AI-aided System and Design Technology Co-optimization Methodology Towards Designing Energy-efficient and High-performance AI Accelerators

by

Kaniz Fatema Mishty

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
December 14, 2024

Keywords: AI accelerator, MRAM, Chiplets, Reinforcement Learning, STCO, DTCO

Approved by

Mehdi Sadi, Chair, Assistant Professor of Electrical and Computer Engineering
Ujjwal Guin, Associate Professor of Electrical and Computer Engineering
Vishwani Agrawal, Professor of Electrical and Computer Engineering
Yin Sun, Associate Professor of Electrical and Computer Engineering
Akond Rahman, Assistant Professor of Computer Science and Software Engineering

Abstract

The rapid growth of artificial intelligence (AI) and deep learning (DL) workloads has created an urgent need for more efficient and high-performance AI accelerators, both at the edge and in cloud data centers. The computational and memory demands of large models, such as ChatGPT and Sora, have far outpaced advancements in semiconductor technology, leading to the emergence of the memory wall and area wall. These challenges necessitate the exploration of new technologies and methodologies. This dissertation presents a comprehensive investigation into emerging memory technologies, innovative architectural designs, and optimization methodologies aimed at improving energy efficiency, performance, and area utilization in AI accelerators.

First, we introduce a high-performance AI accelerator that incorporates spin transfer torque magnetic RAM (STT-MRAM) as the on-chip memory system. Through model-driven design space exploration, we develop a novel scratchpad-assisted buffer architecture that optimizes memory retention time, read/write latency, and energy efficiency by dynamically adjusting for process and temperature variations. Our STT-MRAM-based design (STT-AI) achieves a 75% reduction in area and 3% power savings compared to SRAM-based systems, with minimal trade-offs in accuracy, demonstrating its suitability for modern AI workloads.

Next, we address the limitations of existing accelerators in handling large-batch AI training and inference due to memory bandwidth and capacity constraints. We propose a design technology co-optimization (DTCO)-enabled memory system utilizing spin-orbit torque magnetic RAM (SOT-MRAM) to significantly increase on-chip memory capacity. The limitations posed by STT-MRAM are also addressed by introducing SOT-MRAM. This workload-aware memory system shifts AI accelerators from being memory-bound to achieving system-level peak performance. Our results show an 8× improvement in energy efficiency and 9× reduction in latency for computer vision benchmarks, along with substantial gains

in natural language processing tasks, while consuming just 50% of the area compared to SRAM at the same capacity.

Finally, to address the limitations of large monolithic designs, we explore the potential of chiplet-based architectures for AI accelerators. The vast design space and complex trade-offs between power, performance, area, and cost (PPAC) require a systematic optimization approach. We introduce an optimization framework, Chiplet-Gym, which integrates heuristic-based methods, such as simulated annealing (SA), with learning-based algorithms, such as reinforcement learning (RL), to evaluate and optimize chiplet-based AI accelerator designs by accounting for resource allocation, placement, and packaging architecture. Our results indicate that reinforcement learning demonstrates greater stability and achieves a 16% higher cost model value than simulated annealing. The framework-suggested design choice delivers a 1.52× improvement in throughput, a 0.27× reduction in energy, and a 0.89× lower cost compared to monolithic designs at iso-area, underscoring the potential of chiplet architectures for the next-generation AI hardware.

Acknowledgments

I would like to express my gratitude to my advisor, Dr. Mehdi Sadi, for his support and guidance throughout my PhD journey. I am profoundly grateful to the members of my dissertation committee — Dr. Ujjwal Guin, Dr. Vishwani Agrawal, Dr. Yin Sun, and Dr. Akond Rahman — for their valuable feedback, time, and support in helping me refine my research. Their input has been pivotal in improving the quality of this work.

I would also like to extend my heartfelt thanks to my colleagues and friends, whose collaboration and brainstorming sessions helped refine my ideas and made this challenging journey both manageable and enjoyable.

My deepest gratitude goes to my mother, sisters, and father, whose love and unwavering belief in me provided the strength I needed to persevere through the ups and downs of this journey. To my elder sister, Jawata Afnan Saba, thank you for your endless support, encouragement, and sacrifices. Special thanks to my younger sister, Sumaiya Tohorat, for always keeping me cheerful. This achievement would not have been possible without you.

Finally, I believe that everything has been possible because Almighty Allah (SWT) has paved the way for me. His guidance and blessings have given me the strength, wisdom, and perseverance to accomplish this work.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are in the era of the Fourth Industrial Revolution, characterized by the unprecedented success of Artificial Intelligence (AI), including advancements in Generative AI. These technologies have transformed our lives in every aspect, from performing monotonous tasks such as data entry and spam email filtering to creative endeavors such as creating artwork, producing images and videos, and understanding complex behaviors of time-series data, protein folding, and DNA structure. AI is revolutionizing various sectors, including healthcare, transport, agriculture, finance, business, education, entertainment, security, etc., to name a few. In many instances, AI-provided solutions have proven to be even more effective than those provided by humans, enhancing efficiency, accuracy, and innovation.

As a result, the demand for Deep Learning and Artificial Intelligence (AI) is growing at a rapid pace across a wide range of applications such, as self-driving vehicles, image and voice recognition, medical imaging and diagnosis, finance and banking, defense operations, etc. Because of these data-driven analytics and AI boom, demands in deep learning and AI will emerge at both data centers and the edge [9, 56, 155]. In a recent market research [56], it has been reported that AI-related semiconductors will see a growth of about 18 percent annually over the next few years - five times greater than the rate for non-AI applications. By 2025, AI-related semiconductors could account for almost 20 percent of all semiconductor demand, which would translate into about $67 billion in revenue [56]. As a result, significant R&D efforts in developing AI accelerators - optimized to achieve much higher throughput in deep learning compared to GPUs - are underway from academia, big techs, as well as startups [9]. In AI technology innovation and leadership, high-throughput AI accelerator hardware chips will serve as the differentiator [155, 56].

The great success of AI is fueled by the advancement in the semiconductor industry, the computing and storage capacity of computers, the Internet of Things (IoT), and the evolution of Big Data. We are seeing that more computation power is being made available with newer generations of GPUs and AI accelerators, interconnected at lightning speeds. However, the AI advancements are far outpacing the Moore's law. We continue to see hyperscaling of AI models leading to better performance, with seemingly no end in sight. A few of the prime reasons AI models achieve human-level performance are model size and training data set size. These models with hundreds of billions of parameters are trained on thousands of gigabytes of data. The power-law relationship between accuracy and data set growth and the sublinear relationship between model size and dataset size [1] imply that to keep pace with the accuracy improvement, dataset sizes will need to grow by 33–971×, while model sizes will need to grow by 6.6–456× [3]. Figure 1.1 shows the trend of growing model size, and Table 1 shows the dataset size of the commonly used datasets used to train the AI models in different domains. The large model and dataset sizes eventually translate into hundreds of exaflops of compute and terabytes of data movement while training/running these models. That being said, it is impossible to use (both in inference and training mode) the State-of-the-Art AI models with general purpose CPU. Though general-purpose GPUs are able to handle parallel processing, the huge amount of data movement makes them inefficient as well. As a result, Application Specific Integrated Circuits (ASIC)s have evolved, known as AI accelerators, in the last decade. These AI accelerators are specifically designed to take advantage of the inherent parallel computation and data reuse properties of the Deep Learning (DL) models to ensure performance and energy efficiency.

The existing DL/AI accelerators can sustain the computation demand of the current AI models, thanks to the inherent parallel property of the DL workloads. However, with the rapid growth of model and dataset size, even with the advancement in architectural innovation, it is hard to support the DNN/AI models training and inference. As shown

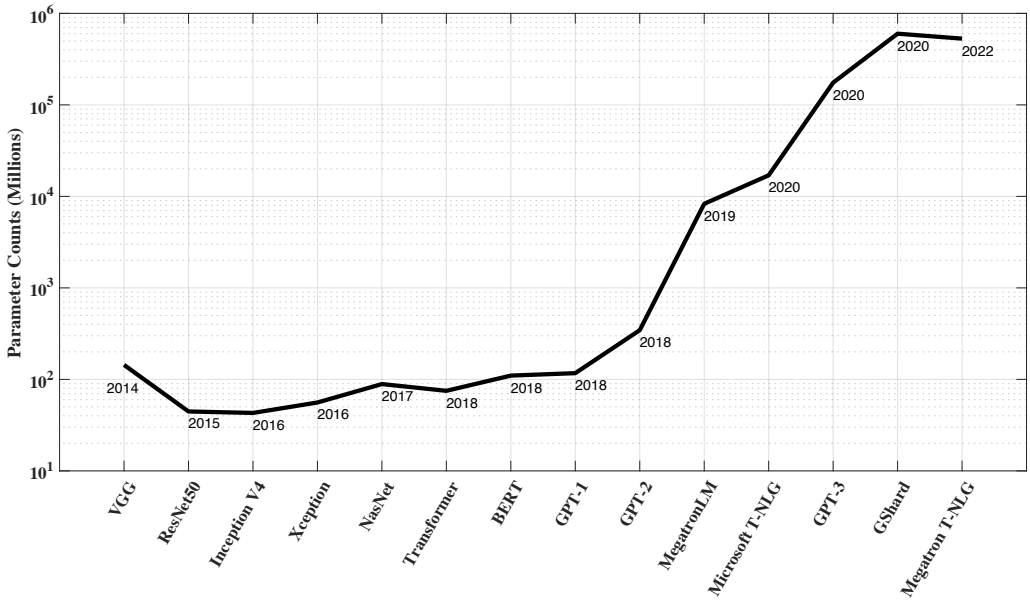


Figure 1.1: Growth trend of Deep Learning models with year

Table 1.1: Commonly known datasets and their sizes

Dataset Name	**Size (Instance or Bytes)**
ImageNet	14M (Image)
Kinetics-700	7M (Video clip)
YFCC100M	100M (Media Objects)
Wikipedia	45TB
MovieLens	25M movie ratings
Jester	4.1M continuous ratings
Alibaba	2.15B product reviews

in Figure 1.2, while the compute throughput has increased by 900×, the memory capacity and the memory bandwidth have increased only by 5×, creating the Memory Wall, which means that the hardware is memory bound. Despite having a higher theoretical throughput, the achievable throughput is less because of less memory bandwidth. This translates into huge data movement between processors, from host to processor, and processor to processor, incurring significant performance loss and energy consumption.

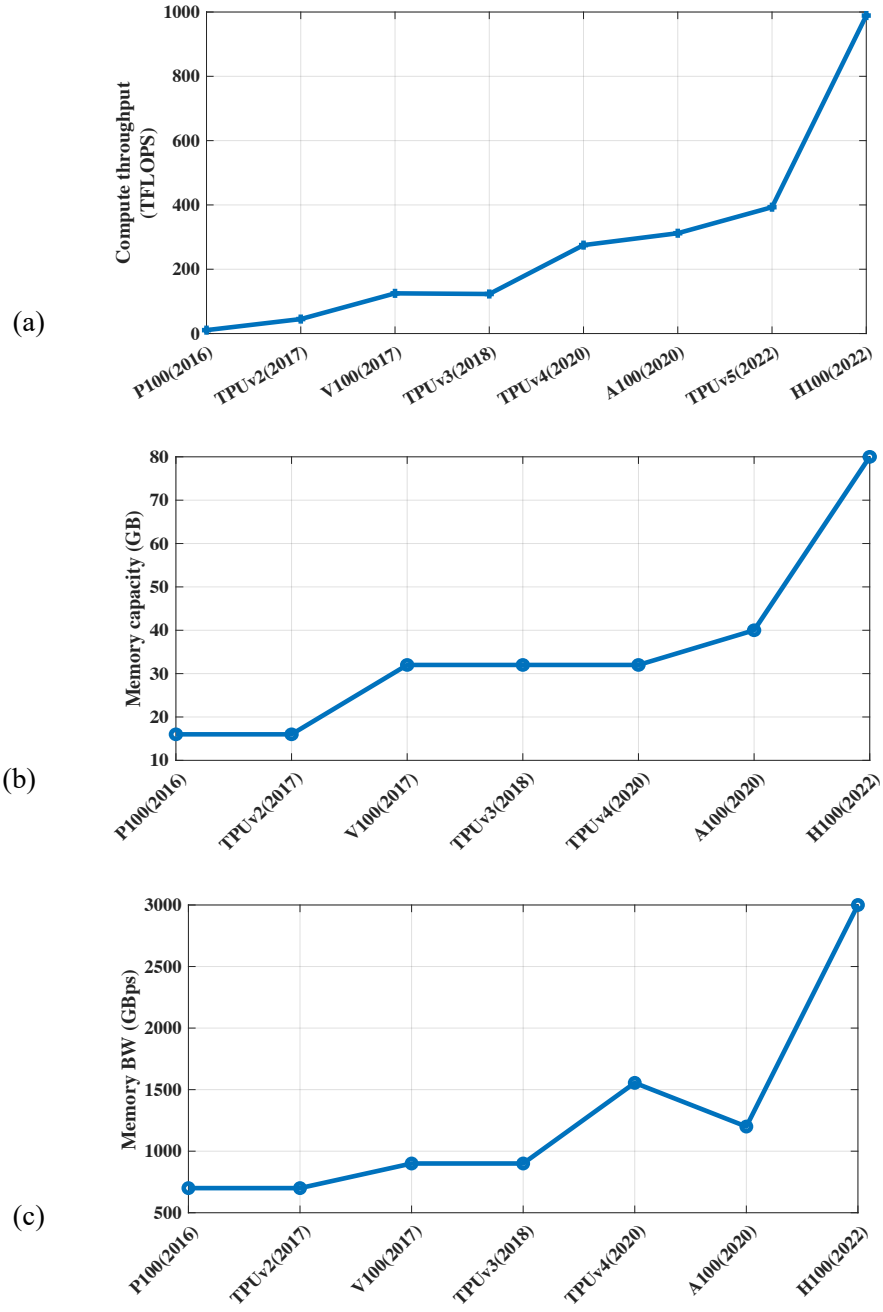


Figure 1.2: Trend of AI hardware accelerators specs over the years: (a) Compute throughput (TFLOPS), (b) Memory capacity (GB), and (c) Memory BW (GBps)

Nowadays, with the growing size of parameters and dataset, the models can no longer be trained with a single GPU as the model parameters cannot fit in the main memory of the

GPU; even the largest GPU cannot hold the parameters. Hence, the models are trained in the data center with racks of powerful GPUs/TPUs interconnected with PCIe or NVLink with a distributed training technique. When it comes to running AI workloads, energy efficiency is of paramount importance, be it inference or training. While the inference devices have limited battery life, data centers are constrained by electricity costs, environmental impact, and carbon footprint. Producing an image using Generative AI, such as OpenAI's DALL-E 3, consumes an equivalent amount of energy as fully charging a smartphone. The carbon footprint of training a single large language model is around 600,000 pounds of CO_2 emissions, which is the equivalent of 125 round-trip flights between New York and Beijing[50] [51]. The energy usage of data centers is growing exponentially, with projections indicating a surge to 1,000TWh by 2026, further escalating to 2500TWh by 2030 [50] [51]. It is also very expensive to train the LLM models. OpenAI's GPT-3 language model, each training run required at least $5 million worth of GPUs [52].

As a result, the key metrics of an AL/AI accelerator are: (i) high performance or high throughput (TOPS), (ii) energy efficiency (TOPS/W), (iii) area efficiency (logics/A), (iv) cost efficiency, (v) scalability, and (vi) generality. To ensure performance, the AI/DL hardware should support for parallel workloads, by having powerful matrix and vector engines. To ensure energy-efficiency, they should have high memory capacity, both on-chip and off-chip DRAM, and high memory bandwidth, less data movement. The cost efficiency is ensured by the high yield, less NRE and RE cost. To ensure generality, they should have the ability to keep pace with the rapid ever-evolving DNN/AI model architectures, support for multi-domain and diverse AI/DL models all in one efficiently. With powerful matrix and vector engines, the existing accelerators/GPU offer high performance. However, because of the smaller memory bandwidth and less memory capacity, they suffer from poor energy efficiency and cost efficiency. The challenge is to balance the need for more computational throughput at lower energy and cost.

The goal of this research is to optimize compute and memory resources to meet workload demands, achieving energy and cost efficiency while maintaining high performance. We hypothesize that memory capacity and bandwidth, both on-chip (cache) and main memory (DRAM), play a significant role in the performance, energy, and cost efficiency of AI accelerators. This thesis presents a System and Design Technology Co-optimization (STCO & DTCO) methodology aimed at achieving energy, area, and cost efficiencies while preserving high performance.

1.1 Thesis Outline and Contributions

Towards achieving energy efficiency, I evaluate the significance of on-chip memory on the energy efficiency and performance of AI accelerators and explore the potential and feasibility of NVM technologies as embedded on-chip memory/cache to increase the on-chip memory capacity. In chapter 2, I design an energy-efficient and high-performance AI accelerator with customized STT-MRAM (Spin Transfer Troque Magnetic Random Access Memory). Based on model-driven detailed design space exploration, a design methodology of an innovative scratchpad-assisted on-chip STT-MRAM based buffer system for high-performance accelerators is presented. Using analytically derived expression of memory occupancy time of AI model weights and activation maps, the volatility of STT-MRAM is adjusted with process and temperature variation aware scaling of thermal stability factor to optimize the retention time, energy, read/write latency, and area of STT-MRAM. In chapeter 3, we address the limitations that we faced with STT-MRAM by introducing SOT-MRAM as the on-chip memory. We develop the memory system with Design Technology Co-optimization (DTCO)- enabled customized Spin Orbit Torque (SOT)-MRAM as large on-chip memory through System Technology Co-optimization (STCO) and detailed characterization of the DL workloads. Together, these chapters provide an important contribution by introducing

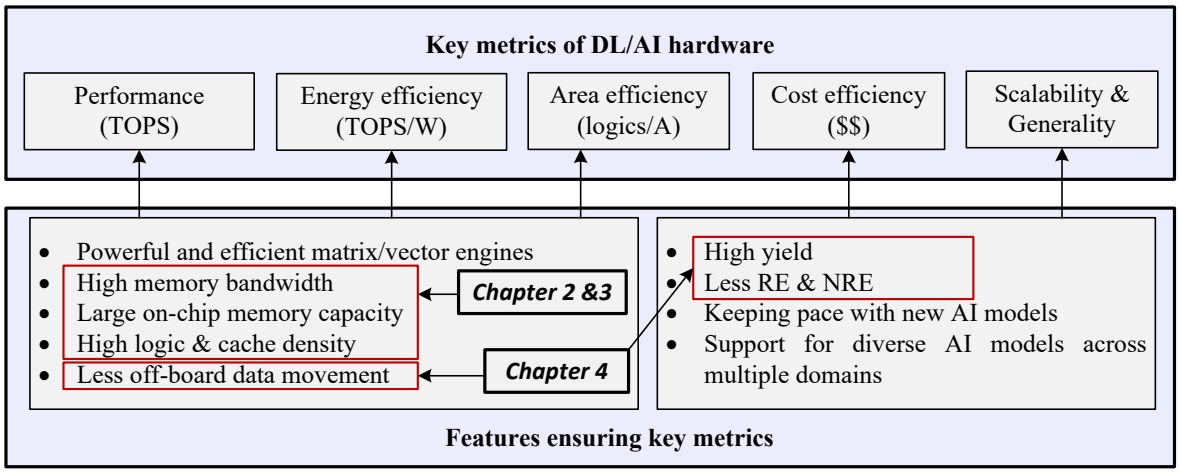


Figure 1.3: Thesis outline and contributions: key metrics of DL/AI accelerators and the chapter-wise distribution of the contribution.

a larger on-chip memory and high on-chip bandwidth towards achieving high performance, energy, and area efficiency.

In Chapter 4, we explore the chipset-based AI accelerator to improve performance, energy, and cost efficiency. We analytically model the PPAC (Power, Performance, Area, and Cost) of the chiplet-based AI accelerator and integrate it into an OpenAI gym environment to systematically navigate the vast design space of chiplet-based AI accelerator. This system and package-level co-design methodology aid the designer in finding the optimum design point regarding Power, Performance, Area, and manufacturing Cost (PPAC) in a time and resource constrained environment.

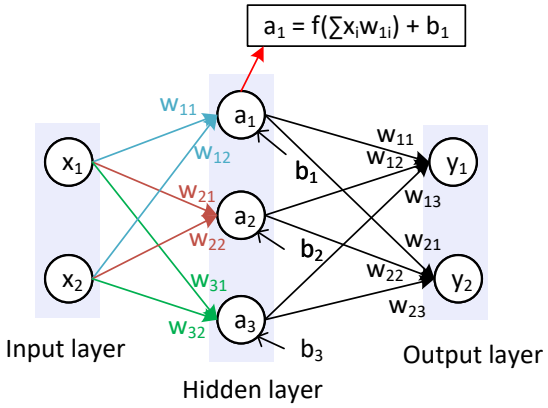
Finally, as shown in Figure 1.3, this dissertation addresses several essential aspects of achieving the primary metrics/qualities of DL/AI accelerators through two straightforward methods: (i) Implementing NVM as on-chip memory to minimize off-chip data movements in chapters 2 and 3, and (ii) Utilizing chiplets to decrease off-package data movement in chapter 4.

1.2 Background and Literature Review

1.2.1 Overview of DNN and Generative AI

The AI that captivates our imaginations today owes its success to the profound capabilities of Deep Neural Networks and Self Attention mechanism. An artificial neuron, mimicking a biological neuron, is a mathematical operation that accepts an input vector, where each element is weighted by a corresponding element of a weight vector. The weighted sum, accumulated with a bias term, is passed through an activation function to generate the final output element. Mathematically, this is expressed as: $y = act.(\sum_{i=1}^{n}(w_i x_i) + b)$. There are various activation functions, such as sigmoid, ReLU, leaky ReLU, etc. Multiple neurons are grouped together to form a layer. A network is formed with an input layer and an output layer, with or without hidden layers. The layers between the input and output layers are known as hidden layers. Networks with many hidden layers are known as Deep Neural Networks (DNNs), and the depth of the network is determined by the number of hidden layers. The simplest network is a Multi layer perceptron (MLP) or fully connected layer. Where, each neuron of the previous layer is connected to each neuron of the next layer. Mathematically, the operations performed in each layer are equivalent to a matrix-vector multiplication. The output vector is generated by multiplying the weight matrix with the activation vector, which is either the input or the output of the previous layer. Figure 1.4 shows a vanilla neural network and the mapping of its operations to matrix-vector multiplication.

Another type of layer, best known for extracting spatial information from an image, is the convolution layer, where multiple kernels/filters are convolved over the inputs to generate the output. Figure 1.5 shows the operations performed in convolution operation. The convolution operation can be converted to matrix-matrix multiplication by performing a Im2Col transformation on the input matrix and unrolling the filter matrices and concatenating them



$$[a_1 \quad a_2 \quad a_3\,] = f([x_1 \quad x_2]\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}^T + [b_1 \quad b_2 \quad b_3])$$

Figure 1.4: Vanilla Neural Network (Multilayer perceptron) and its underlying matrix-vector multiplication representation of the hidden layer.

along the rows [9]. The Deep Neural Networks (DNN) comprising of stacks of multiple convolutional layers connected straight and/or through residual connection [55] to extract the embedded features and one or more Fully connected layers to classify them dominate the Computer Vision (CV) domain of AI. Figure 1.6 shows the typical structure of the CV models. Image classification, captioning, reconstruction and object/instance segmentation are the scopes of CV models. Deep Residual Networks, having convolutional layers at their core, dominate the CV domain.

Language modeling deals with processing sequential data. Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM), and Gated Recurrent Unit (GRU) have been used in language modeling until the state-of-the-art Transformer [111] model is introduced. NLP models are used in machine translation, text summarization, speech recognition, syntactic and semantic parsing, question answering, dialog system etc. Fig 1.7) illustrates the architecture of the Transformer model.

In Generative AI (GenAI) the new realistic data such as text, image, video, etc. are generated by the generative models. These models acquire an understanding of the underlying patterns and structures within their training data, subsequently producing fresh data

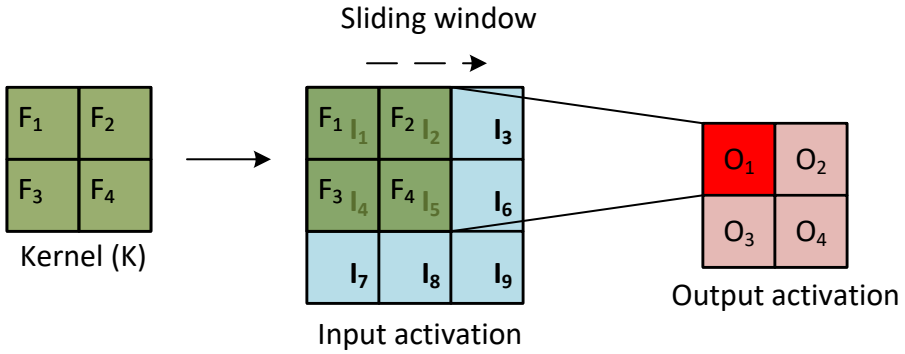


Figure 1.5: Illustration of 1D convolution operation. A 2x2 Output activation is generated by convolving a 2x2 Kernel with a 3x3 Input activation with a stride size of 1. Each element of output activation is generated by taking the element-wise dot product of the Kernel and Kernel-overlapped input activation region.

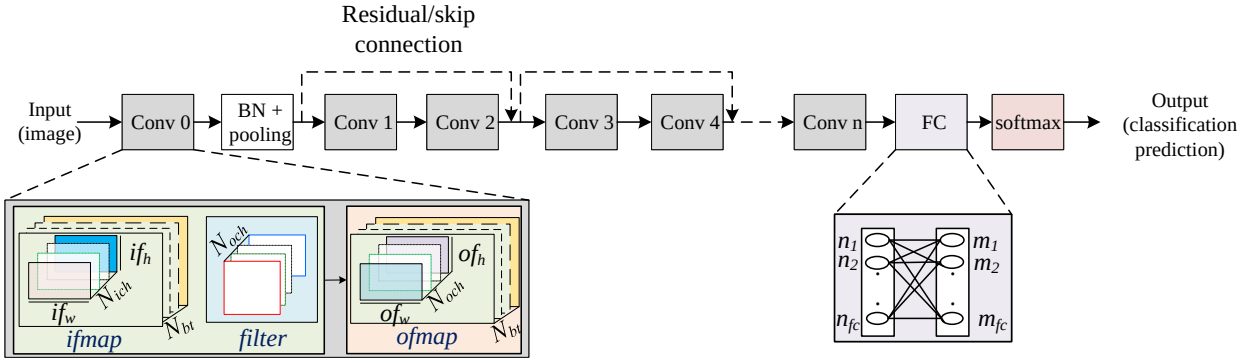


Figure 1.6: CV model (CNN/DNN) abstract architecture. Deep convolution (Conv) layers with residual/skip connection followed by fully connected (FC) layer/s. For symbol meaning please see Table 3.1.

that share similar traits and characteristics of their training data. While there are different approaches to training the generative models, such as Generative Adversarial Networks (GAN), Variational Auto-Encoders (VAE), Transformer based models, and Diffusion models, their building blocks are Deep Neural Networks or the transformer-based self-attention mechanism [2]. As a result, the most compute and memory intensive part of these workloads are also DNN and transformer-based attention mechanism.

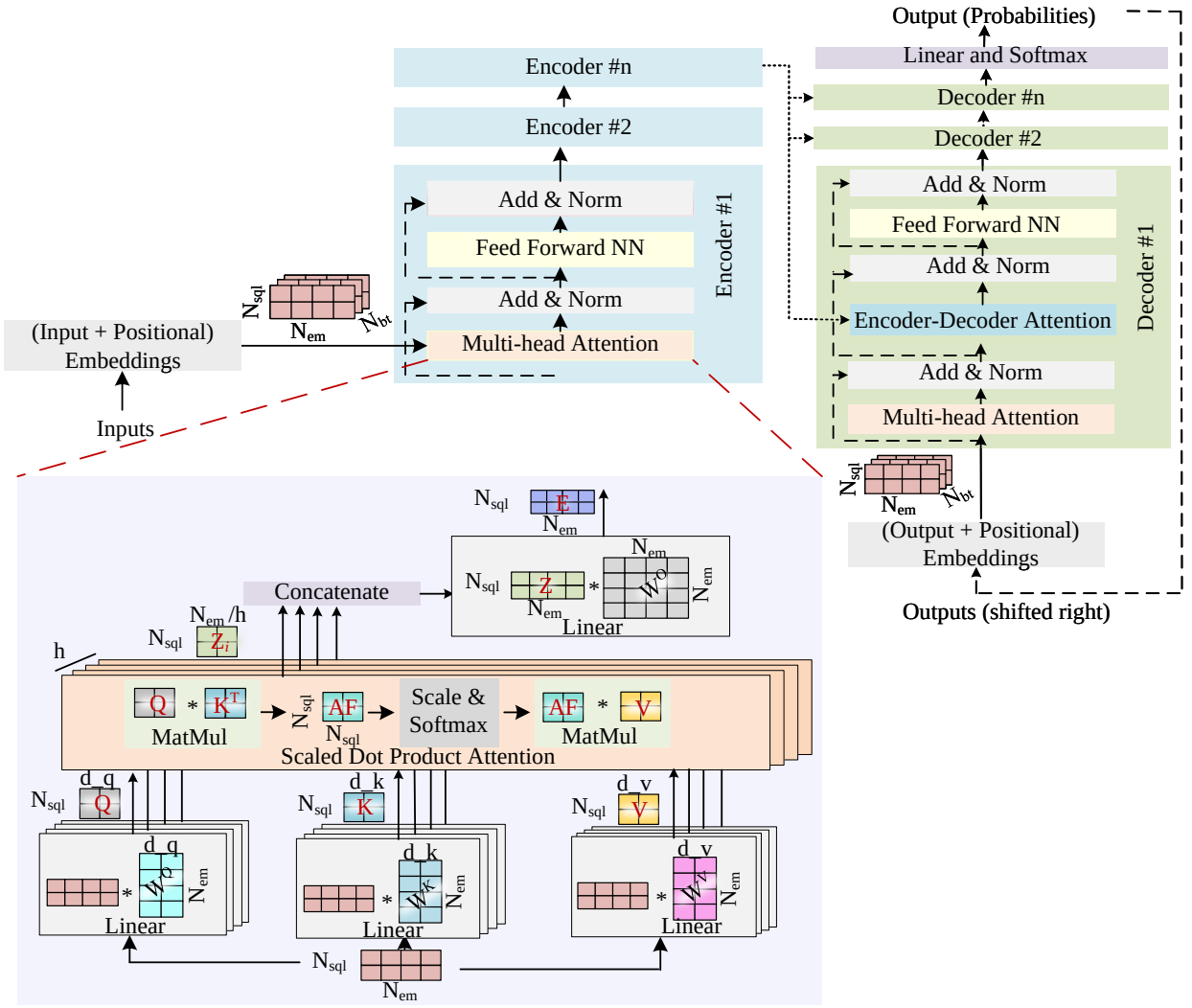


Figure 1.7: Transformer model workflow breakdown

1.2.2 Overview of AI accelerators

The fundamental component of DNN workloads (such as Conv. layer, FC layer, self-attention mechanism) are MAC (Multiply-Accumulate) operations, which are essentially the matrix-vector/matrix multiplication, or can be converted to matrix-vector/matrix multiplication with massive parallelism. CPUs are inefficient in parallel processing. GPU with temporal features such as vector processing (SIMD), parallel threading (SIMT) can accelerate the parallel computations[9]. However, they are inefficient in data movement. In addition to

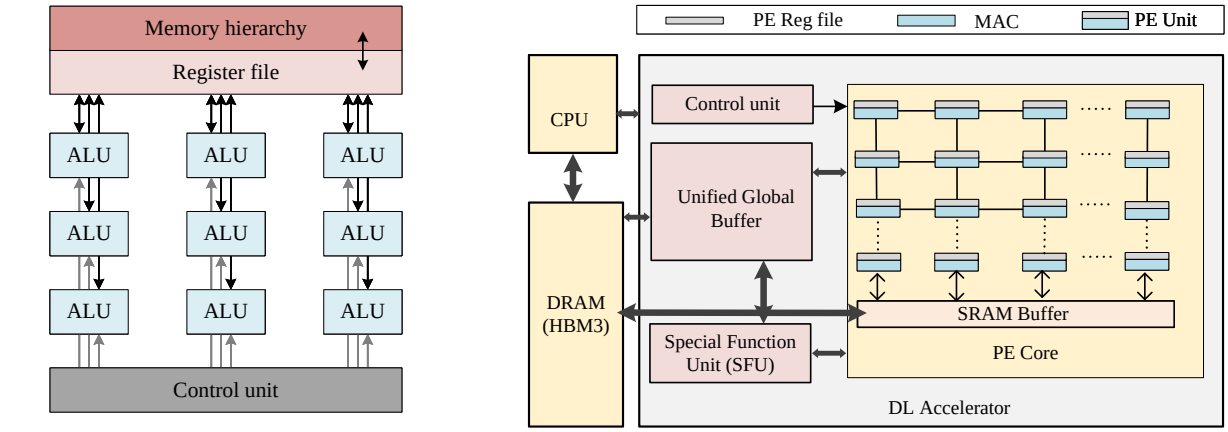


Figure 1.8: (a) SIMD type architecture. (b) Systolic array-based Deep Learning accelerator architecture

the parallel computation, systolic array type architecture [4] with spatial features can also leverage the dataflow architecture to reduce the data movement to a significant amount. As a result, Systolic array-based architecture has been widely used in accelerating DNN workloads[155] efficiently. The typical architecture of AI accelerators is composed of array of Processing Elements (PE) containing MAC (multiplier and adder) modules, with/without register file and control unit local to each PE unit along with the Global bufffer (GLB) to hold to weights and activations. The number of PE units and the size of GLB may differ. Figure 1.8 (a) shows the block diagram of a SIMD-type temporal architecture, and Figure 1.8 shows the typical architecture of a Systolic Array-based Deep Learning accelerator.

Diannao[5] and ShiDianNao[6] are two of the earliest accelerator architectures that focused on reducing memory access to improve energy efficiency and performance by playing with memory hierarchy to extract the reuse of operands for CNNs and MLP networks. Eyeriss[7] is an energy-efficient DNN accelerator that introduces Row Stationary (RS) dataflow to maximize data reuse and minimize data movement, utilizing a hierarchical memory structure to significantly reduce energy consumption and improve performance. To

keep up with the diverse structure of DNN layers and to support multiple mapping strategies, flexible architectures such as MAERI [10], Flexflow [11] have recently been proposed. In addition to the massive parallalism and reuse propetry, the DNN workloads come with another special feature, sparsity, that can be leveraged to accelerate the performance and improve energy-efficiency. Sparsity means that a significant amount of activations and weights are either zero or close to zero (because of activation function, regularization, pruning technique). SCNN[8] proposed by NVIDIA is one of the first ASIC architectures supporting sparsity. It leverages sparse operands by first constructing tiles from sparse matrices and then using a dense accelerator substrate for non-zero operands. CambriconX[12] uses hardware indexing to eliminate redundant computations in sparse matrix multiplications, while Cambricon-S[13] enhances software to reduce sparse activation irregularities, simplifying the hardware indexing unit. SIGMA[14] employs specialized interconnects to efficiently handle large-scale sparse computations with irregular operand matrices.

Advances in machine learning, including GenAI, have led to increasingly large and complex models, outpacing the capabilities of conventional accelerators and necessitating new designs for efficient large-scale computation. DaDiannao [15] is one of the earliest accelerator that introduced interconnected accelerators for large network training and inference. Nvidia's Simba[16] features a scaled-out, still targetting inference, on-chip DNN accelerators linked via a silicon interposer. Stanford's Tangram[17] uses Eyeriss-like PEs with a new interconnect structure.

