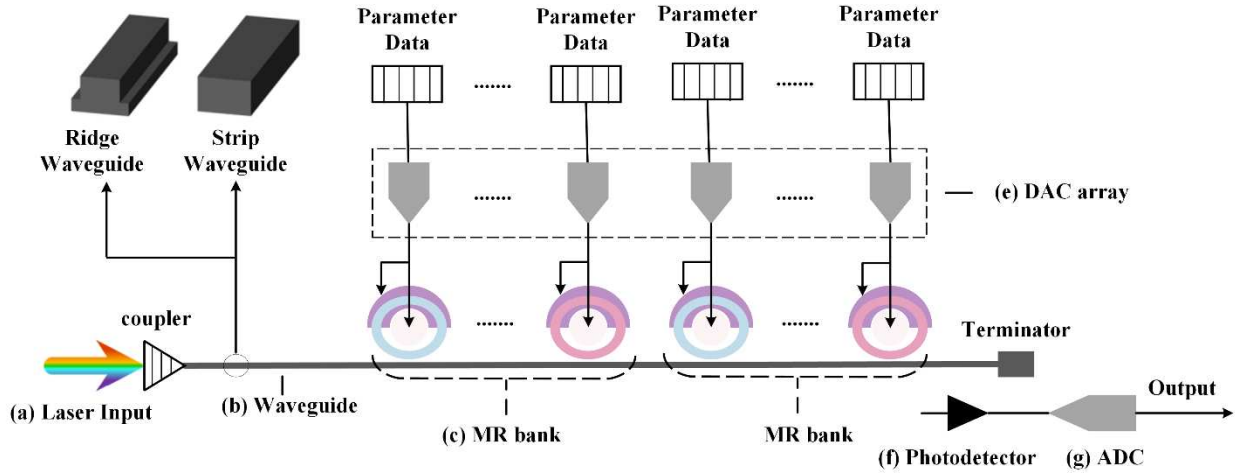


Fig. 5: Overview of non-coherent photonic circuit to implement broadcast-and-weight-based multiplication and accumulation for AI acceleration. This circuit is composed of (a) a laser source, which can be off-chip or on-chip; (b) a waveguide, which can be strip (for passive devices) or ridge (for active devices); (c) microring resonator (MR) banks perform MAC operation; (d) the banks are tuned as per data from the electronic domain, using digital-to-analog converters (DACs); (e) the result from the MAC operation is detected and accumulated using a photodetector; (f) for converting the data to digital domain, for postprocessing or storage, an analog-to-digital converter (ADC) can be used.

1.3.3 SILICON PHOTONIC DEVICES AND CIRCUITS

Figure 5 presents a general overview of the fundamental optical circuit and devices used to perform MAC operations in non-coherent AI accelerators. The following are the main components needed:

- *Lasers:* performing computation and communication in optical circuits, requires using lasers to create optical signals. These lasers can either be on-chip or off-chip. While off-chip lasers have better light emission efficiency, there are significant losses when coupling the optical signals onto on-chip waveguides. Conversely, on-chip lasers, such as vertical cavity surface emission lasers (VCSELs), offer a higher level of integration density and lower losses. There are two types of on-chip lasers: directly modulated lasers that can produce modulated signals and VCSEL arrays that provide unmodulated signals that can be modulated using MZIs or MRs.
- *Waveguides:* silicon photonic waveguides are typically composed of two materials, a high-refractive-index core made of silicon (Si) and a cladding made of silicon dioxide (SiO2). This material combination creates a high-refractive-index contrast, which allows for total internal reflection. The waveguides can be either ridge or strip in shape. WDM allows a single waveguide to support multiple wavelengths simultaneously without any interference. This enables the transmission of ultra-high bandwidth signals.
- *Microring Resonators (MRs):* An MR add-drop filter is an optical modulator that is designed using a ring-shaped waveguide that is fabricated near both an input waveguide and a drop waveguide, resulting in what is known as the add-drop configuration. MRs can be designed to be sensitive to a specific wavelength, which

is called the MR resonant wavelength. MR banks consist of groups of MRs that share a single input waveguide and can be utilized for performing MAC and summation operations in non-coherent architectures.

- *Photodetectors (PD):* detecting processed optical signals and converting them into electrical signals is done using PDs. To be effective, a PD should be able to generate the desired electrical output using a small input optical signal. The input signal power must be greater than the responsivity of the PD. The design of the optical link must consider different types of losses that may occur to ensure that the appropriate optical power is delivered.
- *Tuning circuits:* Tuning circuits are devices designed to control the effective index (n_{eff}) of MR devices to precisely modify an output optical signal. MRs can utilize either electro-optic (EO) or thermo-optic (TO) tuning mechanisms. The TO mechanism involves resistive heat pads placed over a waveguide to generate heat, which then alters n_{eff}. The EO mechanism utilizes doped sections of a waveguide in combination with a biasing circuit to introduce carrier injection, which also modifies n_{eff}.

1.3.4 CHALLENGES IN SILICON PHOTONIC HARDWARE AI ACCELERATORS

Realizing the full potential of silicon-photonic-based AI accelerators necessitates overcoming several challenges. Given the novelty of leveraging silicon photonics for AI computations and acceleration, some barriers still need to be addressed. For example, signal loss and crosstalk noise pose a major challenge when designing AI accelerators. There are several types of crosstalk such as thermal crosstalk and heterodyne and homodyne crosstalk. The

stability of photonic components and energy consumption in non-coherent AI accelerators are affected by thermal crosstalk. TO tuning, which provides significant tuning ranges, can be used to correct phase perturbations caused by FPV or time-varying phase changes due to thermal fluctuations from nearby electric circuitry in photonic components. However, mutual thermal crosstalk can occur in TO tuning circuits, which decreases the stability of MRs and increases energy consumption. Heterodyne and homodyne crosstalk are also both significant in non-coherent architectures that utilize multiple wavelengths at the same time through WDM and may have several devices that are sensitive to the same MR resonant wavelength (λ_{MR}) along the same waveguide [53]. Another main challenge faced when designing silicon-photonic AI hardware accelerators is being able to scale and leverage the silicon-photonic devices and circuits efficiently to be used for specific types of ANNs. This is a non-trivial task especially with complex and advanced ANNs, such as transformers and GNNs, since each neural network has its own unique set of sequential, parallel, dense, and sparse operations and layers, and types of data. The next chapters outline how we were able to overcome such challenges when designing our hardware accelerators: *TRON* and *GHOST*.

1.4 THESIS OVERVIEW

In summary, there is a crucial need for robust, reliable, and advanced AI hardware accelerators, capable of efficiently handling advanced ANNs. Advancements on the software side, including ANN algorithms are moving at a faster pace than advancements on the hardware side. Moreover, current computing platforms are no longer able to efficiently accommodate ANNs workload inference requirements, while abiding by strict power constraints. Electronic-based hardware AI accelerators, including general-purpose processors and application-specific

accelerators, have proven to be inefficient for handling various advanced ANNs workloads and algorithms. On the other hand, silicon photonics offers a promising solution for integrating optical computation and communication within AI hardware accelerators. Nevertheless, leveraging silicon photonics presents several challenges that need to be addressed to be able to acquire a reliable and efficient silicon-photonic-based hardware inference accelerator for ANNs. To address these problems, the main contribution of this thesis is the design of two novel silicon-photonic-based hardware accelerators, *TRON* and *GHOST*, for accelerating transformer neural networks and GNNs, respectively. The rest of the thesis is organized as follows:

In chapter 2, we present *TRON*, the first silicon-photonic-based hardware accelerator for transformer and transformer-based neural networks. The implementation of each layer, as well as most of the operations and functions needed for processing transformer models in the optical domain, are thoroughly explained. We demonstrate how our accelerator architecture is able to efficiently accommodate a diverse set of transformer and transformer-based models while outperforming several computing platforms and state-of-the-art transformer hardware accelerators.

In chapter 3, *GHOST* is presented, which is the first hardware accelerator for GNNs inference, leveraging silicon photonics. We address the various challenges associated with processing GNNs due to their combination of dense and very sparse operations along with the wide range of possible GNN algorithms and graph datasets. We outline the several device, circuit, and architecture level optimizations implemented to achieve efficient GNNs acceleration, outperforming several computing platforms and state-of-the-art hardware accelerators.

Chapter 4 concludes this thesis. We summarize our comprehensive body of research in this chapter and also make recommendations for future work.

1. TRON: TRANSFORMER NEURAL NETWORK ACCELERATION WITH NON-COHERENT SILICON PHOTONICS

Transformer neural networks are rapidly being integrated into state-of-the-art solutions for NLP and computer vision. However, the complex structure of these models creates challenges for accelerating their execution on conventional electronic platforms. We propose the first silicon photonic hardware neural network accelerator called TRON for transformer-based models such as BERT, and Vision Transformers. Our analysis demonstrates that TRON exhibits at least 14× better throughput and 8× better energy efficiency, in comparison to state-of-the-art transformer accelerators.

1.2 MOTIVATION AND CONTRIBUTION

Transformer neural networks have gained significant popularity in the last few years, surpassing the performance of traditional RNNs and CNNs [54]. As the network architecture in transformer models relies on attention mechanisms and positional encodings instead of recurrence, it enables much higher parallelization than RNNs for sequence modeling and transduction problems. Since the introduction of the first transformer in 2017 [23], considerable progress has been made, with the emergence of powerful transformer-based pre-trained NLP models, such as BERT [56] and Albert [57], and computer vision models, such as the Vision Transformer [24]. Despite the remarkable success of the transformer model, its size, number of parameters, and operations still require significant computational resources, hindering its progress and usage in resource-constrained systems. This highlights the main issues with these models, which includes long inference times, large memory footprint, and low computation-to-memory ratio. Existing work on inference acceleration of conventional ANNs, mainly focuses on

compute-intensive operations and optimizations at the layer-level granularity, which makes extending it to transformers—with its unique layer architecture and memory-intensive requirements—challenging. Several transformer-centric accelerators have been proposed in recent years to overcome these challenges with transformer execution [43], [56]-[60]. However, most of the work presented so far either focuses on accelerating a specific transformer architecture or is based on electronic components. Electronic accelerators are susceptible to the limits of the post Moore's law era, where diminishing performance improvements are being observed with technology scaling. Such limitations also present major performance and energy bottlenecks for electronic dataflows [38]. On the other hand, silicon photonics has proven its proficiency as a solution beyond high-throughput communication in the telecom and datacom domains, and it is now being considered for chip-scale communication. Moreover, CMOS-compatible silicon photonic components can be used for computations, such as matrix-vector multiplications and logic gate implementations. Accordingly, the integration of silicon photonics is now actively being considered for deep learning acceleration [7]. In this chapter, we introduce TRON, the first silicon-photonic based transformer accelerator that can accelerate inference of a broad family of transformer models. Our novel contributions are:

- The design of a novel transformer accelerator using noncoherent silicon photonics, with the ability to accelerate any existing variant of transformer neural network models,
- Detailed crosstalk analyses, to improve signal-to-noise ratio (SNR) and tunability for photonic microresonator (MR) banks,
- A comprehensive comparison with GPU, TPU, CPU, and stateof-the-art transformer accelerators.

The rest of the chapter is organized as follows. Section 2.2 presents a background on transformers, ANNs, and their acceleration using silicon photonics. Section 2.3 describes our TRON architecture. Section 2.4 discusses the experimental setup and comparisons with other accelerators, followed by conclusions in Section 2.5.

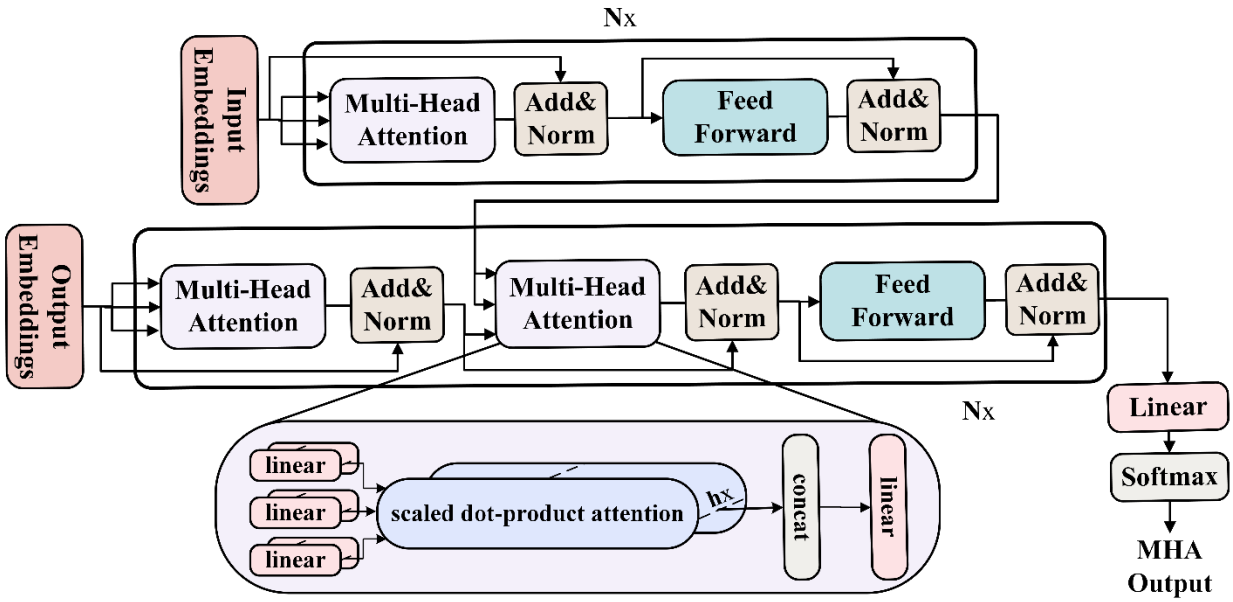


Figure 6: Transformer neural network architecture overview.

2.2 BACKGROUND

2.2.1 TRANSFORMER NEURAL NETWORK MODEL

The attention mechanism has emerged as a prominent technique in sequence learning and NLP, where long-term memory is required. By utilizing the attention mechanism, transformers have outperformed RNNs (LSTMs, GRUs) across many NLP tasks. As shown in Figure 6, the original transformer model [23] designed for sequence learning has two main blocks: encoder and decoder. The encoder is responsible for mapping the input sequence into an abstract

continuous representation. The decoder then processes that representation and gradually produces a single output while also being fed the previous outputs. Before being sent to the encoder, each input sequence is mapped to a vector, and positional encoding is used to embed the position information of each vector in relation to the original input sequence. The processed input is then passed through to the encoder/decoder block.

The encoder and decoder blocks consist of N stacked layers (Figure 6). The main sub-blocks in the encoder and decoder blocks are the multi-head attention (MHA) and feed forward (FF) layer, along with residual connections for each, followed by layer normalization. Self-attention is applied in MHA where it links each element (e.g., word) to other elements (e.g., words) in a sequence. Each MHA has H self-attention heads, and each attention head generates the query (Q), key (K), and value (V) vectors to compute the scaled dot-product attention. Q, K, and V vectors are generated by multiplying the MHA's input sequence X by the query, key, and value weight matrices: WQ, WK, and WV. The self-attention output is then computed through a scaled dot-product operation as follows:

$$Head(X) = attention(Q, K, V) = softmax\left(QK^T/\sqrt{d_K}\right)V, \quad (1)$$

where X is the input matrix and d_k is the dimension of Q and K. The output of the MHA is the concatenation of the self-attention heads' outputs, followed by a linear layer. The FF network is composed of two dense layers with a *RELU* activation in between.

More recent transformer-based pre-trained language models, such as BERT [56] and its variants [57], include the transformer encoder block only, as a cascaded set of N layers, followed by an FF layer, then GELU, and normalization layers. Similarly, the ViT is composed of N

encoder layers, followed by a multi-layer perceptron [24], where the ViT's inputs are sequence vectors representing an image.

2.2.2 TRANSFORMER ACCELERATION

Transformer accelerators to date have focused on accelerating either a specific subset of transformer models or specific transformer layers. For instance, [58] proposed an FPGA-based hardware accelerator, for accelerating MHA and FF layers. Their approach involves efficiently partitioning the weight matrices used in the MHA and FF layers to allow both layers to share hardware resources. In [60], another FPGA-based acceleration framework was proposed with a pruning technique for efficient model compression and an optimized method for storing the sparse matrices. An in-memory processing-based transformer accelerator called TransPIM was presented in [41], with a novel token-based dataflow for optimized data movements along with hardware modifications to the high bandwidth memory. The work in [59] focused on accelerating ViTs and proposed an automated framework, VAQF, that guides the quantization and FPGA resource mapping for a specific ViT. *Unlike prior efforts, TRON supports accelerating a broad family of transformer models for both NLP and computer vision tasks*.

2.2.3 SILICON PHOTONICS FOR ANN ACCELERATION

Due to the significant benefits offered by optical ANN accelerators in terms of performance and energy efficiency, they have garnered a lot of traction from academic and industry researchers [7]. Optical ANN accelerators are either coherent or non-coherent. In coherent architectures, which use a single wavelength, parameters are imprinted onto the optical signal's phase [61] to perform multiply and accumulate (MAC) operations. Alternatively, non-

coherent architectures leverage multiple wavelengths and imprint parameters onto the optical signal's amplitude. Each wavelength can be used to perform operations in parallel. Current research in optical ANN accelerators has focused mainly on CNNs, MLPs, and RNNs [62]. To the best of our knowledge, *TRON* is the first optical accelerator for transformer ANN models.

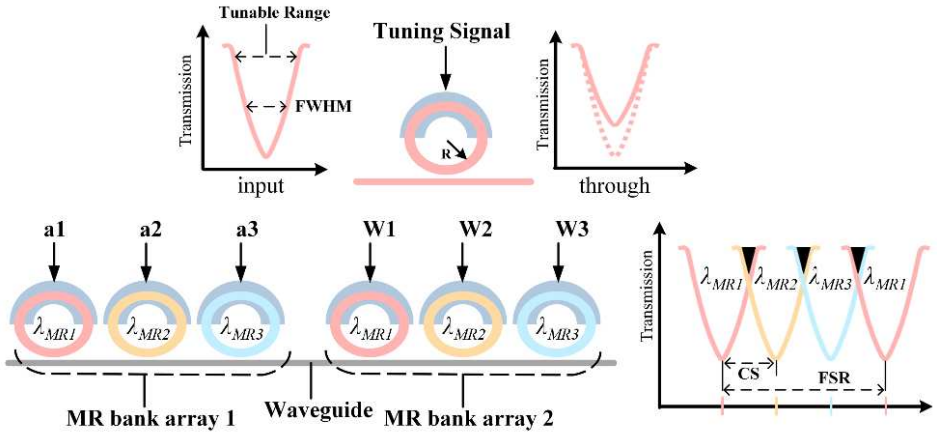


Figure 7: Top MR shows input and through ports' wavelengths after imprinting a parameter onto the signal. Bottom MR bank arrays perform multiplication by imprinting input activations (a_1-a_3), followed by weight vector values (W_1-W_3).