Apart from academia, Google's TPU [155][164], NVIDIA's NVDLA [18], Microsoft's Brainwave [28], Apple's Neural Engine [22], Intel's Nervana Neural Network Processor (NNP) [25], Tesla's FSD [26], Cerebra's Wafer Scale Engine (WSE) [19], SambaNova's Reconfigurable Dataflow Unit (DFU) [23], Groq's tensor streaming processor [24], Huawei's Ascend [29] are the notable and successful accelerators proposed from the industry.

Another domain of AI accelerator research is "In-memory computing" or "Processing-in-memory", where the computation is performed in the memory to reduce the data movement. In this thesis, we only focus on the regular temporal or spatial AI accelerator architecture, not in-memory or processing-in-memory architectures.

Chiplet-based AI accelerator

Recent advances in AI, including GenAI, have led to a substantial increase in computational and memory demands on AI hardware across various domains, from edge to cloud. To address the high demand for computing and memory resource capability, chiplet-based AI accelerators have recently evolved as a viable solution. Unlike monolithic designs, which are often constrained by the fabrication yields, area, power consumption, heat dissipation, etc., chiplet-based AI accelerators provide scalability, area, yield, and cost efficiency. Simba [16] is one of the first chiplet-based AI accelerators for inference, which integrates 36 chiplets on organic package substrate with ground-referencing signaling (GRS) technology for intra-package communication. Centaur [35] integrates CPU and FPGA chiplets on package targeting recommendation system workloads. SPRINT [36] is a 64-chiplet system with photonic interconnect for DNN inference. [33] proposes a chiplet-based AI accelerator with IMC (In-Memory Computing) technology. [40] proposes a chiplet-based ASIC supercomputer for LLM that optimizes the total cost of ownership (TCO) per generated token. [38] proposes Tascade, a task-oriented scalable chiplet architecture for distributed wprkload execution, evaluated across 256 distributed chips. NN-Baton [34] is a tool for analyzing and optimizing the granularity of chips that proposes a 4-chiplet AI accelerator for DNN workloads. Monad [41] and SCAR [39] perform design space exploration for chiplet-based AI accelerator, focusing different aspcects of the design space, such as monad considers architecture and integration, while SCAR considers multi-model dataflow scheduling on heterogeneous multi-chiplet modules.

1.2.3 Optimization & Evaluation of Accelerators

Impact of Design Space Exploration (DSE) and System & Design technology co-optimization (STCO & DTCO)

With the AI models becoming increasingly complex and large, the design space of the AI accelerators are also booming. As a result, the importance of Design Space Exploration (DSE) and system and design technology co-optimizations across the entire stack, starting from the AI application to all the way down to the circuit and device level of the hardware running the AI models, has become pivotal to ensure the high performance, energy, area, and cost efficiency. The standalone optimization of any component of the stack, may lead to sub-optimal accelerator design. Edge devices are constrained by area and energy, while cloud or data centers are constrained by latency and environmental impact. The prediction accuracy of AI models should be given a high priority for Safety-critical applications such as wearable medical devices, autonomous vehicles and (Depart of Defense) DoD applications.

The key attributes contributing to the quality of AI accelerator are dataflow and mapping styles, and hardware resources. Dataflow styles incorporate computation order, loop tiling and parallelization strategies. While HW resources incorporate on-chip compute resources, memory hierarchy, on-chip memory capacity, and bandwidth, main memory capacity and bandwidth. Going from monolithic to multi-chiplet based AI accelerator, the design space further explodes, with different types of packaging and interconnect architectures, to name a few. From architectural perspective, resource allocation, monolithic or multi-chiplet based architecture, memory hierarchy, memory capacity, mapping and dataflow of the DNN workloads across the compute and memory resources should be considered. From communication and integration perspective, chiplet and memory placement, routing protocols, stacking/packaging technologies, interconnect types, bandwidth, and finally from application perspective, system requirement, such as reliability, scalibility etc., should be considered all

at the same time capturing their inter-dependency while optimizing for PPAC (Power, Performance, Area, and Cost).

Table 1.2: Comparative summary of AI accelerator simulator frameworks for design space analysis

	Simulator/Optimizer/ Architecture	**Design space/Architectural details**	**Monolithic/ Chiplet-based**
Scale-sim [27]	Performance Simulator	(i) Dataflow: WS, IS, OS (ii) HW resource: No. of PE, on-chip memory size (iii) Architecture type: Eyeriss, TPU	Monolithic
Timeloop+ accelergy [53]	Mapping optimizer + Performance, energy, and area simulator	(i) Dataflow: WS, IS, OS (ii) HW resource: No. of PE, on-chip memory size (iii) Different memory hierarchies (iv) Architecture type: Eyeriss, Simba	Monolithic
Maestro [186]	Performance and energy Simulator +Optimizer	(i) Dataflow: flexible (ii) HW resources: Number of PEs, NoC BW/Latency, on-chip memory size (iii) Architecture type: NVDLA-like	Monolithic
Astra-sim [31]	Performance Simulator	(i) Distributed training: data, model, hybrid parallelism (ii) Hierarchical collective algorithms: on-load, off-load (iii) Fabric design: number of links & latency/BW per link (iv) Fabric topology: pt-to-pt, 2D/3D Torus	Multi-chip (package/board level)
STONNE [30]	Performance, energy and area Simulator	Architecture type: flexible & reconfigurable architecture	Monolithic
Confucuix[49]	Performance and energy optimizer	(i) HW resource: Number of PE, on-chip buffer size	Monolithic
SIMBA [16]	Performance & power Optimizer + 36-chiplet Architecture	(i) Mapping and tiling strategies (ii) 36 NVDLA chiplets connected in 2D mesh	Multi-chip (package level)
SPRINT [36]	64-chiplet Architecure	64-chiplet architecture with photonic interconnect for interchiplet communication	Multi-chip (package level)
NN-Baton [34]	Simulator + Optimizer	(i) No. of accelerator chiplets: 1, 2, 4, 8 (ii) On-chip memory: 36 to 642KB (iii) Different mapping strategies (iv) Routing topology: Ring	Multi-chip (package level)
Moand[41]	Optimizer	(i) No. of accelerator chiplets (ii) mapping & tiling (iii) packaging (iv) network topology (v) placement	Multi-chip (package level)
TVLSI'20[183]	Simulator	(i) 64-core ROCKET-64 architecture (ii) 2.5D interposer-based centralized NoC	Multi-chip (package level)
Magnet [32]	Area, Power, Performance optimizer and RTL generator	(i) Dataflow: WS, OS, WS-LOS, OS-LWS (ii) HW resources: No. of PE, on-chip memory size (iii) Precision: 4/8 bits (weight/activation), 16/20/24 bits (accumulation) (iv) Operating frequency: 0.5 GHz, 1 GHz	Monolithic
Chiplet-gym (This work Chapter 4 [37])	Performance, power, area and cost Optimizer	(i) No. of AI accelerator chiplets, no. & location of HBM chiplets (ii) package architecture: 2.5D, 3D (iii) Interconnect types & configuration (details in Table 4.1)	Multi-chip (package level)

Use of AI/ML in design space exploration and optimization

With the exponential increase in valid design points within the design space, learning-based approaches, especially Reinforcement Learning (RL), have become more popular than traditional heuristic and metaheuristic-based search algorithms due to their efficiency and performance. Apollo[47] uses black-box optimization methods and transfer learning for

sample-efficient accelerator design. Confuciux [49] uses RL and genetic algorithm to explore the on-chip HW resource allocation on AI accelerator for a given dataflow and reuse. Zeus [45] co-optimizes performance and energy by finding optimal job and GPU-level configurations for recurring DNN training jobs using RL. [46] proposes a Deep Reinforcement Learning (DRL) based frmework for exploring the router-less Network-on-chip (NoC) design space. [48] uses multi-agent reinforcement learning to optimize DRAM-memory controller. Apart from architectural design space exploration, RL has also used in back-end of ASIC design, such as physical design [42], circuit design [43][44].

Existing simulators, optimizer and and DSE frameworks

To evaluate the performance and efficacy of the accelerators, and to accelerate the research, simulators and optimization frameworks have been proposed, specifically tailored for DNN accelerators. Scalesim [27] is one of the earlier simulator for systolic array type architecture. Given the architectural and workload parameters, it outputs the computation cycles, mapping utilization, SRAM and DRAM bandwidth. Timeloop+Accelergy[53][54] projects the performance and energy-efficiency of DNN accelerator for inference exploring different mapping and tiling strategies. While these two simulators work in layer-wise, STONNE[30] is an end-end, meaning that it can simulate the entire model, cycle-level simulator for both flexible architectures such as MAERI[10], SIGMA[14] and conventional architectures, such as Eyeriss[7]. MAESTRO[186] is another simulator which takes DNN models, mapping or dataflow and hardware resources as input, and outputs the performance report such as latency, NoC bandwidth requirement, and cost report such as activity count (energy), buffer size requirement, data reuse amount, etc. ASTRASIM [31] is the only system-level simulator available today, supporting distributed machine learning as well. Table 1.2 shows the framework for design space exploration in the literature.

1.2.4 Impact of memory in AI accelerators & potential of Emerging memory technologies in AI accelerators

When it comes to DNN workload processing, on-chip memory plays a crucial role in performance and energy efficiency. Because data movement from processor to memory is two orders of magnitude more expensive than MAC (Multiply-Accumulate) operation, the dominant arithmetic operation in DNN workloads. Each MAC operation requires three memory reads (filter weights, fmap (feature map) activation, and partial sum) and one memory write operation [9]. Each memory access is at least 200× more energy and latency expensive than the MAC operations. The idea is that the more weights and intermediate activations can be accommodated in the on-chip memory, the off-chip main/DRAM access can be reduced, hence improving the performance and energy efficiency. A comparative analysis between A100 and TPUv4 processors, as shown in Fig. 1.9, effectively illustrates the impact of on-chip memory on performance and energy efficiency. The A100, with specifications of 312 TFLOPS and a 40MB L2 cache, is compared to the TPUv4, which boasts 275 TFLOPS and a 160MB L2 cache. The performance metrics reveal that for the BERT model, the TPUv4 demonstrates a 1.93× reduction in mean power (measured in watts) compared to the A100. Similarly, for the ResNet model, the TPUv4 shows a 1.33× reduction in mean power over the A100. Furthermore, the normalized performance for BERT is 1.15× higher with the TPUv4, while for ResNet, the TPUv4 offers a 1.67× increase over the A100. These data points clearly indicate that TFLOPS is not the sole determinant of performance, highlighting the critical role of memory capacity.

Recent advancements in on-chip memory capacity across industry further underscore this point. NVIDIA's transition from the A100 to the H100 demonstrates an increase in L2 cache capacity from 40MB to 50MB, integrated with a transformer engine, signifying significant improvements in processing efficiency and capability. AMD's MI300 processor

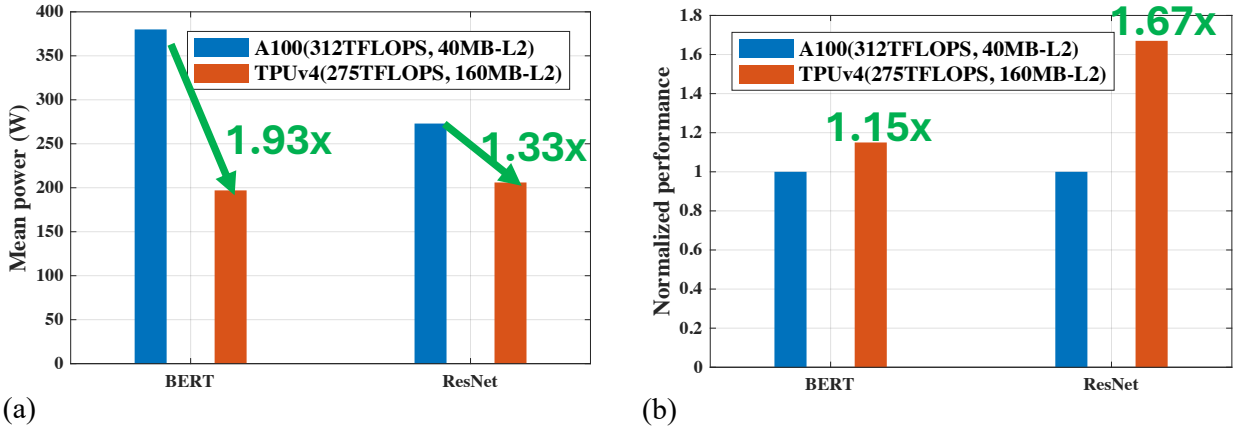


Figure 1.9: Power and performance comparsion of A100 GPU vs TPUv4, showing the impact of on-chip memory.

incorporates 190GB of HBM3 and 3D-vCache technology with 144MB SRAM, representing a leap in memory architecture that enhances computational throughput and efficiency. Cerebras' WSE-2 [19] features 40GB of on-wafer memory, underscoring the shift towards integrating substantial memory directly on the chip to optimize performance. Similarly, Groq's TSP [24] boasts 220MB of SRAM, illustrating the industry's focus on increasing memory capacity to reduce the energy and latency hungry data movement.

SRAM has been predominantly used as the on-chip memory for CPUs and GPUs due to its ultra-fast access times (in pico-second range), unlimited endurance (exceeding 10^{16} cycles), and scalability with technology scaling. However, for such large memory capacity as demanded by AI accelerators, SRAM is an expensive technology in terms of silicon footprint, with a relatively low integration density of only dozens of megabits per millimeter squared. It also suffers from high standby leakage power, ranging from tens to hundreds of picowatts per bit. As a result, to meet the demand of AI accelerators' Global Buffer (GLB), exploration of alternate memory technologies has gained interest. Emerging memory technologies, such as MRAM, RRAM, PCM etc. have shown potentials. Table 1.2.4 shows a comparsion between the properties of emerging memories and SRAM.

Table 1.3: High-level comparison between SRAM and emerging memory technologies

Property	**Resistive RAM (RAM)**	**Phase Change Memory (PCM)**	**Magnetic RAM (MRAM)**	**Static RAM (SRAM)**
Cell area (Storage density)	$4-12F^2$ (High)	$4-30F^2$ (High)	$6-50F^2$ (High)	$160F^2$ (Low)
On/OFF resistance ratio	High	High	Low	High
Retention time (Non-volatility)	>10y (High)	>10y (High)	>10y (High)	N/A
Leakage power	Low	Low	Low	High
Write latency	100ns	150ns	10ns	<1ns
Write energy	2nj	6nj	1nj	0.2nj
Read latency	10ns	10ns	10ns	<1ns
Endurance	10^6-10^{12}	10^9	10^{15}	10^{16}

For example, as shown in Table 1.2.4, while the emerging memories have the advantages of high integration density they also face high write energy and latency. A co-optimization methodology, guided by the demands of AI workloads and applications, is required to extract the benefits while discarding its drawbacks.

1.3 Thesis Organization

The thesis is organized as follows. In Chapter 2, we design an energy-efficient and high-performance deep learning accelerator with customized STT-MRAM. In Chapter 3, the drawbacks of STT-MRAM is mitigated by introducing SOT-MRAM in the memory system and performing a system and design technology co-optimization of the system requirement, workloads, and SOT-MRAM. In Chapter 4, we present Chiplet-gym, a Reinforcement Learning based system and package level co-design methodology for chiplet-based AI accelerator. Finally, the thesis is concluded in Chapter 5 with potential future research direction.

Chapter 2

Energy-efficient and High-performacne AI accelerator with customized STT-MRAM

2.1 Introduction

On-chip memory capacity plays a significant role in the performance and energy efficiency of AI tasks [9, 7, 58, 155]. In AI accelerator, off-chip Dynamic Random-Access Memory (DRAM) accesses can take 200 times and 10 times more energy compared to the local register file and global buffer memory, respectively [7]. Larger on-chip buffer memory is needed to minimize DRAM accesses, and it can improve the energy efficiency and speed of the accelerator. However, conventional Static Random-Access Memory (SRAM) based solutions suffer from area constraints and leakage power at advanced technology nodes [69, 74], which is a major concern for the energy-constraint IoT domain. STT-MRAM has the potential to replace SRAM as the global buffer in high-performance AI accelerators that require large on-chip memory [57, 155]. For AI accelerators used in inference-only applications, the pre-trained weights need to be stored on-chip. As conventional embedded Flash storage suffers from scalability and reliability issues at advanced nodes [74], emerging memory-based solutions are required for AI accelerators. As analyzed in detail in [75], because of weight reuse in Deep Learning, radiation-induced soft errors in the memory block of the accelerator can impact the accuracy of AI models. This is especially a concern for safety-critical applications such as autonomous vehicles with rigid FIT requirements [75], and STT-MRAM can be a better option for these types of applications.

At scaled technologies (e.g., 10nm and newer), static energy loss from the high leakage current dominates the overall energy dissipation in DRAM and SRAM technologies [74]. Although Trench cap based embedded DRAM (eDRAM) has a higher density compared to SRAM, the leakage power and scaling challenges of eDRAM at advanced process nodes make it less competitive in the future technology roadmap [74]. Beyond 28nm node eFlash

faces scaling challenges, and eMRAM technology becomes superior over eFlash because of its lower write voltage and energy, higher endurance, lower area, and faster read/write time [63]. The emerging resistive RAM (RRAM) and Phase Change (PCM) based cross-point memory suffers from endurance, reliability and variability problems [70, 74]. Among all the emerging embedded memory technologies, STT-MRAM is one of the most promising due to its high energy efficiency, write endurance (e.g., more than 1 million cycles), high cell density, high-temperature data retention capability, operating voltage comparable with CMOS logic, and immunity to soft errors [74, 69, 60, 62, 61, 77, 78]. Moreover, STT-MRAM is higly compatible with CMOS and requires only 2 to 6 extra masks in the backend-of-the-line (BEOL) process [62, 69]. Because of the leakage power issue, beyond a certain memory size, embedded MRAM becomes more energy efficient compared to SRAM [69].

While performing Deep Learning/AI tasks, the throughput of the AI accelerator primarily depends on, (i) the number of Processing Elements (PE), and (ii) the size of on-chip buffer memory [9, 155]. As a result, area-efficiency is of paramount importance for AI accelerators, and the critical design goal is to increase PE density and on-chip memory capacity. Because of compact size ($6F^2$ of STT-MRAM vs. $100F^2$ of SRAM [81, 66]), STT-MRAM has the potential to outperform conventional SRAM as the on-chip memory in accelerators. At iso-memory capacities, the MRAM module occupies much lower area compared to SRAM [69]. Additionally, for power constraint mobile/edge/IoT applications, STT-MRAM based AI accelerators can significantly minimize static power compared to SRAMs. However, the higher write energy and write latency of conventional eMRAM can be a deterrent in their full adoption in AI accelerators. In this paper, we present a methodology to design efficient AI accelerators with customized STT-MRAM that can provide high bit cell density while still ensuring fast write speed and decreased write energy. We achieve this feat by analyzing

the volatility requirement of weight and input/output feature-map (ifmap/ofmap) data on-chip, and scaling the eMRAM's retention time accordingly without incurring unacceptable bit error rates.

The key contributions and highlights of this chapter are:

- We present an innovative runtime reconfigurable core design that can be optimized for both dot products of convolution layers and matrix multiplications of fully connected layers.

- We derive the analytical expressions of occupancy times of weights and input/output feature maps in the global memory of the AI accelerator between different stages (i.e., Conv. layer followed by Conv., Conv. layer followed by Fully-Connected (FC) layer, and FC-FC) of AI/Deep Learning operation. Guided by this data activity duration, we scale the retention time of STT-MRAM and customize the design for application as the global buffer memory in energy-efficient AI accelerator. We consider Process variations and runtime Temperature fluctuations in this scaling procedure to ensure negligible read/write Bit Error Rates (BER) and retention failures across all corners.

- Based on detailed design space exploration using state-of-the-art AI/Deep Learning models, an AI accelerator system and MRAM technology co-design framework is presented with the key innovations, - (i) Optimizes STT global buffer size to minimize DRAM accesses. (ii) A novel scratchpad-assisted STT-MRAM based global buffer architecture is presented to minimize the writes to the MRAM by bypassing writes of the partial ofmaps to the scratchpad. (iii) For inference-only tasks, to store the trained weights a specially customized embedded STT-MRAM - as a Flash replacement - with optimized retention time (e.g., 3 to 4 years) and robust BER is used.

- To further improve the energy and area efficiency, we exploited the inherent error tolerance of Deep Learning/AI models and created two STT-MRAM banks for the global buffer. For the first bank, the thermal stability factor is scaled further to a relaxed BER, and the less critical half of the weights/fmap bits (e.g., LSB groups) are assigned to this memory block. The second bank has scaled retention time with a robust BER, and the other remaining half of the bits (e.g., MSB groups) are stored in this bank.

The remainder of this chapter is organized as follows. The background is discussed in Section 2.2. STT-MARM based optimum AI/Deep learning accelerator design methodology is presented in Section 2.3. The AI accelerator-aware eMRAM technology co-design methodology is presented in Section 2.4. We present Simulation Results in Section 2.5, Related work in Section 2.6, and Conclusions in Section 2.7.

2.2 Background

2.2.1 Deep Neural Networks

At the core of Deep Learning/AI is the Deep Neural Network (DNN). Modern state-of-the-art DNN consists of stacks of Convolution layers to extract the objects' features and a few Fully Connected layers at the end to classify them. Convolutions are element-wise dot products between matrix (or vector) and matrix. In convolution, kernels convolve over input feature maps (ifmap) to extract embedded features and generate the output feature maps (ofmap) by accumulating the partial sums (psums) as shown in Fig. 2.1. Each fmap and filter is a 3D structure consisting of multiple 2D planes, and a batch of 3D fmaps is processed by a group of 3D filters in a layer. Activation functions (e.g., ReLU) operate on the results before they go to the MaxPooling layer. The computations of a convolutional layer can be

expressed as:

$$Ofmap[z][u][x][y] = ReLU(B[u] + \sum_{v=1}^{N_{in\_ch}} \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} I[z][v][Sx+i][Sy+j] \times F[u][v][i][j]),$$

$$0 \leq z < N, 0 \leq u < N_{out\_ch}, 0 \leq y < N_{ofmap\_rw}, 0 \leq x < N_{ofmap\_cl},$$

$$N_{ofmap\_rw} = ((I_h - k_h + 2P)/S) + 1, N_{ofmap\_cl} = ((I_w - k_w + 2P)/S) + 1 \tag{2.1}$$

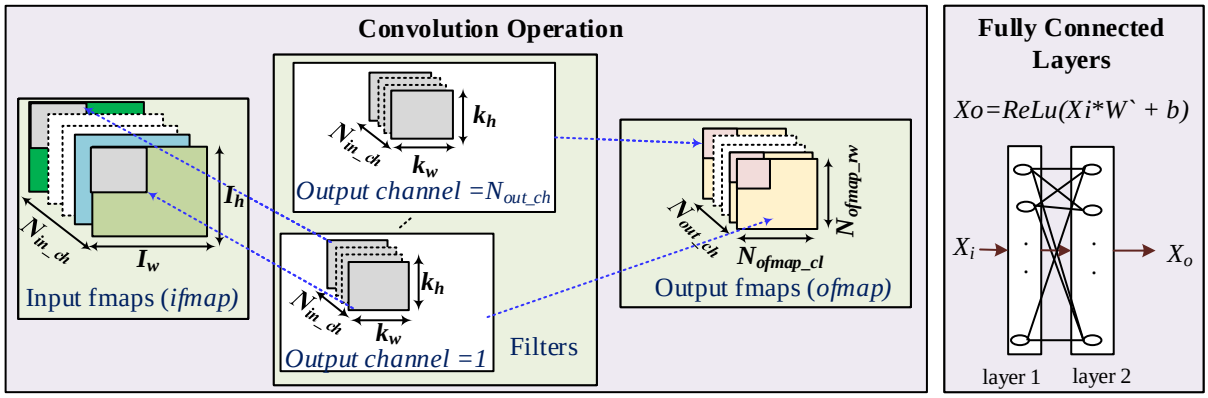


Figure 2.1: Convolution and fully connected layer operations

where, Ofmap, N, F, I, P, S, and B represent output feature maps, number of inputs in a Batch, filter weights, input feature maps, padding, stride size, and bias respectively [7].

Unlike the convolution layers, each neuron of the Fully Connected (FC) layer is generally connected to every other neuron of its previous/next layer with a specific weight (0 for no connection) associated with each connection. The computations of FC layers are matrix/vector-matrix multiplications, where the output activation (X_o) of a layer is obtained by multiplying the input activation (X_i) matrix/vector with the weight matrix (W) followed by the addition of a bias term, and finally passing the result through a non-linear function such as ReLU, $X_o = ReLu(X_i * W + b)$.

2.2.2 Deep Learning/AI Hardware Accelerators

SIMD, or Systolic array based hardware optimized for matrix (or vector)-matrix multiplication, is the present state-of-the-art hardware to accelerate AI operations [9, 155]. The systolic array is only optimized for matrix-matrix multiplication, but it can not perform the dot product necessary for convolution layers. Mapping the convolution dot products into the matrix multiplications by converting the activation maps into the Toeplitz matrix and the kernel weights into a row vector is a popular solution to address this problem. Nonetheless, it involves redundant data in the input feature map which give rise to inefficient memory storage, and complex memory access pattern [9]. More recently, heterogeneous architectures are evolving that have optimized cores for Convolution and FC layers [58]. While this solves the complications regarding Toeplitz matrix conversion, it incurs area overhead. Because when convolution core is active, FC core remains idle, wasting circuit area. In response to the existing issues, in this chapter, we propose a novel concept of a reconfigurable core capable of efficiently performing both convolution dot products and matrix multiplications based on the operation-dependent (i.e., convolution or fully-connected) control signal.

2.2.3 Memory System in AI/Deep Learning Hardware

The memory system is one of the vital metrics in determining the performance of AI hardware. Each off-chip DRAM access is 100 to 200 times more energy costly than any ALU operation or a local memory (e.g., register file/scratchpad) access [9]. As a result, most energy-constraint AI hardware leverage a memory hierarchy of register file, global buffer, and DRAM. Moreover, a significant amount of memory is required to store the pre-trained weights for inference-only applications. The larger the global buffer memory, the more energy-efficient the AI hardware is due to lower DRAM access. Most of the existing DNN hardware use SRAM both as Global Buffer and Register file and eFlash as weight

storage memory [9]. Because of the large size of SRAM and static energy loss due to high leakage at scaled nodes (e.g., 10nm and newer), the global buffer size cannot be increased beyond a certain threshold energy-efficiently. eFlash starts to suffer from scaling challenges even at earlier technology such as 28nm [63]. The benefits of our proposed Δ-customized STT-MRAM as a replacement of both the SRAM-based global buffer and the eflash-based weight storage memory are many-fold: higher memory capacity, lower read-write latency and energy, and higher endurance against soft errors.

2.3 Efficient AI/Deep Learning Hardware

Fig. 2.2 depicts the top-level architecture of the accelerator containing the proposed reconfigurable core and MRAM-based memory system. The following sections describe the dataflow in convolution and systolic mode and formulate the memory occupancy time in each mode.

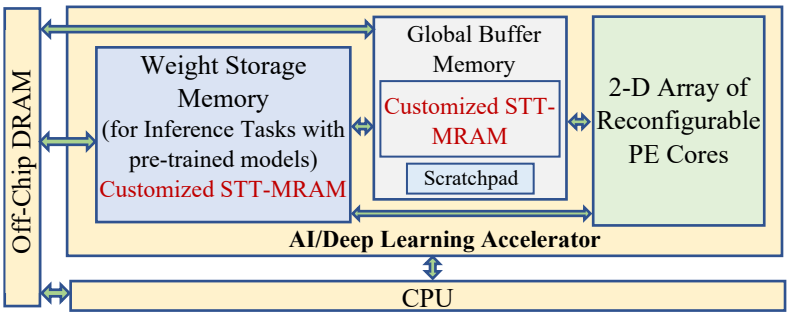


Figure 2.2: AI Accelerator with reconfigurable cores optimized for both Conv. and FC layers, and STT-MRAM based on-chip memory.

2.3.1 Reconfigurable Core

The architecture and workflow of our proposed Reconfigurable Core are quite simple but powerful enough to support both matrix multiplication and convolution dot product at run-time configuration. The reconfigurable core consists of three MAC modules and four

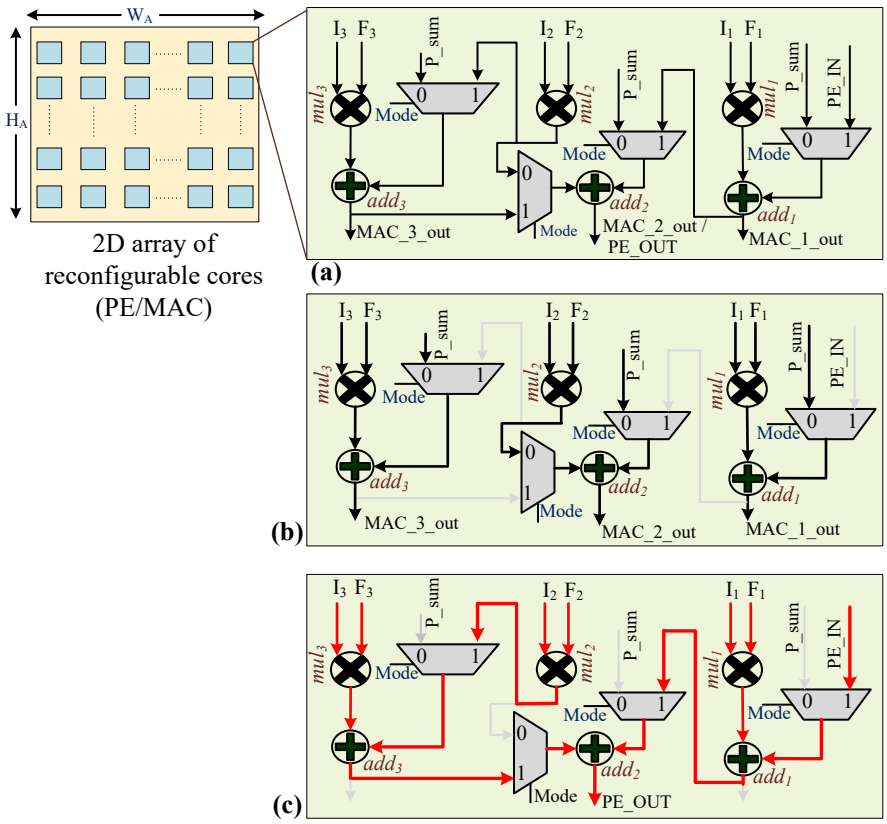


Figure 2.3: (a) Reconfigurable core, (b) Reconfigurable core acting as building block of systolic array when mode is low, and (c) Reconfigrable core acting as convolution PE, when mode is high.

Multiplexers. Each MAC contains a BFloat16 multiplier and an FP32 adder [82, 83] to accommodate both training and inference. If only inference is desired, the hardware can be 8-bit int8 type [9, 155]. The multipliers take input feature maps and filter weights as inputs and pass the results to their neighboring adders to be added with the previous partial sum results. The multiplexers act as mode selectors of the core. When *Mode* is de-asserted, the MACs are disconnected from each other and their outputs are collected downward to reflect the systolic array architecture (Fig. 2.3(b)). On the other hand, when *Mode* is asserted, three MACs collectively act as a convolution block that performs three dot products parallelly and produces one partial sum (Fig. 2.3(c)). In this case, $adder_3$ adds the outputs of $multiplier_3$ and $multiplier_2$ to produce the intermediate sum. Meanwhile, $adder_1$ adds

the $multipler_1$ output with the previous partial sum. These operations occur concurrently, provided that the input activations and filter weights are assigned to the multipliers parallelly. Once the outputs from $adder_3$ and $adder_1$ are ready, $adder_2$ sums them up to produce the PE_OUT. The building block containing three MACs and four Muxes is defined as a Process Element (PE) block for convolution in this work. Fig. 2.3 illustrates the functionality of the Reconfigurable core in systolic array mode (b) and convolution mode (c).

2.3.2 STT-MRAM Based On-Chip Memory System

The prime criteria of memory to sustain as an on-chip memory are: high density, low read/write latency and energy. Conventional STT-MRAM suffers from high read/write energy and latency. However, in the case of on-chip memory/global buffer, the intrinsic non-volatility property of STT-MARM can be compromised to minimize the read/write energy and latency by adjusting the thermal stability factor (Δ). Considering the data retention time in the global buffer, Δ can be scaled down to achieve a significant reduction in read/write energy, latency, and increase in cell density. This subsection will formulate necessary expressions to calculate the data retention time in the global buffer for the most time-consuming AI operations, such as the convolution layer and fully connected layer operations. The derived expressions will help us to precisely determine the maximum data retention time in global buffer, and thus help to scale down Δ.

Deep Learning/AI operations are layer-wise sequential operations, meaning the current layer's output acts as input to the following layer. To formulate the data retention time between two consecutive layers, in inference mode, we define T_1 as the time required by the accelerator to generate the ofmap of one layer. Once the ofmap of one layer is generated, it goes through Maxpooling and Activation functions (e.g., ReLU) to serve as the input to the following layer. We refer T_{pool_relu} as the time required to perform the Maxpooling and ReLU operations. The time required to generate the ofmap of the following layer is termed as T_2.

Finally, T_{ret} is the data retention time in memory between two consecutive convolution (or fully connected) layers.

Retention time for Conv-Conv layers

In convolution mode, each PE block of the array performs the dot product between the input feature maps (ifmaps) and weights. Each unit PE block's size is defined as P_s, where P_s represents the number of elements the MAC module can process. The ifmaps and kernel weights are loaded into the PE array from global buffer memory, and the PE array computations occur in parallel. Without loss of generality, in our analysis, we adopted the Row Stationary data flow where kernel rows are loaded into the PE blocks and kept stationary, and ifmaps are loaded and shifted according to the *stride* size [9, 7]. The partial sums are accumulated vertically to generate the output feature maps (ofmaps). This process is repeated until a complete $ofmap$ is generated. Setting $Mode = 1$ in the Muxes of the PE blocks (Fig. 2.3) ensures that the Reconfigurable core is acting as a Convolution core.

To calculate T_1, we formulate an expression that helps us estimate the time required to generate the output (ofmap) of a convolution layer. We assume that the operations related to the next output channel will be assigned in the accelerator array only after all the MAC operations related to the previous output channel have been completed. In other words, in an iteration of the accelerator array, the input channels present in it are all related to the same output channel. In addition to simplifying the PE scheduling procedure, this assumption also aligns with our goal of obtaining a convolution layer's worst-case completion time.

For layer $n-1$, a single row of a partial ofmap (i.e., ofmap corresponding to one kernel and one input channel) will require $(k_h * \lceil k_w/P_s \rceil)$ PE blocks (symbol meanings are given in Table 1), implying that a partial ofmap for a single input channel will require $N_{ofmp\_rw} * k_h * \lceil k_w/P_s \rceil$ PEs. (The $\lceil\rceil$ symbol means *ceil* operation where the result is

Table 2.1: Parameters & description

Parameter	**Description**
N_{in_ch}	Number of input channels
N_{out_chn}	Number of input channels
N_{bat}	Number of images per mini batch
k_h	kernel height
k_w	kernel width
P_s	PE internal size
W_A	Accelerator array width (PEs)
H_A	Accelerator array height (PEs)
W_{SA}	Systolic array width (# of MACs), $P_s * W_A$
N_{ofmap_rw}	Number of rows in ofmap
N_{ofmap_cl}	Number of columns in ofmap
$N_{cyc\_per\_stp}$	Total clk cycles per iteration in conv./systolic mode of accelerator
n_{fc}	number of neurons in input FC layer
m_{fc}	number of neurons in output FC layer
T_{clk}	Clock cycle time

rounded to the nearerst larger integer). An example of convolution operation inside the core in Conv. mode is shown in Fig. 2.4.

Next, we find out how many partial ofmaps (i.e., how many input channels) can be fitted in the full accelerator array in one step. This number is obtained by dividing the total available PEs in the accelerator array, $W_A * H_A$, by the number of PEs required for one input channel. The total number of required steps (i.e, number of times the complete accelerator array will be used) for all input channels (N_{in_ch}) for one 3D filter (i.e, one output channel), $N_{steps\_per\_out\_ch}$, can be expressed as,

$$N_{steps\_per\_out\_ch} = \lceil \frac{N_{in\_ch} \ * \ k_h \ * \ N_{ofmp\_rw} \ * \ \lceil \frac{k_w}{P_s} \rceil}{W_A \ * \ H_A} \rceil \tag{2.2}$$

The details of the symbols used in Equations (2.2)-(2.6) can be found in Table 1. Inside the accelerator, time for each of the above steps,

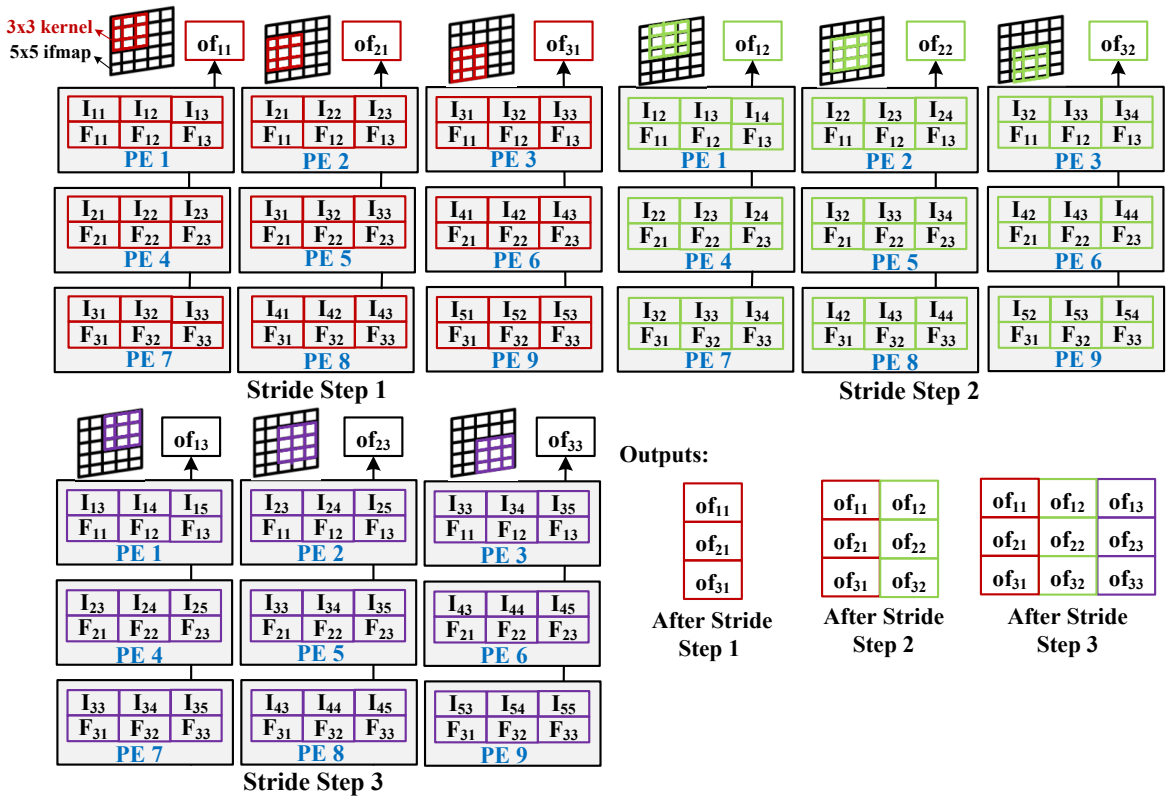


Figure 2.4: A 3×3 kernel ($k_h = k_w = 3$) is convolved (with stride=1) over a 5×5 ifmap to produce 3×3 ofmap ($N_{ofmp\_rw} = N_{ofmp\_cl} = 3$). The size of unit PE block, $P_s = 3$. Total 9 PE blocks are required for this convolution.

$$t_{per\_step} = T_{clk} \ * \ N_{cyc\_per\_stp} \ * \ N_{ofmp\_cl} \ * \ N_{bat} \tag{2.3}$$

$N_{cyc\_per\_stp}$ refers to the total clock cycles required in the accelerator, for one image of the batch, to perform, (i) dot products between the kernel and ifmap elements, (ii) partial sum accumulation of the dot products, and (iii) partial sum of ofmap of previous input channel with current channel. This term depends on the circuit-level implementation of accelerator hardware. The term N_{ofmap_cl} appears in Equation (2.3) because the kernel needs to be shifted (i.e., according to the *stride* parameter of convolution) this many times to generate the partial ofmap for each input channel. N_{bat} appears in the equation since each image from the mini-batch will be serially processed [7]. In between each input channel

operations, the partial sum of ofmaps of the input channels will be stored in the scratchpad to be accumulated to the next input channel's partial ofmaps to finally create the full ofmap output for that particular output channel and filter. The total time required to generate each output channel/ofmap, $t_{per\_out\_ch}$, is given by,

$$t_{per\_out\_ch} = N_{steps\_per\_out\_ch} \ * \ t_{per\_step} \tag{2.4}$$

If there are a total of N_{out_chn} output channels, then the total time required to generate the full ofmap (i.e., for all output channels) is, $T_1 = t_{per\_out\_ch} * N_{out\_chn}$. Using Equations (2.2)-(2.4), the T_1 term can be expressed with the following equation. All parameters in Equation (2.5) are for Conv. layer $n-1$.

$$T_1 = \lceil \frac{N_{in\_ch} * k_h * N_{ofmp\_rw} * \lceil \frac{k_w}{P_s} \rceil}{W_A * H_A} \rceil * T_{clk} * N_{cyc\_per\_stp} * N_{ofmp\_cl} * N_{bat} * N_{out\_chn} \tag{2.5}$$

The ofmap of layer $n-1$ will act as ifmap to next layer n after passing through the ReLU and MaxPool layers. The $ifmap_n$ should be retained in the memory until layer n has finished reading it to generate its output $ofmap_n$. A closer look into the situation will reveal that the input data read time for layer n is related to the $ofmap_n$ generation time. This implies that the $ifmap_n$ data need to be in the memory for a maximum duration of time that is equal to the time required for complete $ofmap_n$ generation. Considering the above facts, we first calculate $ofmap_n$ generation time using the similar methods of Equations (2.2)-(2.5) and then assign it as T_2. All parameters in Equation (2.6) are for Conv. layer n.

$$T_2 = \lceil \frac{N_{in\_ch} * k_h * N_{ofmp\_rw} * \lceil \frac{k_w}{P_s} \rceil}{W_A * H_A} \rceil * T_{clk} * N_{cyc\_per\_stp} * N_{ofmp\_cl} * N_{bat} * N_{out\_chn} \tag{2.6}$$

ReLU and MaxPool layers take relatively much shorter time and also do not involve complex computations as Conv. layers do. Therefore, we can directly estimate T_{pool_relu} from hardware implementation of ReLU and MaxPool layers. Combining T_1, T_2, and T_{pool_relu}, we can estimate the required data retention time, T_{ret}, in memory between two consecutive Conv. layers in inference phase.

$$T_{ret\_conv-conv} = T_1 + T_{pool\_relu} + T_2 \tag{2.7}$$

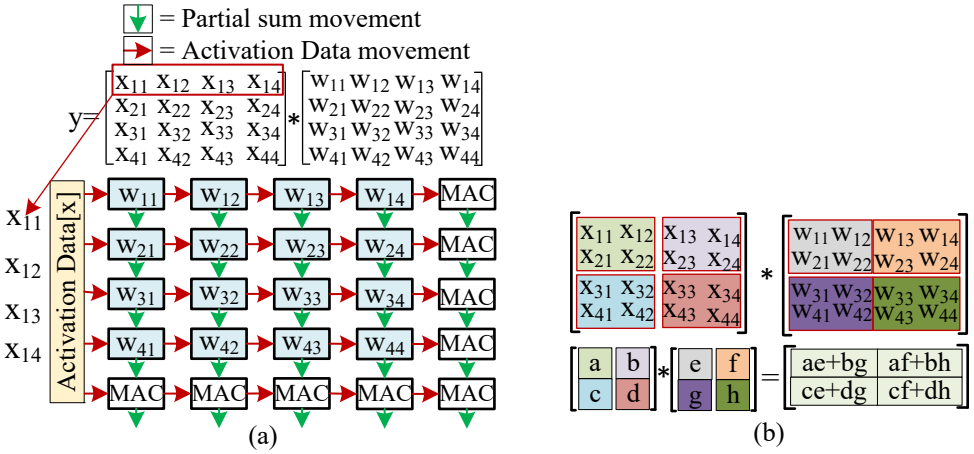


Figure 2.5: (a) Dataflow inside systolic array, (b) Larger matrices can be divided into smaller sub-matrices to fit in the systolic array. An example of dividing two 4×4 matrix into four 2×2 matrices to fit into 2×2 systolic array.

Retention time for FC-FC layers

To perform the computations associated with FC layers, the Reconfigurable Core transforms into the Systolic array. This is achieved by disabling the Mode signal of the Muxes present in the PE blocks. In this section, we formulate an expression to estimate the time required to compute the output of an FC layer. The systolic array shown in Fig. 2.3 (b) has $H_A \times W_{SA}$ MAC modules. Because of the reconfigurable feature of the core, $W_{SA} = P_s * W_A$. The weights are loaded into the array according to the capacity of the systolic array implying

that the number of weights can be loaded into the array in one step is equal to the number of MACs present in the array, $N_{wt\_per\_step} = H_A * W_{SA}$. If the total number of weights is greater than the number of weights the array can accommodate in one step, i.e., $N_{tot\_wt} > N_{wt\_per\_step}$, using the concept of *divide & conquer* in matrix multiplication (Fig. 2.5 (b)) we find out how many steps (i.e., number of times we need to load new weights to the accelerator array) are required to complete the computation with all elements of the weight matrix. The number of steps required to complete the computation with all weights is, $\lceil m_{fc}/H_A \rceil * \lceil n_{fc}/W_{SA} \rceil$. (For symbol meanings see Table 1). In every step, the array is loaded with $N_{wt\_per\_step}$ weights, inputs are streamed from left to right, and the partial sums move downward to be collected in accumulators [155]. The clock cycles required to complete each step are $N_{cyc\_per\_stp}$ that depends on the circuit-level implementation of systolic array hardware and the dimension of systolic array core. Combining the above terms, and considering there are N_{bat} images in the mini-batch, time required to generate the output of FC layer $(n-1)$, T_1, is expressed in Equation (2.8), where all parameters are for FC layer $(n-1)$.

$$T_1 = \lceil \frac{m_{fc}}{H_A} \rceil * \lceil \frac{n_{fc}}{W_{SA}} \rceil * T_{clk} * N_{cyc\_per\_stp} * N_{bat} \tag{2.8}$$

FC layer n will consider the output of previous FC layer $(n-1)$ as its input. The output of $\mathrm{FC_{n\text{-}1}}$ should be stored in the memory until $\mathrm{FC_n}$ has completed reading it for generating its output. With this reasoning, we calculate the output generation time for $\mathrm{FC_n}$ following the above method and assign it as T_2. All parameters of Equation (2.9) are for FC layer n.

$$T_2 = \lceil \frac{m_{fc}}{H_A} \rceil * \lceil \frac{n_{fc}}{W_{SA}} \rceil * T_{clk} * N_{cyc\_per\_stp} * N_{bat} \tag{2.9}$$

Two consecutive FC layers do not have MaxPooling layer in between. Therefore, we can find the data retention time, T_{ret_fc-fc}, for an FC layer followed by another FC as,

$$T_{ret\_fc-fc} = T_1 + T_2 \tag{2.10}$$

2.3.3 Retention time of Convolution layer followed by FC layer

The retention time between a Convolution layer followed by an FC layer is also expressed as,

$$T_{ret\_conv-fc} = T_1 + T_{pool\_relu} + T_2 \tag{2.11}$$

Here, T_1 is the time required to generate the Conv. layer ofmap and T_2 is the time required to generate the FC layer output.

Using the above expressions of weight, ifmap, and ofmap occupancy times in global buffer memory, for a particular accelerator hardware architecture and the operating clock frequency, we can estimate the maximum retention time required for STT-MRAM based global buffers. The MRAM Write and Read times will be added with the above retention time expressions. As the MRAM Read/Write times are orders of magnitude lower (i.e., less than 10ns) compared to the retention times T_1 and T_2 which are in the ms or s range as explained in Section IV, we did not explicitly add the MRAM Read/Write time with the above retention time (T_{ret}) expressions.

2.4 Optimizing STT-MRAM for AI Accelerators

A bit cell of STT-MRAM consists of a Magnetic Tunnel Junction (MTJ) for storing the bit and an access transistor to read/write the bit. The MTJ contains two ferromagnetic layers, one with fixed magnetic orientation, and another free layer whose orientation can be switched externally by an applied current. The orientation of the free layer relative to the

reference layer represents the state of the stored bit; parallel orientation refers to logic 0, while anti-parallel orientation refers to logic 1. Fig. 2.6 depicts the cell schematic, read, and write operations.

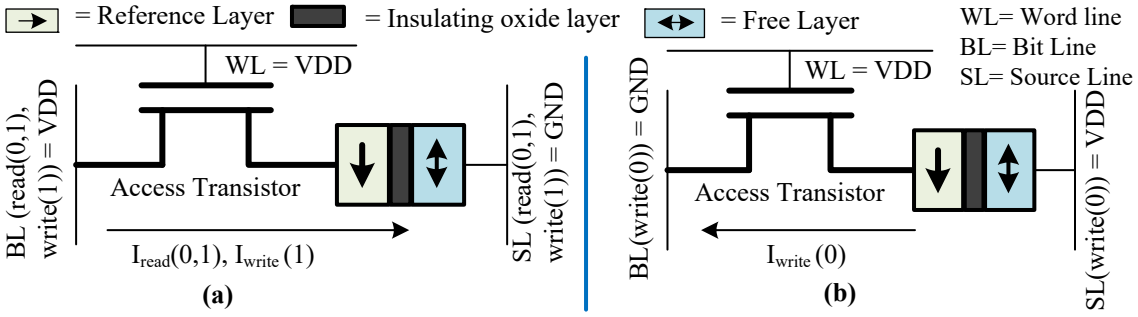


Figure 2.6: Bit cell of STT-MRAM. (a) shows reading from it and writing 1, (b) shows writing 0.

2.4.1 Critical Design and Performance Parameters of MTJ

Thermal Stability Factor and Critical Current

The energy barrier E_b, which the free layer magnetization must overcome to switch its stable state is defined as the thermal stability factor (Δ), and is expressed as [64, 65]:

$$\Delta = \frac{E_b}{k_B T} = \frac{H_K M_S V}{2 k_B T} \tag{2.12}$$

Where, E_b = Energy Barrier of free layer, k_B = Boltzmann Constant, T = Temperature, H_K = Anisotropy field, M_S = Saturation Demagnetization, V = Volume of MTJ.

Critical current, I_c, is defined as the minimum current required to flip the state of the free layer [64, 65, 66]. The critical switching current is modeled as[64, 65]:

$$I_c = (\frac{4 e k_B T}{h}) * \frac{\alpha}{\eta} * \Delta * (1 + \frac{4\pi M_{eff}}{2 H_K}) \tag{2.13}$$

Where, e = electron charge, k_B = Boltzmann Constant, T = Temperature, h = Plank's Constant, α = LLGE damping constant, η = STT-MRAM efficiency parameter, $4\pi M_{eff}$ = Effective demagnetization field, and H_K = Anisotropy field.

Retention Time & Retention Failure

Once data is written, MTJ should retain its state, even if the power source is removed, until any external force is applied to flip the state. However, due to thermal noise, the logic state might get flipped unintentionally. The maximum time MTJ can retain its non-volatility is known as data retention time. The retention failure probability for a given time period t_{ret} is [64, 65]:

$$P_{RF} = 1 - exp\left[-\frac{t_{ret}}{\tau * exp(\Delta)}\right] \tag{2.14}$$

Where, t_{ret}= retention time, τ= technology constant.

Read Disturbance (RD)

To read a bit from STT-MRAM, read current I_r, much less than the critical current I_c, is flown from bit line through the access transistor and MTJ. Fig. 2.6 shows that writing 1 and reading (both 0 & 1) share the same current path. This can sometimes cause the unintentional switching of the bit-cell content resulting in Read Disturbance (RD). For read current I_r and read latency t_r, the probability of RD can be modeled as[64, 65]:

$$P_{RD} = 1 - exp\left[-\frac{t_r}{\tau * exp(\Delta(1 - \frac{I_r}{I_c}))}\right] \tag{2.15}$$

Write Error Rate (WER)

Writing a bit cell requires a write current I_w, larger than I_c, to be flown between BL to SL as shown in Fig. 2.6. Because of the stochastic nature of the write operation, the

switching time of MTJ varies from access to access [64, 65, 66]. If the write current is terminated before the free layer has successfully changed its state, the write operation can be erroneous. For write pulse width t_w, the Write Error Rate (WER) is [64, 65]:

$$WER_{bit} = 1 - exp\left[\frac{-\pi^2\Delta(\frac{I_w}{I_c} - 1)}{4[\frac{I_w}{I_c} * exp\{\frac{t_w}{\tau}(\frac{I_w}{I_c} - 1)\} - 1]}\right] \tag{2.16}$$

2.4.2 Customizing STT-MRAM For AI Accelerators

Scaling Thermal Stability Factor

To achieve typical retention period of 10 years, the thermal stability factor, $\Delta \geq 60$ is required [59, 62, 63, 64, 65, 66]. However, such a long retention time may be unnecessary depending on the application. For example, if MRAM is used as the global buffer memory in AI accelerators, then the retention time can be significantly scaled depending on the weight and input/output feature-map data occupancy time (e.g., ms to s range) in that memory. If STT-MRAM is used as eFlash replacement for pre-trained weight storage for AI inference tasks, then 3 to 5 years retention might be enough instead of 10 years. From Equation (2.12), it is seen that by adjusting the volume (i.e., area and/or thickness) of the MTJ the thermal stability factor (Δ) can be scaled. In other words, considering the target operating temperature range of the AI accelerator and the expected life-time of the data, scaling down of thermal stability factor will improve area efficiency by increasing the memory bit-cell density. Moreover, with scaled Δ and bit-cell area, the cell would require a lower operating current, thus saving energy.

Optimizing Read/Write Latency and Energy at Target WER and RD

Recent state-of-the-art STT-MRAMs can compete or outperform SRAMs in all aspects except write energy and write latency [69, 62, 61]. However, for AI accelerator applications,

by scaling Δ and the retention time of STT-MRAM we can circumvent the write energy and latency limitations. Equation (2.16) implies that write latency, $t_{pw} \propto ln(\Delta)$ at constant write error rate. We can exploit this relationship to reduce the write latency with scaling down of Δ. From Equation (2.14) we infer that retention time t_{ret} is exponentially proportional to Δ. Thus, depending on the desired retention period of STT-MRAM in AI accelerator, we can optimally scale down Δ, and also minimize write latency at that target retention time. However, Equation (2.16) also implies an inverse relationship between write latency and write error rate, which hinders us from aggressively scaling down write latency at the desired Δ. Fortunately, to increase the writing speed at the scaled Δ, we can keep I_w higher (e.g. close to the prescaled value), and this can assist in designing a STT-MRAM with high write speed [66]. Recently, high-speed writing has been experimentally demonstrated in [78] by optimizing the free-layer materials. We can identify the optimum Δ and I_w that minimizes write latency and write energy while still satisfying the WER and retention time requirements for the AI accelerator. As depicted in Equation (2.13), with scaling of Δ, the critical current I_c decreases linearly, and hence read current I_r also decreases. At this scaled Δ and I_r, the read latency can also be scaled by adjusting the sense amplifier reference voltage [66, 69]. Equation (2.15) implies that at scaled Δ, the shortened read pulse duration will also ensure that the Read Disturb rate is within the acceptable target.

2.4.3 Addressing Process and Temperature Variation

The performance of MRAM can degrade due to the process and temperature variations [62, 69, 79]. Process-induced variations in free layer thickness in MTJ, and in access transistor channel length/width and threshold voltage contribute to the performance variations in MRAM. From Silicon measurement data in [69], the standard deviation (σ) of MTJ diameter variation was reported to be 2.1% of the mean. Magnetic Anisotropy field (H_K) is another source of process variation in STT-MRAM.

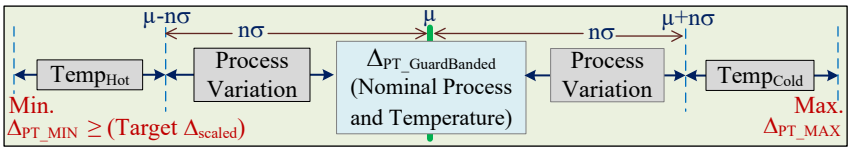


Figure 2.7: Impact of process and temperature variation on thermal stability factor (Δ).

The bit-cells are placed compactly on the layout, as a result, the bit-cell to bit-cell variations within the same die/chip are minimal, and the process variation is dominated by the chip-to-chip variations. Δ increases with an increase in MTJ diameter and H_K due to process variation, and a decrease in temperature from the nominal value (Fig. 2.7 and Equation 2.12). An increase in Δ increases critical current (I_c) which eventually increases the write current (I_w) (Equation 2.13 and 2.16). Given the smaller write pulse, write failure occurs when supplied I_w is less than the required I_w. Worst-case occurs when both, (i) the supplied I_w decreases due to the access transistor being in the slow process corner, and (ii) the required I_w increases due to increase in Δ resulting from Process and runtime Temperature (PT) variations. On the other hand, decrease in Δ beyond a minimum due to PT variation will result in retention failure (Equation 2.14).

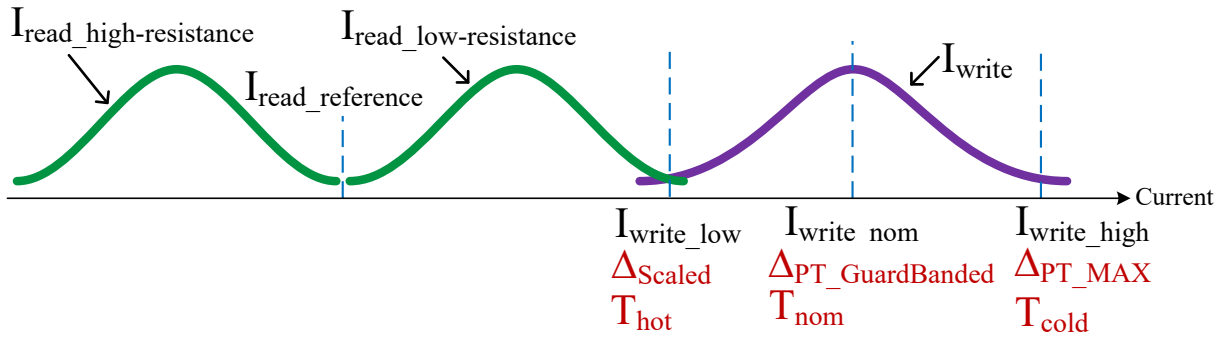


Figure 2.8: Distribution of read/write currents with PT variation. Worst-case occurs when worst process corners experience T_{hot} or T_{cold}.

To protect the desired Δ_{scaled} against the worst-case PT variation, appropriate Guard-Band needs to be added. The $\Delta_{PT\_GuardBanded}$ is chosen to cover both the worst-case 4σ range (i.e., 99.993% of the samples) of process variation and high temperature operating

scenario as shown in Equation 2.17.

$$\Delta_{scaled} \leq (\Delta_{PT\_GuardBanded} - 4\sigma) * (T_{nom}/T_{hot}) \tag{2.17}$$

$$\Delta_{PT\_MAX} = (\Delta_{PT\_GuardBanded} + 4\sigma) * (T_{nom}/T_{cold}) \tag{2.18}$$

The chip samples located on the right side of the process variation distribution (i.e., $\mu + n * \sigma$, where $n \geq 1$) of $\Delta_{PT\_GuardBanded}$, will experience larger Δ as shown in Fig. 2.7 and 2.8. Additionally, at cold temperatures, the Δ will further increase to Δ_{PT_MAX} as shown in Equation (2.18). Although the higher $\Delta_{PT\_MAX} > \Delta_{scaled}$ will be benign for retention time, the required write current will increase in this scenario to confine the write time and Write Error Rate (WER) of these samples within the nominal bound. Designing the write driver for this worst-case scenario will dissipate unnecessary power for all other non-worst-case samples. To address this, we propose a dynamically adjustable write driver depicted in Fig. 2.9. The proposed write-driver circuit provides additional write current in extreme PT conditions. The Process and Temperature Monitor (PTM) block continuously monitors the process and temperature changes. In addition to adjusting the write current according to process variation profile of the MRAM chip/die, the PTM also senses the runtime temperature and adjusts the current dynamically by turning on/off the additional PMOS transistors (Fig. 2.9). For example, at the nominal process and temperature, the extra transistors can be off, however, at PT-induced rise in Δ the transistors can be individually activated to ensure successful write.

In summary, we design the MTJ with higher $\Delta_{PT\_GuardBanded}$ than the desired Δ_{scaled} to accommodate potential degradation in thermal stability from worst-case 4σ process variation

and runtime high temperature, and proposed a write-driver with controllable current drive to address the high write-current demand at cold temperature and slow process corner.

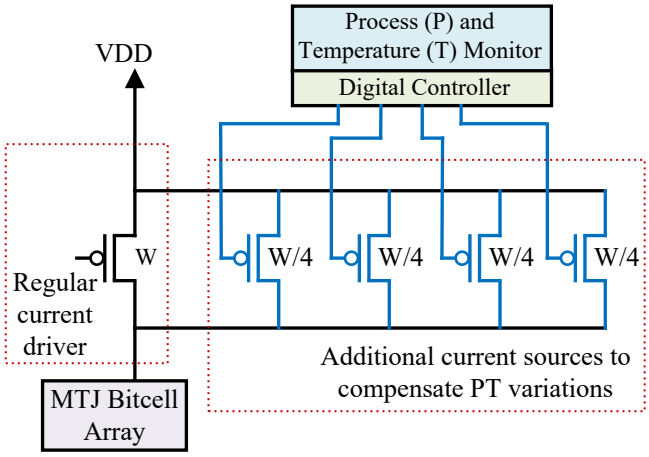


Figure 2.9: Modified Write Driver

2.4.4 MRAM Write Energy Optimization in Accelerator with ScratchPad

Addressing the fact that the write energy of STT-MRAM is higher than the read energy, we propose an innovative scratchpad-assisted MRAM global buffer architecture to minimize the write frequency in STT-MRAM, and thus further optimize the energy. Reduced write-frequency is achieved by using a small global SRAM scratchpad, typically in the KB range (details in Section 2.5), in addition to a large (i.e., MB range) global STT-MRAM buffer. When the accelerator PE array generates partial ofmaps (i.e., ofmap corresponding to each input channel), they need to be stored somewhere in memory to be added to the next partial ofmap to produce the complete ofmap of an output channel. The reason behind these partial-ofmap writes is that the accelerator might not produce the complete ofmap in one step. Between the subsequent steps, the partial ofmap result from the previous step needs to be written in the memory to be subsequently read and accumulated with the partial ofmap result from the following step. Adding this small SRAM scratchpad memory (for intermediate ofmap writes) with MRAM global buffer further improves the energy efficiency

of MRAM-buffer-based deep learning accelerators. In summary, our proposed scratchpad-assisted MRAM memory architecture provides energy efficiency by, (i) minimizing the STT-MRAM write frequency, and (ii) additionally, at a smaller size, SRAM is more energy-efficient than STT-MRAM [69].

2.5 Results and Analysis

2.5.1 Design Space Exploration for Selecting Memory Capacity

Nineteen widely used state-of-the-art deep learning models were analyzed to design and validate our STT-MRAM based AI accelerator with the reconfigurable core. Fig. 2.10 (a) shows the model sizes both in 8-bit int8 (left Y-axis) and 16-bit BrainFloat16 (BF16) [82, 83] (right Y-axis) datatypes. For inference-only accelerator int8 datatype and hardware suffice, however, if full-scale training or transfer-learning is desired then BF16 hardware and data-type are necessary [82, 83]. The models' sizes imply that around 280MB and 140MB of STT-MRAM is required as non-volatile (NVM) weight storage memory to store the pre-trained models using BF16 and int8 datatypes, respectively. The STT-MARM non-volatile weight storage memory can replace the currently used eFlash memory as an efficient alternative. Fig. 2.10 (b) and (c) represent the input/output featuremap and weight size ranges of all models for convolution layers both in int8 (left Y-axis) and BF16 (right Y-axis) formats, and these data helps us to estimate the maximum required global buffer (GLB) memory size to avoid DRAM accesses during each convolution layer operation. In the cases of fully-connected layer operations, only the featuremaps, usually in KB range for most of the models, are stored in the GLB, and the weights, around 200MB in size for the largest model in BF16, are directly assigned from DRAM (or weight-storage NVM) to the systolic array for matrix multiplications. Hence, we ignored the fully connected layers' weight and activation sizes from design space analysis for selecting on-chip GLB memory capacity.

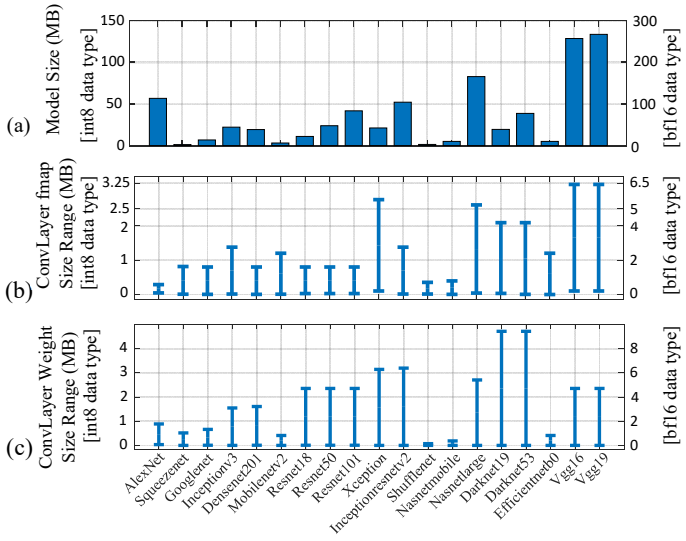


Figure 2.10: (a) Complete sizes of widely used AI models. (b) Activation map (ofmap/ifmap) sizes, (c) Weight sizes for Conv layers.

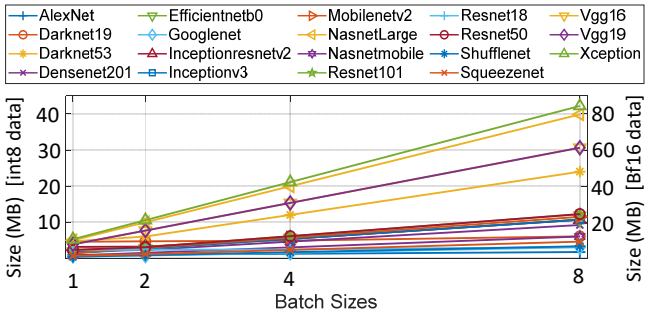


Figure 2.11: Required capacity of global buffer with varying batch sizes to avoid DRAM access during inference.

To fit a Conv. Layer data completely into the GLB, it needs the capacity to store the (i) input fmap (ifmap), (ii) filter weights, and (iii) output fmap (ofmap) of current layer. Fig. 2.11 shows the required GLB size for 19 widely-used deep learning models in int8 (left Y-axis) and BF16 (right Y-axis) for different batch sizes. For smaller batch-size (i.e., ≤ 2), a maximum of 12MB of GLB would be enough for int8 datatype. With 12MB on-chip GLB memory, most of the models, except a few (e.g., Darknet53, VGG19, Nasnetlarge, Xception, etc.), can support larger batch-sizes such as 8. For BF16, 12MB would suffice for batch size 1 for all models. If pruned models [9] are used, batch of more images can be fit into the

GLB. For high-performance accelerators that operate with larger batches of data, the GLB size can be further increased.

When a Conv. layer data - ifmap, weight, and ofmap - do not fit into GLB at one attempt, extra DRAM accesses are needed, incurring extra energy and latency. Fig. 2.12 (a) shows, if a GLB of 12MB is used, even larger batch sizes, such as 8, the extra DRAM access-related latency is zero for most of the models (int8 case), and around $2ms$ for few models. For BF16 datatype, the extra DRAM access latency increases slightly but is within $10ms$. Fig. 2.12 (c) depicts that if the GLB size is 12MB, for most of the models in int8 datatype extra DRAM access-associated energy reduces to zero. For BF16 datatype, most models would need a few extra DRAM accesses (Fig. 2.12 (d)). The DRAM access energy and latency were calculated for dual-channel DDR4-2933 DRAM with 64bit data bus.

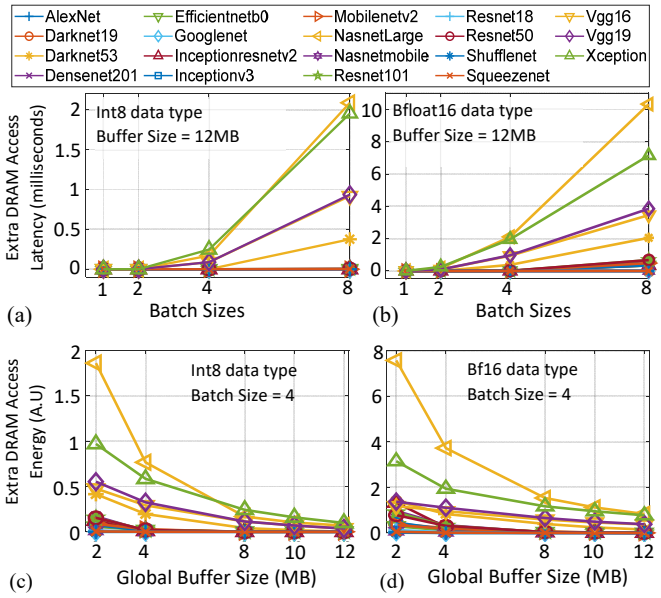


Figure 2.12: For Conv. layers, total extra DRAM access latency for varying batch sizes: (a) int8, (b) BF16 data types; total extra DRAM access energy for varying GLB size: (a) int8, (b) BF16 data types.

2.5.2 Memory Retention Time Estimation for AI Models and Accelerator Architecture

The data retention time in GLB for the models (in BF16 datatype) are calculated using Equations (2.5) - (2.11) (Section 2.3) and the post-layout timing results from the implementation of our proposed reconfigurable accelerator core at 14nm technology (Table 2.2). The results for 42×42 MAC array and batch size 16, presented in Fig. 2.13, show that the maximum data retention time in GLB for all models is less than 1.5s where most models have retention time less than 0.5s. The retention time goes even smaller (in ms range) for int8 datatype as the clock cycle reduces significantly (usually 1-2 clock cycles) in int8 version hardware. Fig. 2.14 (a) shows the maximum retention time for all models (in BF16 datatype) for fixed batch size 16 and varying MAC array sizes, whereas Fig. 2.14 (b) shows the maximum retention time needed for a fixed MAC array size of 42×42 for varying batch sizes. From the figures, it is evident that further reduction in retention time can be achieved by using the proper combination of batch and MAC array sizes.

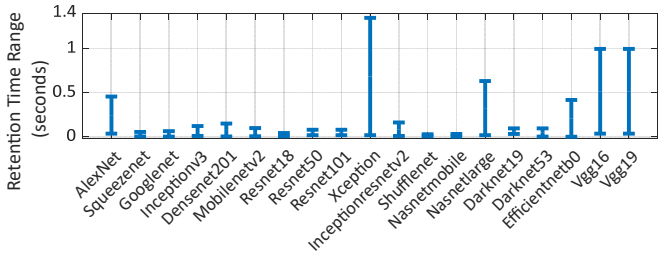


Figure 2.13: Global buffer retention time range for 42x42 MAC array (Bfloat16 hardware, CLK details in Table 2.2) and batch size 16.