TRON is a non-coherent optical accelerator and uses MRs (see Figure 7) as the main opto-electronic device for carrying out key operations. Each MR can be designed and tuned to work at a specific wavelength, called MR resonant wavelength (λ_{MR}), defined as:

$$\lambda_{MR} = \frac{2\pi R}{m} n_{eff}, \tag{2}$$

where R is the MR radius, m is the order of the resonance, and n_{eff} is the effective index of the device. By carefully altering n_{eff} with a tuning circuit, we can modulate electronic data onto an

optical signal passing by (in the vicinity of) an MR. The tuning circuit used is usually based on either thermo-optic (TO) [63] or carrier injection electro-optic (EO) tuning [63]. Both would result in a change in n_{eff}, and hence a resonant shift of $\Delta\lambda_{MR}$ in the MR. In non-coherent networks, computations and, specifically multiplications, are done by tuning an MR's $\Delta\lambda_{MR}$, resulting in a predictable change in the optical signal's wavelength amplitude.

To increase throughput and mimic neurons in ANNs, non-coherent architectures use multiple wavelengths as part of wavelength-division multiplexing (WDM). This entails having multiple optical signals with different wavelengths in a single waveguide using an optical multiplexer [7]. The waveguide would then pass by a bank of MRs, each tuned to a certain wavelength in the waveguide, enabling us to perform several multiplications in parallel. Figure 7 illustrates an example of multiplying an input vector [a_1, a_2, a_3] by a weight vector [W_1, W_2, W_3]. Two MR bank arrays are used: the first imprints input activations onto the optical signals and the second performs the multiplication. The dot product output can thus be calculated by summing the three signals in the waveguide, which can be done by a photodetector (PD).

2.3 TRON HARDWARE ACCELERATOR

TRON is a non-coherent photonic accelerator architecture that can accelerate the inference of a broad family of transformer models. An overview of the architecture is shown in Figure 8. The photonic accelerator core is composed of MHA and FF units. Such composition allows reuse of resources for the encoder and decoder blocks. Interfacing with the main memory, buffering of the intermediate results, and mapping the matrices to the photonic architecture, are all handled by an integrated electronic-control unit (ECU). The following subsections describe

the *TRON* architecture and the hardware optimizations we have considered to efficiently accelerate transformer ANN models.

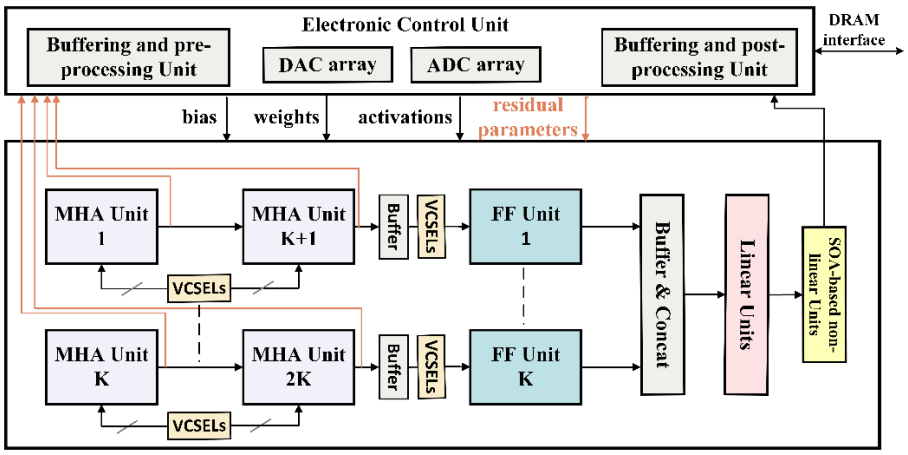


Figure 8: An overview of the proposed *TRON* accelerator architecture.

2.3.1 MR TUNING CIRCUIT

As discussed earlier, MR devices in non-coherent architectures require a tuning mechanism, based on EO or TO. In *TRON*, we employ a hybrid tuning circuit where both TO and EO are used to induce $\Delta\lambda_{MR}$. This enables us to combine the advantages of both while overcoming their disadvantages. EO tuning is faster (≈ns range) and requires less power (≈4 μW/nm), but it cannot be used for large tuning ranges [63]. Conversely, TO tuning accommodates a larger tunability range but at the expense of higher latency (≈μs range) and power (≈27 mW/*FSR*) [63]. Accordingly, in our design, EO tuning is adopted for fast induction of small $\Delta\lambda_{MR}$ in MRs, while slower TO tuning is used only when larger $\Delta\lambda_{MR}$ is required. The effectiveness of this hybrid approach was previously demonstrated in [65]. To further reduce the power overhead of TO tuning, we adopt thermal eigen decomposition method (TED) from [66].

TED entails tuning all MRs within a bank array together, which reduces power consumption. Moreover, the approach uses microheaters to perform thermal tuning which reduces thermal crosstalk noise from heat dissipated from adjoining TO circuits.

2.3.2 MR BANK DESIGN-SPACE ANALYSIS

To ensure error-free MAC operations in the optical domain, it is necessary to manage various sources of noise, namely thermal and crosstalk noise, which can interfere with parameter imprinting and degrade the network performance and accuracy. Our TED-based tuning mechanism alleviates the thermal noise that can arise from TO tuning. But non-coherent architectures, like *TRON*, are inherently noise prone due to multiple wavelengths propagating in the same waveguide which creates inter-channel crosstalk. In inter-channel crosstalk, a portion of the optical signal from neighboring wavelengths can leak into one another, causing signal distortion (see Figure 7; bottom right). This phenomenon is further exacerbated with the presence of multiple MR banks in series, where multiple wavelengths can undesirably drop into an MR. With well-designed channel spacing (CS) and Q-factor in the MR, this can be managed by ensuring that the signal-to-noise ratio (SNR) is better than the detector sensitivity. The design of an MR should ensure adequate Q-factor to improve SNR. Additionally, the MR design should also possess sufficient tunable range, so that necessary parameters can be imprinted free of error. Mathematically, tunable range can be represented as 2×FWHM (full width half maximum) (see Figure 7; top left). We optimize MR design for high FWHM and high SNR. For this optimization, we use the following models from [66]:

$$SNR\ (dB) = 10 \times \log_{10}(P_{signal}/P_{noise}), \tag{3}$$

$$P_{signal} = \Phi(\lambda_i, \lambda_j, Q) P_S(\lambda_i, \lambda_j), \tag{4}$$

$$P_{noise} = \sum_{i=1}^{n} \Phi(\lambda_i, \lambda_j, Q) P_S(\lambda_i, \lambda_j)(i \neq j), \tag{5}$$

where Φ is the crosstalk coefficient corresponding to the inter-channel crosstalk between neighboring channels λ_i and λ_j, which is given by:

$$\Phi(\lambda_i, \lambda_j, Q) = \left(1 + \left(\frac{2Q(\lambda_i - \lambda_j)}{\lambda_j}\right)^2\right)^{-1}. \tag{6}$$

Here, $(\lambda_i - \lambda_j)$ represents the channel spacing CS, i.e., the spectral distance between two adjoining wavelengths. This is also an optimizable parameter within the confines of the free spectral range (FSR) we are considering. P_S in (4) and (5) is the signal power of λ_i that reaches the MR sensitive to λ_j, and can be defined as:

$$P_S = \psi(\lambda_i, \lambda_j) P_{in}(i), \tag{7}$$

where P_{in} is input power to the waveguide, calculated by considering the detector sensitivity and the signal power loss of λ_i before the MR with resonance wavelength λ_j within the bank, represented by ψ. When an optical signal in a waveguide passes by an MR, the crosstalk induced power suppression in its power can be modeled as a through loss, which is defined as γ times the signal power before it passes by the MR. This suppression factor γ and hence ψ can be calculated as follows:

$$\gamma(\lambda_i, \lambda_j, Q) = \left(1 + \left(\frac{2Q(\lambda_i - \lambda_j)}{\lambda_j}\right)^{-2}\right)^{-1}, \tag{8}$$

$$\psi(\lambda_i, \lambda_j) = \prod_{k=1}^{(k-1)<j} \gamma(\lambda_i, \lambda_k, Q)\,. \tag{9}$$

For calculating FWHM, we use the following model:

$$FWHM = \frac{\lambda_{res}}{Q - factor}, \tag{10}$$

where λ_{res} is the resonant wavelength of the MR being considered.Using these models, we can identify the optimal design space for our MR banks which can ensure high SNR and high tunable range (R_{tune}). We must also consider that the lowest optical power level (P_{lpar}) should be higher than P_{noise}, wrt P_{signal}:

$$10\log_{10}\left(\frac{P_{signal}}{P_{lpar}}\right) < 10\log_{10}\left(\frac{P_{signal}}{P_{noise}}\right), \tag{11}$$

where P_{lpar} can be defined in terms of P_{signal} as follows:

$$P_{lpar} = \frac{P_{signal} \times R_{tune}}{N_{levels}}\,. \tag{12}$$

Replacing P_{lpar} in (11) yields the following relation:

$$10\log_{10}\left(\frac{N_{levels}}{R_{tune}}\right) < SNR, \tag{13}$$

where N_{levels} is the number of amplitude levels we need to represent across the available R_{tune}: for an n-bit parameter (ANN weight or bias) representation, N_{levels} will be 2^n. If positive and negative values are represented separately, as in the case with *TRON*, then N_{levels} will be 2^{n-1}. The relationship in (13) can be rearranged to yield the following relationship between R_{tune} and SNR:

$$R_{tune} > N_{levels} \times 10^{-\frac{SNR}{10}} \tag{14}$$

Utilizing these models, we can identify the ideal design space for our MR banks, as discussed later in Section 4.1

2.3.3 MULTI-HEAD ATTENTION (MHA) UNIT DESIGN

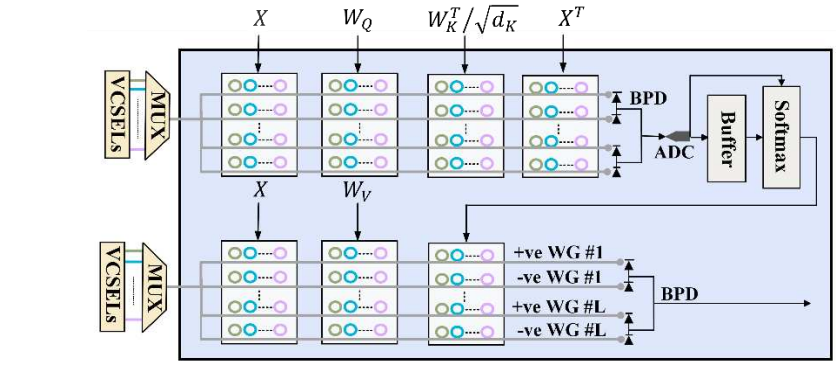
The major challenge with transformer inference acceleration is the time-consuming matrix multiplications (MatMuls). These operations can be decomposed into vector dot-product operations as outlined for optical CNN acceleration in [18]. Looking closely at the self-attention in each head (1), the computation of MatMul $(Q.K^T)$ cannot be performed until the generation and storage of K^Tcompletes. This dependency would infer significant power and latency overhead as we would first need to generate *K* matrix ($K = XW_K$) optically, convert the output to digital domain, buffer the values, generate K^T, then convert the matrix to the optical domain

again to calculate the next MatMul ($Q.K^T$). Alternatively, using MatMul decomposition, we can rewrite the operation as two cascaded MatMul steps as follows:

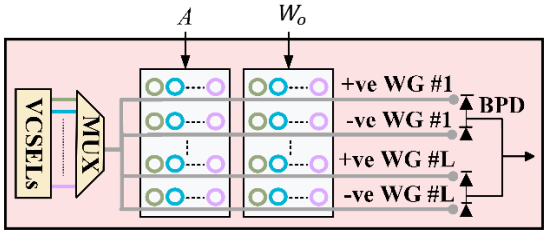
$$Q.K^T = Q.(X.W_K)^T = (Q.W_K^T).X^T \tag{15}$$

As shown by the top four MR bank arrays in Figure 9(a), no intermediate buffering is thus needed to compute $Q.K^T$. The first two MR bank arrays generate Q, then by having W_K^T and X^T previously stored and used to tune the MRs in the following two MR bank arrays, we can directly get the output of (15) optically without any intermediate buffering or expensive opto-electric conversions. To further reduce the latency and power overhead, we propose including the scaling factor in (1) within the weight matrix (W_K^T) storage in the ECU. As such, the individual MR tuning values would be ${W_K^T}_i/\sqrt{d_k}$, instead of having an additional MR bank array to perform the scaling operation. As in most transformer models d_k (dimension of Q and K) is usually 64, a simple 3-bit left shift circuit should be able to efficiently handle the division.

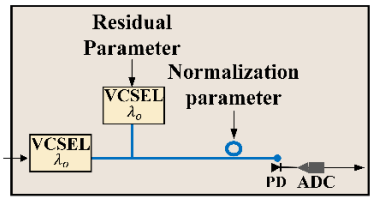
For the MatMul operations, most optical ANN accelerators (such as [62]) calculate them one-by-one, by having separate MAC units with MR bank arrays to perform the multiplication operations. Consequently, they accumulate and add the partial sums. As there are more than two consecutive MatMul operations (1) and (15)) involved in the attention computation, we avoid the accumulation of intermediate values and pass the individual multiplication results generated by the first MR bank array to the following MR bank arrays directly.



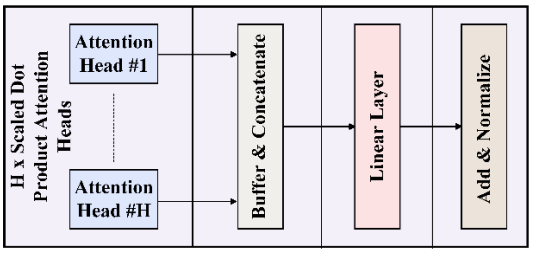
(a)



(b)



(c)



(d)

Figure 9: (a) Attention head unit comprised of seven MR bank arrays for MatMul operations, each with dimension *K*×*N*; (b) Linear layer comprised of an MR bank array with dimension *K*×*N*; (c) Add and Normalization layers using coherent photonic summation and an MR for imprinting the normalization parameter; (d) MHA unit composed of *H* attention heads, buffer and concatenate block, linear layer, and an add and normalize block.

The summation of all the multiplications and partial sums is then done at the end, before the softmax block, as shown in Figure 9(a). This approach avoids the latency and power costs from early summations, intermediate buffering, and associated opto-electric conversions. Moreover, as outlined in Section 3.1, we have ensured minimal crosstalk noise, that would normally be an issue due to such MR arrangement. Following the calculation of $(Q.K^T)$ by the upper MR bank arrays shown in Figure 9(a), all partial sums are accumulated using balanced photodetectors (BPDs). BPDs help accommodate both positive and negative parameter values by placing separate positive and negative arms for the same waveguide. The sum acquired from the negative arm is subtracted by the BPD from the sum from the positive arm. The results are then converted to the digital domain, to undergo softmax computation.

Another challenge in MHA is the softmax operation. It is performed in each attention head and restricts parallelism as all results from the previous MatMul need to be generated first. For its implementation, we propose two optimization solutions. First, we avoid the computationally expensive division and numerical overflow by employing the log-sum-exp trick, used in a few previous works such as [58], as follows:

$$Softmax(\chi_i) = \frac{\exp(\chi_i - \chi_{max})}{\sum_{j=1}^{d_k} \exp(\chi_j - \chi_{\max})} \quad , \tag{16}$$
$$= \exp\left(\chi_i - \chi_{max} - \ln\left(\sum_{j=1}^{d_k} \exp(\chi_j - \chi_{max})\right)\right),$$

where softmax can be divided into four main operations: finding χ_{max}, subtraction, natural logarithm (*ln)*, and exponential *(exp).* Finding χ_{max} and the subtraction can be computed using simple digital circuits. As shown in Figure 4(a), the analog-to-digital converter (ADC) output is

buffered while also being fed to a comparator circuit, so that finding χ_{max} would be computed in parallel to the MatMuls. The natural logarithm (*ln)* and exponential *(exp)* computations can be calculated using look-up tables (LUTs) [68]. This also helps get the final softmax output as an analog value from the memristor cell in the LUT, which can be used to directly tune the MR bank array. Furthermore, our scaled dot-product attention design enables high parallelism because the bottom vertical cavity surface emission laser (VCSEL) array (Figure 9(a)) can be synchronized to only be turned on when the softmax operation is done.

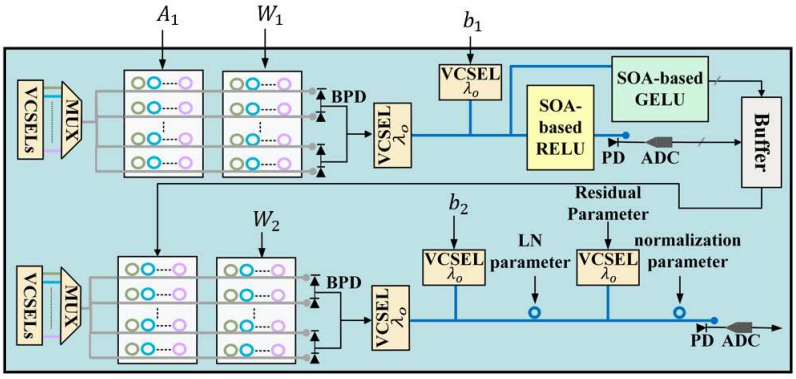
The linear layer in MHA is also implemented optically using two MR bank arrays (Figure 9(b)). For adding the MHA input to its current output (implementing the residual connection), coherent photonic summation is employed, as shown in Figure 9(c), where the output signal from the linear layer is used to directly drive a VCSEL with wavelength λ_o. Another VCSEL with the same wavelength, is driven by value(*i*) from the residual connection, and thus, when the two waveguides meet, they undergo interference, resulting in the summation of the two values. Coherent summation is ensured by using a laser phase locking mechanism [69], which guarantees that VCSEL output signals have the same phase for constructive interference to occur. Lastly, layer normalization (LN) is performed optically using a single MR, tuned by the LN parameter. The entire MHA architecture is shown in Figure 9(d).