2.5.3 Customizing STT-MRAM for AI Accelerator

Using Equation (2.14), we analyzed the impact of thermal stability factor (Δ) on retention time within certain Bit Error Rates (BER). To identify the target BER of STT-RAM

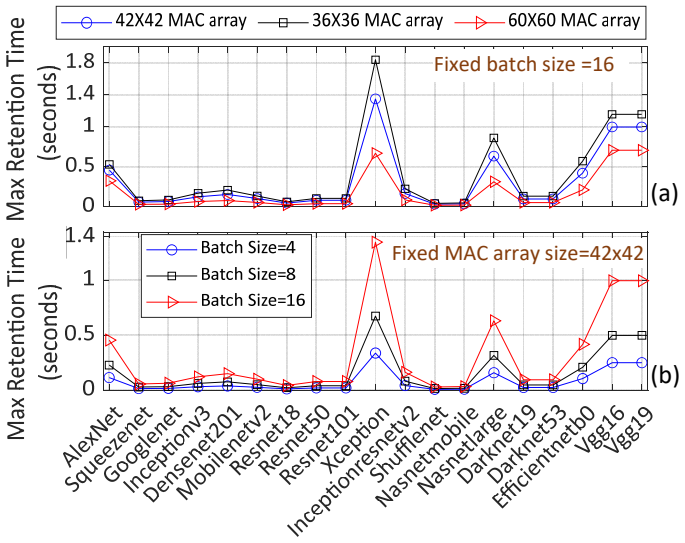


Figure 2.14: The required retention time of MRAM global buffer for Bfloat16 hardware (CLK cycles and frequency given in Table II), (a) varying MAC array capacity. (b) varying batch sizes.

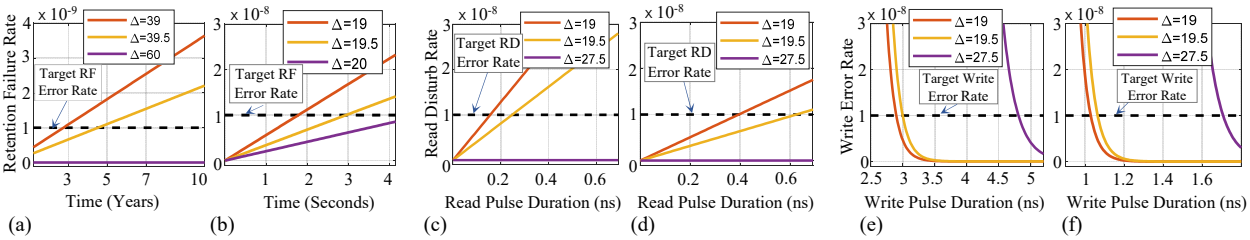


Figure 2.15: (a) Thermal stability (Δ) scaling for 3 years retention time (for pre-trained weight storage NVM application). (b) Δ and retention time scaling for accelerator's global buffer memory design. (c), (d) With scaled Δ, read pulse width scaling while ensuring RD BER is within limit. (e), (f) Write latency scaling with Δ, within target write error rate. Note: (c), (e) uses base-case (10yrs ret. time) from [69], and (d), (f) from [62]. Target BER is chosen to ensure no accuracy impact on AI tasks [76].

for applications in pre-trained weight storage and global buffer (GLB) memory, we first analyzed the size of modern AI models. From Fig. 2.10 (a), it can be seen that a few hundred MBs would be enough to store the pre-trained weights, within this memory capacity we choose BER in the order of 10^{-9} (i.e., 1 bit-flip per 1 billion bits). Given the worst-case cumulative BER can occur from Retention Failure (RF), Read Disturb (RD), and Write Error (WE), the worst-case bit-flips for VGG16 at this BER is about 12 bits. This BER is

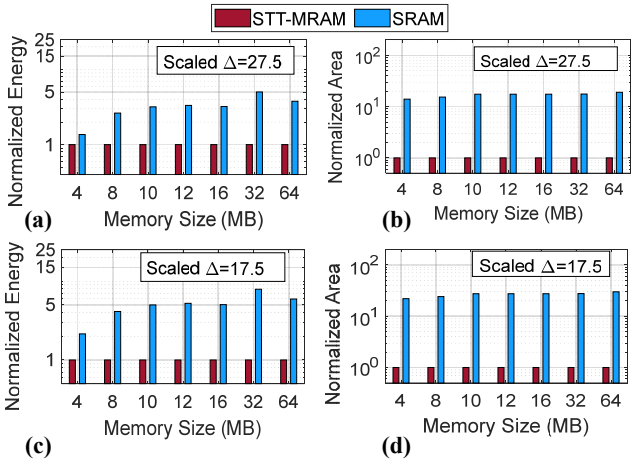


Figure 2.16: Energy and area comparison of SRAM and STT-MRAM for various sizes. Δ scaled: (a),(b) for global buffer; (c), (d) for eMRAM banks to store lower half (i.e., LSB groups) of weight/fmap bits.

negligible and cannot make any impact on the AI task's accuracy [76]. Fig. 2.15 (a) shows that with $\Delta = 39$ we can ensure the loaded pre-trained weight will successfully remain in the accelerator for about 3 years at this target BER, which is enough given that AI models are replaced frequently with better ones. To address Process variation and runtime Temperature fluctuation, we chose $\sigma = 2.1\%$ of mean, $T_{hot} = 120°\text{C}$ $(393K)$ and $T_{cold} = -20°\text{C}$ $(253K)$ in the Equations 2.17 and 2.18 as discussed in Section 2.4.3 and adjust $\Delta = 39$ to $\Delta_{PT\_GB} = 55$ after guard-banding.

For GLB memory, we can lower the Δ and retention time much lower according to the average occupancy time of weight and input/output fmaps in the accelerator's GLB memory. Also, since this memory size is within few tens of MB (e.g., 12MB), we can increase the BER to 10^{-8}, which will cause less than 3 bit-flips in the worst-case (i.e., considering BER from Retention Failure (RF), Read Disturb (RD), and Write Error (WE)) at this memory size. The accuracy impact of deep learning models at this BER and memory size is negligible [76]. In Fig. 2.15 (b), at scaled $\Delta = 19.5$ we can achieve 3 seconds of retention time (which is much higher than the minimum required as shown in Fig. 2.14) at the target BER of 10^{-8}.

Next, we analyze the impact of scaling Δ on the read pulse width. If the read pulse width is large then the chances of RD increase. Moreover, with scaling $\Delta = 19.5$ (after Guardbanding $\Delta_{PT\_GB} = 27.5$), the required read current also decreases. As a result, a significant reduction in read energy is also possible. In our study of Δ scaling impact of read/write latency, as the base-case STT-MARM we used the chip-implemented (for 10 years retention) data of [62, 69]. Fig. 2.15 (c), (e) uses base-case (i.e., $\Delta = 60$) from [69], and (d), (f) from [62]. With the scaling of retention time, the write latency only scales as a factor of $ln(\Delta)$, to further decrease the write latency we can use the write current as another knob as discussed in Section 2.4. The write latency scaling are shown in Fig. 2.15 (e), (f).

We used Destiny memory modeling tool [81] to compare STT-MRAM area and energy with SRAM while Δ is scaled down. Although theoretically, STT-MRAM has a minimum area of $6F^2$, however, silicon results show that MRAM area scaled by 70% compared to SRAM at 14nm node [69]. We modified the Destiny tool to incorporate this silicon observation. The results for scaled Δ at 14nm technology node are shown in Fig. 2.16. We see a significant advantage from STT-MRAM beyond 4MB capacity. Compared to SRAM, the area scales by more than ten times at iso-memory-capacity (Fig. 2.16 (b),(d)). Similarly, for STT-MRAM the relative energy efficiency improves as the memory capacity increases (Fig. 2.16 (a), (c)). These results imply STT-MRAM can offer significant performance gains at future high-performance AI accelerators that will use large on-chip buffer memory.

2.5.4 Energy Optimization with Variable Retention MRAM Banks

We further improved the efficiency in *STT-AI Ultra* accelerator with two separate MRAM banks of, $\Delta = 19.5$ (after PT guard-band $\Delta_{PT\_GB} = 27.5$), and $\Delta = 12.5$ ($\Delta_{PT\_GB} = 17.5$). The first half of the weight/fmap bits are considered significant (MSB group) and stored in $\Delta_{PT\_GB} = 27.5$ bank, and the rest of the LSB groups in $\Delta_{PT\_GB} = 17.5$ bank. For

the LSB group at $\Delta_{PT\_GB} = 17.5$ we relaxed the BER to 10^{-5} as shown in Fig. 2.17. The relative gains in energy and area at $\Delta_{PT\_GB} = 17.5$ are shown in Fig. 2.16 (c), (d).

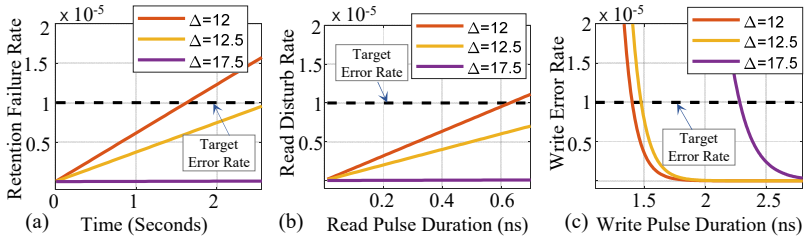


Figure 2.17: Δ scaling with relaxed BER for LSB bit groups. (a) Retention, (b) Read, and (b) Write latency within target BER. (Base case, $\Delta = 60$, data modeled after [62]).

2.5.5 Optimizing Energy with Scratchpad for Partial Ofmaps

Our simulation results show that for STT-MRAM the write energy is about 70% more than the read energy at scaled Δ. As described in Section 2.4.4, using a small SRAM scratchpad for writing the intermediate partial ofmaps instead of the MRAM can significantly reduce the write frequency and save energy. Fig. 2.18 shows the partial ofmap size distribution. For BF16 data type, we see that 52KB (26KB for int8) scratchpad will fit most of the models in one attempt. The normalized energy improvements of proposed scratchpad-assisted MRAM system is shown in Fig. 2.19 for ResNet-50 model and 14nm technology.

2.5.6 Accelerator Implementation

We implemented our AI accelerator architecture with reconfigurable cores (i.e., in Fig. 2.3), at RTL level using BF16 hardware as BF16 can support both inference and training.

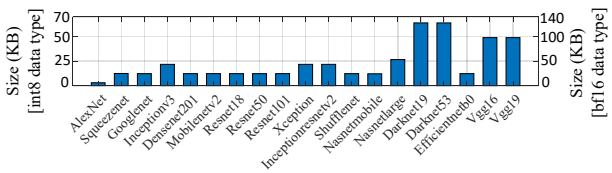


Figure 2.18: Maximum size of partial ofmaps.

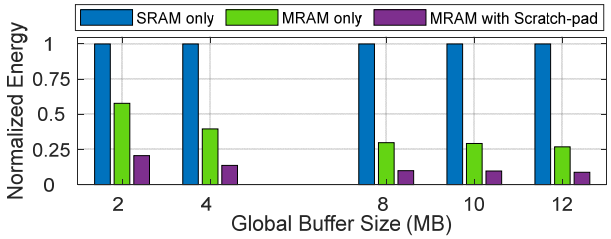


Figure 2.19: Comparison of buffer memory energy dissipation for SRAM, MRAM, and MRAM with scratch pad architectures.

Table 2.2: Reconfigurable PE core details (Bfloat16 hardware, and synthesized with 14nm standard cell library [85])

Reconfigurable Core Mode	**CLK Freq**	**Required CLK Cycles**
Systolic Core (1 MAC)	1 GHz	11
Conv. Core (3 MAC)	1 GHz	17

We used Synopsys 14nm standard cell library [85] to complete synthesis, and place and route of the design. The post-layout CLK cycle data for the PE/MAC are shown in Table 2.2. The top-level view of physical design from ICC2 tool [85] is shown in Fig. 2.20. We used Synopsys 14nm memory compiler to create the SRAMs for our baseline accelerator. Results from post-layout and timing-closed accelerator design are shown in Table 2.3, where Row 7 shows the area and power for the base-line accelerator with 12MB global buffer memory. Next, to implement MRAM based *STT-AI* accelerator, we estimated areas and power data from the Destiny [81] tool at scaled Δ and modeled those as blackbox in the physical design part in Synopsys ICC2 [85] for 14nm node. The 52KB SRAM Scratchpad is divided into two banks with individual CLK/power gating. Rows 4 and 8 show that the *STT-AI* accelerator offers significant area and leakage energy savings. The *STT-AI Ultra* accelerator achieves further improvements in power and area as shown in Row 9 in Table 2.3.

Table 2.3: Accelerator Design Details at 14nm

Module	**Details**	**Area** (mm^2)	**Dynamic Power** (mW)	**Leakage Power** (mW)
Functional Core	Reconfigurable core with 42x42 MACs	4.08	954	0.91
SRAM Block	12 MB SRAM global memory	16.2	48.98	0.21
STT-MARM (Δ=27.5)	12 MB MRAM global memory	1.01	17.61	0.08
STT-MRAM (Δ=17.5, Δ=27.5)	6 MB MRAM (Δ=17.5) 6 MB MRAM (Δ=27.5)	0.93	13.75	0.06
SRAM ScratchPad (for MRAM)	52 KB (two 26KB blocks with CLK/power gating)	0.069	0.2	8E-4
Baseline Accelerator (SRAM-based)	Functional Core and SRAM (Row 3 above)	20.28	1003	1.13
STT-AI Accelerator	Functional core and STT-RAM (Row 4 above)	5.09	972	0.99
STT-AI Ultra Accelerator	Functional Core and STT-RAM (Row 5 above)	5.0	968	0.98

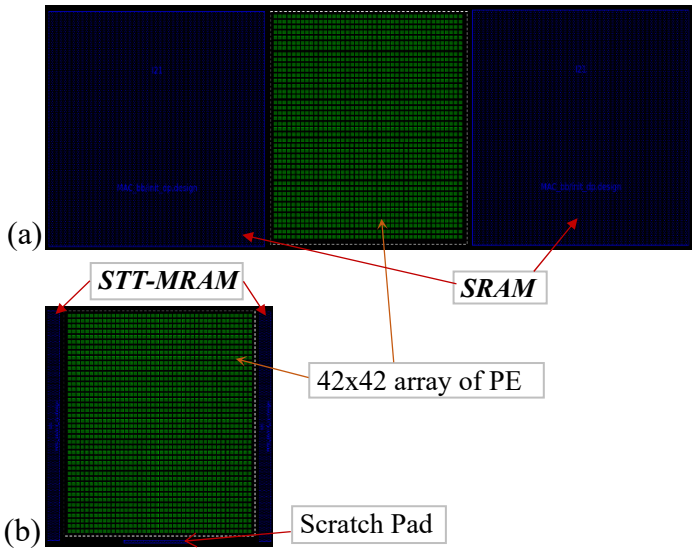


Figure 2.20: Top level floorplan view from ICC2. Accelerator designed with, (a) 12MB SRAM. (b) 12MB STT-MRAM with scratchpad.

2.5.7 Accelerator Performance with ImageNet Dataset

Next, we modeled our hardware and STT-MRAM BERs in PyTorch [84] and ran inference for pre-trained AlexNet, VGG16, and ResNet-50 models with ImageNet benchmark to obtain Top-1 and Top-5 accuracy results. As expected, with STT-MRAM having $\Delta_{PT\_GB} = 27.5$, there were no accuracy loss compared to the baseline SRAM version. However, for *STT-AI Ultra* accelerator, with $BER = 10^{-5}$ in half of the bits (LSB group in lower Δ STT-MRAM bank), we observed negligible (less than 1% normalized) accuracy loss as shown in Fig. 2.21.

2.6 Related Work

Over the last decade, STT-MRAM technology has been extensively researched for its high-endurance, radiation hardness, non-volatility, and high-density memory properties. The prior works on STT-MRAM applications can be broadly categorized into two domains: (i)

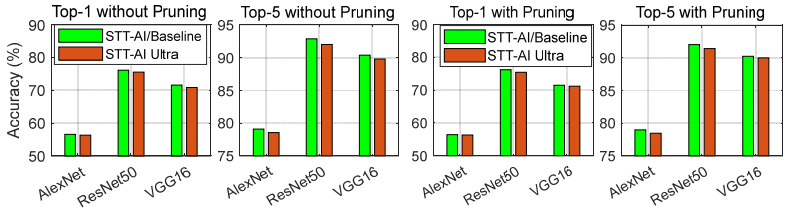


Figure 2.21: Top-1 and Top-5 accuracy comparisons for *STT-AI/Baseline* and *STT-AI Ultra* cases. No accuracy change for *STT-AI/Baseline* cases, and negligible (less than 1% normalized) accuracy change occurs on *STT-AI Ultra* acclerator. Both original and pruned (at 50% pruing rate) [9] model results are shown.

Its use as the last-level cache memory in processors; (ii) Its application in emerging Process-in-Memory (PIM) based computing paradigm.

Several studies [90, 87, 88, 89] have demonstrated the excellence of STT-MRAM over other NVM technologies in PIM setting due to complex and tunable resistance dynamics achieved through its spin-transfer torque mechanism and simultaneous access to multiple word-lines of the same array. [87] proposed an STT-MRAM based crossbar arrays where the internal resistance states - which were used to mimic the weights of the models - of STT-MRAM were tuned to support non-uniform quantization. This study provided energy efficiency and loss reduction, however, additional circuitry, Digital to Analog Converter (DAC) and Analog to Digital Converter (ADC) were needed to support the PIM workflow. Some studies, such as, [91] used the PIM architecture, where the MAC operation was simplified to addition/subtraction, and bit-wise operation by manipulating the models' parameters. [89] mapped the LeNet5 model to a synaptic cross-bar array of STT-MRAM memory cells for inference and showed improved performance in terms of area, leakage power, energy over SRAM for 65nm to 7nm technology node. However, major challenges of PIM over conventional Deep Learning/AI are - (i) requirements of additional hardware circuitry, such as DAC, ADC, which results in area overhead; (2) quantization of weights to be represented with fewer bits resulting in lower precision; (3) in digital PIM, extra manipulation of models' algorithm is needed to replace MAC with the bit-wise operation; and (4) in most of

the cases, PIM is only suitable for inference-only applications. Although PIM-based analog architectures provide fast execution, the performance, energy efficiency, and reliability of analog PIM still lags behind the state-of-the-art DNN/AI models and their corresponding hardware accelerators [9, 155, 70].

While some research leveraged the scalable property of thermal stability factor of STT-MRAM to replace SRAM-based cache memory, others used the error tolerance property of certain applications and designed STT-MRAM based energy-efficient cache with approximate storage. In [72], the retention time of STT-RAM was scaled to implement cache memory that can compete with SRAM-based caches, and DRAM-like refresh was used to compromise the ultra-low retention time. In [73, 94] STT-MRAM based approximate cache was proposed to exploit the error-resilience property of some specific applications. Unlike previous studies, [101] proposed a hybrid STT-MRAM design for cache for different applications depending on the run-time requirements without compromising any reliability degradation. In [59], area-and-retention-time-scaled STT-MRAM was presented as DRAM replacement.

In [71] a hybrid of SRAM and 3D-stacked STT-MRAM based AI accelerator was proposed for real-time learning where eMRAM acted as weight storage memory for infrequently accessed and updated layers, such as all convolutional layers and first few fully connected layers for a Transfer Learning followed by Reinforcement Learning algorithm. However, due to the use of typical slow and write-power-hungry STT-MRAM, this study could not completely exploit STT-MRAM to substitute SRAM and eventually used SRAM for storing weights of the last few fully connected layers which are accessed and updated frequently in transfer learning-based reinforcement learning setting.

In summary, prior notable research on STT-MRAM applications focused on implementing last-level cache memory, designing in-memory computing architectures, and high-capacity 3D-stacked memory as DRAM replacement for DNN hardware. Our work is the first to present a detailed analysis on the feasibility of using STT-MRAM as high bandwidth

on-chip buffer memory in DNN/AI accelerator hardware that can offer much larger capacity at lower energy and area costs compared to SRAM. Moreover, for complete DNN model storage in edge inference devices, a non-volatility relaxed STT-MRAM design is presented as an alternative to e-flash, which suffers from scaling limitations at advanced technology nodes.

2.7 Conclusions

In this chapter, we demonstrated the design of highly efficient AI/Deep Learning accelerators that utilize emerging STT-MRAMs. Based on detailed design space exploration, we designed the STT-MRAM-based global buffer to minimize DRAM access latency and energy, as well as reduce the area and power of the MRAM buffer. We presented an innovative runtime-reconfigurable core optimized for both dot products and matrix multiplication in convolution and fully-connected layers, respectively. A scratchpad-assisted STT-MRAM global buffer design has been demonstrated that reduces the frequency of energy-dominant write operations of the partial ofmaps during convolution. Using actual data occupancy times in memory for AI tasks, we guided the STT-MRAM thermal stability factor scaling. We showed that with *STT-AI* accelerator 75% area and 3% power savings are possible at iso-accuracy. Furthermore, with *STT-AI Ultra*, 75.4%, and 3.5% savings in area and power, respectively, over regular SRAM-based accelerators at minimal accuracy trade-off. While successful integration of this Δ-scaled STT-MRAM as GLB memory depends on advancements in the fabrication process of MRAM technology, the adoption of this technology will transform the AI accelerator industry by enabling sustainable, energy- and area-efficient accelerators for real-time applications.

Chapter 3

System and Design Technology Co-optimization of SOT-MRAM for High-Performance AI Accelerator Memory System

3.1 Introduction

The proliferation of artificial intelligence (AI) and deep learning (DL) has precipitated the computing hardware community to continually design innovative AI/DL accelerators with large data processing capabilities. However, with the ever-growing trend of model size, the bottleneck for state-of-the-art AI/DL accelerators is now memory rather than data and compute availability, and we expect this trend to worsen in the future [9][95][155]. The lack of efficient and high-performance data flow between the computing and memory element (i.e., the memory wall or memory bottleneck) masks the improvement coming from the efficient compute system [96]. One promising solution to the memory bottleneck of AI-specific workload is to increase the on-chip memory capacity[98]. For both training and inference, the on-chip memory capacity in the accelerator needs to be increased to ensure that the intermediate activations, as well as the weights of the current layer, can be loaded. Moreover, significantly more memory is required during training to store the gradients and optimizer states. Inadequate on-chip memory capacity causes frequent DRAM accesses which exacerbates energy costs, as well as stalls the compute cores of AI/DL accelerator until the data is fetched. Because of this large capacity demand, an SRAM-based on-chip memory system can be detrimental due to leakage energy and area inefficiency.

The promising features, such as high density, near-zero leakage power, immunity against radiation-induced soft errors, and CMOS compatibility of emerging Spin-based non-volatile (NVM) magnetic memory (i.e., MRAM) technologies, attracted researchers from academia and industry [74]. Spin Transfer Torque (STT) MRAM, has already shifted its gear from the R&D phase to commercialization as the NAND-based embedded flash replacement [70]

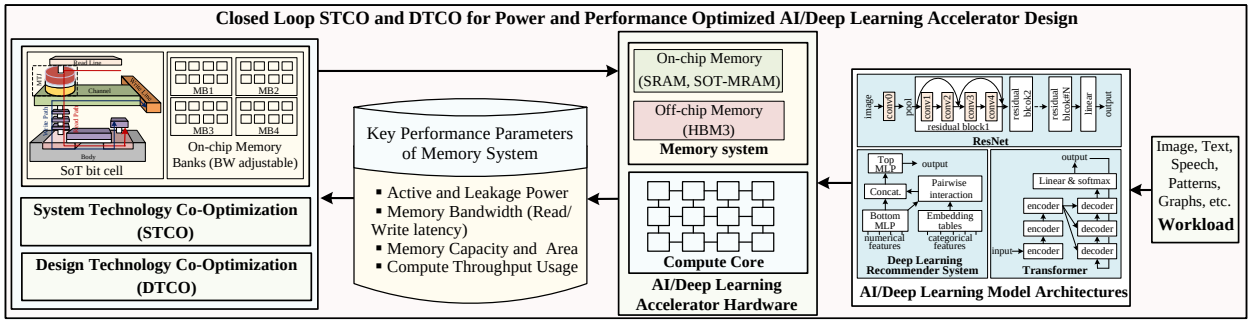


Figure 3.1: Workflow of closed-loop analysis for system and device level optimization for AI/Deep Learning Accelerator Design

[100]. However, STT-MRAM, the memory technology used in Chapter 2, faces several challenges - poor write performance, Read Disturbance (RD), retention failure [100][102]. These challenges stem from two main reasons. First, the high write current flowing through the MTJ accounts for almost 10× energy consumption as SRAM. Large write delay (> ns range) resulting from spin injection symmetry in switching the magnetic orientation of free layer belittles STT-MRAM's feasibility as an on-chip cache [103]. The stress on the dielectric oxide of the MTJ due to the large write current accelerates the time-dependent wear out of the cell [101]. Second, its shared read-write path makes it vulnerable to RD.

SOT MRAM, considered the next generation of STT-MRAM, offers high performance without compromising reliability issues such as RD. SOT-MRAM is a three-terminal memory cell that uses MTJ as the storing element [107]. By splitting the read-write path and using a different switching scheme, SOT-MRAM resolves all the challenges of STT-MRAM while retaining its every benefit [100] [101] [102] [103] [104]. The isolated read-and-write path allows the designer to optimize the read-and-write path independently, decreasing the write current and increasing the read-write operating margin, thus solving the RD-induced reliability issues. Though lacking mass-scale production from foundries due to early-stage manufacturing challenges, [100] [102] [103] [104] [105][106] have demonstrated the successful fabrication of SOT-MRAM with attractive specifications. Its attractive features, such

as high density, reliability and endurance, zero leakage, read-write latency comparable to SRAM, and research effort to enable mass production make it one of the best candidates for AI accelerator memory system where large on-chip memory is a must for training and inference.

The performance of an AI accelerator depends on both the compute and memory throughput of the device. While most accelerators have enough compute throughput, their performance is limited by memory throughput operating in the *memory bound* region. To address the *memory bound* problem of the AI hardware, in this chapter, we perform a closed-loop STCO on AI workloads and DTCO on SOT-MRAM to present a hybrid memory system. To our knowledge, this is the first work that analyzes and evaluates the performance of SOT-MRAM as the on-chip memory of AI accelerators targeting both inference and training. The STCO-DTCO methodology is shown in Fig. 3.1, and the key contributions of the chapter are highlighted as follows.

- We present a power and performance-optimized hybrid memory system for Deep Learning (DL) accelerators through a workload-aware STCO and DTCO. Comprised of off-chip HBM3 DRAM, on-chip SRAMs, and DTCO-enabled SOT-MRAM, the hybrid memory system can support the training and inference of DL workloads. We perform a closed-loop STCO and DTCO by taking into account the (i) System performance attributes (e.g., throughput and energy cost); (ii) Architectural and micro-architectural attributes (e.g., compute resources utilization, memory bandwidth) (iii) Workload attributes at both training and inference (e.g., runtime action counts, dataflow and data reuse) to reach the Pareto optimal solution.

- Using the Deep Learning models' execution profiles, DTCO enables device and circuit level customization of read/write bandwidth, retention time, and capacity of SOT-MRAM memory banks to meet the bandwidth and capacity demands of DL workloads.

Memory banks are individually optimized with various bandwidths and capacities to achieve dynamic runtime optimization of the power and performance of the accelerator hardware for diverse workloads.

- Finally, using various DNN benchmarks, we provide a comparative analysis of the existing SRAM-based memory system and the proposed DTCO-STCO optimized hybrid memory system for AI accelerators.

The chapter is organized as follows. Section 3.2 discusses the background. In Section 3.3, we present the analytical model for DNN workload profiling, followed by the DTCO of SOT-MRAM in Section 3.4. Sections 3.5 and 3.6 present the results & analysis, and related works, respectively, following the conclusion in Section 3.7.

3.2 Background

3.2.1 AI/DL Applications

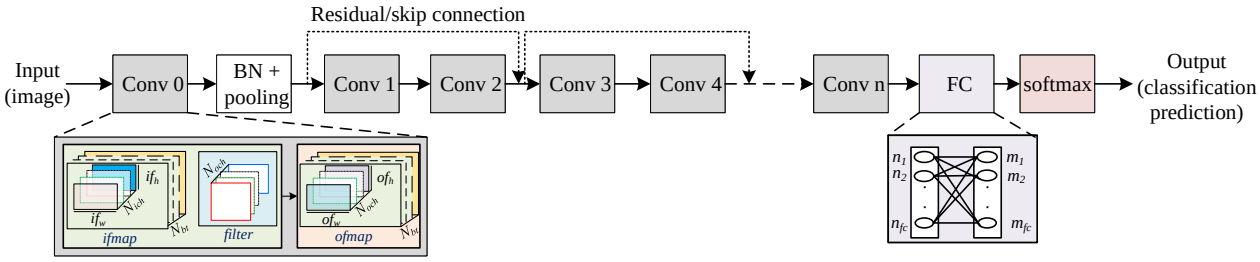


Figure 3.2: CV model (CNN/DNN) abstract architecture. Deep convolution (Conv) layers with residual/skip connection followed by fully connected (FC) layer/s. For symbol meaning please see Table 3.1.

Computer Vision (CV) and Pattern Recognition

CV models, also called Convolutional/Deep Neural Networks (CNN/DNN), are the stacks of convolution layers connected straight and/or through residual connection [55] to

extract the objects' features, and a few Fully Connected (FC) layers at the end to classify the objects. The input images are convolved with the filter weights to produce the output feature map (*OFMAP*). The *OFMAP* goes through the pooling and normalization layers to act as input (*IFMAP*) to the next layer. The linear and softmax layer at the end finally recognizes the image (Fig. 3.2). The size of each data entity (IFMAP, OFMAP, and Weights) depend on the model architecture.

Natural Language Processing (NLP)

Transformer-based [111] models dominate the NLP domain. In Transformer-based models [111], the input sequence propagates through the embedding layer and different sublayers of the encoder stacks to extract different linguistic features and inter-token dependency of the input sequence. The decoder stacks then generate the output sequence by taking the encoded input sequence from the encoder stack and the output sequence generated by itself in the previous timesteps (Fig 3.3). The input sequence multiplied by different layer weights takes different activation names and shapes throughout the model operation.

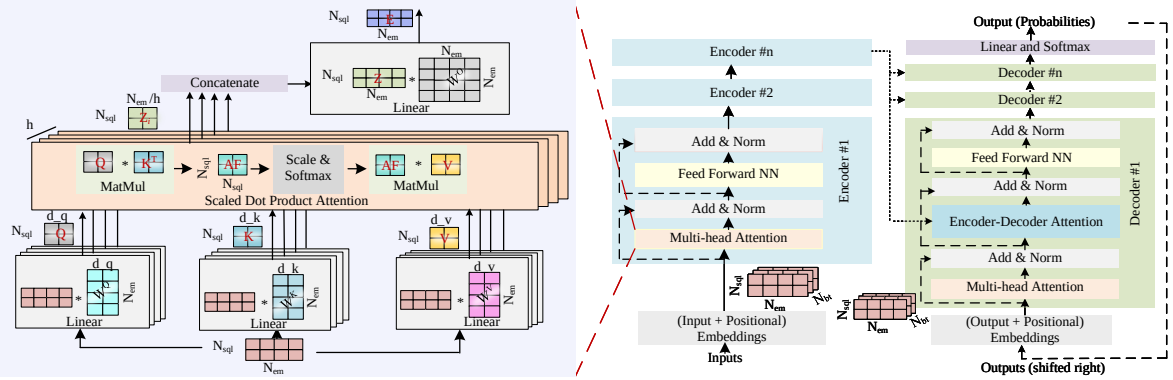


Figure 3.3: Transformer model workflow breakdown

3.2.2 AI/DL Accelerators

At the core of AI/DLs is the matrix-matrix/vector multiplication (GEMM) with massive parallelism. Exploiting this parallelism, Systolic Array (SA) based architecture [155] have been used to accelerate the computations. Different dataflows, such as row stationary, output stationary, weight stationary, have been evolved to maximize the reuse and reduce the data movement. Off-chip DRAM access being 100-200 times more energy and latency expensive than any ALU operation or on-chip access [7] plays a crucial role in determining the overall system performance. Another non-conventioanl type of architecture, In-Memory Computing (IMC) [109] has recently evolved to address the data communication cost for DNN accelerators. However, in this work, we focus on reducing the off-chip memory access for conventional DNN accelerator architectures [7][18][155] by increasing the on-chip Global Buffer (GLB) size with SOT-MRAM.

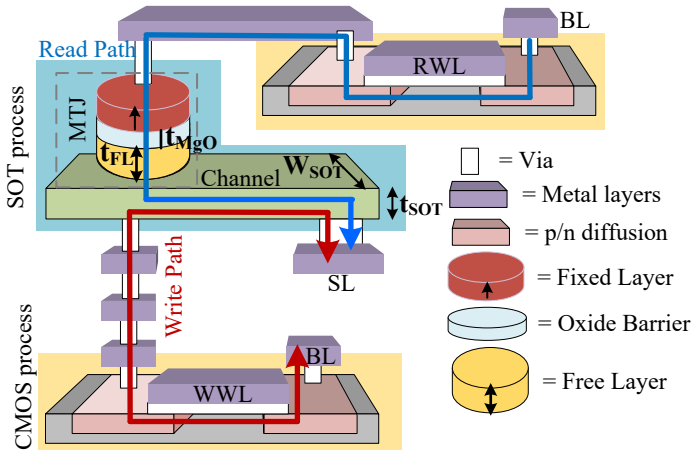


Figure 3.4: Physical structure of a SOT-MRAM bit cell highlighting separate read (along blue line) and write (along red line) path

3.2.3 SOT-MRAM

Physical Structure

With MTJ [64] as storing element, the SOT-MRAM is a three terminal device. Depending on the type of bit cell, there are three to four lines to control the read-write operation. In this work, we consider a two transistor one SOT (2T1SOT) bit cell architecture that requires two access transistors, (i) *Read Wordline (RWL)*, (ii) *Write Wordline (WWL)*, (iii) *Bit Line (BL)*, and (iv) *source Line (SL)* to accommodate separate read-write access path [107] [97] (Fig. 3.4). The MTJ stack, with its free layer at the interface, is placed on top of a SOT layer (i.e., channel) to ensure SOT-induced switching. The SOT layer is composed of heavy metals or topological insulators [110].

Read-Write Operation

Upon the activation of RWL, a small amount of current is passed through BL and grounded SL. The resistive state of the MTJ is captured by sensing the voltage across it and comparing the voltage with a reference value [102]. Low resistive state (R_P) and high resistive state (R_{AP}) represent bit 0 and 1 respectively. The write operation of MTJ-based MRAM involves switching the resistive status of MTJ. In SOT-MRAM, switching occurs due to Spin Orbit Torque (SOT) effect. Unlike STT-MRAM, a current is passed through the SOT layer to change the MTJ resistive state by switching the magnetic orientation of the free layer. A bidirectional write current flows through BL and SL during write operation. The potential of BL and SL changes depending on the bit value written in the cell. For example, to write '1', current flows from BL to SL and vice versa to write '0' [102] [101].

3.3 DNN WORKLOAD PROFILING

Profiling the target workload is a prerequisite for designing an accelerator for the target workload. Assuming that we have a powerful computing system to handle the exhaustive computations of the DL workload, we focus on providing efficient data movement between the compute and memory system to ensure 100% utilization of computing resources by introducing the workload-aware hybrid memory system. We propose the hybrid memory system by analyzing the Deep Learning model workloads from CV and NLP domain. We analytically model the on-chip bandwidth requirement and memory access patterns of different parts of the workload during inference and training, *Memory and Compute Model*, to develop the memory system for TPU-like [155] DNN accelerators.

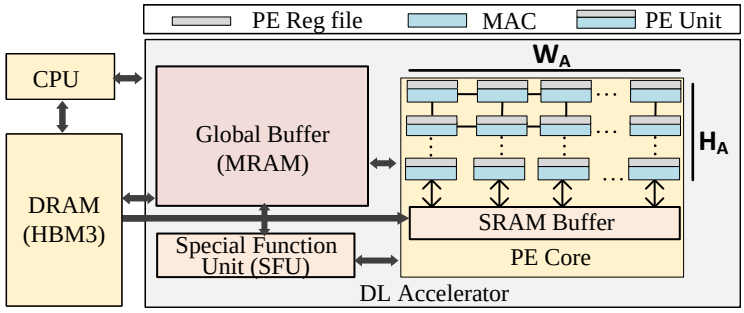


Figure 3.5: Block diagram of Accelerator architecture

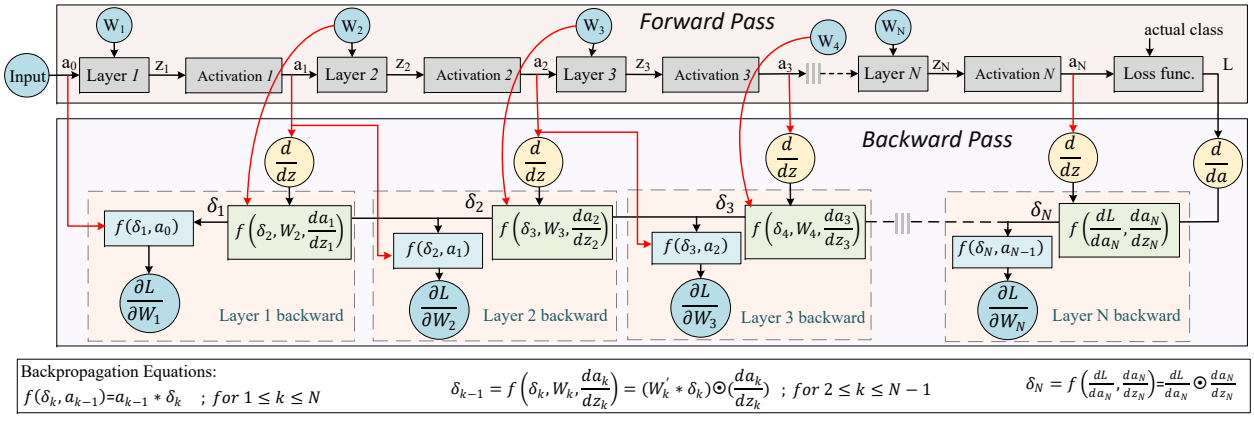


Figure 3.6: Computational graph of DNN training

3.3.1 Memory Bandwidth Expression

We express the required bandwidth (BW) as a function of compute resources and workload. BW (bytes/sec) is defined as the rate at which data needs to be transferred to/from memory by a processor to utilize the computation resources of the processor fully. Mathematically,

$$BW = \frac{F_p}{OI} \tag{3.1}$$

Where F_p = Theoretical peak performance (ops/sec) = number of operations the accelerator performs per sec. The F_p of a $H_A \times W_A$ Processing Element (PE) array (Fig. 3.5):

$$F_p = H_A * W_A * F_{acc} \tag{3.2}$$

F_{acc} = Operating frequency of the accelerator. OI = Operational Intensity of Workload (ops/byte) = number of operations performed per byte accessed. It is a measure of parallelism of the workload. In the subsequent subsections, we will formulate the OI of Conv. and FC layer to find their BW, respectively. Note that the read and write bandwidth will not be the same for these workloads.

Read Bandwidth (BW_{RD}) of Conv. layer

To formulate an expression for OI of convolution workload: First, we determine the total number of MAC operations, T_{MAC}, performed by a $H_A \times W_A$ PE array per clock cycle

$$T_{MAC} = H_A * W_A \tag{3.3}$$

Second, we figure out how many bytes should be read from memory to utilize all PEs of the accelerator in one clock cycle. In a row stationary dataflow [7], it takes $(k_h * k_w + of_h * of_w) * d_w$ bytes of data (d_w = data type in bytes, i.e., FP32, BF16 etc.) and $\#(of_h * of_w * k_h * k_w)$ PEs to generate the partial ofmaps corresponding to one input channel. Depending on the size of the PE array, in each iteration (one complete use of accelerator), multiple input channels can be fit. The input channels (i.e., no. of partial ofmaps) computed by the PE array in each iteration:

$$N_{ich\_per\_stp} = \frac{H_A * W_A}{of_h * of_w * k_h * k_w} \tag{3.4}$$

Total bytes read from memory to utilize all PEs:

$$T_{byte} = \frac{H_A * W_A}{k_h * k_w * of_h * of_w} * (k_h * k_w + if_h * if_w) * d_w \tag{3.5}$$

We divide the total number of MAC operations, T_{MAC}, by the total bytes accessed, T_{byte}, to find OI:

$$OI = \frac{k_h * k_w * of_h * of_w}{d_w * (k_h * k_w + if_h * if_w)} \tag{3.6}$$

Substituting the expression of OI in equation (3.1) gives BW_{RD} as a function of array size and workload:

$$BW_{RD} = \frac{(k_h * k_w + if_h * if_w) * d_w}{k_w * k_h * of_h * of_w} * H_A * W_A * F_{acc} \tag{3.7}$$

For the symbol meanings, please see Fig. 3.2 and Table 3.1.

Table 3.1: CNN and systolic array parameters nomenclature

W_A, H_A	Accelerator array width & height (PEs)
k_w, k_h	Kernel width & height
of_w, of_h	Output feature map width & height
if_w, if_h	Input feature map width & height
N_{ich}, N_{och}	No. of input & output channel
N_{bt}	Batch size
n_{fc}, m_{fc}	No. of neurons in input & output FC layer

Write Bandwidth (BW_{WR}) of Conv. Layer

Partial ofmap of a single input channel requires $\#(of_h * of_w * k_h * k_w)$ PEs. Therefore, $H_A \times W_A$ PEs generate $(H_A * W_A)/(of_h * of_w * k_h * k_w)$ ofmaps in each iteration. Each partial ofmap contains $of_h * of_w$ elements. The total output bytes generated by the PE array in one iteration is, equivalently, the write bandwidth:

$$BW_{WR} = \frac{H_A * W_A * F_{acc} * d_w}{k_h * k_w} \tag{3.8}$$

BW_{RD} & BW_{WR} of FC layer

The systolic array is a widely used architecture to perform GEMM operation [155]. Depending on the array dimension ($H_A \times W_A$) and operand matrix dimension (input matrix: $K \times M$, weight matrix: $M \times N$, and output matrix: $K \times N$), we formulate required Read and Write GLB bandwidth for four different cases: (i) Weight matrix dimensions (both) are less than the systolic array dimensions ($M < H_A, N < W_A$), (ii) Height of weight matrix is less than the height of systolic array, but the width of weight matrix is larger than or equal to the width of the systolic array ($M < H_A, N \geq W_A$), (iii) Height of weight matrix is larger than or equal to the height of systolic array, but width of the weight matrix is less than the width of the systolic array ($M \geq H_A, N < W_A$), and (iv) Both height and width of

Table 3.2: RD/WR bandwidth expression of FC layer for different cases

Cases		$\mathbf{BW_{RD}}$	$\mathbf{BW_{WR}}$
$M < H_A;\ N < W_A$	$K < W_A$	$\frac{M*N+K*M}{N+K}$	$\frac{K*N}{2*N+K-1}$
	$K \geq W_A$	$\frac{M*N+W_A*M}{N+W_A}$	$\frac{W_A*N}{2*N+K-1}$
$M < H_A; N \geq W_A$	$K < W_A$	$\frac{M*W_A+K*M}{N+K}$	$\frac{K*W_A}{2*W_A+K-1}$
	$K \geq W_A$	$\frac{M*W_A+W_A*M}{2*W_A}$	$\frac{{W_A}^2}{2*W_A+K-1}$
$M \geq H_A; N < W_A$	$K < W_A$	$\frac{H_A*N+K*H_A}{N+K}$	$\frac{K*N}{2*N+K-1}$
	$K \geq W_A$	$\frac{H_A*N+W_A*H_A}{W_A+N}$	$\frac{W_A*N}{2*N+K-1}$
$M \geq H_A; N \geq W_A$	$K < W_A$	$\frac{H_A*W_A+W_A*H_A}{W_A+K}$	$\frac{W_A*N}{2*N+K-1}$
	$K \geq W_A$	$\frac{H_A*W_A+W_A*H_A}{2*W_A}$	$\frac{W_A^2}{2*W_A+K-1}$

weight matrix are larger than or equal to the height and width of systolic array respectively ($M \geq H_A, N \geq W_A$).

In a weight stationary dataflow, it takes N clock cycles to load the weight matrix into the systolic array. Once the weights are loaded, the input matrix is streamed from left to right and the outputs are collected downward. The input matrix's first column reaches the weight matrix's last column at $2N$ clock cycles. The last (or K^{th}) column of the input matrix reaches the last column of weight matrix after $2N + K - 1$ clock cycles and generates the output matrix, $K \times N$. Based on the above dataflow and mapping, the peak read-write bandwidth per clock cycle for different cases is summarized in Table 3.2. The expressions are shown for weight stationary dataflow.

From the transformer-based NLP model architecture (Fig. 3.3), we observe that the dominant operations are the GEMM operations. As a result, we model the read-write bandwidth requirement for different layers of the transformer-based model same as the read-write

Table 3.3: Parameter nomenclature for Algorithm 1 and 2

I, O, W	$ifmap,\ ofmap,\ weight$ size in MB
RD_{DRAM}	DRAM Read access counts
WR_{DRAM}	DRAM Write access counts
RD_{GLB}	GLB Read access counts
WR_{GLB}	GLB Write access counts
GI, GO, GW	$ifmap, ofmap, weight$ Gradient size in MB
$mbpa$	MB of data fetched per memory access
$layer\_f$	Layer size (MB) combining $ifmap$, $ofmap$ & $weights$
$layer\_b$	Layer size (MB) in backprop combining upstream, $ofmap$ & $weight$ gradient
$cum\ layer$	Cummulative size of layer
rd_f, rd_b	DRAM read access during forward & backward pass
wr_f, wr_b	DRAM write access during forward & backward pass

bandwidth of FC layer. Another dominant operation after GEMM is softmax operation. Softmax operation is performed mostly on the scaled attention filter matrix, AF of size $N_{sql} \times N_{sql}$; $\sigma(AF)ij = \frac{e^{AF_{ij}}}{\sum_{i=1}^{N_{sql}} e^{AF_{ij}}}$. The softmax operation is generally performed in the Special Function Unit (SFU)[95] (Fig. 3.5). The bandwidth requirement of the softmax operation depends on the hardware architecture, mapping, and AF matrix dimension. Assuming that the SFU contains $1 \times H_A$ units, each capable of performing one exponential operation, followed by an accumulator for accumulating the exponentials, and a regular ALU for performing the division the bandwidth of softmax operation on SFU is estimated as $BW_{softmax} = d_w * H_A$.

3.3.2 Memory Access Patterns

Our proposed memory system consists of HMB3 (off-chip DRAM memory), a large GLB with multiple SOT-MRAM banks, a smaller double-buffered SRAM, and PE reg file specific to each PE unit (Fig. 3.5). The banks inside SOT-MRAM are optimized through a DTCO between the SOT-MRAM parameters and the workload requirements. The double-buffered

Algorithm 1: DRAM & GLB access count at Inference

for $i = 1$ **to** *no. of layers* **do**
  $RD_{GLB} \leftarrow \frac{I_i}{mbpa_{GLB}}$
  if $i = 1$ **then**
    $WR_{GLB} \leftarrow \frac{I_i + O_i}{mbpa_{GLB}}$
    if $(I_i + W_i) \leq GLB$ **then**
      $RD_{DRAM} \leftarrow \frac{I_i + W_i}{mbpa_{DRAM}}$
    else
      $RD_{DRAM} \leftarrow \frac{I_i + W_i}{mbpa_{DRAM}} + \frac{I_i + W_i - GLB}{mbpa_{DRAM}}$
    end
  else
    $WR_{GLB} \leftarrow \frac{O_i}{mbpa_{GLB}}$
    if $O_{i-1} \leq UB$ **then**
      if $W_i \leq GLB$ **then**
        $RD_{DRAM} \leftarrow \frac{W_i}{mbpa_{DRAM}}$
      else
        $RD_{DRAM} \leftarrow \frac{W_i}{mbpa_{DRAM}} + \frac{W_i - GLB}{mbpa_{DRAM}}$
      end
    else
      $RD_{DRAM} \leftarrow \frac{I_i + W_i}{mbpa_{DRAM}} + \frac{(I_i + W_i) - GLB}{mbpa_{DRAM}}$
    end
  end
  if $i =$ *no. of layers* **then**
    $WR_{DRAM} \leftarrow \frac{O_i}{mbpa_{DRAM}}$
  else
    if $O_i > GLB$ **then**
      $WR_{DRAM} \leftarrow \frac{O_i - UB}{mbpa_{DRAM}}$
    else
      $WR_{DRAM} \leftarrow 0$
    end
  end
end

SRAM holds the weights and partial outputs. In this subsection, we analyze the memory access patterns of CV and NLP models for the proposed memory system.

The required number of main memory accesses depends on the GLB size, weight, activation size, and dataflow. Assuming a fixed dataflow, weight stationary in this case, we model the memory access counts during inference and training as a function of the model's workloads and the GLB size in Algorithm 1, and 2 for inference and training, respectively. During inference, inputs (e.g., images, tokens) are read from HBM3, written to GLB, and

Algorithm 2: DRAM & GLB access count at Training

$cum\ layer \leftarrow 0$;
$tmp \leftarrow 0$;
for $i = 1$ **to** $no.\ of\ layers$ **do**
  $layer\_f_i \leftarrow I_i + O_i + W_i$;
  $layer\_b_i \leftarrow GI_i + GO_i + GW_i$;
  $layer\ (i) \leftarrow layer\_f_i + layer\_b_i$;
  $cum\ layer(i) \leftarrow tmp + layer\ (i)$;
  $tmp \leftarrow cum\ layer(i)$;
  $RD_{GLB} \leftarrow \frac{3*I_i+O_i+5*W_i}{mbpa_{GLB}}$
  $WR_{GLB} \leftarrow \frac{2*I_i+2*O_i+3*W_i}{mbpa_{GLB}}$
  if $cum\ layer(i) \leq GLB$ **then**
    if $i = 1$ **then**
      $rd\_f(i) \leftarrow \frac{I_i+W_i}{mbpa_{DRAM}}$
    end
    if $i = no.\ of\ layers$ **then**
      $wr\_f(i) \leftarrow \frac{O_i}{mbpa_{DRAM}}$
    end
    $rd\_f(i) \leftarrow \frac{W_i}{mbpa_{DRAM}}$;
    $rd\_b(i) \leftarrow 0$;
    $wr\_f(i) \leftarrow 0$;
  else
    if $(i \neq 1)\ AND\ (O_{i-1} \leq GLB)$ **then**
      $rd\_f(i) \leftarrow \frac{W_i}{mbpa_{DRAM}}$
    else
      if $I_i + W_i \leq GLB$ **then**
        $rd\_f(i) \leftarrow \frac{I_i+W_i}{mbpa_{DRAM}}$
      else
        $rd\_f(i) \leftarrow \frac{I_i+W_i}{mbpa_{DRAM}} + \frac{I_i+W_i-GLB}{mbpa_{DRAM}}$
      end
    end
    if $(GI_i + GO_i + GW_i \leq GLB)$ **then**
      $wr\_f(i) \leftarrow 0$
      $rd\_b(i) \leftarrow 0$
    else
      $wr\_f(i) \leftarrow \frac{GI_i+GO_i+GW_i}{mbpa_{DRAM}}$
      $rd\_b(i) \leftarrow \frac{GI_i+GO_i+GW_i}{mbpa_{DRAM}}$
    end
  end
  $wr\_b(i) \leftarrow \frac{W_i}{mb\_per\_acs}$
end

read from GLB to be operated inside PEs core. The read-only weights are directly loaded from HBM3 to the register file of each PE unit, bypassing the GLB. Using double-buffered

SRAM, while the array is computing with loaded weights, the next set of weights is temporarily written to the SRAM buffer to hide the off-chip access latency behind the PE array computation latency. Suppose the GLB size is large enough to hold all samples in the mini-batch. In that case, the data entity can be read all at once, resulting in the memory accesses equal to the algorithmic minimum memory accesses. Algorithmic minimum memory access represents the number of elements in the data entity [53]. For weight gradient calculation, during backpropagation of the training, the inputs are read from GLB to PE core, assuming that the GLB is large enough to hold the input images along with the generated ofmap of the current layer, thus avoiding the DRAM accesses during backward pass. In convolution, the inputs can be reused multiple times for convolutional and filter reuse. It can also be reused multiple times during backpropagation to calculate the gradients of different filters. In Transformer-based NLP models, the embedded input can be reused thrice as input to Key, Query, and Value linear layer. It can also be reused thrice during backpropagation to calculate the weight gradient of the Key, Query, and Value linear layer. The training workflow is complicated and requires many more memory accesses (both off-chip and on-chip) compared to inference. For example, to calculate the weight gradients of Layer 1, it requires the current layer's activation gradient $\frac{da_1}{dz_1}$, input (a_0), next layer's weight (W_2) and the upstream gradient from Layer 2 (δ_1) (Fig. 3.6).

The pseudo code of Alg. 1 models the inference memory access patterns. The inputs and weights must be loaded from DRAM for the first layer. Depending on the combined size of input & weight matrix size, and GLB size, it requires either algorithmic minimum read accesses or more than that (lines 3-9 of Alg. 1). For the rest of the layers, if the $ofmap$ of the previous layer can fit in GLB, then no read accesses are required for input activation, as the $ofmap$ of the previous layer will act as the $ifmap$ to the next layer. Only the weights are read from DRAM for such layers (lines 12-20 of Alg. 1). The opposite case applies to write accesses: the $ofmap$ of the last layer must be written to the DRAM. For other layers,

it needs to be written to DRAM depending on its size and GLB size (lines 22-30 of Alg. 1). No write accesses are required for weight matrices during inference. As the weights bypass the GLB during inference, the GLB read accesses are calculated from the $ifmap$ size for each layer (line 2). The write accesses are calculated from the $ofmap$ except for the 1st layer (line 11, 4). See Table 3.3 for symbol meanings.

The pseudo code of Algorithm 2 models the training behavior. We initialize several temporary variables: $layer\_f_i$ (comprising of $ifmap$, $ofmap$, and $weight$ matrices of i^{th} layer), $layer\_b_i$ (comprising of upstream gradient, $ofmap$, and $weight$ matrix gradients of i^{th} layer), and $layer_i$ (combining $layer\_f_i$ and $layer\_b_i$) as shown in lines 4-6 in Alg. 2. $cum\ layer(i)$ contains all layers' all entities up to i^{th} layer (line 7-8, Alg. 2). If the GLB is large enough to hold $cum\ layer(i)$, we just need to read the $ifmap$ of the first layer & $weight$ of all layers from DRAM during the forward pass and write all layers' updated $weight$ during the backward pass and last layer's $ofmap$ to DRAM during the forward pass (lines 11-20 & 39, Alg. 2). Otherwise, the forward pass is the same as the inference (lines 22-30, Alg. 2). During the backward pass, depending on the size of upstream gradients, $ofmap$, and $weight$ gradients, it accesses the gradients from DRAM (lines 31-37, Alg. 2). The GLB read-write accesses are shown in lines 9-10 in Alg. 2. The $ifmap$ of each layer needs to be read twice, once during the forward pass and once during the backward pass. The upstream gradient, equal in size as $ifmap$, must be read once during the backward pass. The $ofmap$ is read once during the backward pass to calculate the upstream gradient. The weight is read 5 times (once during the forward pass, 4 times during the backward pass). The $ifmap$ and $ofmap$ are written twice, once during the forward pass and once during the backward pass. The $weight$ is written thrice, twice during forward pass and once during backward pass.

3.4 DTCO of SOT-MRAM

To ensure overall system performance for AI workloads, the memory system should have large on-chip memory to avoid frequent DRAM accesses, and the on-chip memory should have high bandwidth to prevent the system from being memory-bound while being energy efficient. In this section, we perform a DTCO of SOT-MRAM in bit-cell level based on the workload profiling done in section 3.3.

3.4.1 Optimizing critical switching current I_c

In SOT-MRAM, the magnetic orientation of the free layer is switched by Spin-Orbit Torque induced by spin Hall and interfacial effects between the channel (i.e., SOT layer) and free layer (FL) of MTJ. An in-plane charge current is flown through the channel to generate a spin current that exerts a spin torque on the free layer, which rotates the free layer's magnetic orientation. The critical current density required to switch the magnetic orientation of FL is expressed as [116]

$$j_c = \frac{2e\mu_0 M_{s,FL} t_{FL}}{\hbar \theta_{SH}} \left(\frac{H_{k,eff}}{2} - \frac{H_x}{\sqrt{2}}\right) \tag{3.9}$$

Where $H_{k,eff}$ is the effective anisotropy field, H_x is the applied field, $M_{s,FL}$ is the saturation magnetization of free layer, and t_{FL} is its thickness. Our interest is in lowering the switching current to achieve low write energy. Here, the free layer thickness t_{FL} and spin Hall efficiency θ_{SH} act as a control knob for critical switching current. θ_{SH} is a material-specific parameter and its higher value is expected to reduce the switching current. The typical value of θ_{SH} in heavy metal alloys ranges between 0.1 to 0.5 [110]. However, recent topological insulators as SOT layer can have a very large θ_{SH}. [117] demonstrated $\theta_{SH} = 152$ with $BiSb$ thin films.

3.4.2 Optimizing read-write pulse width

Read pulse width

The reading of SOT-MRAM involves sensing the resistance of the MTJ. A small amount of current is passed through the MTJ stack and the voltage across the stack V_{data+} or V_{data-} is compared against a reference voltage $V_{ref} = \frac{1}{2}(V_{data+} + V_{data-})$ to read out the stored bit. The read Sensing Margin $SM = |V_{ref} - V_{data}|$ is typically very small. Sensing and amplifying this small difference requires a strong and complex Sense Amplifier that contributes to most of the read latency and energy. The SM is determined by the Tunnel Magneto Resistance ratio ($TMR\ ratio = \frac{R_{AP}-R_P}{R_P}$) of MTJ. A higher TMR ratio produces a larger SM by making V_{data+} higher and V_{data-} lower. Thus the TMR ratio is inversely proportional to the read latency[118]. The higher the TMR window, the higher the read speed and the less effort required on the periphery. The typical range of the TMR ratio is between 100 to 300%. The TMR is tunable by oxide thickness [119] as shown in Fig. 3.15 (a). In SOT-MRAM, we can increase the oxide thickness, thanks to the decoupled read-write path of SOT-MRAM, to achieve a high TMR and increase the read speed without worrying about the large incubation time [120].

Table 3.4: DTCO control parameters & their impact on Power, Performance and Area (PPA)

DTCO Parameters	**Impact on PPA**
Spin Hall angle θ_{SH}	$\theta_{SH} \uparrow$, $j_c \downarrow$, Switching energy $\downarrow$
Free layer thickness t_{FL}	$t_{FL} \downarrow$, $j_c \downarrow$, Switching energy $\downarrow$, Area $\downarrow$
SOT layer dimension A_{SOT}	$A_{SOT} \downarrow$, $\tau_p \downarrow$, Area $\downarrow$, Write Bandwidth $\uparrow$
Oxide thickness t_{MgO}	$t_{MgO} \uparrow$, TMR $\uparrow$, Read Bandwidth $\uparrow$

Write pulse width τ_p

The width of the write current pulse for switching is inversely proportional to the magnitude of the applied current density in the SOT layer j_{sw} [110]

$$\tau_p \propto \frac{1}{j_{sw}} \tag{3.10}$$

As the area of the SOT layer (A_{SOT}) is scaled down, the effective current density increases, $j_{sw} \propto 1/(A_{SOT})$. Successful switching should take place when $j_{sw} > j_c$. We can increase j_{sw} by reducing the SOT layer dimension and decrease j_c by increasing θ_{SH} or by decreasing t_{FL}. Thus we can achieve successful switching in much shorter pulse width (equation 3.10). [121] demonstrated the switching at 180ps, [122] at 400ps, and [123] at 210ps. Switching in shorter pulse width ensures larger write bandwidth which is essential for memory systems used in AI/Deep Learning hardware. The key DTCO parameters of SOT-MRAM and their impact on Power, Performance and Area (PPA) are listed in Table 3.4.

3.5 Results and Analysis

In this section, we provide the results and analysis of the STCO on the CV and NLP workloads during inference and training and present the optimum Power, Performance, and Area results by performing the DTCO of SOT-MRAM. We developed a MATLAB-based framework to implement our analytical *Memory and Compute Model* to capture the relationship between the memory access counts and the memory hierarchy sizes in typical systolic array based AI accelerators. Unlike ScaleSim [27] and Timelooop [53] simulator, which only

Table 3.5: Parameters of NLP models

Model	**Enc. layer**	**Dec. layer**	**Attention head**	**Word Embedding** (N_{em})	**Intermediate dimension** (d_{ff})	**Seq. length** (N_{sql})	**Vocab. size** (N_{vocab})
Transformer	12	6	8	512	2048	1024	37000
BERT	12	-	12	768	3072	512	30522
Distil BERT	6	-	12	768	3072	512	30522
Mobile BERT	24	-	4	128	512	512	30522
Squeeze BERT	12	-	12	768	3072	512	30522
Visual BERT	12	-	12	512	3072	512	30522
GPT	-	12	12	768	2048	512	40478
GPT-2	-	12	12	768	2048	1024	50257
GPT-3	-	96	96	12288	49152	2048	50257
GPT-Neo	-	24	16	2048	8192	2048	50257
GPT-J	-	28	16	4096	16384	2048	50400

support profiling DNN workloads in inference mode to date, our model captures both training and inference behavior of CV and NLP models. We also verified our model's results with Timeloop in inference mode.

3.5.1 Bandwidth Demand

In Fig. 3.7 (a), (b), we plot the read-write on-chip bandwidth demand in *bytes/cycle* of 18 widely used CV models. Resenet101 and Resnet50 running on a 256×256 PE array will demand the highest read bandwidth, 4017 bytes/cycle, from GLB, whereas Squeezenet will demand the lowest bandwidth, 1028 bytes/cycle. Naturally, as the PE array size increases, the computation capacity per cycle T_{MAC} increases which demands more data from memory to keep all PEs active. From the workload perspective, we observe that the most contributing factor to the read bandwidth demand is its inverse relationship with the filter and ofmap size. We explain the inverse relationship of filter and ofmap size with the read bandwidth

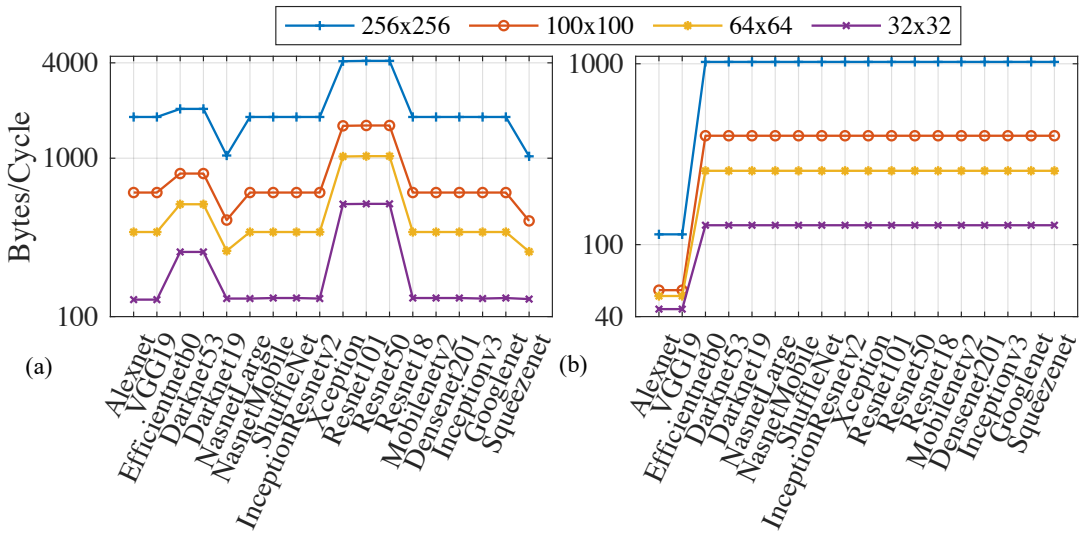


Figure 3.7: Bandwidth requirement of CV models for different PE array sizes. (a) Read Bandwidth, (b) Write Bandwidth. Bandwidth varies from model to model because of their variation in layer size and type.

using the convolutional reuse concept. As the filter size decreases, the scope of convolutional reuse decreases. The ofmap again depends on the filter and ifmap size. With the decrease of filter size and ofmap size, the convolutional reuse decreases, giving rise to more bandwidth demand. The layer of Resnet101 that requires the most bandwidth (4017 bytes/cycle) has the ofmap dimension (7×7) and filter dimension (1×1). On the other hand, the most demanding (1028 bytes/cycle) layer of Squeezenet has the ofmap dimension (18×18) and filter dimension (1×1). Another observation is that though 1×1 convolution reduces the computation complexity, it requires more bandwidth from memory, i.e., becomes memory intensive. The write bandwidth is also inversely proportional to the filter size. However, in 1×1 convolutions, it depends on the number of outputs generated by the PE array. The write bandwidth is always smaller than the read bandwidth (Fig. 3.7 (b)) as it takes more than one operands to generate one output. For example, in a 3×3 convolution, it takes 18 operands to generate a single output; in a 1×1 convolution, it takes two operands.

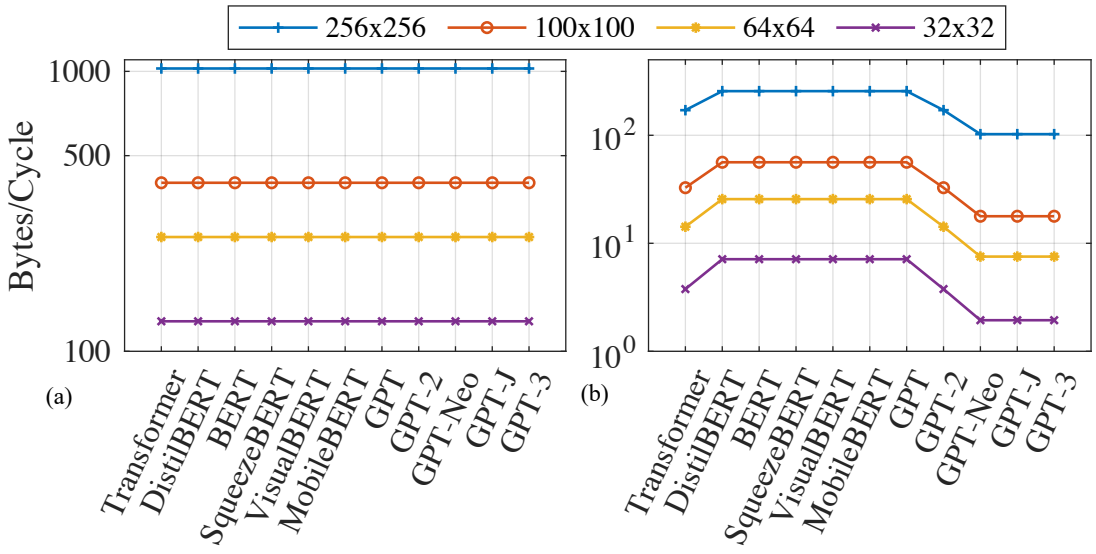


Figure 3.8: Bandwidth requirement of NLP models for different PE array size. (a) Read Bandwidth (for GEMM and softmax operation), (b) Write Bandwidth. Read bandwidth is the same across all the models because it is limited by the PE array dimension, whereas the write bandwidth varies across models because of their different sequence lengths

.

As mentioned in section 3.3.1, the bandwidth requirement for transformer-basded model are calculated using the expressions of Table 3.2. The dimension of the operand matrices is larger than the PE array dimension, hence following Case IV (Table 3.2, Section 3.3.1), the read bandwidth of all models depends on the PE array size (Fig. 3.8 (a)). The write bandwidth depends on the PE array dimension and the input sequence length. The softmax read bandwidth depends on the SFU width, and matches with the GEMM read bandwidth. As different models are trained with different input sequence lengths[108], their write bandwidth demand is not the same across all models. The parameter sizes and settings of the models used in this work are shown in Table 3.5. The models having the highest sequence length (2048) have the lower write bandwidth demand 102 bytes/cycle running on a 256×256 PE array (Fig. 3.8 (b)).

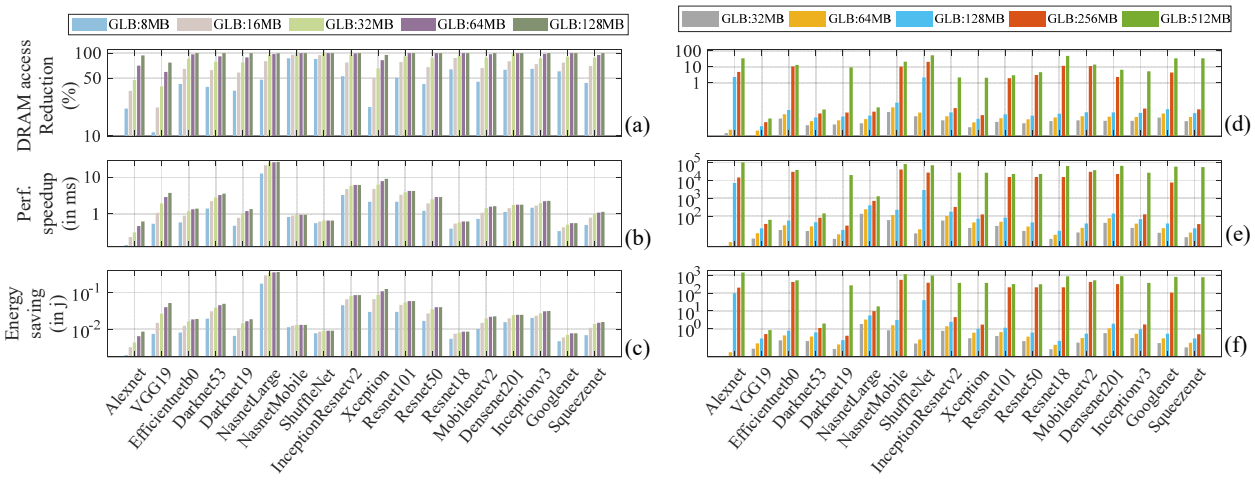


Figure 3.9: Impact of larger GLB memories on performance and energy efficiency for CV models at inference and training. Percentage reduction in DRAM accesses at inference (a) and training (d). Performance Speedup from DRAM access reductions at inference (b) and training (e). Energy savings from reduced DRAM accesses at inference (c) and training (f). Both cases compare results to a baseline of 2MB GLB running 16 samples.

3.5.2 Impact of on-chip memory

Compared to a GLB size of 2MB, the DRAM access counts for all CV models decrease significantly if we increase the GLB size. In inference, reaching the 100% reduction in access means it only needs to read the initial inputs, weights for each layer, and write the final layer output, no DRAM access is needed for the intra and inter-layer operations. Further increase in GLB size will not improve the performance in these cases. Considering 16 samples, DRAM access is reduced by 100% for 14 models at 128MB, and most models experience a reduction of $> 80\%$ at 64MB (Fig. 3.9 (a)). Fig. 3.9 (b), (c) show the performance speed up and energy saving coming from these DRAM access reductions.