2.3.4 FEED FORWARD (FF) UNIT DESIGN

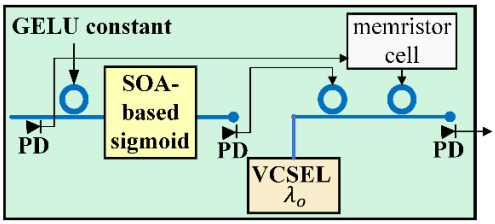
The FF Unit (Figure 10(a)) is composed of two fully connected (FC) layers, with a non-linear activation in between. Each FC layer is accelerated using two MR bank arrays, with dimensions $K \times N$: one to imprint the input activations and the second to compute the MatMul between the inputs and the weight matrices. The bias values are added using coherent photonic

summation, discussed in the previous section. For the non-linear unit, we implemented an optical *RELU* unit, with semiconductor-optical-amplifiers (SOAs). When the gain in an SOA is adjusted to a value close to 1, the behavior becomes almost linear, resembling the *RELU* operation. The work in [70] demonstrated how SOAs can be exploited to implement other non-linear functions such as *Sigmoid* and *tanh*. This expands the scope of *TRON* and enables us to implement the *GELU* operation (used in ViT) instead of the *RELU*, optically. The *GELU* operation can be approximated as follows [71]:

$$GELU(x) = x\Phi(x) = 0.5x(1 + \tanh\left[\sqrt{2/\Pi}\,(x + 0.044715x^3)\right]) \\ = x\sigma(1.702x). \qquad (17)$$



(a)



(b)

Figure 10: (a) FF block composed of four-MR bank arrays with dimensions $K \times N$, SOA-based RELU and GELU units, and bias and residual connection additions, done with coherent photonic summation; (b) GELU unit composed of three MRs, a semiconductor-optical-amplifiers (SOA), and a VCSEL.

As shown in Figure 10(b), the first multiplication $(1.702x)$ is implemented using a single MR, and the sigmoid function is computed using the SOA implementation, described above. The last multiplication of the input with the sigmoid output is calculated using two MRs. To store the input signal and use it to tune the second MR, a low-power, local storage mechanism is used where the analog input signal from the PD is stored in a memristor cell to directly tune the last MR.

The output from the non-linear unit is then buffered and used to tune the MRs in the first bank array of the second FC layer (Figure 10(b)), to be multiplied by the weight matrix (W_2). Following the second FC layer, the normalization layer is implemented using an MR, the residual connection is added through coherent photonic summation, and the final normalization layer is implemented with another MR.

2.3.5 TRON ARCHITECTURE

The architecture of *TRON* (Figure 8) is designed to accelerate various transformer models. Given a transformer's sequential nature and unique structure, *TRON* is designed to support efficient sequential operations where needed, while also implementing parallel hardware to accelerate parallelizable operations. *TRON* is composed of two sets of MHA units and one set of FF units. Each set has a dimension of *L*. Such an arrangement enables both the encoder and decoder blocks to easily reuse most of the units. In case of the encoder block, the first VCSEL array will be used to drive the input to the second set of MHA units only. The MHA unit can be divided into two parts: before and after the softmax operation. As softmax (see (1)) cannot be

computed till the first part is completed, both parts cannot be parallelized. However, the MatMul operations in the second part can be parallelized with the MatMul operations in the FF unit. For the decoder block, the first VCSEL array is used to drive the input to the first set of MHA units. Its output is used as the input to the second MHA unit whose output then drives the FF unit. Moreover, VCSEL-reuse, as described, reduces the laser power consumption. Accordingly, single VCSEL arrays are shared among rows in each MR bank array and used to imprint the input activations.

2.4 EXPERIMENTS AND RESULTS

We performed detailed simulation-based analyses to assess the efficiency of our proposed *TRON* architecture. Four transformer models were considered in our analyses: Transformer-base [23], BERT-base [56], Albert-base [57], and ViT-base [24]. The model parameters are shown in Table 1, where d_{model} and d_{ff} are the dimensionality of input/output and FF layers. We developed a simulator in Python to estimate the area, performance, and energy costs associated with running each model. The area, performance, and energy estimates for all electronic buffers used in *TRON* were estimated using the CACTI tool [72] at 28nm; while the electronic circuit in softmax, was synthesized using Xilinx Vivado at 28 nm and the resulting power/delay estimates were used in our analyses. Tensorflow 2.9 was used to train and analyze each model's accuracy.

Table 1: Transformer models and parameter counts

Model	Parameters	Layers	Heads	d_{model}	d_{ff}
Transformer-base	52M	2	8	512	2048
BERT-base	108M	12	12	768	3072
Albert-base	12M	12	12	768	3072
ViT-base	86M	12	12	768	3072

The achieved accuracies and the datasets associated with each model are as shown in Table 2. As shown, Transformer, BERT, and Albert models were used for NLP tasks (language translation and sentiment analysis). ViT was evaluated using an image classification task, with pre-training on ImageNet and fine-tuning on Cifar-10. Our analysis concluded that 8-bit model quantization results in comparable algorithmic accuracy to models with full (32-bit) precision; thus, we targeted the acceleration of 8-bit precision transformer models.

Table 2: Transformer model performances

Model	Dataset(s)	Accuracy (32-bit)	Accuracy (8-bit)
Transformer-base	Ted_hrlr_translate	66.73%	70.4%
BERT-base	Sentiment-Analysis-of-IMDB-Movie-Reviews	85.8%	85.8%
Albert-base	Sentiment-Analysis-of-IMDB-Movie-Reviews	88.3%	88.7%
ViT-base	ImageNet/Cifar-10	97.7%	98.0%

Table 3: Parameters used for *TRON* analysis

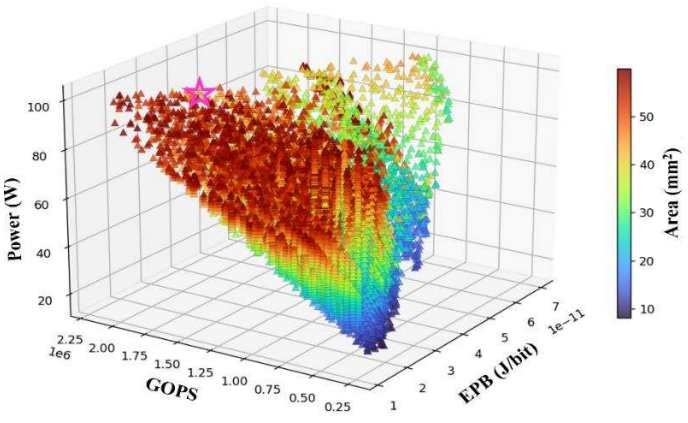
Devices	Latency	Power
EO Tuning [63]	20 ns	4 μW/nm
TO Tuning [64]	4 μs	27.5 mW/*FSR*
VCSEL [62]	0.07 ns	1.3 mW
Photodetector [62]	5.8 ps	2.8 mW
SOA [62]	0.3 ns	2.2 mW
DAC (8 bit) [77]	0.29 ns	3 mW
ADC (8 bit) [78]	0.82 ns	3.1 mW
Memristor cell [62]	0.1 ns	0.07μW

The optoelectronic parameters examined for *TRON's* simulation-based analysis are shown in Table 3. We considered various factors that contribute to photonic signal losses such as: waveguide propagation loss (1 dB/cm), splitter loss (0.13 dB [73]), combiner loss (0.9 dB [74]), MR through loss (0.02 dB [75]), MR modulation loss (0.72 dB [76]), EO tuning loss (6 dB/cm [63]), and TO tuning loss (27.5 mW/*FSR* [64]). Increasing the number of wavelengths and the

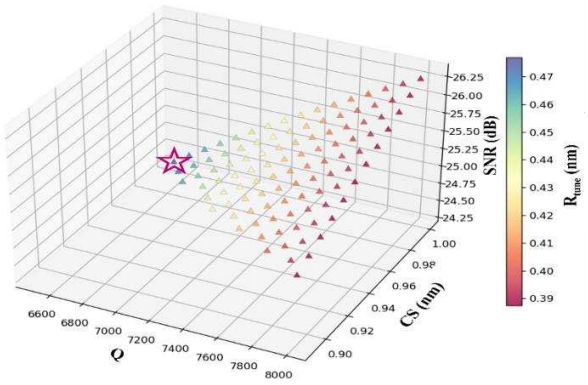
waveguide length will in turn increase the MR count, photonic loss, and the required laser power consumption. Accordingly, we modeled the required laser power used in our architecture for each source as:

$$P_{laser} - S_{detector} \geq P_{photo\_loss} + 10 \times \log_{10} N_{\lambda}, \quad (19)$$

where P_{laser} is the laser power in dBm, $S_{detector}$ is the PD sensitivity in dBm, N_{λ} is the number of laser sources/wavelengths, and P_{photo_loss} is the total optical loss encountered by the signal, due to the factors discussed. In the next subsection, we describe our analyses to determine the optimal values for *TRON's* architectural parameters *H, L, K,* and *N*, which were discussed in section 3.



(a)



(b)

Figure 11: (a) Architectural optimization for *TRON*, to find the optimal [*H, L, K, N*] configuration with the best energy-efficiency and throughput. The best configuration, [4,2,51,17], has the lowest EPB/GOPS value and a maximum power of 100W is shown with the pink star; (b) MR bank optimization for *TRON*, aiming to identify optimal [R_{tune},*Q,SNR,CS*] design point. The best point [0.45,6500,24.3,1] with highest R_{tune} value, is shown with the pink star.

2.4.1 TRON ARCHITECTURE DESIGN OPTIMIZATION

The *TRON* architecture design is dependent on four key parameters, as discussed in Section 3: *H* (the number of heads in the MHA unit*), L* (the number of layers), *K* (the number of rows), and *N* (the number of columns in each MR bank array). We performed an exploration to determine the optimal [*H,L,K,N*] configuration for *TRON,* defined as the configuration with lowest EPB/GOPS, where EPB is energy-per-bit and GOPS is giga-operations-per-second. We also set a maximum power limit of 100W for the configuration. The result of this exploration is presented in the scatterplot in Figure 11(a). The optimal configuration [4,2,51,17], is highlighted with the pink star. This configuration is used in the comparative analysis in the following subsections.

For the MR-bank design, the models described in Section 3.2 were used to perform another exploration study. Using the SNR model (3), with the R_{tune} constraint (14), we explored the MR bank design space to find the parameters [R_{tune},*Q,SNR,CS*], with the aim of maximizing tuning range R_{tune}. We considered N_{level} of 2^{8-1}, an FSR of 20 nm, Q-factor ranging from 2000 to 8000, and channel spacing ranging from 0.1 to 1 nm. The result of the exploration is as shown in Figure 11(b), where we have selected the data point with the best R_{tune}: [0.45, 6500, 24.3, 1].

2.4.2 TRON ARCHITECTURE COMPONENT-WISE ANALYSIS

To understand the performance of the major components within the *TRON* architecture, we present a breakdown in terms of power and latency for these components in Figure 12. For the power, it is evident that MatMul operations in the attention heads contribute to more than half of the architecture's power overhead. This is because of the large dimensions of the matrices being multiplied in the MHA blocks, in each attention head. This requires many digital-to-analog converters (DACs), whose power consumption is considerable. Moreover, the sequential dependency in the attention head also contributes notably to the latency overhead. As Albert shares all attention and FF parameters across layers [57], this leads to a minimization of the number of active DACs, reducing the overall power consumption for the Albert model.

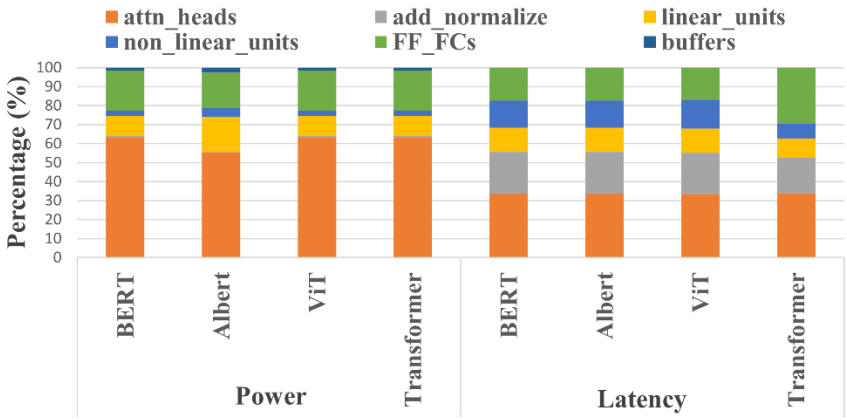


Figure 12: Power and latency breakdown across *TRON* components.

2.4.3. COMPARISON TO STATE-OF-THE-ART ACCELERATORS

We compared *TRON* execution on multiple processors and state-of-the-art transformer accelerators: Tesla V100-SXM2 GPU, TPU v2 [79], Intel Xeon CPU, TransPIM [41], FPGA transformer accelerator in [58] (FPGA_Acc1), VAQF [59], and FPGA transformer accelerator in

[60] (FPGA_Acc2). VAQF focuses on vision transformers and FPGA-Acc2 on traditional encoder-decoder transformer architectures and transformer-based language models; results for these two platforms are thus restricted to the models they are targeted for. We used power, latency, and energy values reported for the selected accelerators, and results from executing models on the GPU/CPU/TPU platforms to estimate the EPB and GOPS for each model. The *TRON* architectural configuration used in the comparisons has a maximum power of 100W, and is the one described in Section 4.1.

Figure 13 shows the GOPS comparison between *TRON* and the other architectures considered. Our architecture achieves on average 262×, 1631×, 1930×, 14×, and 55× better GOPS than GPU, TPU, CPU, TransPIM, and FPGA_Acc1, respectively. When comparing transformer model-specific accelerators, *TRON* has on average 352× higher GOPS than FPGA_Acc2 for transformer, BERT, and Albert models, and 846× higher GOPS than VAQF for ViT. The higher throughput over all compute platforms can be explained in terms of *TRON*'s high-speed execution in the optical domain and the minimal computations in the digital/electric domain.

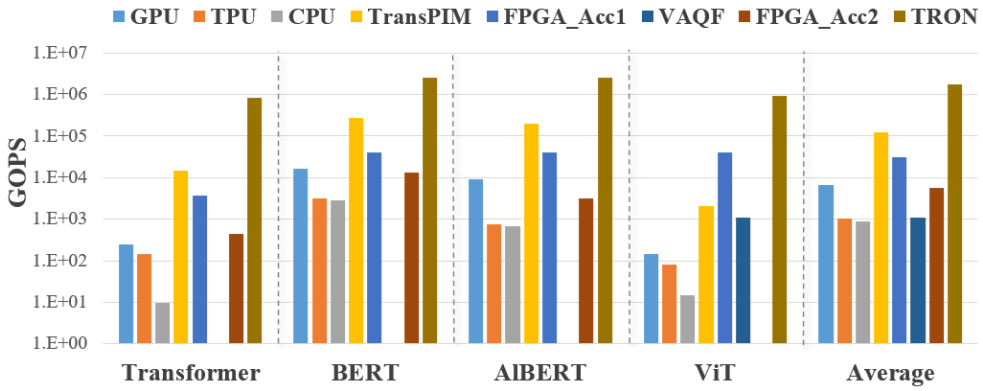


Figure 13: Throughput comparison between transformer accelerators and platforms

Figure 14 presents the energy-per-bit (EPB) comparison. On average, *TRON* attains 4231×, 12397×, 10971×, 14×, and 8× lower EPB than GPU, TPU, CPU, TransPIM, and FPGA_Acc1. For model-specific accelerators, we achieve on average 802× lower EPB than FPGA_Acc2 for transformer, BERT, and Albert models, and 32× lower EPB than VAQF for ViT. These EPB improvements can be attributed to *TRON*'s significant low latency operation, and the relatively lower power compared to some of the computation platforms considered.

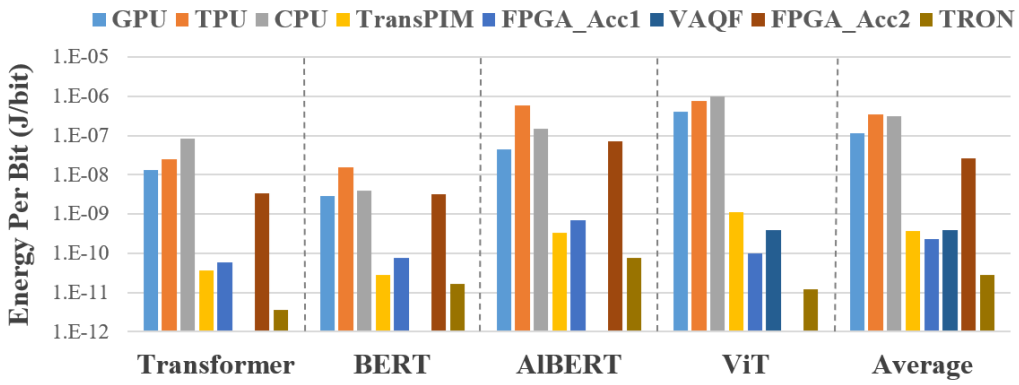


Figure 14: EPB comparison between transformer accelerators and platforms.

2.5 CONCLUSION

In this chapter, we presented the first non-coherent silicon photonic hardware transformer accelerator, called *TRON*. Our proposed accelerator architecture exhibited throughput improvements of at least 14× and energy-efficiency improvements of at least 8× when compared to eight different processing platforms and state-of-the-art transformer accelerators. These results demonstrate the promise of *TRON* in terms of energy-efficiency and high-throughput inference acceleration for transformer neural networks. This work focused on the hardware architecture design with silicon photonics. When combined with software optimization techniques that aim to

reduce a transformer's large memory footprint, significantly better throughput and energy efficiency can be achieved.

2. GHOST: A GRAPH NEURAL NETWORK ACCELERATOR USING SILICON PHOTONICS

GNNs have emerged as a powerful approach for modelling and learning from graph-structured data. Multiple fields have since benefitted enormously from the capabilities of GNNs, such as recommendation systems, social network analysis, drug discovery, and robotics. However, accelerating and efficiently processing GNNs requires a unique approach that goes beyond conventional ANN accelerators, due to the substantial computational and memory requirements of GNNs. The slowdown of scaling in CMOS platforms also motivates a search for alternative implementation substrates. In this chapter, we present *GHOST*, the first silicon-photonic hardware accelerator for GNNs. *GHOST* efficiently alleviates the costs associated with both vertex-centric and edge-centric operations. It implements separately the three main stages involved in running GNNs in the optical domain, allowing it to be used for the inference of various widely used GNN models and architectures, such as graph convolution networks (GCNs) and Graph Attention Networks (GATs). Our simulation studies indicate that *GHOST* exhibits at least 10.2× better throughput and 3.8× better energy efficiency when compared to GPU, TPU, CPU and multiple state-of-the-art GNN hardware accelerators.