We observe a slower improvement in the DRAM access reduction during training unless the GLB size is large enough, at least 256MB for most models (Fig. 3.9 (d)). However, even the smaller percent reduction in DRAM access results in significant performance and energy improvement (Fig. 3.9 (e), (f)). This is because training requires at least $2\times$ DRAM accesses as inference. The smaller percent reduction of a large number of DRAM accesses translates

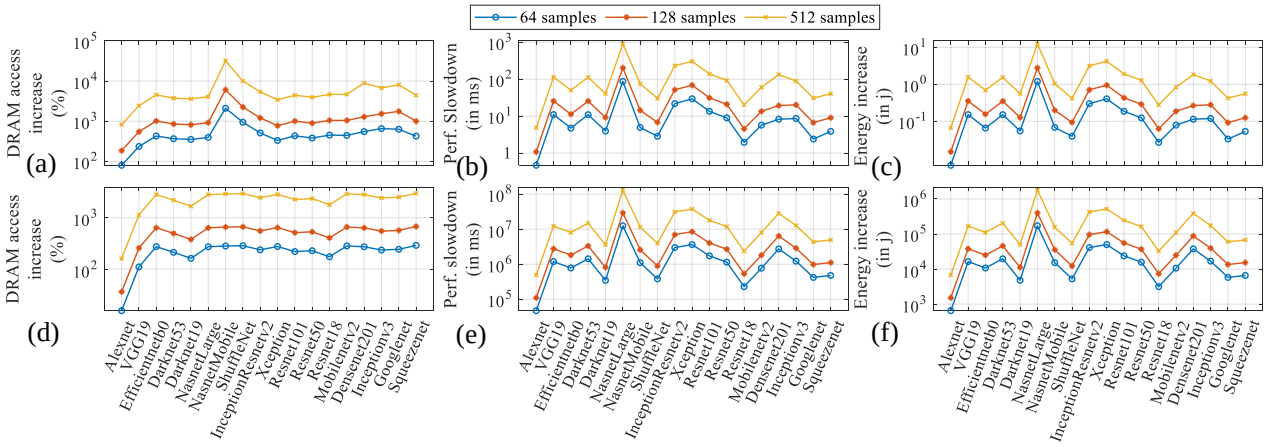


Figure 3.10: Impact of batch size on performance and energy efficiency for CV models at inference and training. Percentage increase in DRAM accesses at inference (a), at training (d). Performance slowdown (latency increase) from extra DRAM accesses at inference (b), at training (e). Energy increase from extra DRAM accesses at inference (c), at training (f). In both cases, results are compared to a baseline of 16 samples running with 4MB GLB.

to a significant energy and latency improvement. A similar trend is observed for NLP models. Transformer-based NLP models are usually larger than the CV models. This is the reason we achieve more performance speedup and energy reduction even at smaller DRAM access reduction rate (Fig. 3.11).We also observe that DNN models learn faster if we increase the batch size. However, for a fixed GLB size, the DRAM access count increases significantly at larger batch size, causing performance slowdown and more energy consumption. Fig. 3.10 and Fig. 3.12 (a, b, c for inference and d, e, f for training) show the increase in DRAM access count and its associated impact on performance and energy for CV and NLP models respectively at different batch sizes.

The key takeaway from this analysis is that we can reduce the energy and latency associated with DRAM accesses if we increase the GLB size. For larger batch sizes, the energy and latency improvement is even more. Because at large batch sizes, throughput increases at the cost of DRAM accesses. As we increase the GLB size, DRAM accesses reduce, and we achieve latency and energy reduction.

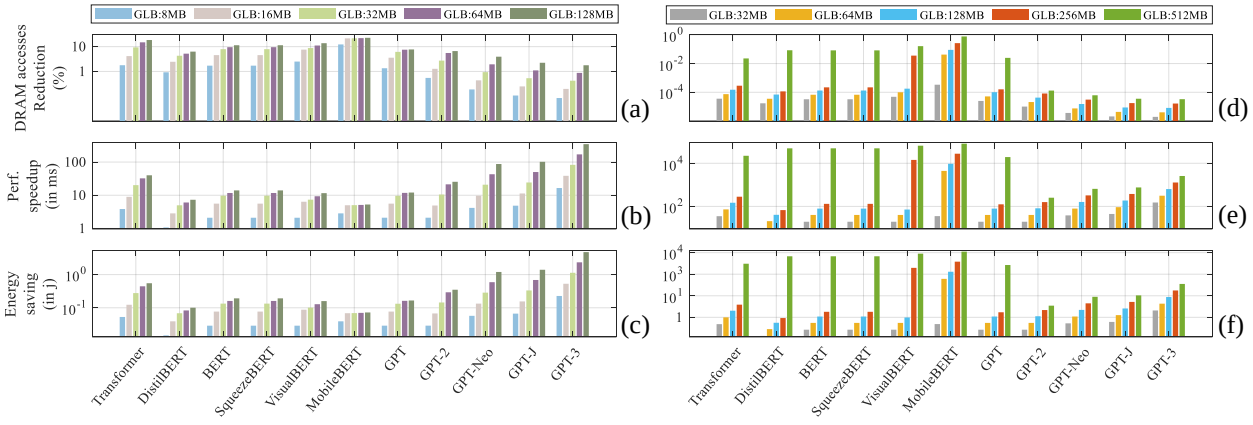


Figure 3.11: Impact of larger GLB memories on performance and energy efficiency for NLP models at inference and training. Percentage reduction in DRAM accesses at inference(a), at training (d). Performance Speedup from DRAM access reductions at inference (b), at training (e). Energy savings from reduced DRAM accesses at inference (c), at training (e). In both cases, results are compared to a baseline of 2MB GLB running 16 samples

3.5.3 DTCO of SOT for PPA Optimization

From section 3.5.2 we see that the GLB size of 64MB (for inference) and 256MB (for training) offer significant energy and performance improvement. However, it is not feasible and efficient to use such large SRAMs because of its area and leakage power, even if the low-power techniques are employed. Section 3.5.1 implies that we need approximately 4000bytes/cycle bandwidth between GLB and PE array for larger array size (256×256). In this subsection, we provide the SOT-MRAM DTCO results and observation meeting the requirements stated in the above two subsections. We perform the DTCO in *Cadence Virtuoso* tool using the compact SOT-MRAM model from [107], and use *Synopsys* 14nm library [85] for the CMOS transistors and peripheral circuits.

I_C optimization

To realize the impact of SOT efficiency θ_{SH} on I_c, we sweep θ_{SH} from 0.1 to 100 (Fig. 3.13 (a)). With $\theta_{SH} \geq 100$, I_c goes as low as 0.5uA. Even though the widely used SOT layers

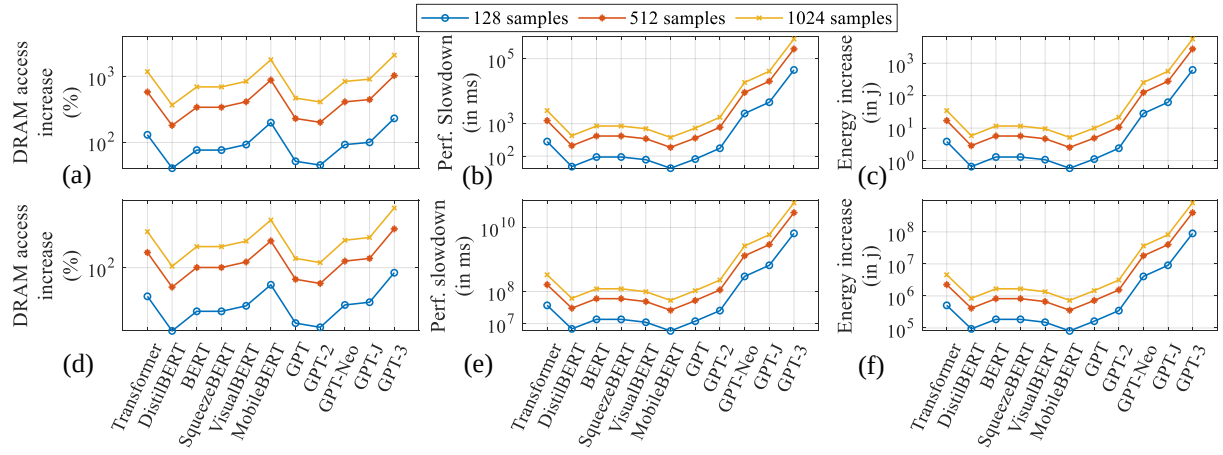


Figure 3.12: Impact of batch size on performance and energy efficiency for NLP models at inference and training. Percentage increase in DRAM accesses, inference (a), and training (d). Performance slowdown (latency increase) from extra DRAM accesses at inference (b), at training (e). Energy increase from extra DRAM accesses at inference (c), at training (f). Results are compared to a baseline of 16 samples running with 4MB GLB.

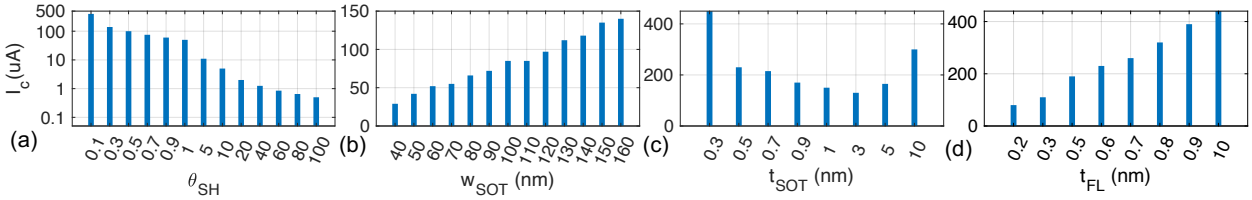


Figure 3.13: Critical current vs θ_{SH}(a), w_{SOT}(b), t_{SOT}(c), and t_{FL}(d).

are made of heavy metal alloys having smaller θ_{SH} (e.g., 0.1 to 0.5), recent advancement in material engineering demonstrates that in topological insulator θ_{SH} can go as high as 152 [117]. We recommend using topological insulators as the SOT layer to achieve a lower switching current.

Next, we analyze the impact of SOT layer geometry on the switching current (Fig. 3.13 (b), (c)). I_c scales down linearly with the decrease of SOT layer width, and w_{SOT} can be set to desired value based on the performance and reliability requirement (Fig. 3.13 (b)). While I_c scales linearly with the width of the SOT layer, the thickness of the SOT layer has an interesting effect on the switching current. The SOT layer should be relatively thin but bulk

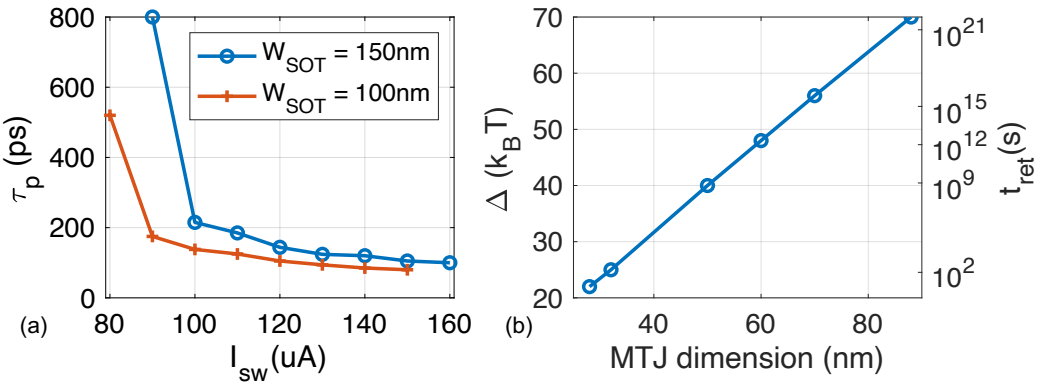


Figure 3.14: (a) Switching pulse width τ_p vs applied switching current I_{sw}. (b) Thermal stability factor Δ (left Y-axis) and retention time t_{ret} (right Y-axis) vs MTJ dimension for a fixed retention failure rate, $P_{RF} = 10^{-9}$. At $\Delta = 70$, MTJ dimension = 88nm, retention time is > 10 years [124].

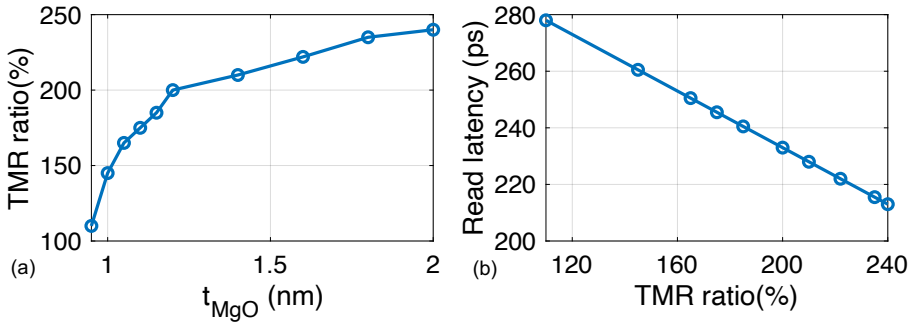


Figure 3.15: Impact of (a) oxide thickness on TMR, (b) TMR on read latency.

enough for heavy metal layers to experience the bulk effect to achieve high SOT efficiency. Once it crosses optimum thickness, which is 3nm (Fig. 3.13 (c)), many of the charges that are injected into the metal do not contribute to the switching, and I_c increases.

The smaller the free layer thickness, t_{FL}, the smaller the switching current (Fig. 3.13 (d)). We also scale the diameter of MTJ, d_{MTJ}, to reduce the MTJ area. However, with the scaling down of d_{MTJ} together with t_{FL}, the thermal stability factor Δ also scales down, reducing the memory's data retention time t_{ret}. Non-volatility is a great feature of MRAM, but it can be compromised to achieve higher density, higher bandwidth, and lower energy when the target application is a cache. Because, in the cache even for AI workloads, the

data lifetime is much shorter, typically in the seconds range [99]. Fig. 3.14(b) shows Δ and t_{ret} as functions of free layer volume. While scaling down t_{FL} to optimize I_c, and d_{MTJ} to optimize area, we keep an eye on the reliability of the stored data. We consider a retention failure rate of 10^{-9} (i.e., 1 bit flip per billion).

Bandwidth optimization

As shown in Fig. 3.15 (a), TMR ratio of the MTJ device can be increased by increasing the oxide thickness [119]. We increase the oxide thickness to decrease the read latency (Fig. 3.15 (b)). The write pulse width is inversely proportional to the applied switching current. While we want to lower the applied current to achieve low energy, the higher amplitude of the applied current is required for faster magnetization reversal. However, switching occurs at smaller pulse width at the iso-current if we scale down the SOT layer width. This is because of the smaller critical current at smaller geometry (Fig. 3.13 (b,d)). Fig. 3.14(a) shows that switching pulse width can be reduced significantly by scaling down the SOT layer width. Thus, we can achieve higher write bandwidth by scaling down the SOT layer width to meet the high BW demand from AI workloads.

Table 3.6: SOT-MRAM DTCO optimized parameters. 30% guard-band are added with thickness and width for process variations.

Parameter	**Value**	**Parameter**	**Value**
Spin Hall angle θ_{SH}	1	TMR	240%
Free layer thickness t_{FL}	0.5nm	MTJ diameter d_{MTJ}	55nm
SOT width w_{SOT}	130nm	SOT thickness t_{SOT}	3nm
Oxide thickness t_{MgO}	3nm	Thermal stability factor Δ	45

3.5.4 Process & Temperature Variation and Bitcell Simulation

In this subsection, we perform Process and Temperature variation on the DTCO-optimized parameters, design the peripheral circuits, and test the read-write operation on the bit cell at scaled parameters.

Process and Temperature variation

To incorporate process variations, we model MTJ diameter, free layer thickness, and SOT layer width as Gaussian variables in the Verilog A model of SOT-MTJ [107]. We assume standard deviations (σ) as 5% of their respective means (μ) and perform Monte Carlo simulations with 5000 samples within 4σ variation. We also consider the temperature variations. The extreme point at the right side of the scaled target parameter is $\mu + 4\sigma, T_{cold}$ (Fig. 3.16). From equations 3.9 and 3.10, I_{sw} and τ_p are independent of Temperature. As a result, the worst case for write operation (highest I_{sw} and longest τ_p) is at $\mu + 4\sigma$. This point is, however, benign to the read operation and retention failure. As we scale down d_{MTJ} and t_{FL}, Δ also reduces, reducing t_{ret}, and I_{data}. Δ reduces further as temperature increases [64]. Thus, the worst case for read operation (smallest I_{data}) and retention failure (smallest t_{ret}) is at $\mu - 4\sigma, T_{hot}$ (see Fig. 3.16). As I_{data} reduces, the difference between I_{data1} and I_{data0} becomes even smaller and difficult to sense.

To ensure the reliability of the SOT-MRAM bit cell, we add a 30% guard band on the scaled SOT device parameters: 20% for process variation and 10% for temperature variation. The optimized DTCO parameters after adding the PT induced 30% guard-band are shown in Table 3.6.

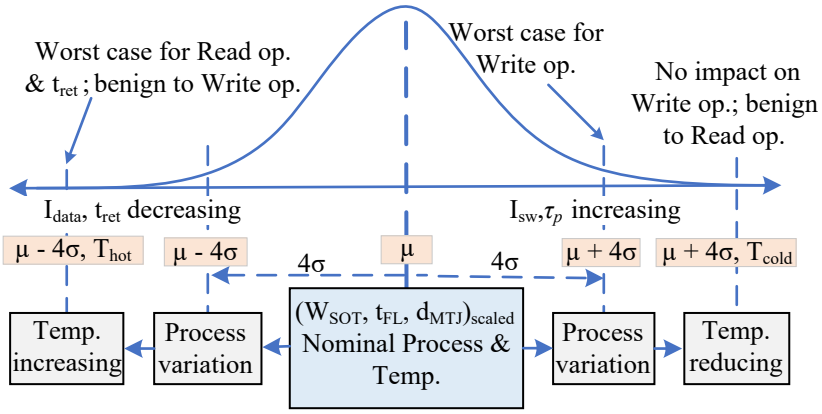


Figure 3.16: Impact and distribution of Process and Temperature variation on scaled parameters.

Write operation

To write SOT-MRAM bitcell, we bias BL with the data-to-be-written and SL with the complement of data-to-be-written. Assuming that the magnetization state of the Reference layer is -1, to write 1 into the MTJ bitcell, we switch the magnetic orientation of the Free layer to +1 state resulting in a high resistive state. To achieve this state, we turn on the WWL, connect BL to VDD and SL to the ground. The resultant current switches the free layer's magnetic orientation from -1 to +1. The opposite bias is applied to write 0. We do not need any additional peripheral circuits for the write operation of SOT-MTJ.

Read operation

Read operation involves sensing the current passing through MTJ at P and AP states. For our SOT-MRAM bitcell, with the parameters shown in Table 3.6, $I_{data0} = 20uA$ and $I_{data1} = 33uA$. We design and optimize the read circuitry to sense this small differential current, as shown in Fig. 3.17. Our proposed read sensing circuit only contains an additional current mirror block (to amplify current), and it does not require the precharge circuits

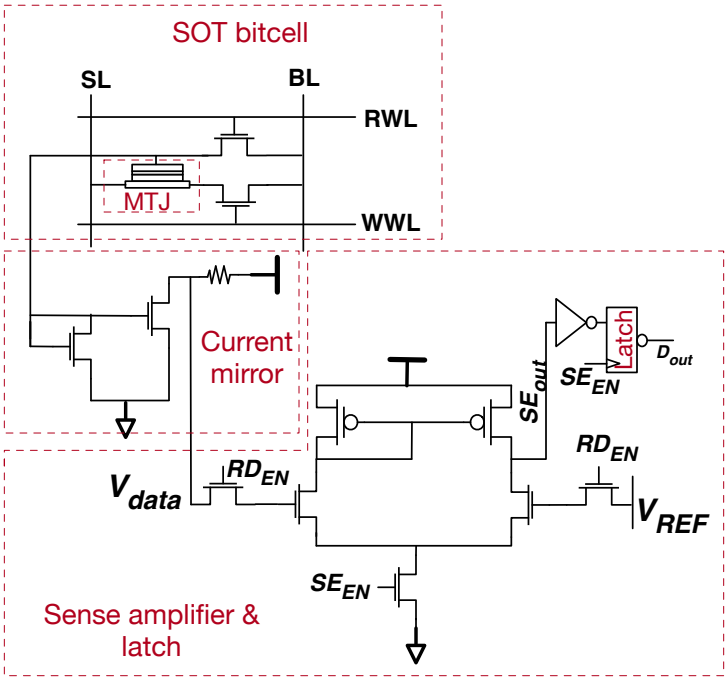


Figure 3.17: SOT-MTJ bitcell with read sensing circuitry.

compared to SRAM. Hence, there is no additional area overhead in the periphery compared to SRAM. The dynamic power consumption are shown in Table 3.7

To capture the stochastic nature of MTJ switching, we simulate the bit cell for 1000 bitstream. We achieve a read and write yield of 100%, and at 250ps and 520ps, respectively. This results in read bandwidth of 4 Gbps and a write bandwidth of 1.9 Gbps. We then dynamically allocate the memory bus width on-demand to satisfy the bandwidth requirement for different workloads and PE array size stated in section 3.5.1.

Table 3.7: Dynamic Power consumption (in uW) of SRAM and SOT-MRAM. (1/0) means the corresponding power to access bit 1 and 0.

	Read(1/0)	**Write(1/0)**
SRAM	426	373
SOT-MRAM	150/368	325/300

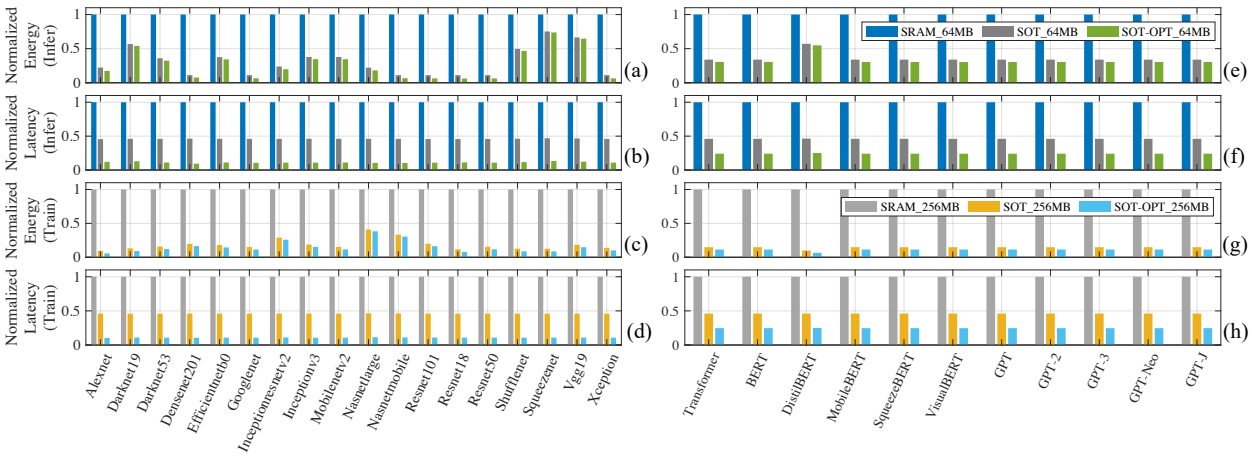


Figure 3.18: System level energy improvement with SOT-MRAM and DTCO-optimized-SOT-MRAM over SRAM at the same size for CV (a-d) and NLP (e-h) models. The top plots show energy (a, e) and latency (b, f) for inference, and the bottom plots show energy (c, g) and latency (d,h) for training.

3.5.5 System level performance evaluation of SOT-MRAM based Memory

In this subsection, we analyze the PPA (Power, Performance, and Area) metrics at the system level on the DNN/CNN benchmarks with SRAM, SOT-MRAM, and DTCO-optimized-SOT-MRAM. We use the Destiny [81] memory simulator to find the array-level data for both SRAM and SOT-MRAM. We modify Destiny source code to reflect: (i) SOT switching mechanism, (ii) special read sensing circuit for SOT-MRAM, and (iii) 14nm CMOS technode. Then, we feed the extracted bitcell-level data of SOT-MRAM in the *.cell* file to find the PPA at the desired memory capacity.

Based on the array-level results from Destiny, and DRAM & GLB access counts from Algorithms 1, and 2, we estimate the system-level power and performance. Finally, we analyze the area of the memory modules of different technologies (14nm SRAM, SOT-MRAM, and DTCO-opt-SOT-MRAM) at iso-capacity. This analysis only incorporates the PPA metrics from the memory system (DRAM and GLB), assuming that the PPA of the compute unit is constant. With SOT-MRAM as GLB, we see significant energy and latency improvement

over SRAM at 64MB (for inference) and 256MB (for training) (see Fig. 3.18 (a-d) for DNN benchmarks and (e-h) for NLP benchmarks). On average, the 64MB SOT-MRAM offers 5× energy reduction and 2× latency reduction over 64MB SRAM across all CNN models at inference. Our DTCO-optimized-SOT-MRAM offers further improvement, 7× energy, and 8× latency reduction over SRAM at iso-capacity. For latency improvement, the most contributing factor is the DRAM access reduction with large GLB and the smaller read/write latency of SOT-MRAM at larger capacity compared to SRAM. At smaller capacity, SRAM is way faster than SOT-MRAM [101, 105]. We observe that the most contributing factor in energy reduction ($> 50\%$) is the near-zero leakage power of SOT-MRAM compared to high leakage power of SRAM. The improvement is even more in training mode; 6× (8× with SOT-opt.) energy reduction and 2× (9× with SOT-opt.) latency reduction. With 64MB SOT-MRAM, NLP models in inference mode experience 2× (3× with SOT-opt.) energy reduction and 2× (4× with SOT-opt.) latency reduction than 64MB SRAM. Like CV benchmarks, with 256MB SOT-MRAM, NLP benchmarks also experience more energy improvement, 6× (8× with SOT-opt.), and latency improvement, 2.5× (4.5× with SOT-opt.), in training mode. The more improvement in training mode is because of two reasons: (1) GLB size increases from 64MB to 256MB, and (ii) GLB access counts are significantly large (at least 5×) in training. Our DTCO-opt-SOT-MRAM further adds value to PPA by its smaller silicon area, 0.54× at 64MB and 0.52× at 256MB of 14nm SRAM at iso-capacity (Fig. 3.19).

3.6 Related Work

SOT-MRAMs have been widely studied as the next generation of STT-MRAM to leverage all benefits of MRAMs as embedded memory [100][102][103][104][105][106]. However, very few studies have evaluated the performance of SOT-MRAM as on-chip memory in

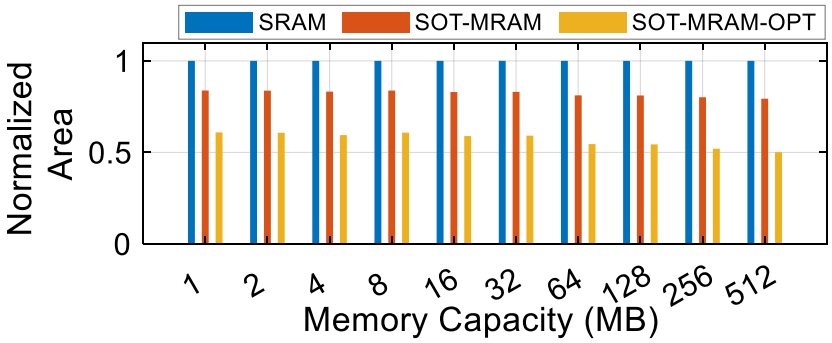


Figure 3.19: Area improvement of SOT-MRAM and SOT-MRAM-OPT

system-level for AI accelerators. [101] and [112] demonstrated the performance improvement of SOT-MRAM as L2 data cache compared to SRAM L2 cache on MiBench, SPEC2000 and SPEC2006 benchmarks. SOT-MRAMs have also been explored in the context of DL accelerators as a promising technology for In-Memory Computing (IMC) or Computing-In Memory (CIM) [113][114][115] [125][126]. IMC/CIM over conventional AI accelerator has pros and cons, and the detailed comparison between these two domains is outside the scope of this work. Our work, where we use SOT-MRAM as the cache storage element, differs from crossbar-based in-memory computing. While the scope of SOT-MRAM has been explored both as regular CPU cache and IMC for DL accelerator to some extent, to the best of our knowledge, unlike IMC, this is the first work that presents a comprehensive analysis of SOT-MRAM as on-chip memory for application in AI/DL accelerators.

3.7 Conclusion

In this research, we presented a System and Design Technology Co-optimization methodology for efficient and high-performance memory system design with SOT-MRAM for modern AI accelerators. Guided by detailed target workload characterization, our memory system comprises of HBM3 DRAM, a DTCO-enabled SOT-MRAM GLB and a small SRAM buffer.

Our large SOT-MRAM GLB significantly reduces the energy and latency by reducing expensive DRAM accesses while still having acceptable on-chip access energy and latency, achieving overall system-level high performance. We finally demonstrate that our memory system performs 8× and 9× better in terms of energy and latency respectively on CV benchmarks in training (7 and 8 times better in inference) and 8× and 4.5× better in terms of energy and latency respectively on NLP benchmarks in training (3 and 4 times better in inference) while consuming only around 50% of SRAM area at iso-capacity.

Chapter 4

Chiplet-Gym: Optimizing Chiplet-based AI Accelerator Design with Reinforcement Learning

4.1 Introduction

As Large Language Models (LLMs), such as chatGPT, GPT-4, LLaMA [174], etc., gain widespread use, there is a growing demand for energy-efficient hardware that can deliver high throughput. To support hundreds of trillions of operations and hundreds of gigabytes of data movement, the high-performance and energy-efficient hardware demands more silicon area, accommodating more compute cores and memory capacity. Training any state-of-the-art AI or Deep Learning (DL) model with a single GPU or accelerator is nearly impossible due to extreme computing and memory demands. The data centers are equipped with clusters of powerful computers and GPUs connected via PCIe, NVLink, etc.[45][134]. Even though these supercomputers can deal with large workloads, they consume a significant amount of energy [45] and involve longer latency. Because off-board communications consume at least one order of magnitude more power and time than any on-package communications [170]. The ideal scenario would be a hardware capable of housing the entire model parameters and intermediate activations on-chip[172], promising optimal performance and energy efficiency. Unfortunately, this is not feasible due to the stagnation of Moore's law and Dennard scaling, die size reaching the reticle limit, and the prohibitive manufacturing cost and yield limitations[134]. Consequently, researchers endeavor to replicate this 'hypothetical ideal' hardware concept by integrating multiple smaller chiplets at the package level, allowing near-ideal performance while minimizing costs and energy consumption.

With the advent of advanced packaging technologies, the chiplet-based heterogeneous integration has opened up a new dimension of chip design, **More-than-Moore** [134]. In chiplet-based system, multiple chiplets (i.e., SoCs) of diverse functionalities (e.g., logic dies,

memories, analog IPs, accelerators etc.) and tech nodes (e.g., 7nm or beyond) from different foundries are interconnected in package level using the advanced packaging technologies, such as CoWoS, EMIB, etc. [134]. The value proposition of chiplet-based architectures is manifold. Compared to multiple monolithic SoCs interconnected via off-package or off-board links such as PCIe, NVLink, CXL etc. [134], package-level integration of multiple monolithic SoCs via 2.5D or 3D has accelerated performance and lower energy consumption, alleviating off-package communications. Chiplet-based systems offer lower RE (Recurrent Engineering) cost by providing higher yield and lower NRE (Non-Recurrent Engineering) by enabling IP reuse and shortening IC design cycle [149].

The commercial chiplet-based general purpose products [140] [138] are designed and developed at vertically integrated companies without exposing much knowledge about the chiplet-based architectures' design space. Unlike these general purpose products, chiplet-based AI accelerators demand extensive design space exploration to hit the target Power, Performance, Area, and Cost (PPAC) budget. From architectural perspective, designers must consider the resource allocation, mapping and dataflow of the DNN workloads. From communication and integration perspective, chiplet placement, routing protocols, stacking/packaging technologies, interconnect types, and finally from application perspective, system requirement, such as reliability, scalibility etc., should be considered all at the same time while optimizing for PPAC [41]. The existing works often focus either on the architectural or integration aspects as a separate design flow: explore routing and packaging given chiplets [141][143][144] or explore chiplets architecture given the packaging [16][34] [36][172]. An isolated approach, addressing individual aspects independently, may result in sub-optimal designs due to the inter-dependency among these factors. For instance, varying resource allocation impacts communication demands, influencing the choice of packaging and its configuration, consequently leading to cost variations.

Currently, many flavors of packaging technologies, both from 2.5D and 3D, are available from the industry leaders, which makes it difficult for system designers and integrators to choose the optimum set of configurations from the vast design space based on the system requirements [134]. The various packaging technologies differ in fabrication cost and complexity, performance, and underlying integration technologies [134]. As a result, no single package technology can be marked as superior to others. Each of the other domains, such as resource allocation, chiplet granularity, placement, Network on Package (NoP), and interconnect architectures, to name a few, also has an extensive design space. A proper co-optimization across all these domains based on the system and application requirements at the available cost is necessary for a successful chiplet based system design.

Optimizing all possible domains results in a combinatorial explosion where brute force search is not an option and random search might not result in the optimum point. The expensive simulation environment of chip design exacerbates this problem.

To overcome these limitations, in this chapter, we make the following contributions: bridging the gap between the system requirements and design aggregation, planning, and optimization for chiplet-based architecture.

- We develop a co-design methodology for chiplet-based AI accelerators. The co-design task contemplates resource allocation, such as the number of AI chiplets, memory capacity, and bandwidth; partitioning and placement of chiplets such as aspect ratio of the accelerator chiplet arrays, and logical placement of accelerator and memory chiplet; different packaging technologies (i.e., CoWoS, EMIB, SoIC, and FOVEROS [134]) and their attributes such as bandwidth, bump pitch density, cost and complexity, to optimize the system-level Power, Performance, Area, and Cost (PPAC) of the chiplet-based AI accelerators.

- We formulate an analytical cost model for assessing the chiplet-based architectures. This analytical model enables us to assess the chiplet-based AI accelerator in a time-and-resource-constrained environment.

- To optimize throughput, energy efficiency, and cost, we identify the inter-dependency of the design space parameters and formulate the optimization problem as a Reinforcement Learning (RL) problem. We also explore non-RL based optimization approaches, such as simulated annealing, and combine these two approaches to ensure the robustness of the optimizer.

- Finally, we validate our methodology by comparing the performance of our optimized design against state-of-the-art monolithic GPU on MLPerf benchmark and justify the performance improvement.

The rest of the article is organized as follows. Section 4.2 presents the background. Section 4.3 describes the analytical modeling and design space exploration. The optimization framework is presented in section 4.4 followed by experiments and results in Section 4.5, related works in Section 4.6, limitations and future works in Section 4.7 and conclusion in Section 4.8.

4.2 Background

4.2.1 AI workloads and Accelerators

AI workloads

The primary domains of AI encompass Computer Vision (CV), Natural Language Processing (NLP), Recommender Systems, and Reinforcement Learning. The integration of

these domains has led to the emergence of Generative AI, enabling models to generate diverse content, including text and images. In Generative or Multi-modal AI, diverse AI/DNN (Deep Neural Network) models are fused together to generate an output.

While the architectural characteristics and parameters of LLM and CV models may differ, their fundamental components share similarities with the structure of Transformer[111] for NLP and ResNet[55] for CV, respectively. The critical operations in CV models involve regular convolution, Depth-wise or Point-wise convolution, residual blocks, FC (Fully Connected) operations, whereas the scaled-dot product attention operations, and FC operations dominate in LLM. These operations can be expressed as or converted to matrix-matrix/vector multiplication (GEMM) with massive parallelism.