3.1 INTRODUCTION

Deep learning has become a vital pillar in our lives due to its ability to solve many complex problems efficiently across diverse fields, including autonomous transportation, healthcare, industrial automation, and network security. This success of deep learning owes tremendously to the evolution of neural networks variants that are tailored for specific learning tasks. For example, convolution neural networks (CNNs) [80] and RNNs [81] have proven their

efficiency in pattern recognition for images and sequence data, by extracting knowledge from the spatial and temporal dimensions of data. While these examples are prominent solutions for many tasks, they are limited ins cope to non-arbitrary structured and Euclidean data. Arbitrary structured data, including graphs, require different techniques for efficient processing. Graph data processing is critical for many problems, e.g., social network analysis, recommender systems, and drug discovery [82].

GNNs have emerged in recent years and established their proficiency in dealing with graph-structured data. These models can extract information from the graph structure and discover patterns in the data that may be difficult to identify with other deep learning methods [32]. Accordingly, many applications now benefit greatly from GNNs, and hence a lot of recent efforts have focused on enhancing GNN algorithms and improving their efficiency in handling large and various graphs. The continuing progress of GNN algorithms and models necessitates hardware platforms capable of providing GNN-specific support with high performance, while abiding by strict power constraints. Although hardware acceleration for neural networks such as CNNs and RNNs have been extensively studied, the processing of GNNs presents unique challenges due to their combination of dense and vastly sparse operations, diversity of input graphs, and the various types of GNN algorithms and models [83]. Thus, hardware accelerators tailored to accelerate the processing of conventional neural networks cannot be directly and efficiently applied to GNNs.

Moreover, relying on traditional electronic accelerators creates limitations as these platforms face challenges in the post-Moore era due to high costs and diminishing performance improvements with semiconductor-technology scaling. Moving data through metallic wires is a well-known bottleneck in these accelerators, as it restricts the achievable performance in terms of

bandwidth, latency, and energy efficiency [103]. Silicon photonics technology provides a promising solution to this data-movement bottleneck, offering ultra-high bandwidth, low-latency, and energy-efficient communication [84]. Optical interconnects, which are now being considered for chip-scale integration, have already replaced metallic ones for light-speed data transmission at almost every level of computing. It is also possible to use optical components for computations, such as matrix-vector multiplication [85]. The emergence of chip-scale optical communication and computation has thus made it possible to design photonic integrated circuits that offer low-latency and energy-efficient optical-domain data transport and computation. Furthermore, prior work, from both academia and industry, has demonstrated the significant benefits resulting from using silicon photonics for the acceleration of neural networks, as in [85], [86], [62].

In this chapter, we introduce *GHOST*, the first silicon-photonic-based GNN accelerator that can accelerate inference of diverse GNN models and graphs. The key contributions in this chapter are:

- The design of a novel GNN accelerator hardware architecture using silicon photonics with the ability to accelerate multiple existing variants of GNN models;
- Detailed photonic device and circuit-level optimizations to mitigate crosstalk noise so that error-free GNN operations can be ensured in the accelerator;
- A detailed architectural optimization for the efficient handling and acceleration of diverse graph structures and GNN model architectures on the proposed hardware accelerator; and
- A comprehensive comparison with GPU, TPU, CPU, and state-of-the-art GNN accelerators.

The rest of the chapter is organized as follows. Section 2 provides a background on GNNs (different models, their acceleration challenges, and previous efforts on GNN acceleration) and on performing computations using silicon photonics. Section 3 describes the *GHOST* architecture and our optimization efforts at the device, circuit and architecture layers. Details of the experiments conducted, simulation setup, and results are presented in Section 4. Lastly, Section 5 presents concluding remarks.

3.2 BACKGROUND

3.2.1 GRAPH NEURAL NETWORKS

Prior to the emergence of GNNs, graph processing was mostly limited to traditional ML and graph algorithms. However, these methods had limitations in capturing the non-linear and complex relationships between vertices in a graph [32]. With the advent of GNNs, graph processing has been revolutionized, and there has been a significant improvement in graph-based ML tasks, such as node classification, link prediction, and graph classification. GNNs are a type of deep learning algorithm that can learn complex graph structures and relationships, and have now broadened the scope of ANNs to encompass non-Euclidean and irregular data found in graphs [83].

GNNs exploit the connections within a graph to understand and represent the relationships between vertices. They utilize an iterative approach that relies on a graph's structure and take in edge, vertex, and graph feature vectors that represent the known attributes of these elements. The general operations of a GNN can be broadly summarized in three main steps: 1) pre-processing, 2) iterative updates, and 3) readout. *Pre-processing* is an optional initial step that is typically performed offline for purposes such as sampling the graph, rearranging the graph to

simplify the algorithm's processing and complexity, or encoding the feature vectors. *Iterative updates* are where the main GNN computations occur through two main phases: aggregation and combination. The aggregation phase accumulates all the edges in a graph, and then for each vertex, it reduces all its neighbors and its own feature vectors into a single set. This feature set is combined and through performing linear transformations and non-linear activation functions, a new updated feature vector for each vertex is obtained. GNNs can be composed of several layers and the iterative process in a single layer updates every edge and vertex with information received from immediate neighbor vertices. This means that the relationships with nodes and edges that are progressively farther away can be gradually considered as more layers are processed. Finally, *readout* is employed when a graph possesses a global feature vector, and it is updated once after the edge and node updates have been executed, usually in graph classification tasks.

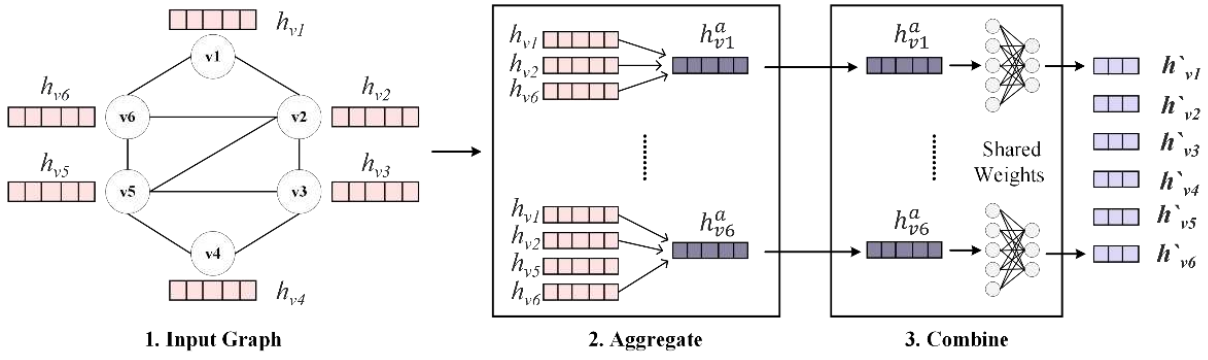


Figure 15: An example of GNN inference showing 1) Input graph to be processed; 2) Aggregation phase, where each vertex's neighbors are reduced to one feature vector; 3) Combine and Update phases, where each vertex is linearly transformed and updated using a non-linear activation function.

Figure 15 illustrates an example of processing the first layer in a GNN. As shown, the aggregation phase iteratively gathers the neighbors of each vertex and then reduces all data into a single vector, h_{vi}^{a}. The reduce operation in this phase can be a variety of arithmetic functions, e.g., summation, mean, or maximum. This vector is then passed through the combination phase, which usually involves a neural network. Unlike conventional ANNs, where each layer has a different set of weights, vertices in a GNN all share the same weights.

Since GNNs were initially introduced in [87], multiple GNN algorithms and models have emerged. Graph convolution network (GCN) [27] expands the idea of convolution in the graph space. In contrast to CNNs, where the convolutional operation is defined on regular grid-like data, the convolutional operation in GCNs is defined on irregular graph structures. GraphSAGE [88] and Graph Isomorphism Network (GIN) [89] are two models that are also based on graph convolutions. GraphSAGE employs custom sampling techniques to obtain a fixed number of neighbors for each vertex while GIN learns the isomorphism invariant representation of graphs by using a learnable parameter ε_l to adjust the weight of the central vertex. Graph Attention Networks (GATs) [28] are another class of GNNs that have demonstrated noteworthy results. GATs update node features through a pairwise function between nodes, which incorporates learnable weights. This results in an attention mechanism that can determine the usefulness of the edges.

3.2.2 GRAPH NEURAL NETWORK ACCELERATION

Processing GNNs presents many challenges. A system processing a GNN needs to have the capabilities to efficiently handle both dense and very sparse computations, adapt its execution and operations based on the specific input graph structure and the GNN algorithm variant

employed, and scale effectively to extremely large graphs. Due to the irregularity and large size of most real-world graphs, GNNs often require very high memory bandwidth and multiple irregular memory accesses. Further, the unique combination of computing characteristics from deep learning and graph processing in GNNs results in having alternate execution patterns [83]. Such challenges are typically absent when processing traditional ANN models. Thus, utilizing ANN accelerators for GNNs can be inefficient and lead to low performance and high energy costs. While overcoming these challenges is a non-trivial task, many recent efforts have tackled this problem and advanced the field of GNN processing, as discussed below.

On the software side, several frameworks and graph libraries have been proposed to aid in the acceleration of GNN models. A few examples of relevant libraries are PyTorch Geometric (PyG) [90], Deep Graph Library (DGL) [91], and NeuGraph [92]. Several programming models that aim to abstract GNN operations have also emerged, such as SAGA [92] and GRETA [93].

On the hardware side, multiple electronic hardware accelerators for GNNs have been proposed. The accelerator in [94] presents a modular architecture where the core unit of the accelerator is a tile composed of an aggregator module (AGG), a DNN accelerator module (DNA), a DNN queue (DNQ), and a Graph Processing Element (GPE). The main component in their AGG module is a bank of ALUs, and the DNA exhibits the architecture of existing spatial accelerators. Another electronic accelerator EnG [95] has a unified architecture that handles GNNs in a single dataflow as a concatenated matrix multiplication of feature vectors, adjacency matrices, and weights. An array of clustered PEs is utilized. To aggregate the results, each column of PEs is connected in a ring, and the results are passed along and added based on the adjacency matrix. HyGCN [96] is another electronic accelerator for GCNs, composed of two dedicated engines that handle the aggregation and combination stages, along with a control

mechanism that coordinates the sequential execution of both processes. The combination stage, which is dense, is computed using a conventional systolic array approach. In contrast, the aggregation stage has a more complex architecture that includes a sampler, an edge scheduler, and a sparsity eliminator. Lastly, GRIP [97] utilizes the GReTA programming model [93] to create an electronic accelerator with specialized units and accumulators for edges and vertices, which are separate and adaptable.

Several GNN accelerators based on ReRAM and processing-in-memory (PIM) have also been presented. For example, ReGNN [98] leverages analog PIM (APIM) and digital PIM (DPIM). The authors decompose the computations in the combination phase into multiple matrix-vector multiplications (MVM) and handle them through a dedicated combination engine composed of an APIM ReRAM array, while non-MVM operations are processed by the DPIM array. ReGraphX [99] is another ReRAM-based architecture that can be used for both training and inference acceleration of GNNs.

Unlike previous efforts, *GHOST* is the first GNN accelerator that leverages silicon photonics. It also supports accelerating a broad family of GNN models, adapts efficiently to different graph shapes and sizes, and mitigates typical GNN memory and performance bottlenecks.

3.2.3 OPTICAL COMPUTATION

Optical ANN accelerators have gained considerable attention from both academic researchers and industry in recent years because of their notable advantages in terms of energy-efficiency and performance [7]. There are two possible classes of optical ANN accelerators: coherent and non-coherent. In coherent architectures, a single wavelength is utilized to imprint parameters onto the optical signal's phase, which allows for multiply and accumulate (MAC)

operations. Non-coherent architectures utilize multiple wavelengths and imprint parameters onto the optical signal's amplitude, enabling parallel operations to be performed using each wavelength. The current focus of research in optical ANN accelerators is mainly on CNNs, MLPs, and RNNs. To the best of our knowledge, *GHOST* is the first optical accelerator for GNN models.

Most of *GHOST's* core operations are performed using opto-electronic tuning devices called microring resonators (MRs; see Figure 16). Each MR can be specifically designed and adjusted to work at a particular wavelength, known as the MR resonant wavelength (λ_{MR}), defined as $\lambda_{MR} = \frac{2\pi R}{m} n_{eff}$, where, R is the MR radius, m is the order of the resonance, and n_{eff} is the effective index of the device. Electronic data can be modulated onto the optical signal passing an MR by carefully adjusting n_{eff} (and hence λ_{MR}) with a tuning circuit. Typically, the tuning circuit is based on thermo-optic (TO) [63] or carrier injection electro-optic (EO) tuning [64], both of which cause a change in the effective refractive index n_{eff} and result in a resonant shift of $\Delta\lambda_{MR}$. In non-coherent systems, computations, particularly multiplications, are performed by adjusting an MR's $\Delta\lambda_{MR}$, which leads to a predictable alteration in the amplitude of the optical signal's wavelength. To enhance throughput and emulate neurons in ANNs, MR-based photonic architectures employ wavelength-division multiplexing (WDM) with multiple wavelengths. This involves combining multiple optical signals with different wavelengths into a single waveguide using an optical multiplexer [7]. Different wavelengths in the input waveguide pass through a series of MRs, with each MR tuned to a specific wavelength, allowing for several multiplications to be executed simultaneously in parallel.

Figure 16 illustrates how matrix-vector multiplication (MVM) can be performed in MR-based non-coherent optical computing systems. As shown in Figure 16(b), two arrays of MR

banks are utilized, with the first array imprinting input activations [a_1, a_2, a_3] onto the optical signals, and the second array performing multiplication with the first row of a weight matrix [w_1, w_2, w_3]. Different optical wavelengths on the waveguide can then go through a photodetector (PD, not shown in Figure 16) to accumulate the output of the dot product.

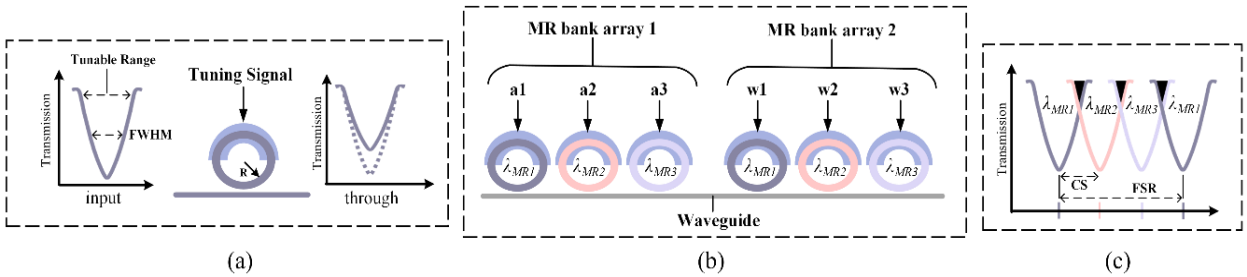


Figure 16: (a) MR input and through ports' wavelengths after imprinting a parameter onto the signal; (b) MR bank arrays used to perform multiplication by imprinting input vector (a_1-a_3), followed by weight vector (w_1-w_3); (c) MR bank response and heterodyne crosstalk shown in black, where CS is channel spacing and FSR is free spectral range.

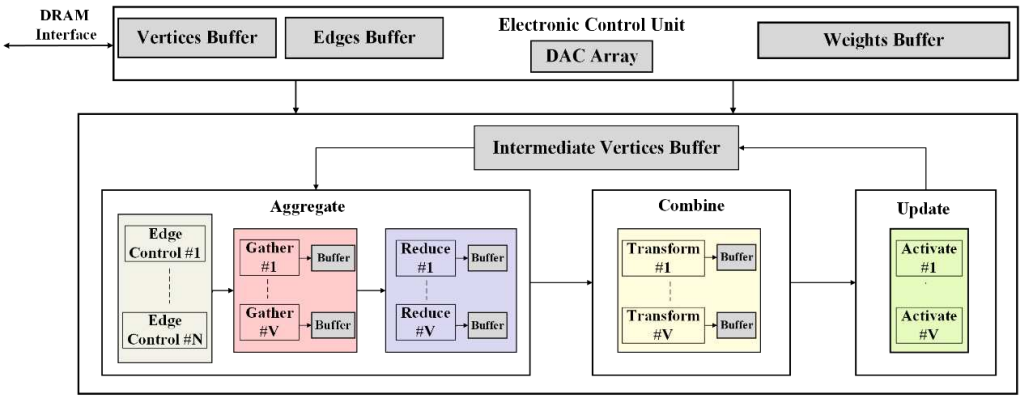


Figure 17: Overview of *GHOST* accelerator architecture showing the ECU, Aggregate, Combine, and Update blocks.

3.3 GHOST HARDWARE ACCELERATOR

GHOST is a silicon photonic architecture that can accelerate the inference of a diverse family of GNN models. An overview of the architecture is shown in Figure 17. The photonic accelerator core is composed of aggregate, combine, and update blocks, enabling the execution of a wide range of GNN models and real-world graph datasets. Interfacing with the main memory, buffering the input graph, identifying the needed resources, and mapping the weight matrices to the photonic architecture are all handled by an integrated electronic-control unit (ECU). The following subsections describe the *GHOST* architecture and the device, circuit, and architecture layer optimization solutions we have considered to efficiently accelerate GNN models.

3.3.1 TUNING CIRCUIT DESIGN

As discussed earlier, MR devices require a tuning mechanism, which can be achieved using either EO or TO methods. In *GHOST*, we have implemented a hybrid tuning circuit that utilizes both methods to induce $\Delta\lambda_{MR}$. By doing so, we can capitalize on the advantages of each approach while mitigating their drawbacks. EO tuning is quicker (≈ns range) and consumes less power (≈4 μW/nm); however, it cannot be used for large tuning ranges [63]. Conversely, TO tuning offers a larger tunability range, but with the drawback of higher latency (≈μs range) and power consumption (≈27 mW/FSR) [64]. To address these challenges, we have integrated EO tuning to quickly induce small $\Delta\lambda_{MR}$ and reserved the slower TO tuning for cases where larger $\Delta\lambda_{MR}$ is required. The effectiveness of this hybrid approach has been previously demonstrated in [64]. To further reduce the power consumption of TO tuning and also reduce thermal crosstalk, we have employed the thermal Eigenmode decomposition method (TED) from [66]. We

analyzed thermal interference between MR heaters and using Eigenmode decomposition, thermal tuning levels across heaters which do not cause thermal interference in MR banks were determined. Tuning the MR heaters according to the tuning levels obtained with this approach allowed us to mitigate thermal crosstalk and minimize TO tuning power.

3.3.2 MR DEVICE OPTIMIZATION

To ensure that the MVM operations performed are error free, so that the deployed GNN can be executed correctly, it is necessary to manage various sources of noise in the analog photonic domain. There are several major noise sources in the photonic computing substrate, including thermal crosstalk, heterodyne (or incoherent) crosstalk, and homodyne (or coherent) crosstalk. The thermal crosstalk between TO tuning circuits is mitigated using our TED-based tuning mechanism (Section 3.1). But we still have to mitigate the impact of heterodyne and homodyne crosstalk in our design.

The presence of multiple wavelengths (or channels) in the same waveguide causes heterodyne or inter-channel crosstalk, where a portion of an optical signal from neighboring wavelengths (say λ_1 and λ_3) can leak into the MR spectrum of another wavelength (say λ_2) (see Figure 16(c)). This power leak causes MR2 output to have λ_1 and λ_3 content, and the MR downstream (in this example MR3), will receive lower signal power than designed for. Thus, the output from MR2 and MR3 will be erroneous. For this design optimization, we model heterodyne crosstalk using the following equations:

$$P_{signal} = \Phi(\lambda_i, \lambda_j, Q_{factor}) P_s(\lambda_i, \lambda_j), \tag{20}$$

$$P_{het\_noise} = \sum_{i=1}^{n} \Phi(\lambda_i, \lambda_j, Q_{factor}) P_S(\lambda_i, \lambda_j)(i \neq j), \quad (21)$$

where Φ is the crosstalk coupling factor, i.e., the spectra overlap between the two neighboring wavelengths, Q_{factor} is the quality factor or Q-factor of the MR, and P_s is the input signal power to the MR.

Heterodyne crosstalk impacts signals with spectral overlap. To mitigate heterodyne crosstalk, this overlap should be minimized. This can be achieved by a well-designed channel spacing and Q-factor tuning while ensuring that the signal-to-noise ratio (SNR) in the output is higher than the photodetector sensitivity. Another factor to be considered is the tunable range of the designed MRs. The MRs should provide adequate Q-factor to improve SNR, but should also possess sufficient tunable range, i.e., 2×FWHM (FWHM=full width half maximum), so that necessary parameters can be imprinted error free. To mitigate heterodyne crosstalk, we optimize our design for high FWHM and SNR, where SNR is expressed as:

$$SNR = 10 \times \log_{10}(P_{signal}/P_{noise}), \quad (22)$$

and, given λ_{res} as the resonant wavelength of the MR being considered, FWHM can be modeled as:

$$FWHM = \lambda_{res}/Q_{factor}. \quad (23)$$

Homodyne or coherent crosstalk is a result of undesired mode coupling among signals of the same wavelength [64]. In some of the computation circuits in *GHOST* we rely on coherent signal processing. In such circuits, part of the signal on the same wavelength may leak through a device and experience a different phase. Such leaked signals interfere with the output signal (based on their phase difference with the output signal) as coherent crosstalk noise. The presence of homodyne crosstalk, similar to heterodyne crosstalk, impacts the SNR of the non-coherent optical circuitry. The homodyne crosstalk noise power can be modeled as follows:

$$P_{hom\_noise} = \sum_{i=1}^{n} P_{in} \cdot X_{MR}^{i}(\rho) \cdot L_{P}^{n-i}, \tag{24}$$

where P_{in} is the input optical power, $X_{MR}^{i}(\rho)$ is the crosstalk contribution from the i^{th} MR in a bank of n MRs, and ρ is the optical phase of the crosstalk signal, which is a function of the EO tuning voltage. ρ does not take into account phase errors from thermal crosstalk, as TED is employed to address those errors. Finally, L_{P}^{n-i} is the passing loss that the crosstalk signal experiences as it propagates through MRs in the coherent circuit.

For homodyne crosstalk mitigation we may increase the cross over coupling by increasing the gap between the input waveguide and ring waveguide. This reduces the amount of crosstalk signal being coupled over from the MR to the main waveguide, reducing the impact of crosstalk on the output signal. For achieving this, while meeting SNR constraints (discussed later) the Q_{factor}, attenuation in MR (a), and cross-over coupling coefficient (κ) has to be fine-tuned as follows [64]:

$$Q_{factor} = \frac{\pi n_g L \sqrt{(1-\kappa^2)a}}{\lambda_{MR}(1-a(1-\kappa^2))} \,. \tag{25}$$

Using our noise models described in (20), (21), and (24), we can identify the optimal design space for our MR banks which can ensure high SNR and a high tunable range (R_{tune}). We must also consider that the lowest optical power level (P_{lpar}) should be higher than P_{noise}.

$$P_{lpar} > P_{noise} \,, \tag{26}$$

$$\frac{P_{signal}}{P_{lpar}} < \frac{P_{signal}}{P_{noise}}, \tag{27}$$

$$10\log_{10}\left(\frac{P_{signal}}{P_{lpar}}\right) < 10\log_{10}\left(\frac{P_{signal}}{P_{noise}}\right), \tag{28}$$

where P_{lpar} can be defined in terms of P_{signal}, from (1), as follows:

$$P_{lpar} = \frac{P_{signal} \times R_{tune}}{N_{levels}} \,. \tag{29}$$

Replacing P_{lpar} in (28) yields the following relation:

$$10\log_{10}\left(\frac{N_{levels}}{R_{tune}}\right) < SNR. \tag{30}$$

Here, N_{levels} is the number of amplitude levels we need to represent across the available R_{tune}. For n-bit GNN parameter representation, N_{levels} will be 2^n. If positive and negative values are represented separately, as in the case with *GHOST*, then N_{levels} will be 2^{n-1}. The relationship in (30) can be rearranged to as follows:

$$\frac{2 \times \lambda_{MR}}{Q_{factor}} > N_{levels} \times 10^{-\frac{SNR}{10}} \tag{31}$$

Utilizing these models, we can identify the design space for our MRs and the MR banks they constitute, in terms of the Q_{factor}, N_{levels}, and *SNR*. We can obtain values for $\Phi(\lambda_i, \lambda_j, Q)$ in (1) and (2) and $X_{MR}(\rho) \cdot L_P^{n-i}$ in (5) through multi-physics simulations. The results from our exploration studies using our detailed models and the simulation tool suite from Ansys Lumerical [100] are presented later in Section 4.2.

3.3.3 GHOST ARCHITECTURE DESIGN

As illustrated in Figure 17, the main units in the *GHOST* architecture (aggregate, combine, update) are divided into *V* execution lanes. During inference, each lane is assigned one output vertex to process, in parallel with all the other lanes. The aggregate block gathers all neighbor vertices and the associated edge data, and performs a reduce function for each of the assigned output vertices. The combine block then applies a linear transformation on each aggregated vertex feature vector h_v^a. Finally, the update block applies a non-linear activation function to obtain the updated vertex feature vectors $h_v^{`}$. At the start of processing a GNN model, when processing the first layer, the aggregate block reads the graph data from the vertex and edge buffers in the ECU, which interface directly with main memory. As updated vertex data is

computed, it is placed in the intermediate vertex buffer and thus, the aggregate block would read the vertex data when processing the next layers from the intermediate vertex buffer.

3.3.3.1 AGGREGATE BLOCK

As most graphs can be extremely sparse and irregular, regulating their memory accesses to improve performance is a challenge. *GHOST* alleviates this bottleneck through employing a "buffer and partition" optimization technique which is explained in Section 3.4.1. In this technique, the source and destination vertices in the input graph are split into blocks of *N* and *V*. The aggregate block is thus composed of *N* edge control units, *V* gather units, and *V* reduce units. In each cycle, the *V* execution lanes in *GHOST* are assigned to one output vertex group at a time. The edge control units fetch *N* input nodes simultaneously and then each unit forwards its fetched node and edge data to all gather units to convert the vertex data to analog signals, which are used to tune the MRs in the reduce unit. The corresponding edge data is used by the gather units to define whether an input vertex is a neighbor of the assigned output vertex. Accordingly, the total delay of the aggregate block is dependent on the node with the largest number of neighbors.

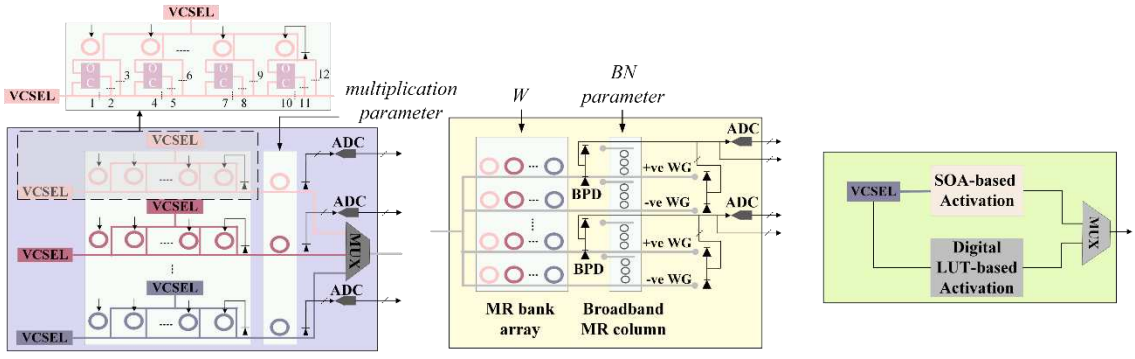


Figure 18: (a) Reduce unit showing the needed changes in in each feature lane to support the max aggregation operation using an optical comparator; (b) detailed view of transform unit; (c) detailed view of activate unit.

The reduce unit is an optical unit configured as a coherent summation block. Each row in the reduce unit is assigned a feature from the vertex feature vectors, while each column is assigned a neighbor input vertex (see Figure 18(a)). The rows and columns have the sizes R_r and R_c, respectively. Thus, one reduce unit can aggregate R_r features of R_c neighbor vertices. Different VCSEL and waveguide colors in the figure represent different optical wavelengths. Each VCSEL source generates an optical signal which is split into R_c signals. These R_c signals are then passed through the MR bank to imprint the neighbor nodes' feature values onto the signals. When the different waveguides carrying signals with the same wavelength meet, they undergo interference, resulting in the summation of the two values. Using a VCSEL phase locking mechanism, it can be guaranteed that all output signals have the same phase, ensuring the occurrence of constructive interference [62]. Since for a particular vertex the number of its neighbors may be greater than R_c, multiple mappings of different source vertices may be needed. Accordingly, the output of each row in the reduce unit is converted to an analog signal using a photodetector (PD), and used to tune the last MR in each lane (Figure 18(a)) such that the sum that is output from that cycle will be added to the feature values in the next cycle.

The organization of the reduce unit allows for the flexibility of implementing a wide range of reduce operations, which encompass most, if not all, GNN models. After the summation step as explained above, the output of the reduce unit is ($h_v^a = h_v + \sum_{u=0}^{n} h_u$). The last MR after the coherent summation block is used to implement the mean operation where the output summation value would be adjusted by the MR such that it is multiplied by

$\frac{1}{number\ of\ neighbors\ (n)}$, resulting in a reduce unit output of ($h_v^a = h_v + \frac{1}{n}\sum_{u=0}^{n} h_u$). Further, for implementing other reduce operations, such as maximum, the reduce unit includes an optical comparator [101]. An example of one lane in the reduce unit in that case would be as shown at the top of Figure 18(a). By activating blockers 1, 3, 4, 6, 7, 9, 10, and 12, the output of the reduce unit becomes the maximum value among all nodes. The optical output from all the rows in the reduce unit would then be combined into a single waveguide. This optical waveguide is connected directly to transform units in the combine block (discussed in the next subsection) to undergo the needed linear transformations. The same output values may need to be passed to the transform unit multiple times, depending on the size of the weight matrix. Accordingly, as the output from the reduce unit is passed directly to the transform units, it will be converted to the digital domain and buffered.

3.3.3.2 COMBINE BLOCK

The combine block accumulates the results from the aggregate block and performs linear transformation using the learned weight parameters. This generates a new learned, more expressive representation that captures the important structural information of the graph. Additionally, the linear transformation usually results in a reduced dimensionality for the vertex data representation, making the model more efficient as the dimensionality of some graphs can be very large (as shown later in Table 5). The transform unit's linear transformation is computed in the optical domain using MR bank arrays, as shown in Figure 18(b). Since linear transformation operations in GNNs are mainly MVMs, where the feature vector of the vertex being computed is multiplied by the weight matrix, as discussed in Section 2.4, such operations can be computed in the optical domain using MR bank arrays by passing the weight values to the

transform unit as analog signals to tune each MR, while the feature vectors values are imprinted onto the optical signals in the waveguide from the reduce units. The number of rows in a reduce unit R_r is equal to the number of columns in a transform unit such that the number of optical signals in the waveguide is equal to the number of MRs in one row in the transform unit.

As batch normalization (BN) is commonly used in many GNNs after the linear transformation, *GHOST* leverages broadband MR devices to perform BN in the optical domain where the BN parameter is used to tune the broadband MRs to adjust the optical signals as needed to reflect the BN operation. The efficiency of performing BN optically with this configuration was demonstrated in [101]. It is important to note that the BN unit can be bypassed if not needed. The output from each row is then accumulated using balanced photodetectors (BPD). BPDs are photodetectors that have two separate arms for the same waveguide, one for positive and the other for negative signal polarities. This allows them to accommodate both positive and negative parameter values by detecting the absolute difference between the two signals. The BPD sums the output signal from the positive arm and the output signal from the negative arm separately. Then, the BPD subtracts the output signal from the negative arm from the output signal from the positive arm to obtain the net difference signal.

If the size of rows in the weight matrix is smaller than or equal to the number of columns in the transform bank array and only one mapping for each weight matrix row is needed, the output from the transform unit after the BPDs will be passed directly to the activate units. Otherwise, the output will need to be converted to the digital domain and buffered till all needed values are computed and accumulated, and then passed to the activate units in the update block (discussed in the next subsection). *GHOST*'s versatility to adapt to different model architectures and sizes, and not having to always convert the values to the digital domain, greatly

reduces the latency and power costs associated with analog-to-digital converters (ADCs) and buffering.

3.3.3.3 UPDATE BLOCK

The update block is composed of V update units to apply a non-linear activation function to the output from the transform units. The work in [102] demonstrated how semiconductor-optical-amplifiers (SOAs) can be exploited to implement multiple non-linear functions such as *RELU*, *sigmoid*, and *tanh*. For example, when the gain in an SOA is adjusted to a value close to 1, the behavior resembles the *RELU* operation. Accordingly, such non-linear operations are implemented optically, resulting in considerably improved performance. The analog signals output from the transform units are used to directly drive VCSELs, generating optical signals with the output value imprinted into its amplitude, which is then passed through the SOA-based non-linear unit. For the non-linear activation functions that are harder to implement optically (such as softmax), a digital activation unit, such as the one described in [103] can be integrated to accommodate these functions using look-up tables (LUTs) and simple digital circuits, such as add and subtract. One update unit consists of T_c rows of activate units, in compliance with the number of rows used in each transform unit. Accordingly, the output from each row of the transform unit corresponds to one value in the vertex data vector, as shown in Figure 18(c).

3.3.4 ORCHESTRATION AND SCHEDULING OPTIMIZATIONS

GHOST supports four main optimizations for efficient orchestration and scheduling of GNNs, namely 1) graph buffering and partitioning, 2) execution pipelining and scheduling, 3) weight digital-to-analog converters (DACs) sharing, and 4) workload balancing. While

performing computations in the optical domain already offers significant performance and energy benefits, efficient optimizations targeting improved memory bandwidth utilization and enhanced execution flow are imperative for designing a scalable and robust GNN accelerator.

3.3.4.1 GRAPH BUFFERING AND PARTITIONING

Retrieving the entire graph from memory and processing it all at the same time entails tremendous memory bandwidth, resources, and computational costs. Hence, dividing the graph into several partitions and executing them separately is a widely used GNN optimization. But utilizing this approach alone can lead to increased latency as, in the worst case, while processing one vertex, it might necessitate loading another partition for each of its neighbor vertices. *GHOST*, on the other hand, uses a modified partitioning algorithm that builds on the one presented in [99]. As information regarding the graph edges is input, the adjacency matrix for the graph is generated. Columns of the adjacency matrix are identified as destination/output vertices while rows are the source/input vertices. Output and input vertices are then partitioned into groups of sizes V and N, respectively. Consequently, the edges are also grouped into $V \times N$ chunks. For each partition where the group V_i is assigned to the vertex processing lanes, if for input nodes N_i, the corresponding group of edges contains one or more connected edges, N_i is prefetched and assigned to the edge control units, while all-zero blocks are skipped entirely. Generating the partition matrices and determining the memory access and fetching order is done once offline, as part of a graph preprocessing step. This significantly reduces sparsity in the graph and the need for complex techniques to deal with performing sparse operations. It also helps overcome the GNN's memory bottleneck and eases greatly the traffic and access frequency to and from the main memory.

3.3.4.2 EXECUTION PIPELINING AND SCHEDULING

GHOST can efficiently adapt the pipelining, computation scheduling, and execution ordering depending on the GNN model, as each model can require a different sequence of execution. For example, GCNs usually mandate having all nodes gathered first, reduced, transformed, and then updated. In contrast, GATs initially compute the attention coefficients, which involves gathering the nodes, performing linear transformations, and then applying a non-linear activation. In this case, the reduce step is performed at the end.

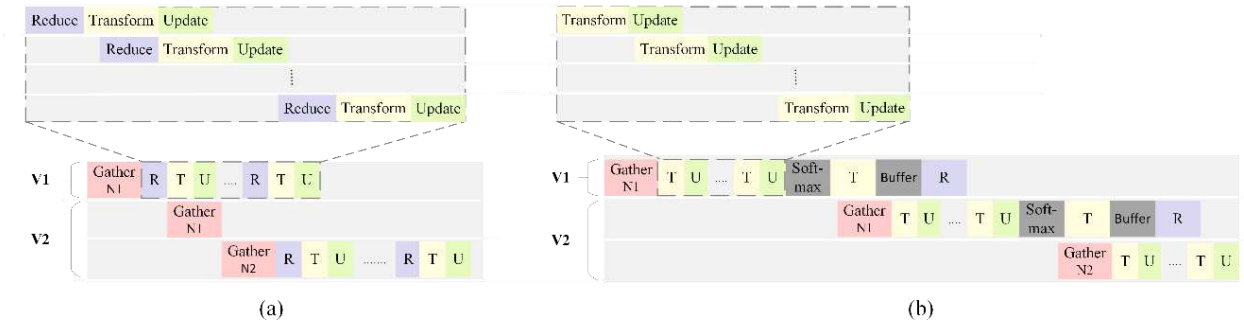


Figure 19: *GHOST* pipelining within one output vertex group V_1 (top), and the pipelining between output vertex groups V_1, V_2, where V_1 requires input vertices in N_1 while V_2 requires N_1 and N_2 for (a) GCN, GraphSAGE, GIN; and (b) GAT.