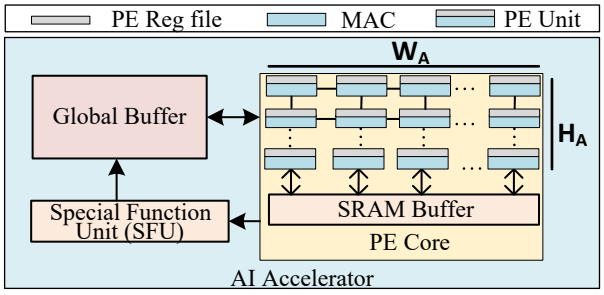


Figure 4.1: AI accelerator chiplet architecture

AI Accelerator

Systolic array [164] type architecture, leveraging the inherent parallelism of DNN workloads, has been used as the core of AI accelerators. A typical AI accelerator is composed of arrays of Processing Element (PE) for computation and on-chip buffer to hold the weights and activations. PEs are composed of Multiplier-Adder (MAC) units and small register file for each MAC units to hold the stationary data, depending on the dataflow. The size of the PE array, memory hierarchy, and memory size are critical design parameters of an AI accelerator. Fig. 4.1 shows an AI accelerator with a PE core, Special Function Unit (SFU),

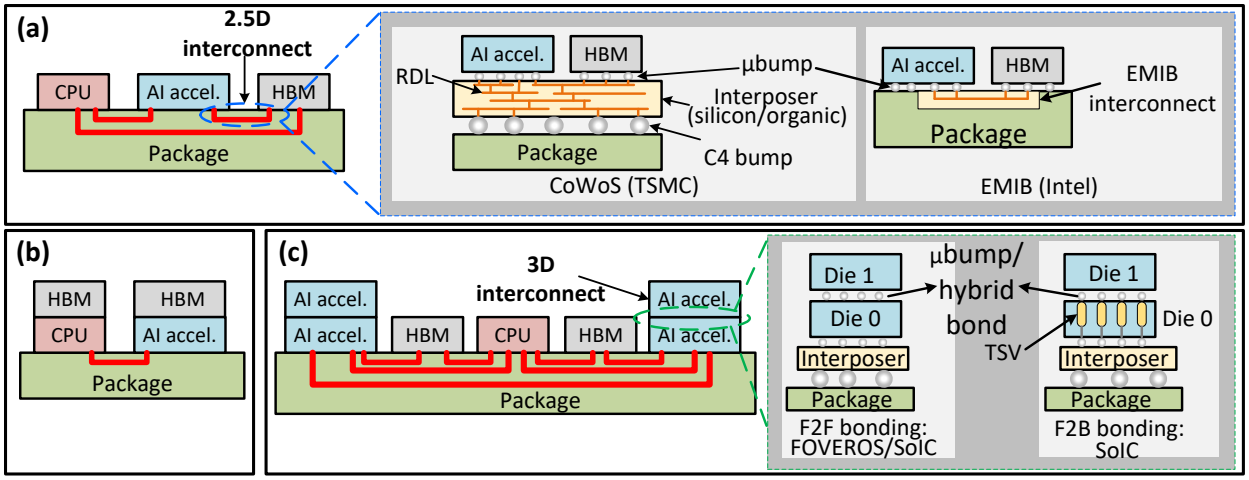


Figure 4.2: Top-level system architecture for different scenarios. (a) CPU, AI accelerator and HBM chiplets are connected in package level through 2.5D interconnects. CoWoS and EMIB are two options of 2.5D interconnects. (b) CPU and AI accelerator chiplets are connected through 2.5D interconnects and HBM is stacked on top of CPU and AI accelerator through 3D interconnects. (c) Two AI accelerator chiplets are stacked on top of each other through 3D interconnects and they are interconnected to CPU, HBM and other AI chiplets pair through 2.5D.

and Global Buffer. The PE core contains a small SRAM buffer and bunch of PE units. Each PE consists of a MAC unit and a reg. file [178].

4.2.2 Chiplets and Heterogeneous Integration

2.5D architecture

In 2.5D architecture, two or more chiplets, fabricated separately, are connected side-by-side with each other in package-level through interposer (silicon/organic) or silicon bridge. Two commercial 2.5D interconnects are Chip on Wafer on Substrate (CoWoS) from TSMC [129] and Embedded Multi-die Interconnect Bridge (EMIB) from Intel [130]. In CoWoS, two side-by-side dies are connected with each other and with package substrate through an intermediate interposer layer [129]. Interposer can be active and passive. Active interposer contains embedded logics and Re-Distribution Layers (RDL) where as passive interposer

containing RDLs are only used for routing purpose. CoWoS typically employs passive interposer for 2.5D integration. In contrast, Intel's EMIB utilizes thin silicon pieces with multilayer BEOL interconnects (Silicon Bridge) embedded in the organic package substrate for high-density localized interconnects, eliminating the need for a separate interposer layer [130]. CoWoS and EMIB architectures are illustrated in Fig. 4.2 (a).

3D architecture

In 3D, two or more separately fabricated chiplets are stacked on top of each other through 3D interconnects formed with copper micro-bumps, or hybrid wafer bonding [156]. Depending on the bonding interface orientation of the interconnected dies, different bonding configurations are possible, such as face-to-face (F2F), face-to-back (F2B), back-to-back (B2B) etc. Intel's FOVEROS [131] uses F2F bonding where the face of the top die is bonded to the face of the bottom die (active interposer) through Cu micro-bump connections. Bottom die is connected to the package through TSV [131]. TSMC has the option of both F2F and F2B bonding configuration in their System on Integrated Chips (SoIC), however, they use hybrid bonding instead of Cu μ-bumps [133]. The latest upgrade of FOVEROS, FOVEROS-Direct, also leverages direct cu-cu hybrid bonding for inter-die interconnection. Recently, both 2.5D and 3D can be integrated on the same package and these architectures are known as 5.5D [180].

4.3 Throughput formulation and Design space exploration

In this section, we formulate the cost model for chiplet-based AI accelerators, including throughput, energy, and cost. We perform design space exploration to comprehend the influence of various design parameters on the cost model.

4.3.1 Top level Architectural exploration

We explore two architectural approaches: (i) 2.5D architecture, where all chiplets are connected with each other at the package level through 2.5D interconnects (Fig. 4.2(a)). (ii) 5.5D (combining 2.5D and 3D)[180] where two or more 3D-stacked (connected via 3D interconnects) chiplets are further linked through 2.5D interconnects (Fig. 4.2 (b) & (c)). In all cases, the architecture of the AI accelerator chiplet is a regular systolic-array composed of PE array and dedicated on-chip buffer shown in Fig. 4.1[178]. However, the number of PE units and on-chip buffer size varies with the number of allocated chiplets, as we consider a fixed package size.

2.5D architecture

In 2.5D architecture, we consider that CPU, AI accelerator, and HBM chiplets are connected at the package level through 2.5D interconnects (Fig. 4.2 (a)). We explore two 2.5D integration technologies, EMIB and CoWoS, and their different configurations.

5.5D architecture (combining 2.5D and 3D)

5.5D architecture is divided into two cases: (i) memory-on-logic, where HBMs are stacked on top of CPU and/or AI chiplets as shown in Fig. 4.2 (b), and (ii) logic-on-logic, where two AI chiplets are 3D-stacked on top of each other. These 3D-stacked AI chiplets are connected to CPU and/or HBM and other 3D-stacked AI chiplets through 2.5D interconnects as shown in Fig. 4.2 (c). To avoid temperature-induced breakdowns [156], we limit our exploration to only 2-tiers. We explore the off-the-shelf 3D integration techniques, SoIC and FOVEROS, and their different configurations. Depending on the integration technology and their configuration settings, these architectures offer different bandwidths, energy efficiency, area efficiency, and cost.

4.3.2 Throughput and Energy efficiency formulation

Throughput

We define system throughput as *tasks completed per second*,

$$T = \frac{tasks}{sec} \tag{4.1}$$

tasks represents different entities depending on the DNN domain and its mode of operations. During inference, *tasks* represents the number of $inferences$. During the training of CV and NLP models, *tasks* means the number of *images* and *tokens* processed per second, respectively. $tasks/sec$ can be decomposed into [150]

$$\frac{tasks}{sec} = \frac{ops}{sec} \times \frac{1}{(\frac{ops}{task})_G} \times \frac{1}{(\frac{ops}{task})_{nG}} \times M_{eff} \tag{4.2}$$

Here, ops/sec depends on both DNN hardware and DNN models. GEMM operations per task, $(ops/task)_G$, and non-GEMM operations per task $(ops/task)_{nG}$ depend on only DNN models, and M_{eff}, mapping efficiecny depends on DNN models and hardware, along with mapping strategies. *ops* means the MAC operation. The GEMM operations are performed in the systolic array. The non-GEMM operations such as softmax is performed in the SFU of the accelerator. Dropout and residual operations manifested as Element-wise multiplication and addition, are also performed using the MAC modules. Layer normalization and other reduction or control flow operations are taken care of in the ALU or scalar unit of the SFU.

For a system comprising multiple AI accelerator chiplets, the $operations/sec$ is expressed as,

$$(\frac{ops}{sec})_{sys} = (\frac{ops}{sec})_{AI\_chip} \times AI\_chip_{tot} \times U_{sys} \tag{4.3}$$

Where $(ops/sec)_{AI\_chip}$ is the peak throughput per AI chiplet, AI_chip_{tot} = total number of AI chiplets, and U_{sys} = system utilization factor. It represents the effective fraction of the active chiplets out of the total chiplets. It depends on the interchiplet communication bandwidth (BW_{AI-AI}), determined by choice of the packaging architecture, package type, and their different configuration. In section 4.3.4, we describe this in detail. The peak throughput per AI chiplet is expressed as

$$(\frac{ops}{sec})_{AI\_chip} = (\frac{1}{\frac{cycles}{op}} \times \frac{cycles}{sec}) \times PE_{tot} \times U_{AI\_chip} \tag{4.4}$$

Where,

$$\frac{cycles}{op} = cycle_{comm} + cycle_{op^*} \tag{4.5}$$

$cycle_{comm}$ = chiplet-to-chiplet communication latency, $cycle_{op^*}$ = arithmetic operation latency of the chiplet microachitecture, and $cycles/sec{=}f$, frequency of the AI accelerator chiplets. $cycle_{comm}$ depends on the distance between the data source and destination. It is impacted by the chiplet allocation, chiplet array dimension (i.e., number of AI chiplets in X and Y dimension) and the physical location of the AI and HBM chiplets. $cycle_{op^*}$ depends on the micro-architecture of the chiplet (design of PE array, MAC unit) and the type of operations. We assume that all AI chiplets can operate at the same frequency and have the same architectural and functional configuration. However, the frequency of each chiplet can be further controlled based on the data traffic and location of chiplets to optimize system throughput and energy. PE_{tot} = total number of PEs per AI chiplet, and U_{AI_chip} = chiplet utilization representing the fraction of PEs utilized during computation. U_{AI_chip} depends on mapping of the AI model tasks to the accelerator.

Energy efficiency

Energy efficiency is paramount when processing DNN at edge devices and cloud data centers. Edge devices are usually constrained by battery life and thermal budget, and data centers are typically constrained by electricity bills, thermal budget, and environmental impact [45]. Data centers are mainly focused on achieving higher throughput, which requires higher energy budget. In this work, we closely monitor energy efficiency while maximizing the throughput.

We define energy efficiency (E_{eff}) of a system as *tasks completed per joule*:

$$E_{eff} = \frac{tasks}{joule} = \frac{1}{\frac{joules}{ops}} * \frac{1}{\frac{ops}{task}} \tag{4.6}$$

$joules/operations$ depends on both DNN hardware and DNN models, whereas $operations/task$ depends only on the considered DNN model. We break down the energy per operations, E_{op}, (i.e. $joules/operations$) into its constituent parts:

$$E_{op} = E_{comm} + E_{op^*} \tag{4.7}$$

E_{comm} is the energy required to transfer data from chiplet-to-chiplet and E_{op^*} is the energy to perform an arithmetic operation. E_{comm} depends on the choice of packaging architectures (e.g., 2.5D, 3D), interconnect types (e.g., EMIB, CoWoS, FOVEROS, SoIC), and the data traffic, whereas E_{op^*} depends on the microarchitecture.

4.3.3 Chiplet allocation and Placement

In the context of chiplet-based accelerator design, determining the number of chiplets, area allocated to each chiplet, and their placement becomes pivotal, as they impact the

throughput, energy, and cost. Here we will delve into the relationship between yield, area, cost, communication latency across various chiplet configurations.

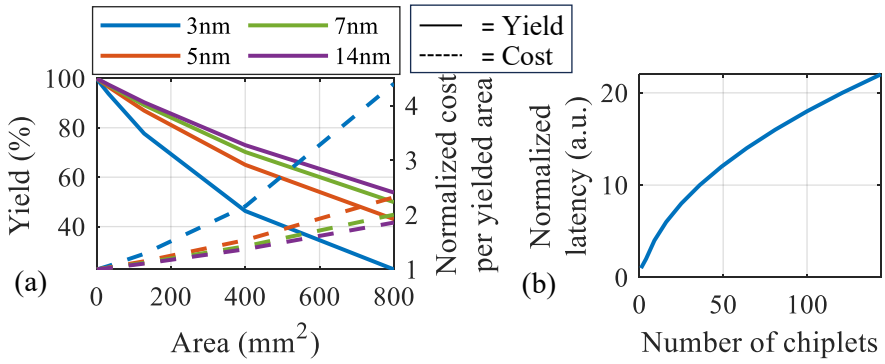


Figure 4.3: (a) Yield (left y-axis) and normalized cost per yielded area (right y-axis) vs area at different tech nodes. (b) Normalized latency vs number of chiplets.

Yield and Cost vs Area

Intuitively, as the chip area increases, its compute and memory capacity increases, ensuring high performance and energy-efficiency. However, as shown in Fig. 4.3 (a), we are limited by the fact that in advanced tech nodes, as the chip area increases, yield decreases, resulting in increased cost per area [149]. The yield of the manufactured chip, Y_{die} is expressed as the following Negative Binomial model:

$$Y_{chip} = (1 + \frac{dA}{\alpha})^{-\alpha} \tag{4.8}$$

where d is the defect density of the tech node, A is the area of the chip, and α is the cluster parameter. Assuming P_0 as unit price, we can also estimate the cost per yielded area as

$$C_{yield} = \frac{P_0}{Y_{chip}} \approx P_0(1 + dA + \frac{\alpha - 1}{2\alpha} d^2 A^2) \tag{4.9}$$

Inter-Chiplet Communication Latency

As mentioned earlier, the chiplet-to-chiplet data communication latency, $cycle_{comm}$, impacts the system performance by contributing to *cycles/operations*.

Data transfer between chiplets occurs through the package-level interconnects such as CoWoS, EMIB, FOVEROS, SoIC etc. Considering that the data might be supplied from another AI-chiplet or directly from HBM, we estimate both AI-AI chiplet communication latency, L_{AI-AI}, and HBM-AI communication latency, L_{HBM-AI}.

$$cycle_{comm} = \begin{cases} L_{AI-AI} & \text{if data moves from AI to AI chip} \\ L_{HBM-AI} & \text{if data moves from mem. to AI chip} \end{cases} \tag{4.10}$$

Impact of AI chiplet count. As the number of chiplet increases, the physical distance between the source and destination chiplet increases, resulting in increased communication latency. We consider 2D-mesh topology, which is widely used in tile-based architecture for its simplicity and scalability. Routing in the package substrate is more intricate than on-chip. As a result, tile-based chiplet architectures have been architected with mesh topology[16]. Fig. 4.3 (b) shows that communication latency increases drastically with the number of chiplets for a mesh topology.

Impact of Chiplet array dimension.

The longest AI-to-AI chiplet communication latency is expressed as

$$L_{AI-AI} = H_{AI-AI} \times t_w + H_{AI-AI} \times t_r + T_c + T_s \tag{4.11}$$

As we consider a 2D mesh of AI accelerator chiplets, $H_{AI-AI} = m+n-2$ denotes the number of hops between the source-destination pair. m, n represent the number of AI chiplets in the X and Y dimension of the array, respectively. t_w is per-hop wire delay, t_r, T_c, and T_s are

router delay, contention delay, and serialization delay, respectively [151]. Here, t_w, t_r, T_s are design time metrics, that depend on tech. node, interconnect technologies, circuit, and microarchitecture design, T_c depends on workload/data traffic. For a fixed number of chiplets and routing topology, H_{AI-AI} depends on the chiplet array X and Y dimension. We try to keep the aspect ratio of the chiplet array as close as possible to 1 to reduce the communication latency. In addition, the physical dimension of the chiplet array impacts the system performance by affecting the choice of dataflow and workload mapping strategies [34]. For a fixed dataflow and mapping strategy, the system performance largely depends on the chiplet array dimension as shown in Fig. 4.4.

Impact of HBM/CPU count and location. We analyze the impact of dividing the allocated HBM into multiple chiplets and placing the chiplets in multiple positions on system latency. Partitioning a large chunk of memory into multiple memory chiplets (instead of placing the large memory in one place) and placing these multiple memory chiplets in different locations improves the system latency. Unlike, AI chiplet counts, as the number of HBM chiplets increases, communication latency decreases. Because the communication latency depends on the physical location of the data [16]. Fig. 4.4 illustrates how chiplet partitioning and placement improve the system latency.

As we consider a 2D mesh of AI accelerator chiplets, there are 6 locations: left, right, top, bottom, middle, and 3D stacking, to place the HBM chiplets around the AI chiplets array. These locations result in $2^6 - 1$ combinations for HBM/CPU placements. We model $L_{HBM/CPU-AI}$ same as equation 4.11, where H_{AI-AI} is replaced by $H_{HBM/CPU-AI}$.

We use the model presented in [179] to calculate $H_{HBM/CPU-AI}$ for different locations of HBM/CPU pair. We consider a 16GB (8-stack, each stack 16Gb) HBM3 chiplet [135], giving the highest capacity of 80GB with 5 chiplets. We assume that each HBM chiplet has a dedicated memory controller and NoC router integrated within it [182]. As a result,

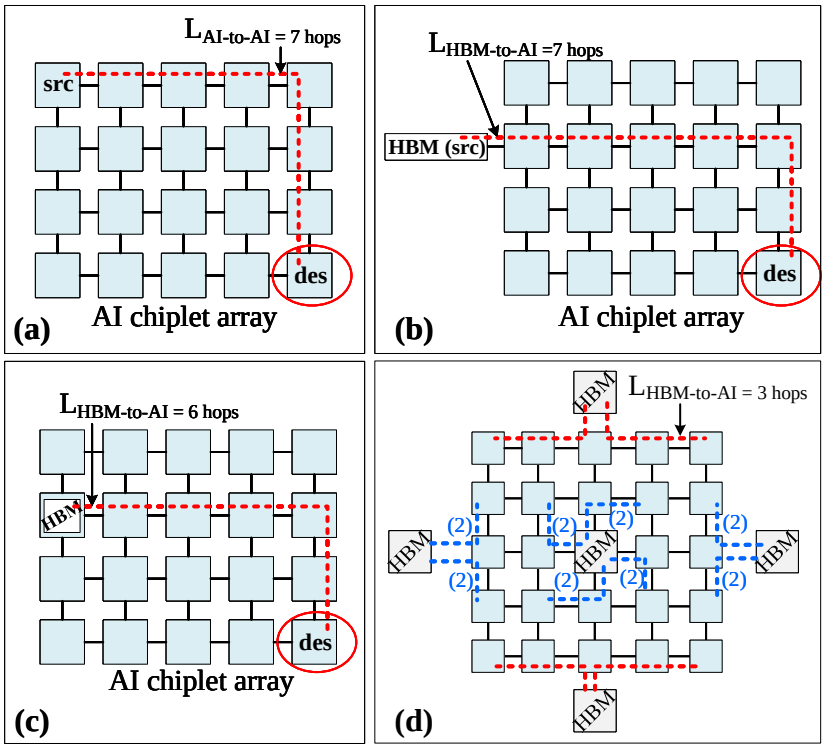


Figure 4.4: Illustration of latency (in terms of hop) calculation. (a) AI2AI chiplet communication, considering the farthest chiplets as source-destination pair. (b) One HBM chiplet, located at the left connected in 2.5D, and the farthest AI chiplet as source-destination pair. (c) One HBM chiplet, 3D-stacked on top of a left-most AI chiplet, and the farthest AI chiplet as source-destination pair. (d) 5 HBM chiplets are placed in 5 different positions. The highest latency decreases from 6 hops (case (c)) to 3 hops with most of the AI chiplets can be provided with data in 2 hops by nearest HBMs.

at iso-memory-capacity (i.e, same number of HBMs with integrated memory controller) the cost associated with HBM for both monolithic and chiplet systems is equivalent.

The host CPU is primarily responsible for dispatching the workloads to the accelerator chiplets. The package area is shared by accelerator chiplet, HBMs as well as CPUs. However, the majority of the package area is used for AI computing and HBM memories [182][16]. Hence, in this work we only focus on the AI accelerator and HBMs.

The above discussion suggests that, for cost-effective integration of more functionalities, we should partition the total chip area into multiple chiplets, each with smaller areas.

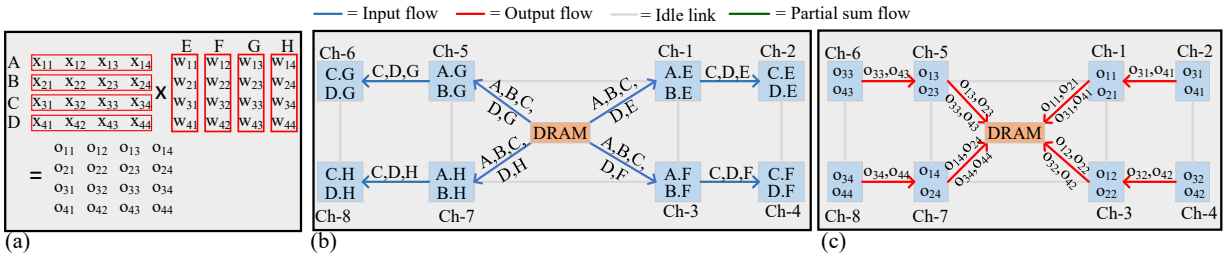


Figure 4.5: Illustration of mapping and dataflow. (a) Splitting the matrices into smaller parts for different chiplets. (b) Initial data supply from DRAM. Once the chiplets are loaded with required data, computation begins. (c) Final output collection to the DRAM. In this dataflow, there is no inter-chiplet communication during computation for partial sum.

From the yield and cost perspective, the more the number of chiplets, the better throughput and less cost. However, this also introduces another consideration: an increase in the number of chiplets results in higher inter-chiplet communication latency, ultimately diminishing throughput and energy efficiency. Therefore, a balance must be struck between dividing the area into an appropriate number of chiplets to enhance functionality and ensure the associated communication latency does not compromise overall system performance and efficiency.

4.3.4 Package architectures and configurations

We explore different packaging architectures, interconnects, and their different configurations [129][130][131][133] to analyze their impact on the system performance and budget.

Inter-chiplet communication bandwidth

The system utilization term, U_{sys}, of equation 4.3 depends on the inter-chiplet communication bandwidth. We define U_{sys}:

$$U_{sys} = \frac{BW_{act}}{BW_{req}} \tag{4.12}$$

Where, BW_{act} is the actual bytes of data transferred per sec and BW_{req} is the required bytes to keep all the neighboring AI chiplets at 100% utilization, i.e., no stalling for data. For the layout of AI and HBM chiplets we consider in this work, the HBM chiplet needs to deliver data to its 4 neighboring AI chiplets simultaneously at most, and any AI chiplet needs to deliver data to its 1 neighboring chiplets at most. However, it can change with the mapping strategies. As the communication between CPU and AI chiplet primarily involves the instruction dispatch and output accumulation, the communication bandwidth is dominated by bandwidth requirements of the AI accelerator to HBM chiplet.

Chiplet mapping exploration. For large sequence lengths and batch sizes of NLP/LLM models as well for large FC/Conv. layers of DNN models, the matrix sizes get larger, which need to be split temporally in the monolithic chips if the monolithic chip does not contain enough PE units and memory. Having multiple chiplets, the matrices can be split spatially and mapped to multiple chiplets, performing parallel computation.

As illustrated in Fig. 4.5(a-c), the input matrix is split along rows (A, B, C, D), and the weight matrix is split along columns (E, F, G, H). Chiplets 1, 3, 5, 7 handle data chunks A and B, while Chiplets 2, 4, 6, 8 handle C and D. The weight matrix portions (E, F, G, H) are distributed to all chiplets accordingly. During initialization, the DRAM supplies data $4 \times [A, B, C, D]$, and $[E, F, G, H]$ simultaneously to chiplets 1, 3, 5, 4, with A and B reaching neighboring chiplets in one hop and C and D reaching distant chiplets in the next hop. Data chunks E, F, G, H reach neighboring and distant chiplets in one hop and two hops, respectively. The outputs are collected back to DRAM once the computations are completed. Outputs from neighboring chiplets (ch-1, ch-3, ch-5, ch-7) reach DRAM in one hop, while outputs from distant chiplets reach DRAM in two hops. No inter-chiplet communication is required for partial sum accumulation, however, the required AI-HBM bandwidth (or number of channels) is higher in this mapping strategy, as DRAM needs to

broadcast $[A, B, C, D]$ to all four neighboring chiplets. According to the above mentioned mapping and dataflow, the required bandwidth is formulated as

$$BW_{req} = \begin{cases} 4 \times N_o \times d_w \times f \times (\frac{ops}{sec})_{AI\_chip} & \text{if src. is HBM} \\ 1 \times N_o \times d_w \times f \times (\frac{ops}{sec})_{AI\_chip} & \text{if src. is AI chip} \end{cases} \tag{4.13}$$

Where, N_o is the number of operands required to perform a MAC operation, which is 2 in general (two multipliers for the multiplication and no new external operands are needed for addition). d_w is the data width and $(ops/sec)_{AI\_chip}$ is the peak throughput of the AI chiplet, and f is the frequency of the accelerator. If $BW_{act} \geq BW_{req}$, then there is no stalling in initializing the chiplets' PE array with data. However, if $BW_{act} < BW_{req}$, then there will be $\lceil \frac{BW_{req}}{BW_{act}} \rceil$ cycle stalling for operand data to start the computation. We penalize the overall system throughput with these stalling periods while estimating the system throughput. From equation 4.13, the required bandwidth is smaller if the peak throughput of the AI chiplet is low, resulting in less penalty.

Impact of Data rates and Link count. The data rate per pin (in Gbps), DR, and the number of links assigned for data transfer, L, of different package type determine the active bandwidth, BW_{act},

$$BW_{act} = DR \times L \tag{4.14}$$

DR and L depend on the interconnect technology. It plays a significant role in the system throughput by contributing to the system utilization.

Inter-chiplet communication energy

Interchiplet communication energy E_{comm} depends on the packaging architecture and the data transfer volume. We model it as

$$E_{comm} = E_{bit\_pkg} \times bit_{tot} \tag{4.15}$$

E_{bit_pkg} is the energy per bit data communication for different interconnect technologies, and bit_{tot} is the data traffic required for the desired operation.

Impact of trace length and no. of RDL layers. For a specific data rate and link count, E_{bit_pkg} again depends on trace length, tr_len, (link-to-link distance between two interconnected dies). To achieve a specified data rate over a longer trace length, intricate circuit techniques and more RDL layers are required resulting in the $E_{bit\_pkg} \propto tr\_len$ relationship [129].

Packaging cost

The packaging cost (C_P) depends on the packaging architecture and interconnect type. For the same package type, the packaging cost again depends on (i) package area (A_P), (ii) number of layers (i.e., core and RDL), and (iii) link count (L) and modeled as [153] :

$$C_P = \mu_0 A_P + \mu_1 L + \mu_2 \tag{4.16}$$

Where μ_0, μ_1, and μ_2 are the regression parameters based on the number of core and RD layers. In this work, we consider a fixed package area of 900mm^2, leaving the packaging cost dependent on the number of package layers and link density.

The above discussion suggests that, based on the BW_{req}, which also depends on the number of chiplets, energy and cost budget, appropriate allocation of DR and L requires

co-optimization, such that the hardware is not suffering from under-utilization while not spending too much budget unnecessarily.

4.4 Optimizing Chiplet-based Architecture

In this section, we build a framework to efficiently navigate the search space, as detailed in Table 4.1, aiming to optimize throughput, energy, and cost efficiency. We formulate the objective function as:

$$\max_{X \in D} \quad \alpha T(X) - \beta E(X) - \gamma C(X) \tag{4.17}$$

where T, E, and C are the throughput, energy, and cost, respectively, expressed as the function of design parameters, X, within design space, D. α, β, γ are the user-specified constants which let the users put specific weightage on the desired metrics.

Table 4.1: Parameters and values of Design Space

Parameter	**Values**
Architecture type	2.5D, 5.5D: (i) memory-on-logic (ii) logic-on-logic
No. of chiplets	1 to 128 @ step of 1
No. & location of HBMs	Left, right, top, bottom, middle, 3D stacked; 2^6 -1 location
AI2AI interconnect 2.5D	CoWoS, EMIB
AI2AI data rate 2.5D (Gbps)	1 to 20 @ step of 1
AI2AI link count 2.5D	50 to 5000 @ step of 50
AI2AI trace length (mm) 2.5D	1 to 10 @ step of 1
AI2AI interconnect 3D	SoIC, FOVEROS
AI2AI data rate 3D (Gbps)	20 to 50 @ step of 1
AI2AI link count 3D	100 to 10,000 @ step of 100
AI2HBM interconnect 2.5D	CoWoS, EMIB
AI2HBM data rate 2.5D (Gbps)	1 to 20 @ step of 1
AI2HBM link count 2.5D	50 to 5000 @ step of 50
AI2HBM trace length 2.5D (mm)	1 to 10 @ step of 1

Comprising of 14 parameters and their possible values, our parameter space has more than 2×10^{17} design points which poses challenges for exhaustive search due to its time and

resource-intensive nature. To address this, we explore learning-based and meta-heuristic search approaches to efficiently reach global or near-global optima.

Because of the inherent stochastic nature of Reinforcement Learning (RL) and Simulated Annealing (SA) algorithms, we observe slight variations in the achieved objective function values. To enhance the robustness of the optimizer, we train multiple RL models and SA algorithms with different seed values. Subsequently, we perform an exhaustive search across the outcomes of these algorithms to pinpoint the optimum solution (refer to Alg. 3). An overview of the optimization framework is presented in Fig. 4.6. It takes the design space and constraints as input and outputs the optimized design points.

Algorithm 3: Proposed optimization algorithm

```
t ← Trial_max;
obj_best ← −inf;
while t ≤ Trial_max do
    param_SA, obj_SA ← SA();
    if obj_SA > obj_best then
        param_best, obj_best ← param_SA, obj_SA;
    end
    param_RL, obj_RL ← RL();
    if obj_RL > obj_best then
        param_best, obj_best ← param_RL, obj_RL;
    end
end
return param_best, obj_best
```

4.4.1 RL problem formulation

RL tries to mimic human learning behavior to learn about a new environment. In RL, an **Agent** continuously interacts with an **Environment**, takes **Actions** by observing the present **State** of the environment, receives feedback as a form of **Reward** from the environment, and updates its underlying **Policy** to take new actions to maximize reward. After enough interaction with the environment, the agent can take a specific set (sequence) of

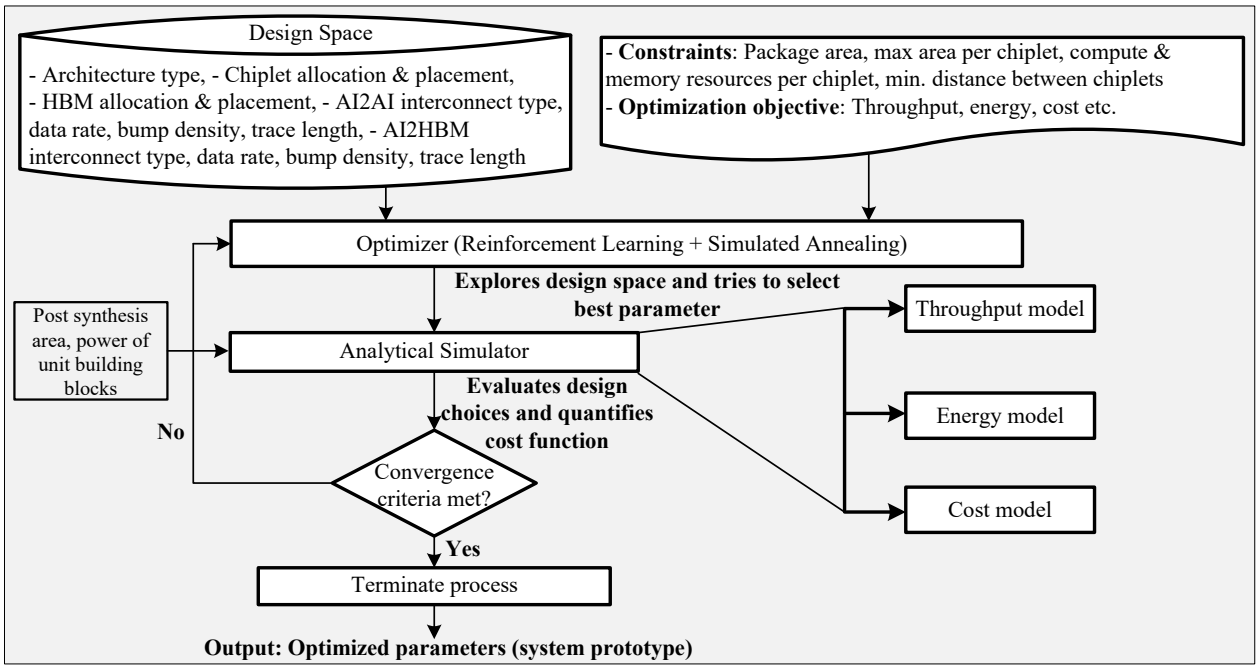


Figure 4.6: Optimization framework overview

actions that maximize the reward in the given environment. Formulating a Markov Decision Process (MDP) consisting of a tuple of five key elements: $< \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \delta >$ is at the core of formulating an RL problem. Where $\mathcal{S}$ = State space, $\mathcal{A}$ = Action space, $\mathcal{P}$ = Transition probability matrix of going to S_t from S_{t-1} by taking action A_{t-1}, r = Reward function, and δ = discount factor that takes any value between $[0, 1]$ [154].

Environment provides feedback to the agent by quantifying the rewards. In our case, we incorporate our analytical expressions discussed in Section 4.3 into a Gym[166] environment, known as Chiplet-Gym, to assess the performance of the action taken by the agent.

State or Observation space contains the set of all possible states of the environment. It should include all the information for the agent to take the next action, making the process an MDP. In our case, the observation space contains the following items: {*maximum package area, the maximum area allowed per chiplet, current area per chiplet, ai2ai communication*

latency, ai2hbm communication latency, current communication energy, current packaging cost, current throughput}.

Action space defines the set of all possible actions available to the agent each time step. Our action space, consisting of a combination of discrete integers and categorical values, corresponds to the parameters we aim to optimize. Given the state of the environment and the reward, the agent selects values for each of the parameters in Table 4.1 to maximize the reward.

Reward is provided to the agent as a form of feedback in response to every action it takes. We formulate the reward function same as the objective function we want to maximize

$$r = \alpha T - \beta E - \gamma C \tag{4.18}$$

Where T, E, C represent the throughput, communication energy, and packaging cost respectively. α, β, γ are the user-defined constants that let the users put specific weight on specific parameters of the objections function, such as throughput, cost, energy-efficiency during optimization. Based on the reward, which is formulated from the analytical expressions of Section 4.3, RL finds the optimum design choices considering complex trade-offs of chiplet area, bandwidth, chiplet-to-chiplet communication.