Given the buffering and partitioning optimization discussed in Section 3.4.1, *GHOST* performs pipelining at two levels of granularity. The first pipelining technique entails pipelining the operations within one output vertex group V_i. Initially as the output vertices are assigned to the execution lanes, all of their neighbor nodes are gathered. In case of models such as GCN,

GraphSAGE, and GIN, which perform aggregation first and then the transform and update, *GHOST* pipelines the reduce, transform, and update executions. As soon as all the needed input vertices are gathered, the reduce units are initiated. Transform units are activated when R_r feature values are ready and output from the reduce units, without the need to wait for all feature values to be computed. Similarly, the update units begin updating the vertex data directly after T_r values are linearly transformed. For the second pipelining technique between different vertex groups, the operations for a group V_{i+1}, are pipelined with those of group V_i. This scheduling approach ensures that the initial reduce for V_{i+1} is activated after the last reduce for group V_i. The pipelining model is illustrated in Figure 19(a).

For other GNN models such as GATs, where transforming the vertex data occurs prior to the aggregation, the pipelining model is tailored to fit the model's specifications. Figure 19(b) shows an example of pipelining for a GAT model. For pipelining with one output vertex group, as the transformation of each vertex fetched by the edge control units is independent of the other vertices, gather operations are pipelined with the transform operations without the need to wait till all needed partitions are fetched and processed. Hence, the first group of input vertices is gathered, and the first linear transformation of multiplying the input with matrix W is handled by transform units, followed by attention vector multiplication. The output is then concatenated with transformed output vertices updated with a *leakyRELU* activation function by the activate units. When all the neighbor vertices $\in V$ are transformed, softmax is computed. Reduce is then computed at the end after the second transformation. Alternatively, when processing all output vertex groups V, the gather operations for the next output vertex group V_{i+1} are pipelined with the transform and update operations for vertex output group V_i.

3.3.4.3 WEIGHT DAC SHARING

A key characteristic of GNNs is that the same set of weights is applied across all vertices during processing. Accordingly, when all transform units in *GHOST* are operating with the same speed and processing vertices with the same feature sizes, the same weight values can be shared among all transform units. *GHOST* leverages this idea to implement a weight DAC sharing optimization. DAC devices are needed in *GHOST* to convert the digital values of weights that are read from the buffers in the ECU into analog signals. These analog signals are then used to tune optical MR devices to perform the linear transformations in each transfer unit. The DAC sharing optimization shares DACs across weights, thereby reducing the total number of DAC devices required, which is a significant factor in the power and latency budget of silicon-photonic accelerators, as normally one DAC device would be needed for each MR. With DAC sharing V (number of transform units) MRs would share the same DAC device and thus the total number of DAC devices is reduce by $\frac{\#MRs\ in\ combine\ block}{\#MRs\ in\ one\ transfer\ unit}$.

3.3.4.4 WORKLOAD BALANCING

Exploiting the buffer and partitioning optimization discussed in Section 3.4.1 implies having certain units/blocks in idle states during specific times. Due to some graphs' irregularity, the number of neighbors for each vertex within the same output vertex group V_i can vary considerably. As a result, when processing a GCN model for example, some gather units will be waiting in an idle state till the gather unit with the highest dimensionality vertex receives all its neighbor vertices. Our workload balancing optimization allows each lane to operate at its own rate without the need to wait while other lanes with higher dimensionality are still gathering their

needed vertices. Accordingly, when such lanes complete processing their assigned vertices, the workload of the other lanes will be split among the completed lanes. As a result, this can notably reduce the overall latency of the executing GNN model, especially when dealing with highly sparse and irregular graphs.

3.3.5 PROGRAMMING MODEL

GHOST's programming model is based on GReTA [93], an abstraction model tailored to processing a broad family of GNNs. GReTA utilizes four stateless user-defined functions (UDFs) to break down computation in each GNN layer, namely gather, reduce, transform, and activate. These UDFs are then performed in a series of three main execution phases: aggregate, combine, and update. Algorithm 1 explains the execution flow of the GReTA programming model as implemented by *GHOST*. *GHOST* takes as input the graph(s) as a set of edges and vertices represented by the notation *G(V,E)*, where edges are defined as two lists of vertices: destination/output (V), and source/input vertices (U). The aggregate phase iterates over all edges and invokes gather and reduce UDFs. The *Gather()* UDF collects each output vertex feature vector (h_v), its neighbor input vertices $(\forall h_u \in N(v))$ and the edges features $(h_{u,v})$ associated with the current output vertex being processed, and prepares a message value. The *Reduce()* UDF collects the messages that are output by gather, related to the same output vertex, and reduces them into a single value. The combine phase invokes the *Transform()* UDF where it takes in all the accumulated values for each vertex, employs the learned weight parameters, and performs linear transformation. Lastly, the update phase invokes the *Activate()* UDF and computes a non-linear activation function for each output vertex to generate the updated feature vectors for all vertices.

ALGORITHM 1: GHOST Programming Model

Input: Graph *G(V, E)*, source vertex feature data h_v, vertex feature data h_u, edges feature data $h_{u,v}$, weights *W*.

Output: Updated vertex feature data h_v'

1: // Edges Accumulate Phase
2: **for** each (u, v) in E:
3: $h_{v\_r} = \text{Reduce}(h_v, \text{Gather}(h_u, h_v, h_{u,v}))$
4: // Vertices Accumulate Phase
5: **for** each v in V:
6: $h_{v\_t} = \text{Transform}(h_v, W)$
7: // Update Vertices Phase
8: **for** each v in V:
9: $h_v' = \text{Activate}(h_{v\_t})$

3.4 EXPERIMENTAL RESULTS

3.4.1 SIMULATION SETUP

To evaluate our proposed *GHOST* accelerator, we developed a comprehensive simulator in Python to estimate the power and latency of the accelerator. *GHOST* was simulated with attention to both software mapping and hardware mapping. For the software mapping, graph data is used to generate the partition matrix and retrieve needed information about the graph, such as the maximum number of neighbors in each partition. Then, we consider the layer-wise mapping and operation of each GNN model, and the architectural requirement for the mapping. For the hardware mapping, we modeled optoelectronic and electronic devices and circuits, and composed these into the blocks of the accelerator architecture. Compact models were used to analyze the losses associated with the device operation and also to determine device latency. The performance and energy estimates of all buffers used in *GHOST* were obtained using CACTI

[72]. However, since CACTI only supports down to 20 nm technology, the obtained latency and energy values were scaled down to 7 nm using the set of scaling relations from [104]. For the softmax circuit needed for the GAT model using the update block, the design with a LUT and a maximum frequency of 294 MHz from [103] was utilized.

Table 4 displays the optoelectronic device and circuit parameters that were used during *GHOST*'s simulation-based analysis. Various factors were taken into account for assessing photonic signal losses, including waveguide propagation loss (1 dB/cm), splitter loss (0.13 dB [73]), combiner loss (0.9 dB [74]), MR through loss (0.02 dB [75]), MR modulation loss (0.72 dB [76]), EO tuning loss (6 dB/cm [63]), and TO tuning power (27.5 mW/FSR [64]). Also, as the number of wavelengths and waveguide length increase, so does the MR count, photonic loss, and required laser power consumption. Thus, the laser power required for each source used with multiple wavelengths in our architecture is modeled as follows:

$$P_{laser} - S_{detector} \geq P_{photo\_loss} + 10 \times \log_{10} N_{\lambda}, \tag{32}$$

where P_{laser} is the laser power (dBm), $S_{detector}$ is the PD sensitivity (dBm), N_{λ} is the number of laser sources/wavelengths, and P_{photo_loss} is the total optical loss (dB) due to the factors discussed.

Table 4: Parameters considered for *GHOST* analysis

Devices	Latency	Power
EO Tuning [63]	20 ns	4 μW/nm
TO Tuning [63]	4 μs	27.5 mW/*FSR*
VCSEL [62]	0.07 ns	1.3 mW
Photodetector [62]	5.8 ps	2.8 mW
SOA [62]	0.3 ns	2.2 mW
DAC (8 bit) [49]	0.29 ns	3 mW

ADC (8 bit) [50]	0.82 ns	3.1 mW

A diverse set of GNN models, tasks, and graph datasets were used in our analysis. The following GNN models were considered: GCN [27], GraphSAGE [93], GIN [23], and GAT [24]. Each model processed four different graph datasets with the properties outlined in Table 5. The node-classification graph datasets (Cora, PubMed, Citeseer, Amazon) were processed with GCN, GraphSAGE, and GAT, while GIN was used for processing the graph-classification datasets (Proteins, Mutag, BZR, IMDB-binary). Further, GCN and GraphSAGE were implemented with two layers, while the MLP in GIN was implemented with eight layers. For the GAT model, two layers were implemented with the first one leveraging eight attention heads while the second used one attention head. The PyG library [90] was used to train and analyze each model's accuracy as shown in Table 6. Our analysis indicated that 8-bit model quantization results in comparable algorithmic accuracy to models with full (32-bit) precision; thus, we targeted the acceleration of 8-bit precision GNN models.

Table 5: Graph Datasets Parameters

Dataset	(avg) #Nodes	(avg) #Edges	#Features	#Labels	#Graphs
Cora	2,708	10,556	1,433	7	1
PubMed	19,717	88,651	500	3	1
Citeseer	3,327	9,104	3,703	6	1
Amazon	7,650	238,162	745	8	1
Proteins	39	73	3	2	1113
Mutag	18	40	143	2	188
BZR	34	38	189	2	405
IMDB-Binary	20	193	136	2	1000

Table 6: GNN Models Performances

Model	Dataset	Accuracy (32-bit)	Accuracy (8-bit)

GCN	Cora	88.70%	88.90%
	PubMed	87.40%	87.30%
	Citeseer	74.90%	74.40%
	Amazon	94.00%	93.70%
GraphSAGE	Cora	71.70%	70.80%
	PubMed	77.50%	76.90%
	Citeseer	63.30%	65%
	Amazon	77.10%	76.90%
GAT	Cora	78.30%	77.90%
	PubMed	76.70%	77.90%
	Citeseer	70.20%	69.10%
	Amazon	94.38%	94.64%
GIN	Proteins	74.00%	73.40%
	Mutag	94.74%	94.74%
	BZR	65.85%	65.85%
	IMDB-Binary	77%	73%

In the following subsections, we present results from our analyses and experiments to determine optimal photonic device level configurations in *GHOST* (Subsection 4.2), the optimal values for *GHOST's* architectural parameters, which were discussed in Section 3.3, including V, N, R_r, R_c, and T_r, (Subsection 4.3), sensitivity analysis to assess the impact of the orchestration and scheduling optimizations that were discussed in Section 3.4 (Subsection 4.4), and comparison with GNN accelerators proposed in prior work (Subsection 4.5).

3.4.2 DEVICE-LEVEL ANALYSIS

To ensure error-free photonic device operation in *GHOST*, we must reduce various noise sources in the analog domain, and ensure higher SNR than that dictated by (30) (see Section 3.2). We performed our device-level optimization analysis using optoelectronic simulation tools from Ansys Lumerical [100]. For obtaining the operational characteristics of the active MRs, i.e., MRs with EO tuning as opposed to passive MRs without them, we used the FDTD, CHARGE, MODE solver, and INTERCONNECT tools [100]. FDTD was used to obtain the operational characteristics of the passive MR. The doping levels for the tuning junction's p and n doped

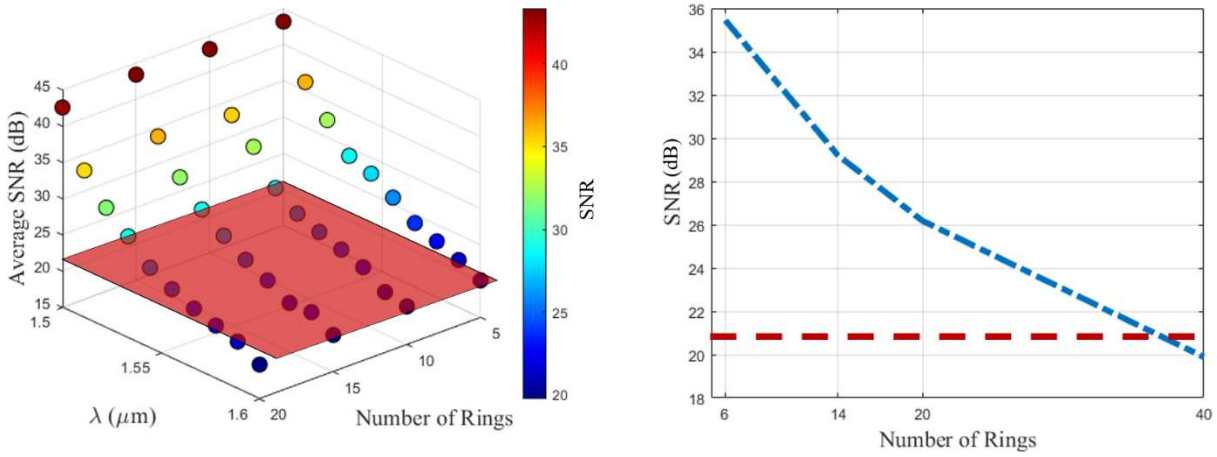
regions was assumed to be 4e19 in our CHARGE simulations, to obtain the carrier distribution in the ring, under various voltage conditions (0V to 10V range). Using the carrier distribution to voltage relationship obtained from CHARGE, we performed MODE solver simulations to obtain the shift in effective refractive index (neff) over the voltage range. Finally, the data from these simulations was used in our INTERCONNECT simulations to obtain the operational characteristics of the MR under different biasing voltages. These analyses were performed over a range of design parameters for the MR and λ_{MR} values. From these simulations, we obtained our ideal MR design to have: ring and input waveguide width at 450 nm, radius of 10μm, Gap of 300 nm, and a Q-factor of 3100. Using these parameters and (31) we can calculate the SNR required to be 21.3 Db.

The data from these tools was used for crosstalk and SNR analysis as mentioned previously in Section 3.2. Depending on the SNR requirements, the gap between the input and the ring waveguide and the width of the waveguides (input and ring) were explored and adjusted to achieve the required trade-off between Q-factor and the SNR at the output of the design as per (12). N_{levels} in our design is fixed from our software-level quantization to be 2^7, since we consider 8-bit quantization in our models (see Section 3.2 for discussion on this).

Using the device operational characteristics and the noise models from Section 3.2, we perform a sweep to determine the viable MR bank sizes for our coherent circuits ($P_{noise} = P_{\text{hom }\_noise}$) and non-coherent circuits ($P_{noise} = P_{het\_noise} + P_{\text{hom }\_noise}$), the results of which are shown in Figures 20(a) and 20(b).

For coherent MR banks we need to sweep for the wavelength to be used in the circuit, the number of MRs, and the SNR for the design. The cutoff SNR is shown as a red plane in the figure. From Figure 20(a), we can observe that it is possible to have up to 20 MRs in the

coherent summation circuit, when the resonant wavelength is 1520 nm, while satisfying the SNR requirements (red plane). Similarly, exploration for non-coherent circuits was also conducted and results are shown in Figure 20(b). The number of rings (MRs) along the x-axis is 2 times the number of wavelengths in the waveguide, as we need two MR banks to perform MAC operations. We considered the first wavelength to be 1550 nm and used a channel spacing of 1 nm between wavelengths. The red line in Figure 20(b) indicates the cut-off SNR. From this analysis, we determined that the waveguide can host 36 MRs, or 18 wavelengths (1550 nm to 1568 nm) for non-coherent operation. These results were used to size and design the MR banks within the main architectural blocks of *GHOST*, to meet SNR goals while maximizing performance.



(a) Coherent MR bank design sweep

(a) Non-coherent MR bank design sweep

Figure 20: Design space exploration for (a) coherent and (b) non-coherent MR banks for *GHOST* architecture.

3.4.3 ARCHITECTURE DESIGN SPACE EXPLORATION

The *GHOST* architecture relies on five main parameters, as outlined in Section 3: *N, V, R_r, R_c* and *T_r*. *N* refers to the number of edge control units or the size of each input vertices group

in the partition matrix, while V refers to the number of execution lanes, which is also the size of each output node group in the partition matrix. R_r is the number of rows in the coherent sum MR array in the reduce units, which is also the number of columns for the MR bank array in the transform units. Lastly, T_r is the number of rows for the MR bank array in the transform units. To identify the optimal configuration for *GHOST*, which is determined by the combination of $[N, V, Rr, Rc, Tr]$ that offers the lowest EPB/GOPS (where EPB is energy-per-bit and GOPS is giga-operations-per-second), we conducted a detailed design space exploration as shown in Figure 21. Using the *GHOST* simulator described in Section 4.1, the EPB and GOPs values were obtained for each GNN model and each accompanying graph dataset for a wide set of possible values for $[N, V, Rr, Rc, Tr]$. The average EPB/GOPs values across all the GNN models and datasets for each set of parameters were then obtained and the optimal configuration [20,20,18,7,17] was identified as the one with the lowest EPB/GOPS value.

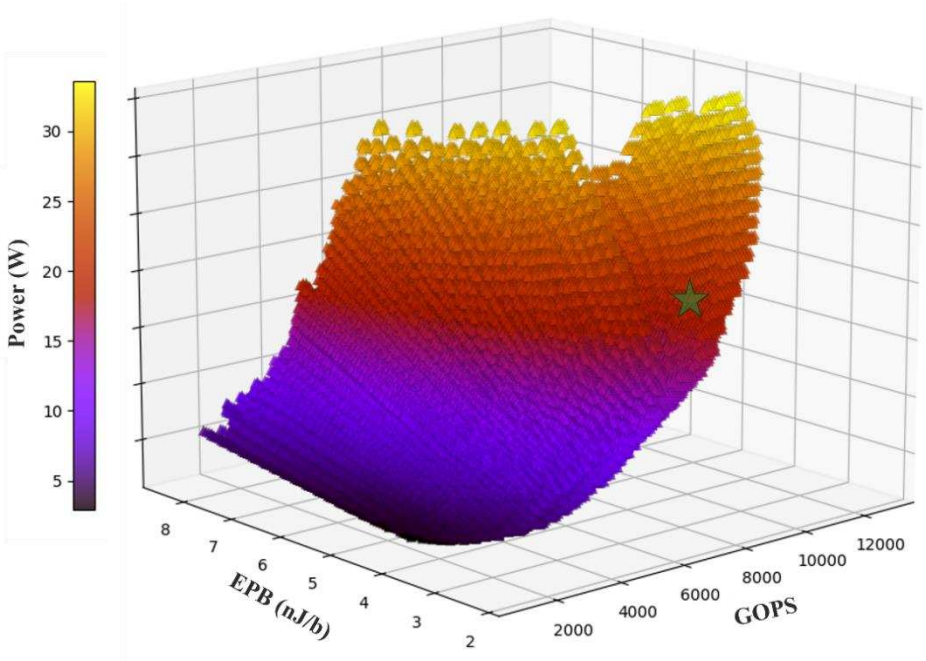


Figure 21: Architectural design-space exploration for *GHOST*, to find the optimal $[N, V, Rr, Rc, Tr]$ configuration with the best EPB/GOPS. The best configuration, $[20, 20, 18, 7, 17]$ is shown with the green star.

3.4.4 ORCHESTRATION AND SCHEDULING OPTIMIZATION ANALYSIS

We conducted a sensitivity analysis to assess the impact of each of the orchestration and scheduling optimizations described in Section 3.4. The normalized energy results are shown in Figure 22. The baseline configuration does not utilize any of the optimizations and each gather unit requests the needed neighbor vertices sequentially from the ECU. The buffer and partition (BP), pipelining (PP), and DAC weight sharing (DAC_Sharing) optimizations and their viable combinations were explored in this analysis. For workload balancing (WB), implementing it in isolation was found to be inefficient as the memory and buffer accesses are not synchronized and occur sequentially and on-demand. For instance, the processing lane that handles the vertex with the smallest dimensionality may not be the first to finish execution, as the order of memory access is also a critical factor. Moreover, employing WB necessitates having each lane possibly operating at different speeds, making it difficult to utilize the weight DAC sharing optimization. Therefore, implementing WB in isolation is impractical, and we only show results of considering WB alongside BP and PP to observe its benefits.

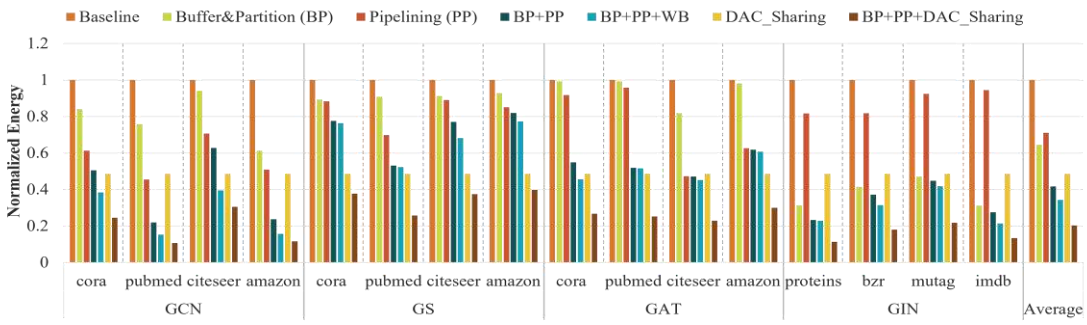


Figure 22: Impact of each orchestration and scheduling optimization on normalized energy consumption in *GHOST*.

The results shown in Figure 22 are normalized to the energy consumption of the baseline model. As can be observed, employing BP, PP, and DAC sharing optimizations simultaneously results in the least energy values for all the GNN models as well as all the graph datasets used. On average, when using DAC sharing combined with BP and PP, the energy consumption is reduced by 4.94× compared to the baseline. On the other hand, using BP, PP, and WB reduces the overall energy consumption by 2.92×. Hence, while *GHOST* supports all four optimizations described in Section 3.4, we leverage BP, PP, and DAC sharing for our *GHOST* configuration that is used in the subsequent sections for comparisons with other GNN accelerators.

Figure 22 also indicates that the different optimization techniques have different impacts across the various GNN models and graph datasets used. The optimizations have a greater impact with datasets having large number of vertices, and a very high degree of sparsity (e.g., PubMed). This demonstrates the scalability of *GHOST* to larger and more complex graphs and how the optimization techniques can alleviate the expensive memory and computational costs associated with GNN inference.

Another key observation is the effect of BP and PP when processing different graphs. When processing larger graphs such as Cora, PubMed, Citeseer, and Amazon, PP results in lower energy consumption values than BP. Conversely, for datasets used with GIN, BP leads to lower energy values. While the graph datasets used with GIN are each composed of multiple graphs, each individual graph in the datasets is considerably smaller than the other graphs used with GCN, GraphSAGE and GAT. Accordingly, as pipelining is determined based on each

individual graph, the impact of PP with small graphs diminishes. On the other hand, BP reduces sparsity and number of memory accesses. Consequently, as we need to offload an entire graph from memory every time *GHOST* starts processing a new graph, employing BP results in notable energy reductions.

3.4.5 COMPARISON WITH COMPUTING PLATFORMS AND STATE-OF-THE-ART GNN ACCELERATORS

GHOST is compared against multiple computing platforms and state-of-the-art GNN hardware accelerators: GRIP [17], HyGCN [96], EnG [95], HW_ACC [94], ReGNN [98], ReGraphx[99], Google TPU v4, Intel Xeon CPU, and NVIDIA A100 GPU. We used power, latency, and energy values reported for the selected accelerators, and directly obtained results from executing models on the GPU, CPU, and TPU platforms to estimate the EPB and GOPS for each model and graph dataset.

We compared each hardware accelerator on the models supported by them, as outlined in their papers. For the models used in our analysis (Table 6), the GRIP and HyGCN accelerators support processing GCN, GraphSAGE, and GIN; EnG supports GCN and GraphSAGE; and HW_ACC supports GCN and GAT. The ReRAM-based hardware accelerators, ReGNN and ReGraphx, were used for the GCN and GraphSAGE model comparisons. As mentioned, one of the key attributes of *GHOST* is its versatility in accommodating a diverse set of GNN models, enabling it to support the different models used in our analysis.

3.4.5.1 THROUGHPUT COMPARISON

Figure 23 shows the GOPS throughput comparison for *GHOST* with the computing platforms and GNN hardware accelerators considered. Our accelerator achieves on average 102.3×, 325.3×, 40.5×, 10.2×, 12.6×, 150.6×, 1699.0×, 1567.5×, 584.4× better GOPS when compared to GRIP, HyGCN, EnG, HW_ACC, ReGNN, ReGraphx, TPU, CPU, and GPU, respectively. While *GHOST* demonstrates promising improvements across all GNN models and datasets, the largest GOPS improvements are observed with the GIN graph dataset used for graph classification tasks. Across all datasets, processing GIN yielded on average 87.4× more GOPS when compared to the GNN hardware accelerators and 2168.9× when compared to GPU, CPU and TPU. This can be attributed to the small sizes of the graphs in each GIN dataset. These results are also consistent with those in Section 4.4, which illustrated that the partitioning optimization had the greatest impact on processing the graphs associated with the GIN model, leading to significant speedup. These findings highlight *GHOST*'s proficiency in handling diverse graph processing tasks. Also, there are significant GOPS improvements with *GHOST* for the GraphSAGE model, where it performed on average 100.9× better than other GNN accelerators and 1743.1× better than the GPU, CPU and TPU. This showcases how our accelerator is efficiently able to handle complex models, as it was able to overcome the complexity associated with supporting the sampling technique used in GraphSAGE.

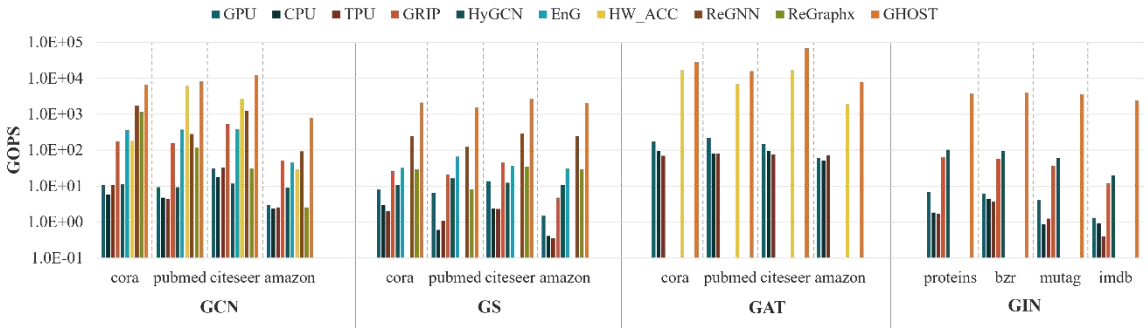


Figure 23: Throughput comparison between GPU, CPU, TPU, GNN hardware accelerators and *GHOST*.

3.4.5.2 ENERGY EFFICIENCY COMPARISON

The EPB comparison results for *GHOST* with the computing platforms and GNN accelerators considered are shown in Figure 24. On average, *GHOST* attains 11.1×, 60.5×, 3.8×, 85.9, 15.7×, 313.7×, 24276.7, 6178.8×, 2585.3× lower EPB compared to GRIP, HyGCN, EnG, HW_ACC, ReGNN, ReGraphx, TPU, CPU and GPU, respectively. Our accelerator exhibits lower EPB values across all the GNN models and the graph datasets. In particular, GCN, which is the most widely used GNN model, achieved the lowest EPB values, with an average EPB reduction of 116.7× with *GHOST*, when compared to the GNN hardware accelerators and an average reduction of 7120.4×, in comparison to GPU, CPU and TPU. This is due to GCN's uniform operation pofile, which involves processes each vertex independently based on its immediate neighbors only, without considering other vertices in the graph. Overall, the improved energy efficiency can be explained in terms of *GHOST*'s significant low latency operation in the optical domain, as well as its relatively low power consumption of 18W compared to the other hardware accelerators and compute platforms.

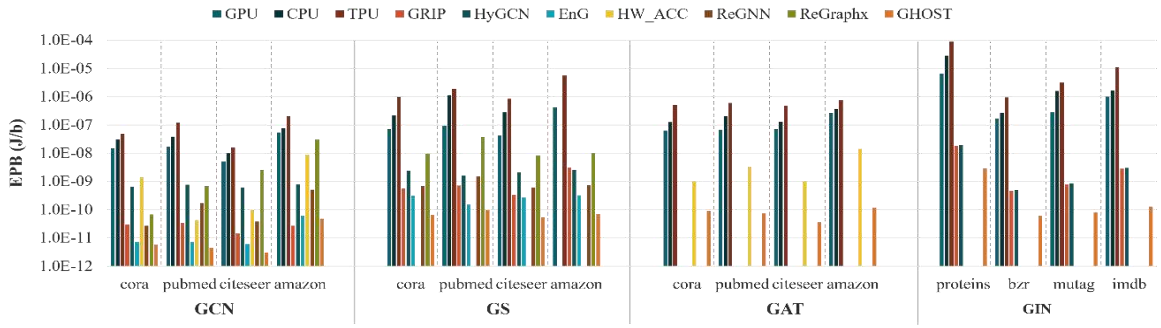


Figure 24: EPB comparison between GPU, CPU, TPU, GNN hardware accelerators and GHOST.

3.4.5.3 EPB/GOPS COMPARISON

The primary motivation for the design of *GHOST* is to obtain both an energy-efficient and high-throughput GNN acceleration. Accordingly, to showcase *GHOST*'s performance and energy improvements together, we show the EPB/GOPS comparison for *GHOST* against the computing platforms and GNN accelerators from prior work in Figure 25. It can be seen that *GHOST* achieves exceptionally lower EPB/GOPS values across all models and datasets. On average, *GHOST* attains 2.7e3×, 190.3e3×, 197.2×, 1.9e3×, 1.9e3×, 90.1e3×, 121.4e6×, 22.8e6×, 4.8e6× lower EPB/GOPS, compared to GRIP, HyGCN, EnG, HW_ACC, ReGNN, ReGraphx, TPU, CPU, and GPU, respectively. These results demonstrate how the cross-layer optimizations employed in *GHOST* across the circuit-level, device-level, and architecture-level along with performing most of the GNN operations in the optical domain, help to mitigate the various GNN inference challenges.

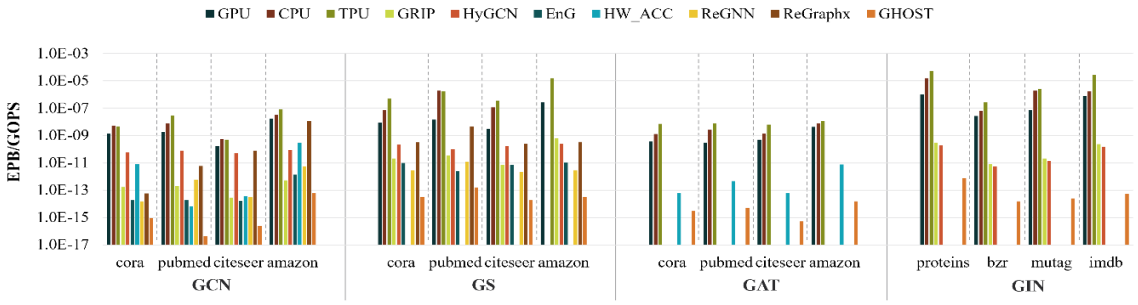


Figure 25: EPB/GOPS comparison between GPU, CPU, TPU, GNN hardware accelerators and *GHOST*.

3.5 CONCLUSION

In this chapter, we presented the first silicon photonic GNN accelerator, called *GHOST*. In comparison to nine computing platforms and state-of-the-art GNN accelerators, our proposed accelerator exhibited throughput improvements of at least 10.2× and energy-efficiency improvements of at least 3.8×. These results demonstrate the promise of *GHOST* in terms of energy-efficiency and high-throughput inference acceleration for GNNs. This work focused on the hardware architecture design with silicon photonics and employed various device-, circuit-, and architecture-level optimizations. When combined with software optimization techniques to reduce the high memory requirements in GNNs, we expect that even better throughput and energy efficiency can be achieved.

3. CONCLUSION AND FUTURE WORK SUGGESTIONS

4.1 RESEARCH CONCLUSION

In this thesis we presented two novel silicon-photonic-based accelerator architectures for transformers and GNNs. To the best of our knowledge, no previous work, in academia or the industry, has leveraged optical computations for accelerating such networks. Through our research, we demonstrate that employing optical computations and silicon-photonic-based architectures can significantly alleviate the challenges associated with transformer and GNNs inference.

As part of our first major contribution, we have proposed a non-coherent silicon photonic transformer accelerator called *TRON* (chapter 2). *TRON* efficiently supports computing most of the transformers and transformer-based models' operations in the optical domain. We were able to overcome energy and performance limitations that are normally seen with conventional electronic accelerators. Specifically, our proposed accelerator architecture has demonstrated significant improvements in both throughput and energy efficiency, outperforming eight different processing platforms and state-of-the-art transformer accelerators. We observed at least a 14× increase in throughput and an 8× improvement in energy efficiency. These results demonstrate the great promise of *TRON* in providing highly efficient and accelerated inference for transformer neural networks. Moreover, unlike existing state-of-the-art transformer accelerators, *TRON* can accelerate any existing variant of transformer neural network models.

Our second major contribution is *GHOST* (chapter 3), which is a GNN hardware accelerator using silicon photonics. *GHOST* efficiently alleviates various GNN inference challenges associated with both, edge-centric and vertex-centric operations. This is achieved

through leveraging optical computing and employing various optimization techniques across the device, circuit, and architecture levels. We have demonstrated that our proposed accelerator surpasses state-of-the-art GNN accelerators and nine computing platforms, achieving at least ×10.2 improvement in throughput and ×3.8 increase in energy efficiency. These results highlight the potential of GHOST to deliver highly efficient and accelerated inference for a broad family of GNN models, with significant improvements in energy efficiency and throughput.

4.2 SUGGESTIONS FOR FUTURE WORK

While both hardware accelerators presented in this thesis (*TRON* and *GHOST*) exhibited exemplary results and improvements in energy efficiency and throughput, the problem of designing hardware accelerators for such DNNs will keep getting more complex and require more demands. Hence, we envision the following research directions for future work:

- *Reliable hardware-software co-optimization aware design techniques:* both hardware accelerators presented in chapters 4 and 5, focus mostly on hardware accelerator design using silicon photonic devices and circuits. While *TRON* and *GHOST* exhibited notable results, combining them with software optimization techniques can further improve throughput and energy efficiency, and enable them to efficiently scale to future models and added complexities. Model compression software techniques, specifically, can complement silicon-photonic-based hardware accelerator architectures and considerably improve their performance. This is due to the limited precision most devices used in silicon-photonic architectures such as MRs, DACs, and ADCs, can accommodate.
- *Expand architecture design space and use optimized search method:* For both hardware accelerators, TRON and GHOST, we focused on a reasonable subset of possible

configurations in each accelerator's architecture design space and exhaustively explored all possible combinations of parameters in that subset. The optimal configuration was then identified as the design point with the lowest EPB/GOPs value. One way to possibly identify a more scalable and reliable configuration is to expand the design space such that it encompasses other parameters in the architectures and wider ranges for each chosen parameter. A search algorithm can then be adopted instead of exhaustively exploring the entire design space and a new objective can also be introduced such as minimizing area.

- *Non-volatile memory cells:* one of the main bottlenecks when designing silicon-photonic-based AI accelerators, is the need for intermediate buffering that requires multiple opto-electric conversions. These conversions are carried out using PDs, ADCs, and DACs devices which result in considerable latency and power overheads. One possible method to mitigate this issue is by using PCM-based photonic memory cells. The work in [105] proposes the design and implementation of an optical memory array and explores its scalability and power consumption when using different optical memory cells. Leveraging such a design would potentially improve the throughput and energy consumption of silicon-photonic-based hardware systems significantly.
- *Explore and leverage more optimizations tackling silicon-photonic device-level challenges*: in addition to the silicon photonic challenges we tackled in chapters 2 and 3, several other device-level challenges can be addressed to improve the overall performance of silicon photonic accelerators. Many efforts have already targeted elevating such challenges through various optimization techniques that can be used in hardware accelerators using silicon photonic devices. For example, the work in [47] describes how the effect of FPVs can be mitigated using techniques such as optical

channel remapping, intra-channel wavelength tuning and variation-aware routing, balanced homodyne locking, and using redundant devices.

BIBLIOGRAPHY

[1] C. Zhang, and Y. Lu. "Study on artificial intelligence: The state of the art and future prospects." *Journal of Industrial Information Integration* 23, 2021, p. 100224.

[2] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. “Revisiting unreasonable effectiveness of data in deep learning era”, *IEEE international conference on computer vision*, 2017, pp. 843–852.

[3] N.J. Nilsson, *Principles of artificial intelligence*, Morgan Kaufmann, 2014.

[4] V. Sze, Y. H. Chen, T. J. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in *Proceedings of the IEEE*, vol. 105, no. 12, Dec 2017, pp. 2295-2329.

[5] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, 1943, pp. 115–133.

[6] F. Rosenblatt, “The perceptron, a perceiving and recognizing automaton”, Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[7] F. Sunny, E. Taheri, M. Nikdast, and S. Pasricha, “A Survey on Silicon Photonics for Deep Learning”, *ACM Journal on Emerging Technologies in Computing Systems* (JETC), Vol. 17, Iss. 4, Oct 2021.

[8] Z. Zhan, J. Li, J. Zhang, “Evolutionary deep learning: A survey”, *Neurocomputing*, Apr 2022, pp. 42-58.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[10] Hofmann, Martin, Florian Neukart, and Thomas Bäck. "Artificial intelligence and data science in the automotive industry." *arXiv preprint arXiv:1709.01989*, 2017.

[11] D. Ravì, C. Wong, F. Deligianni, M. Berthelot, J. Andreu-Perez, B. Lo, and G. Z Yang, "Deep learning for health informatics." *IEEE journal of biomedical and health informatics* 21.1, 2016, pp. 4-21.

[12] S. Pasricha, V. Ugave, C. W. Anderson, and Q. Han, "LearnLoc: A framework for smart indoor localization with embedded mobile devices," *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2015, pp. 37-44 .

[13] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "INDRA: Intrusion detection using recurrent autoencoders in automotive embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2020, pp. 3698-3710.

[14] Abiodun, O. Isaac, A. Jantan, A. Omolara, K. Dada, N. Mohamed, and H. Arshad. "State-of-the-art in artificial neural network applications: A survey." *Heliyon* 4, no. 11, 2018, p. e00938.

[15] T. Takeuchi, A. J. Duszkiewicz, and R. G. Morris, "The synaptic plasticity and memory hypothesis: encoding, storage and persistence," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 369, no. 1633, 2014, p. 20130288.

[16] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, A. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." *IEEE Signal processing magazine*. Oct 2012, pp. 82-97.

[17] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning", *Nature* 521, no. 7553, 2015, pp. 436-444.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei. “ImageNet large scale visual recognition challenge”. *International Journal of Computer Vision*, 2016, pp. 211-252.

[19] I. Sutskever, O. Vinyals, and Q. Le. “Sequence to sequence learning with neural networks”. *Advances in neural information processing systems*, 2014, pp. 3104-3112.

[20] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur. “Recurrent neural network based language model”. *Eleventh annual conference of the international speech communication association*. Sep 2010, pp. 1045-1048.

[21] B. Widrow and M. E. Hoff, "Adaptive Switching Circuits", *IRE Transactions on Circuit Theory*, 1962, pp. 213-222. 1962.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors.” *Nature*, 1986, pp. 533-536.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser Ł, Polosukhin I. “Attention is all you need”, *Advances in neural information processing systems (NIPS)*, 30, 2017.

[24] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, “An image is worth 16x16 words: Transformers for image recognition at scale,” *International Conference on Learning Representations (ICLR)*, 2021.

[25] D. Bahdanau, C. Kyunghyun, and B. Yoshua. "Neural machine translation by jointly learning to align and translate." *arXiv preprint arXiv:1409.0473,* 2014.

[26] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, “Graph neural networks: A review of methods and applications”, *AI open*. Jan 2020, pp. 57-81.

[27] T. N. Kipf, and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks”. *International Conference on Learning Representations (ICLR),* 2017, pp. 1-14.

[28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, “Graph Attention Networks”, *International Conference on Learning Representations (ICLR '18)*, 2018, pp. 1-1.

[29] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, D. Amodei et al. “Language models are few-shot learners”. *Advances in Neural Information Processing Systems* (pp. 18704-18716). 2020.

[30] A. J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. “Language models are unsupervised multitask learners”. *OpenAI blog*, Feb 2019, p. 9.

[31] GPT-3: Language Models are Few-Shot Learners. OpenAI 2020. https://openai.com/blog/gpt-3-ai-language-model/. Last accessed 4/19/2023.

[32] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S. Y. Philip, “A comprehensive survey on graph neural networks”, *IEEE transactions on neural networks and learning systems*. Mar 2020, pp. 4-24.

[33] Personalized engagement on Amazon Alexa. Amazon Alexa Team, Amazon Developer, 2018. https://developer.amazon.com/en-US/docs/alexa/engage-your-users/personalized-engagement.html. Last accessed 4/19/2023.

[34] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. “Graph convolutional neural networks for web-scale recommender systems”. *24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining,* 2018, pp. 974-983.

[35] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina. “An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks”. *Future Internet*, 2020, p. 113. 2020.

[36] Y. Cao, X. Liu, and H. Xie, "FPGA-based accelerator for deep convolutional neural networks," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 161-170.

[37] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, “Deep learning with limited numerical precision”, *International conference on machine learning,* Jun 2015, pp. 1737-1746.

[38] M. M. Waldrop, “The chips are down for Moore's law”, *Nature News 530*, 2016, p. 7589.

[39] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, K. Yelick, “A case for intelligent RAM”, *IEEE Micro*. Mar 1997, pp. 34-44.

[40] S. Bavikadi, P. Sutradhar, K. Khasawneh, A. Ganguly, and S. Dinakarrao. “A Review of In-Memory Computing Architectures for Machine Learning Applications." *Great Lakes Symposium on VLSI (GLSVLSI '20)*, 2020, pp. 89-94.

[41] X. Qiao, X. Cao, H. Yang, L. Song, and H. Li, "AtomLayer: A Universal ReRAM-Based CNN Accelerator with Atomic Layer Computation," *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1-6.

[42] Y. Long, T. Na, and S. Mukhopadhyay, "ReRAM-Based Processing-in-Memory Architecture for Recurrent Neural Network Acceleration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Dec 2018, pp. 2781-2794.

[43] M. Zhou, W. Xu, J. Kang, T. Rosing, "TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer", *IEEE International Symposium on High-Performance Computer Architecture (HPCA),* Apr 2022, pp. 1071-1085.

[44] C. Liu, H. Liu, H. Jin, X. Liao, Y. Zhang, Z. Duan, J. Xu, H. Li, "ReGNN: a ReRAM-based heterogeneous architecture for general graph neural networks", *59th ACM/IEEE Design Automation Conference*, Jul 2022, pp. 469-474.

[45] N. Jouppi, C. Young, N. Patil, D. Patterson, "Motivation for and evaluation of the first tensor processing unit", *IEEE Micro*, May 2018, pp. 10-9.

[46] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, "Tensorflow: a system for large-scale machine learning", *Osdi*, Nov 2016 pp. 265-283.

[47] S. Pasricha and M. Nikdast, "A survey of silicon photonics for energy-efficient manycore computing", *IEEE Design & Test*. Mar 2020, pp. 60-81.

[48] S. A. Mittal, "Survey of ReRAM-Based Architectures for Processing-In-Memory and Neural Networks". *Mach. Learn. Knowl. Extr.* 2019, pp. 75-114.

[49] Lightmatter. https://lightmatter.co/. Last accessed 4/16/2023.

[50] C. Ramey, "Silicon Photonics for Artificial Intelligence Acceleration: HotChips 32." *2020 IEEE Hot Chips 32 Symposium (HCS)*, 2020, pp. 1-26.

[51] Intel Silicon Photonics Technology Overview. Intel Corporation, 2021. https://www.intel.com/content/www/us/en/architecture-and-technology/silicon-photonics/silicon-photonics-overview.html. Last accessed 4/16/2023.

[52] Have you seen the light?. Ayar Labs, 2021, https://ayarlabs.com/have-you-seen-the-light/#artificial-intelligence. Last accessed 4/16/2023.

[53] Hewlett Packard Labs. Hewlett Packard Enterprise Development LP, 2021, https://www.hpe.com/us/en/hewlett-packard-labs.html. Last accessed 4/16/2023.

[54] F. Sunny, M. Nikdast, S. Pasricha, "Cross-Layer Design for AI Acceleration with Non-Coherent Optical Computing", to appear, *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2023.

[55] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling", In *NIPS Workshop on Deep Learning,* December 2014.

[56] J. Devlin, M. W. Chang, K. Lee, K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", *arXiv preprint arXiv:1810.04805*, Oct 2018.

[57] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut,"Albert: A lite bert for self-supervised learning of language representations," *International Conference on Learning Representations (ICLR)*, 2019.

[58] S. Lu, M. Wang, S. Liang, J. Lin, Z. Wang, "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer", *IEEE 33rd International System-on-Chip Conference (SOCC),* Sep 2020, pp. 84-89.

[59] M. Sun, H. Ma, G. Kang, Y. Jiang, T. Chen, X. Ma, Z. Wang, Y. Wang, "VAQF: fully automatic software-hardware co-design framework for low-bit vision transformer", *arXiv preprint arXiv:2201.06618*, Jan 2022.

[60] P. Qi, E. H. Sha, Q. Zhuge, H. Peng, S. Huang, Z. Kong, Y. Song, B. Li, "Accelerating framework of transformer by hardware design and model compression co-optimization", *IEEE/ACM International Conference On Computer Aided Design (ICCAD),* Nov 2021, pp. 1-9.

[61] Z. Zhao, D. Liu, M. Li, Z. Ying, L. Zhang, B. Xu, B. Yu, R. T. Chen, and D. Z. Pan, "Hardware-software co-design of slimmed optical neural networks", *24th Asia and South Pacific Design Automation Conference,* Jan 2019, pp. 705-710.

[62] F. Sunny, M. Nikdast, and S. Pasricha, "RecLight: A Recurrent Neural Network Accelerator with Integrated Silicon Photonics", *IEEE Computer Society Annual Symposium on VLSI (ISVLSI),* Jul 2022, pp. 98-103

[63] S. Abel, T. Stöferle, C. Marchiori, D. Caimi, L. Czornomaz, M. Stuckelberger, M. Sousa, B. J. Offrein, J. Fompeyrine, "A hybrid barium titanate–silicon photonics platform for ultraefficient electro-optic tuning", *Journal of Lightwave Technology*. Apr 2016, pp. 1688-93.

[64] P. Pintus, M. Hofbauer, C. L. Manganelli, M. Fournier, S. Gundavarapu, O. Lemonnier, F. Gambini, L. Adelmini, C. Meinhart, C. Kopp, F. Testa, "PWM-Driven thermally tunable silicon microring resonators: design, fabrication, and characterization", *Laser & Photonics Reviews*, Sep 2019, p. 1800275.

[65] L. Lu, X. Li, W. Gao, X. Li X, L. Zhou, J. Chen, "Silicon non-blocking 4× 4 optical switch chip integrated with both thermal and electro-optic tuners", *IEEE Photonics Journal*. Sep 2019, pp. 1-9.

[66] M. Milanizadeh, D. Aguiar, A. Melloni, and F. Morichetti, "Canceling thermal cross-talk effects in photonic integrated circuits", *Journal of Lightwave Technology.* Jan 2019, pp. 1325-32.

[67] S. Chittamuru, I. Thakkar, and S. Pasricha, "HYDRA: Heterodyne crosstalk mitigation with double microring resonators and data encoding for photonic NoCs", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. Sep 2017, pp. 168-81.

[68] C. Li, C. Graves, X. Sheng, D. Miller, M. Foltin, G. Pedretti, and J. Strachan, "Analog content-addressable memories with memristors", *Nature communications*. Apr 2020. p. 1638.

[69] C. Wang, C. Li, J. Dai, Z. Wang, "Study on in-chip phase locked high brightness bottom emitting Talbot-VCSELs array", AOPC 2020: *Advanced Laser Technology and Application (AOPC),* Nov 2020 pp. 7-12.

[70] K. Vandoorne, J. Dambre, D. Verstraeten, B. Schrauwen, P. Bienstman, "Parallel reservoir computing using optical amplifiers", *IEEE transactions on neural networks*, Jul 2011, pp. 1469-81.

[71] D. Hendrycks, K. Gimpel, "Gaussian error linear units (gelus)", *arXiv preprint arXiv:1606.08415*, Jun 2016.

[72] HP Labs. CACTI. https://www.hpl.hp.com/research/cacti/. Last accessed 4/15/2023.

[73] L. Frandsen, P. Borel, Y. Zhuang, A. Harpøth, M. Thorhauge, M. Kristensen, W. Bogaerts, P. Dumon, R. Baets, V. Wiaux, and J. Wouters, "Ultralow-loss 3-dB photonic crystal waveguide splitter", *Optics letters*, Jul 2004, pp. 1623-5.

[74] Y. Tu, P. Fu, and D. Huang, "High-efficiency ultra-broadband multi-tip edge couplers for integration of distributed feedback laser with silicon-on-insulator waveguide", *IEEE Photonics Journal*, Jun 2019, pp. 1-3.

[75] S. Pasricha, S. Bahirat, "OPAL: A multi-layer hybrid photonic NoC for 3D ICs", *16th Asia and South Pacific Design Automation Conference (ASP-DAC),* Jan 2011, pp. 345-350.

[76] H. Jayatilleka, M. Caverley, N. Jaeger, S. Shekhar, and L. Chrostowski, "Crosstalk limitations of microring-resonator based WDM demultiplexers on SOI", *IEEE Optical Interconnects Conference (OI),* Apr 2015, pp. 48-49.

[77] C. M. Yang and T. H. Kuo, "A 3 mW 6-bit 4 GS/s subranging ADC with subrange dependent embedded references," *IEEE Transactions on Circuits and Systems II: Express Briefs (TCAS)*, Jan 2021.

[78] L. Kull, T. Toifl, M. Schmatz, P. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. Andersen, Y. Leblebici, "A 3.1 mW 8b 1.2 GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32 nm digital SOI CMOS", *IEEE Journal of Solid-State Circuits*, Sep 2013, pp. 3049-58.

[79] "System architecture | cloud TPU | google cloud," Google. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm. Last accessed 4/15/2023.

[80] L. Kull, T. Toifl, M. Schmatz, P. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. Andersen, and Y. Leblebici, "A 3.1 mW 8b 1.2 GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32 nm digital SOI CMOS", *IEEE Journal of Solid-State Circuits*, Sep 2013, pp. 3049-58.

[81] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors", *Nature*, Oct 1986, pp. 533-6

[82] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications", *AI open*, Jan 2020, pp. 57-81.

[83] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, "Computing graph neural networks: A survey from algorithms to accelerators", *ACM Computing Surveys (CSUR)*, Oct 2021, pp. 1-38.

[84] A. Ziabari, J. Abellán, R. Ubal, C. Chen, A. Joshi, and D. Kaeli, "Leveraging silicon-photonic noc for designing scalable gpus", *29th ACM on International Conference on Supercomputing*, Jun 2015, pp. 273-282.

[85] F. Sunny, A. Mirza, M. Nikdast, and S. Pasricha, "CrossLight: A cross-layer optimized silicon photonic neural network accelerator", *58th ACM/IEEE Design Automation Conference (DAC),* Dec 2021, pp. 1069-1074.

[86] V. Bangari, B. Marquez, H. Miller, A. Tait, M. Nahmias, T. De Lima, H. Peng, P. Prucnal, and B. Shastri, "Digital electronics and analog photonics for convolutional neural networks (DEAP-CNNs)", *IEEE Journal of Selected Topics in Quantum Electronics*, Oct 2019, pp. 1-3.

[87] F. Scarselli, M. Gori, A. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model", *IEEE transactions on neural networks*, Dec 2008 pp. 61-80.

[88] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs", *Advances in neural information processing systems (NeurIPS)*, 2017, pp. 1024-1034.

[89] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[90] M. Fey, and J. E. Lenssen. "Fast graph representation learning with PyTorch Geometric". *International Conference on Learning Representations (ICLR)*, 2019.

[91] M. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs", *ICLR workshop on representation learning on graphs and manifolds*, Jan 2019.

[92] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, “NeuGraph: Parallel Deep Neural Network Computation on Large Graphs”, *USENIX Annual Technical Conference,* Jul 2019, pp. 443-458.

[93] K. Kiningham, P. Levis, and C. Re. “GReTA: Hardware optimized graph processing for GNNs”. *Workshop on ReCoML’20*. 2020.

[94] A. Auten, M. Tomei, and R. Kumar. “Hardware acceleration of graph neural networks.” *Design Automation Conference (DAC)*, 2020.

[95] L. Shengwen, W. Ying, L. Cheng, H. Lei, L. Huawei, X. Dawen, and L. Xiaowei, “EnGN: A High-Throughput and Energy-Efficient Accelerator for Large Graph Neural Networks”, *IEEE Trans. Comput.*, Sept. 2021, pp. 1511–1525.

[96] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, “Hygcn: A gcn accelerator with hybrid architecture”, *IEEE International Symposium on High Performance Computer Architecture (HPCA),* Feb 2020, pp. 15-29.

[97] K. Kiningham, P. Levis, C. Ré, “GRIP: A graph neural network accelerator architecture”, *IEEE Transactions on Computers*, Aug 2022.

[98] C. Liu, H. Liu, H. Jin, X. Liao, Y. Zhang, Z. Duan, J. Xu, H. Li, “ReGNN: a ReRAM-based heterogeneous architecture for general graph neural networks”, *59th ACM/IEEE Design Automation Conference (DAC),* Jul 2022, pp. 469-474.

[99] A. Arka, J. Doppa, P. Pande, B. Joardar, and K. Chakrabarty, “ReGraphX: NoC-enabled 3D heterogeneous ReRAM architecture for training graph neural networks”, *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Feb 2021, pp. 1667-1672.

[100] Online: https://www.ansys.com/products/photonics. Last accessed: 3/19/2023.

[101] D. Dang, S. Chittamuru, S. Pasricha, R. Mahapatra, and D. Sahoo, "BPLight-CNN: A photonics-based backpropagation accelerator for deep learning", *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, Aug 2021, pp. 1-26.

[102] K. Vandoorne, J. Dambre, D. Verstraeten, B. Schrauwen, and P. Bienstman, "Parallel reservoir computing using optical amplifiers", *IEEE transactions on neural networks*, Jul 2011, pp. 1469-1481.

[103] Z. Wei, A. Arora, P. Patel, and L. John, "Design space exploration for softmax implementations", *IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP),* Jul 2020, pp. 45-52.

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

[105] A. Shafiee, S. Pasricha, and M. Nikdast, "Design-Space Exploration in PCM-based Photonic Memory", *to appear, ACM GLSVLSI, 2023.*