RL algorithm We use Proximal Policy Optimization (PPO) algorithm [162] implemented by Stable-Baselines3 [163] because of its simplicity, computational efficiency, and compatibility with the action and state space of our problem. PPO is a on-policy policy gradient method that combines the idea of having multiple workers from Advantage Actor-Critic (A2C) algorithm and the idea of using trust region to improve the current policy from Trust Region Policy Optimization (TRPO) algorithm [163].

4.4.2 Simulated Annealing

In addition to RL, we also explore meta-heuristic search approaches, such as simulated annealing, to evaluate their efficacy in navigating the design space. Simulated annealing adds an exploitation step on top of random search. It randomly samples the design points and in addition to accepting the better design points, based on the acceptance criterion, it also accepts the design points that worsen the objective function. We modify the simulated annealing algorithm by slightly changing the acceptance criterion for our problem. The algorithm is shown in Algorithm 4. We optimize the same objective function as shown in Equation 4.17: $\max_{X \in D} \ \alpha T(X) - \beta E(X) - \gamma C(X)$.

Algorithm 4: Modified simulated annealing algorithm

```
1  iteration ← T_max;
2  temp ← temperature;
3  st_sz ← step_size;
4  X_curr ← randomly choose initial solution;
5  O_curr ← evaluate initial solution;
6  X_best, O_best ← X_curr, O_best;
7  while iterations ≤ T_max do
       /* find candidate solution                                   */
8      X_cand ← X_curr + uniform(−1, 1) ∗ st_sz;
       /* evaluate candidate solution                               */
9      O_cand ← f(X_cand);
10     if O_cand > O_best then
11         O_best ← O_cand;
12         X_best ← X_best;
13     end
14     t ← temp/iterations;
15     if O_cand > O_curr OR rand() < t then
16         X_curr, O_curr ← X_cand, O_cand;
17     end
18 end
19 return X_best, O_best
```

Finally, we deploy RL and SA algorithm multiple times, followed by conducting a comprehensive search on the outputs produced by SA and RL agents.

While demonstrated explicitly for AI accelerators and mesh routing topology, the proposed optimization framework can be generalized to diverse chipset-based designs and routing topology, requiring users to model their architectures and network topology in equations 4.4, 4.10, 4.11, and 4.13 to find the correct blend of the package and interconnect architecture. For example, I/O chiplets provide signal transmission and regeneration. Their performance can be modeled as extra latency in our framework.

4.5 Experiments and Results

4.5.1 Experimental method

As shown in Fig. 4.6, at the core of the optimizer we implement PPO and simulated annealing algorithm. The optimizer explores the design space and tries to select the best parameters sticking to the design constraints and user-given optimization objective, such as throughput optimization, energy, and/or cost optimization. To evaluate the optimizer's objective function, we implement our cost model, explained in Section 4.3, in an OpenAI Gym[166] environment named as Chiplet-Gym.

Table 4.2: Per hop wire length and delay for 2.5D and 3D architecture [157][130]

Packaging arch.	**Per hop wire length (mm)**	**Delay, t_w (ps)**
2.5D	1	17.2
3D	0.08	1.6

Table 4.3: Interconnects' properties[129]

Interconnect	**Bond/bump pitch (μm)**	**TSV pitch pitch(μm)**	**Energy (pJ/bit)**	**Implementation cost**
CoWoS	30 - 40	-	0.2 ∼0.5	Medium
EMIB	55 - 45	-	0.17 ∼0.7	Low
SoIC	9	9	0.1 ∼0.2	High
FOVEROS	< 10	-	< 0.05	Highest

We consider a fixed amount of package area, 900mm^2, dedicated for AI and HBM chiplets [182]. To avoid thermal hotspot, we place the chiplets at 1mm apart from each other in a mesh topology [159]. This leaves $(900 - (m + n + 2)mm^2)$ of area for the chiplets. The optimizer will select the number of chiplets such that it maximizes the throughput while sticking to the area constraint. The area per chiplet is calculated as the total package area available for AI chiplets over the number of chiplets. Analyzing the yield vs area curve (Fig. 4.3) we set the maximum allowable area per chiplet to 400mm^2 as a constraint. Because, at 14nm, for the die area beyond 400mm^2 the yield is even lower than 75%. Inspired by the recent trend of higher on-chip memory to reduce the DRAM accesses [164], we allocate 40% of the chiplet area to the compute resources, 40% to the on-chip SRAM, and rest 20% to other blocks such as control, IO, NoC, routing etc. For 3D architecture, we have to sacrifice some of the area of the chiplet for the TSV and its associated keepout zone. From SoIC TSV pitch of 9um[129], > 12K TSVs can be fit into 1mm^2. So we keep at most 2mm^2 for TSV in 3D architecture. Which is enough for both signal and power supply [160]. We use the values shown in Table 4.2 and 4.3 in our throughput, cost, and energy model to calculate the cost function of the design points.

4.5.2 Implementation details

The optimization framework[1] is written in Python v3.9. and run on an Intel hexa-core i5-9500 @ 3 GHz machine.

RL

The Chiplet-Gym environment is constructed by integrating our analytical simulator into OpenAI Gym v0.26.2 [166] to establish a unified interface between the RL algorithm and the analytical simulator. We define the action space as MultiDiscrete and observation

[1]https://github.com/KFM135/chiplet-optimizer

space as Box space. The current state (which includes design metrics like throughput, energy, and cost) is mapped to the action space through the PPO policy network that outputs a probability distribution over possible actions in the MultiDiscrete action space. This distribution allows the PPO agent to select a combination of parameters that define a design point for the simulator. After the action is taken, the simulator updates the state with new design metrics and calculates a reward based on performance. This state-reward feedback is then used by the PPO's critic network to refine the policy, encouraging actions that yield higher rewards in future steps. PPO's clipping mechanism ensures stable updates by constraining drastic changes in the policy, promoting a smooth and reliable mapping of states to actions to maximize overall rewards.

Policy-Value network. PPO utilizes the Multi-Layer Perceptron (MLP) as both its policy and value network. The architecture of the actor or policy network is defined as [10, 64, 64, 810], and the architecture of the critic or value network is set as [10, 64, 64, 1], employing the tanh activation function. The size of the input for both networks is determined by the dimension of the observation space, while the output layer size of the policy network is determined by the action space. The output layer size of the value network is set to 1.

Impact of episode length on RL convergence. The algorithms are trained with an episode length of 2. While a longer episode length often results in a higher mean episodic reward, it does not guarantee a superior cost model value for the optimized parameters. Although longer episodes are generally associated with increased exploration, our hypothesis is that, in our specific case, the agents lean towards exploitation to maximize rewards. This hypothesis arises from the fact that our reward values span from a large negative value to a positive one. Once the agent discovers a positive value, it tends to exploit that particular action to maximize the mean episodic reward neglecting further exploration of the design space. Figure 4.7 (a) shows that the agent achieves a mean episodic reward of 800 at episode length of 10, where as the cost model value of these actions are less than 100 (Fig. 4.7 (b)).

On contrary, at episode length of 2, the mean episodic reward is around 300 and the cost model value is around 150. (Note: The cost model value at each timesteps are calculated as *mean_episodic_reward/episode_length*.)

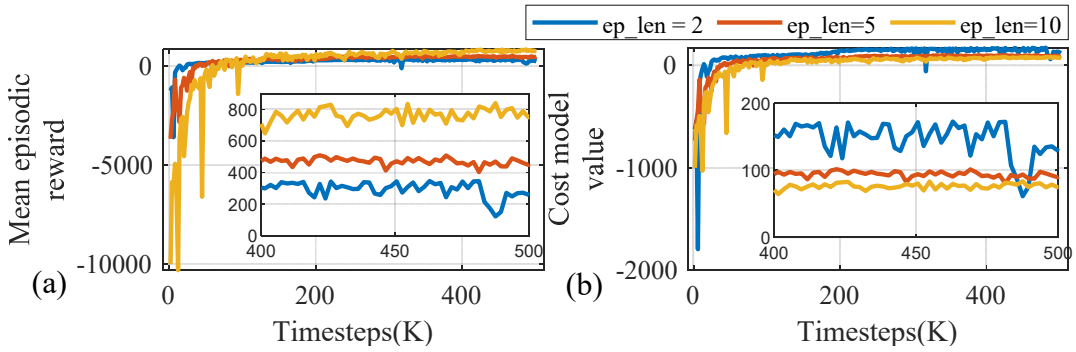


Figure 4.7: Impact of episode length in convergence (PPO algorithm). Inset shows the zoomed-in version of each plot.

Impact of entropy coefficient on RL convergence Another hyper-parameter impacting the exploration and exploitation balance is entropy coefficient. Serving as a regularizer, entropy coefficient plays a crucial role in shaping the behavior of the RL agent during training. A larger entropy coefficient implies that all actions are equally likely, fostering exploration, while a smaller entropy coefficient indicates that one action's probability within the policy dominates, emphasizing exploitation. Fig. 4.8 (a) shows that when entropy coefficient is set to 0, the agent stabilizes to a lower reward value more rapidly. However, when the entropy coefficient is increased to 0.1, the agent achieves a higher reward value, albeit with a slightly less stable trajectory. In this case, we use an entropy coefficient of 0.1 to reach higher convergence value. Other significant hyperparameters of PPO algorithm are shown in Table 4.4.

Simulated Annealing

We employ Algorithm 4, initializing it with a randomly chosen candidate solution from the design space. Like PPO, SA's performance is also sensitive to initial temperature, a

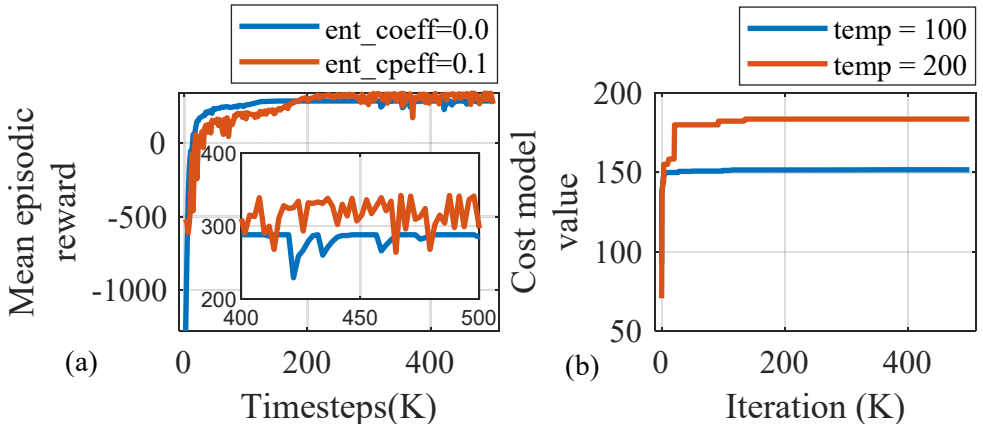


Figure 4.8: (a) Impact of entropy coefficient in RL convergence and (b) impact of temperatue on SA convergence . Inset shows the zoomed-in version of each plot.

measure of exploration vs exploitation. As shown is Fig. 4.8 (b), SA achieves significant higher cost model value with higher temperature value. Higher temperature value ensures more exploration by increasing the probability of accepting a worse trial point. As a result, the initial temperature to 200, and a step size of 10 is employed for locating the neighboring points. We do not use the general Metropolis acceptance criterion, $metropolis = exp - \{(O_{curr} - O_{cand})/t\}$, due to the potential for $(O_{curr} - O_{cand})$ to become very large or very small, leading to the *metropolis* evaluating to either infinity or 0. Instead, we solely utilize the parameter t to statistically accept poorer solutions in the early stages, facilitating exploration of the search space. O_{curr}=cost model value for current design point and O_{cand}=cost model value at candidate design point.

Table 4.4: PPO hyper-parameters & their values

n_steps	2048	n_epoch	10
batch_size	64	learning rate	0.0003
clip range	0.2	value func. coef.	0.5
entropy_coeff.	0.1	discount factor	0.99
bias-variance trade-off factor			0.95

Table 4.5: Optimized parameters for $\alpha, \beta, \gamma = [1, 1, 0.1]$ found by PPO algorithm

Parameter	**Case (i): 64 chiplets as upper bound**	**Case (ii): 128 chiplets as upper bound**
Architecture type	5.5D-Logic-on-Logic	5.5D-Logic-on-Logic
No. of chiplets	60 (30 3D chiplet pairs arranged in 5X6 2.5D mesh)	112 (56 3D chiplet pairs arranged in 7X8 2.5D mesh)
Package area; per-chiplet area	900 mm^2; 26mm^2	900 mm^2; 14mm^2
HBM location & capacity	4 16GB HBM chiplets @ top, bottom, right, and middle of 5X6 chiplet pairs with a total capacity of 64GB	4 16GB HBM chiplets @ left, right, bottom, and middle of 7X8 chiplet pairs with a total capacity of 64 GB
AI2AI 2.5D interconnect	EMIB	EMIB
AI2AI 2.5D data rate	20 Gbps	20 Gbps
AI2AI 2.5D link density	3100	1450
AI2AI 2.5D trace length	1 mm	1 mm
AI2AI 3D interconnect	SoIC	FOVEROS
AI2AI data rate 3D	42 Gbps	34 Gbps
AI2AI 3D link density	3200	4400
AI2HBM 2.5D interconnect	EMIB	EMIB
AI2HBM 2.5D data rate	20 Gbps	20 Gbps
AI2HBM 2.5D link density	4900	3850
AI2HBM 2.5D trace length	1 mm	1 mm

4.5.3 Results

Performance and runtime analysis of optimizer

In our investigation of the design space, we consider two distinct scenarios: case (i), wherein the upper limit for the number of AI chiplets is set to 64, and case (ii), where this upper limit is increased to 128. We ran each of the algorithms multiple times for each cases with different seed values to ensure their convergence stability. Fig. 4.9 and 4.10 (a) and

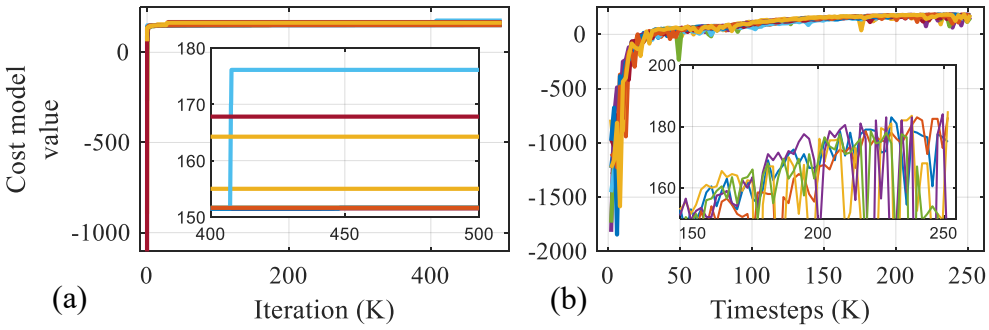


Figure 4.9: Convergence behavior of (a) SA and (b) RL for multiple runs with 10 different seed values for case (i) (i.e., 64 chiplets). Inset shows the zoomed-in version of each plot.

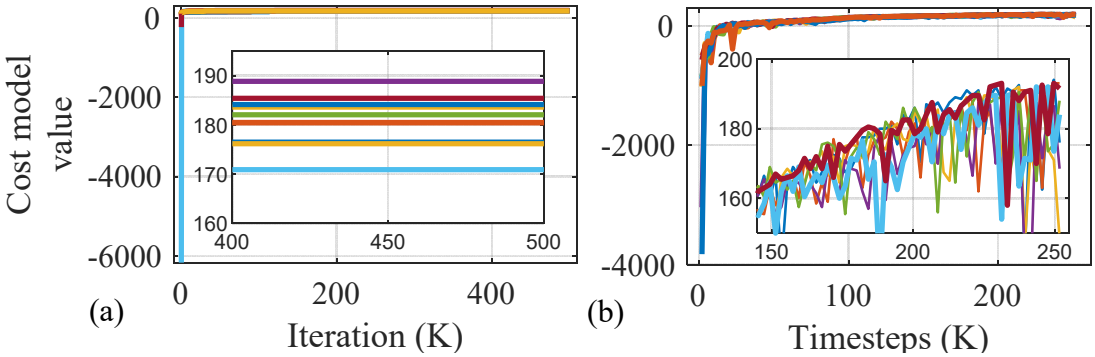


Figure 4.10: Convergence behavior of (a) SA and (b) RL for multiple runs with 10 different seed values for case (ii) (i.e., 128 chiplets). Inset shows the zoomed-in version of each plot.

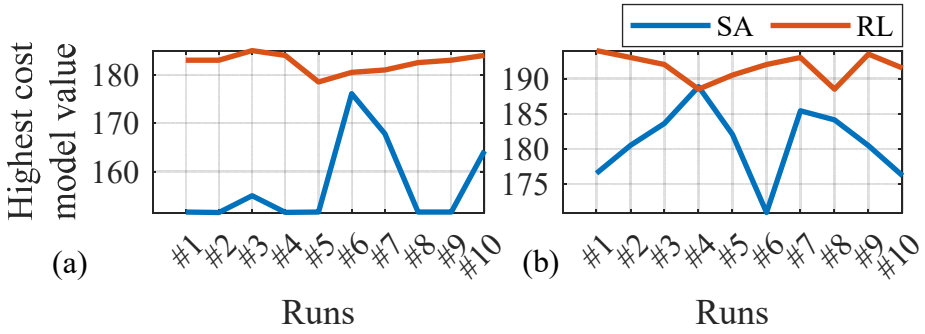


Figure 4.11: Highest cost model value achieved by the SA and RL algorithms for multiple runs: (a) for 64 chiplets and (b) for 128 chiplets.

Table 4.6: DNN benchmark features

Benchmark model	**Domain**	**Dataset**	**Ops. per forward pass**
Resnet50	Image classification	Imagenet	4 GFLOPs
Efficientdet	Light weight object detection	COCO 2017	410 GFLOPs
mask-RCNN	Heavy weight object detection	COCO 2014	447 GFLOPs
3D-UNet	Biomedical image segmentation	KiTS19	947 GFLOPs
BERT	Natural Language Processing	Wikipedia 2020	32 GFLOPs

(b) show the convergence behavior of SA and PPO algorithm for case (i) and case (ii) for 10 runs, respectively. As expected, both algorithms achieve a better cost model value for case (ii) because of its higher throughput, however, due to large packaging cost, case (i) 64 chiplets as the upper bound, is considered more practical. Fig. 4.11 (a) and (b) show the highest cost model value achieved by SA and RL algorithm over 10 runs for case (i) and (ii), respectively. We observe that RL achieves higher cost model values each run and more stable over multiple runs ranging from 178 - 185 for case (i) and 188 - 194 for case (ii). Where SA achieves 151 - 176 and 170 - 188 for case (i) and case (ii), respectively.

The run time of SA for 500K iterations is less than a minute and the run time to train the PPO agent for 250K timesteps is $<$ 20 mins. We finally integrated several trained RL agents and performed SA optimization on-the-go, and performed an exhaustive search among those SA and RL agents. The final optimizer with 20 SAs and 20 RL trained RL agents take around 10 mins to report the optimized parameter. As the RL is used in inference mode, here the SA dominates the runtime.

Optimized architecture evaluation

Table 4.5 shows the optimized parameter found by the optimizer for both cases for a specific α, β, γ value (user-defined weights on the objective function as explained in Eqn. 4.18). We observe that the RL PPO algorithm found the best parameter. Please note that multiple design configurations may coexist, achieving almost identical cost model value.

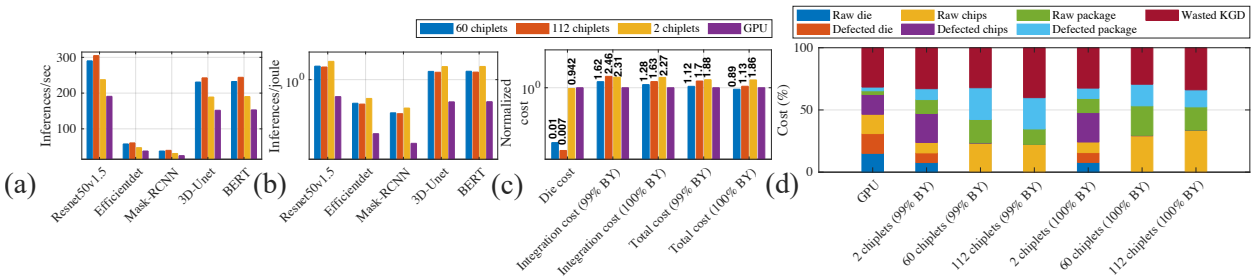


Figure 4.12: Comparison of 60-chiplet, 112-chiplet, 2-chiplet and monolithic system: (a) Inferences/sec, (b) Inferences/joule for MLPerf benchmark, and (c) cost. (d) Cost breakdown of monolithic, 2-chiplet, 60-chiplet, and 112-chiplet system at 99% and 100% package bonding yield (BY = bonding yield).

The optimal design point for case (i) consists of 30 3D AI chiplet pairs arranged in a mesh topology 5×6, resulting in 60 chiplets in total. 2 chiplets (forming a pair) are connected with SoIC 3D integration technology with a data rate of 42Gbps per link and link count of 3200 providing up to 131.25 Tbps of bandwidth. Each chiplet pair is connected with other chiplet pair with 2.5D EMIB integration with a data rate of 20Gbps and a link count of 3100 delivering up to 60 Tbps of bandwidth. Four 16GB HBM chiplets, located at top, right, bottom, and middle of the 5×6 mesh topology, are connected to 2 to 4 neighboring AI chiplets with EMIB 2.5D integration technology with a data rate of 20Gbps per link and a link count of 4900, resulting in a bandwidth of 95 Tbps. The trace length for each 2.5D interconnect is selected as the minimum trace length possible (minimum chiplet-to-chiplet distance). In case (ii), when we increase the maximum number of chiplets to 128, we observe that the optimum design configuration contains 112 chiplets (56 chiplet pairs) and the communication bandwidth decreases for all cases. This is because, as the number of chiplets increases, area per chiplet decreases, resulting in smaller throughput per chiplet, less bandwidth demand, and high system utilization. We observe that 3D architecture, even with area penalty for TSV and TSV-associated keep-out zone[181], achieves 1.52× more logic density than its 2D/2.5D counterpart at the same package size.

We synthesize the chiplet module, found by the optimizer, with Synopsys Fusion Compiler using their 14nm PDK [85] at 1GHz clock frequency and obtain the peak throughput per chiplet, $(ops/sec)_{AI\_chip}$, and energy consumption per MAC operation, E_{op*}. We use these values in our analytical model to estimate the throughput and energy efficiency of the 60 and 112 chiplet system. For cost estimation, we use the model from [149].

Fig. 4.12 compares the 60-chiplet, 112-chiplet, 2-chiplet and monolithic GPU for MLPerf benchmark [169]. The benchmark features are briefly summarized in Table 4.6. We observe that 3D 112-chiplet, 60-chiplet, and 2-chiplet systems achieve 1.60×, 1.52×, and 1.24× higher throughput of the monolithic one, respectively (Fig. 4.12 (a)). The higher throughput of the chiplet-based system can be explained with three facts. First, higher logic density in 3D logic-on-logic systems in the same area footprint increases the peak theoretical throughput. This explains why all chiplet-based systems show higher throughput than the monolithic one. Second, as the peak theoretical throughput per chiplet (or chiplet pair) increases, the required inter-chiplet communication bandwidth increases. If the active bandwidth cannot sustain the required inter-chiplet (both AI2AI and AI2HBM) bandwidth, system utilization decreases, resulting in decreased achieved throughput. This explains why the 112-chiplet system has the highest throughput compared to the 60-chiplet and 2-chiplet systems. The per-chiplet throughput in the 112-chiplet system is smaller, requiring less inter-chiplet bandwidth and ensuring higher system utilization. Conversely, the 2-chiplet system requires a higher inter-chiplet bandwidth due to its higher per-chiplet throughput, leading to under-utilization and reduced overall throughput. Third, as the number of AI chiplet increases, the inter-chiplet communication latency increases. However, the lower bandwidth penalty of the 112-chiplet system outweighs the higher latency penalty, resulting in a superior overall throughput compared to the 60-chiplet and 2-chiplet systems.

The 2-chiplet, 60-chiplet, and 112-chiplet systems are 4.63×, 3.76×, and 3.62× energy-efficient (inverse of energy consumption) compared to the monolithic, respectively (Fig. 4.12

(b)). The monolithic system is less energy-efficient than the 3D chiplet based system at iso-throughput. Because, to achieve equal throughput, more than one monolithic chips need to be connected off-board on the PCB, consuming at least one order of magnitude more energy[170] than on-package communication. Among the 3 chiplet-based configurations, 2-chiplet system achieves slightly higher energy-efficiency, 1.23× and 1.28× compared to 60-chiplet and 112-chiplet system, respectively, as it requires less inter-chiplet communication. However, handling the thermal hotspot and heat removal for such large and high-throughput 3D stacked chiplets presents a significant challenge.

Fig. 4.12 (c) shows the cost comparison of the monolithic vs chiplet based systems at different bonding yields. The raw die costs of 60-chiplet, 112-chiplet, and 2-chiplet configuration are 0.01×, 0.007×, and 0.94×, respectively, of the monolithic system. This significant cost difference arises from the low yield (48%) of the monolithic chip of 826mm^2, compared to the 97% and 98% die yield of the 60 and 112 chiplet systems, with a die size of 26mm^2 and 14mm^2, respectively, at 7nm node. In addition to that, the cost of Known Good Dies (KGD) is inversely proportional to the number of KGD (N_{KGD}). As the die area (A) increases, the number of good dies (N_{KGD}) decreases, leading to a substantial increase in cost. The relationship between the cost and die area can be approximated as $cost_{KGD} \propto A^{\frac{5}{2}}$ (taking up to 2 terms of Taylor series expansion of die yield) [170][149].

We estimate the packaging cost of chiplets at 99% and 100% inter-chiplet bonding yield. With better process control and TSV/pad repair techniques, TSMC reported that the bonding yield can reach 100%[133][185]. Although the raw die cost is smaller, for chiplet based configurations, the integration cost, including all the defected and wasted chips and packages, of chiplet based system are 1.62× (for 60-chiplet), 2.46× (for 112-chiplet), and 2.31× (for 2-chiplet) higher than the monolithic system at 99% bonding yield. The integration cost improves with the 100% bond yield. Finally, combining the die and integration cost, we observe that the total cost for the 60-chiplet system can achieve the 0.89× cost of the

monolithic system with the 100% bonding yield, while the total cost for 112-chiplet configurations is slightly higher (1.13×) than that of the monolithic system. The 2-chiplet system is the most cost inefficient, as it does not benefit from the lower raw die cost and also suffers the high 3D integration cost. Fig. 4.12 (d) shows the breakdown of the total cost of the different configurations. For all configurations, wasted KGD (i.e., wasted chips) consumes a significant amount of total cost, with a maximum of 40% of the total cost for the 112-chiplet system at 99% bonding yield and a minimum of 29% of the total cost for the 60-chiplet at 100% bonding yield. In monolithic and 2-chip systems, die and chip costs (raw and defective) contribute equally to the total cost, and package-related costs (raw and defective) only consume 6% and 20% of the total cost, respectively. On the other hand, in 60 and 112 chiplet-based configurations, the cost of the defected dies and chips are less than 1%. The cost of raw chips and packages (raw and defected) dominates the total cost in these cases. We implement our chiplet in synopsys 14nm free PDK. However, we estimate the cost for 7nm to have a fair comparison between the monolithic one, which was fabricated in 7nm technode[95].

4.6 Related works

4.6.1 Chiplet-based architecture exploration

DNN accelerator

SIMBA [16] is a pioneering work in chiplet-based AI accelerator, that integrates 36 NVDLA-like accelerator chiplets on a package. Centaur [35] integrates CPU and FPGA chiplets on package, specially for recommendation system workload. SPRINT [36] is a 64-chiplet system with photonic interconnect for DNN inference. There have been few works in chiplet based architecture focusing on different aspect of design space exploration. NN-Baton[34] proposes a framework for DNN workload mapping and chiplet granularity in small

scale (1 to 8 chiplets), however, they do not consider the packaging integration aspect and fabrication cost. While Monad [41] incorporates mapping, resource allocation, communication and different package substrate to optimize for PPA and fabrication cost, their packaging integration design space is limited to 2.5D, excluding 3D. [172] proposes ChipletCloud for LLM inference, however, their chiplets are connected in board-level instead of package level.

General purpose

Some works focus on the exploration of Network-on-Package (NoP) and reliable routing protocols [141] for chiplet-based architecture. [143] explores network topology and cost-aware chiplet placement for 2.5D architecture. [149] puts forward a cost model for evaluating the 2.5D manufacturing cost. [170][158] suggest the importance of chiplet design space exploration for performance, energy, cost, reliability enhancement.

4.6.2 RL in Design-space exploration

Deep Reinforcement Learning has gained popularity in exploring the design space exploration and optimization of the EDA domain, spanning from front-end (i.e., planning and architectural exploration) [45][49][144][147] to back-end (i.e., implementation, physical design and circuit design)[42][43] [176]. To the best of our knowledge, this work is the first to perform a comprehensive design space search, encompassing resource allocation, placement, packaging architectures (both 2.5D and 3D), and their configurations to optimize for Power, Performance, Area, and Cost (PPAC) using Deep Reinforcement Learning (DRL).

4.7 Limitations and Future Works

In order to keep the design space concise and tractable, we limit it as mentioned in Table 4.1 and make several assumptions as mentioned in Section 4.5.1. We also assume

that the HBM3e chiplets have their integrated memory controller and NoC router that can be used as a node in the mesh topology. The analysis of the cost of additional chips with the NoC+memory interface, exploring other routing topology such as p2p with photonic interconnects, H tree, bus, ring etc., exploring more heterogeneous architectures, multi-tier 3D-stacks, placement of host CPU chiplets and exploring their different layouts are future works.

4.8 Conclusion

This chapter proposes Chiplet-Gym to explore the design space of chiplet-based AI accelerators to optimize for Power, Performance, Area, and Cost (PPAC). To evaluate the design points, we analytically model the power, performance, and cost for chiplet-based AI accelerator. With reinforcement learning and simulated annealing, the optimizer is robust and efficient in locating the global or near-global optima of the design space for PPAC. The results show that the optimizer finds the design point that achieves 1.52× throughput, 0.27× energy, and 0.89× cost of its monolithic counterpart in iso-area.

Chapter 5

Conclusions and future works

5.1 Conclusions

With the ever-increasing demand for AI and generative AI, AI/DL accelerators capable of running the AI/DL workloads optimally become inevitable. This dissertation identifies the key metrics of AI/DL accelerators and presents system- and circuit-level design and optimization methodologies to achieve some of the key metrics of AI/DL accelerators.

We showed that the larger on-chip memory plays a significant role in the performance boost and energy efficiency of the AI/DL accelerators. To enable larger on-chip memory/cache, we explored the potentials and feasibility of emerging NVM technologies as embedded on-chip memory. Chapter 2 presents a design methodology of an innovative scratch-pad assisted on-chip STT-MRAM based buffer system based on the model-driven detailed design space exploration. The STT-MRAM based AI accelerator achieves 75% area and and 3% power saving over regular SRAM-based AI accelerator at iso-accuracy of the DNN models. Chapter 3 explores SOT-MRAM to address the limitations posed by STT-MRAM. The SOT-MRAM based memory system was developed by performing a closed-loop STCO and DTCO by taking the system performance attributes, architectural and micro-architectural attributes, workload attributes into account while performing the device and circuit-level optimization on SOT-MRAM. The proposed SOT-MRAM-based memory system achieves 8× energy and 9× latency savings compared to the SRAM-based memory system on the CV and NLP benchmarks in training mode while consuming only 50% of the SRAM area at iso-capacity. Together, these 2 chapters provide high performance and energy efficiency over the baseline SRAM-based accelerator by increasing on-chip memory and the use of MRAM over SRAM provide area efficiency at iso-capacity.

To improve the logic density cost-efficiently and to reduce the off-board communication we explore chiplet-based 3D accelerators, in chapter 4. However, the design space explodes in the chiplet-based accelerator. We analytically model the Power, Performance, Cost and Area of chiplet based AI accelerator and to efficiently navigate the vast design space, we introduce Reinforcement Learning to find the optimal design point. The optimizer suggested design point achieves 1.52× throughput, 0.27× energy, and 0.89× cost of its monolithic counterpart in iso-area.

Overall, this work provides a solid foundation for future research and development of energy-efficient, high-performance AI hardware. The methodologies and findings presented here can serve as a valuable resource for both academic research and industry applications, contributing to the ongoing evolution of AI accelerator technologies.

5.2 Future works

5.2.1 Dynamic Re-configuration of Hardware Resources at Runtime to Optimize Energy and Throughput

This work focuses on design-time optimization for chiplet-based architectures based on workload type (e.g., inference, training). However, for multi-modal DNN models and Generative AI, dynamic run-time reconfiguration of hardware resources could further improve energy efficiency and throughput. This approach could also enhance the scalability and generality of AI accelerators.

The existing run-time dynamic re-configurations approaches such as dynamic voltage and frequency scaling (DVFS) are designed considering a single device operation (i.e., standalone CPU/GPU). As a result, conventional DVFS techniques cannot be applied to multi-accelerator systems to optimize performance-per-watt. The design and exploration of a dynamic resource management framework - consisting of (i) memory resource, (de)activation,

and (ii) DVFS policies - for multi-accelerator systems to optimize energy and throughput that is aware of model/data-parallel execution paradigm will be interesting future work.

5.3 Photonic interconnects in chiplet-based AI accelerators

As the communication demands continue to rise in the data-intensive and parallel computing of AI workloads, existing interconnect solutions face scalability challenges, impacting both performance and power consumption. Photonic interconnects, with their high bandwidth density, low power per bit, and resilience to distance-dependent latency, offer a promising alternative. However, their successful deployment in chiplet architectures requires overcoming specific challenges, including thermal sensitivity, device reliability, and optimal integration with existing electronic systems. Future research should focus on high-performance, energy-efficient, and scalable chiplet architectures that leverage the full potential of silicon photonics for next generation computing. These include the exploration of adaptive, application-aware photonic interconnect solutions that enhance not only dataflow and reconfigurability across diverse workloads but also the power and thermal constraints that hinder the scalability in large chiplet systems.

5.4 Reconfigurable memory system

In-memory computing, where computation is performed directly within the memory arrays, represents a promising paradigm for addressing the memory bottleneck in data-intensive applications, especially in deep learning and artificial intelligence (AI). By eliminating the need to transfer data back and forth between memory and processing units, in-memory computing can significantly reduce data movement and thus improve energy efficiency. However, despite its potential, in-memory computing faces several challenges that limit its current

utility, such as handling training, accuracy loss, and the additional circuit complexity of ADC-DAC converters.

A potential solution to these limitations is the development of a reconfigurable memory system, where individual cells can be dynamically configured to function as either storage or computation units, depending on the workload requirements. By dynamically adjusting the mode of operations of the memory cells, such a system could flexibly allocate resources depending on the workload features, computation, and memory requirements. For example, during inference tasks, a subset of cells could be reconfigured to perform matrix multiplications while other cells store intermediate results. In contrast, during training, more cells could revert to their storage roles while the matrix multiplications are performed in the traditional MAC core. Ultimately, a reconfigurable in-memory computing architecture, which can dynamically balance computing and storage functions, has the potential to advance the performance and energy efficiency of AI and data-intensive applications significantly, supporting different types of DNN applications.

Bibliography

[1] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. Mostofa, Y. Yang, and Y. Zhou, "Deep Learning Scaling is Predictable, Empirically" in arXiv preprint arXiv:1712.00409, 2017.

[2] S. Singh, A. Bin, S. Kumar, and F. Carroll, "Generative Artificial Intelligence: A Systematic Review and Applications" in arXiv arXiv:2405.11029, 2024.

[3] J. Hestness, D. Newsha and G. Diamos, "Beyond human-level accuracy: computational challenges in deep learning," in ACM Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming, Pages 1-14, February 2019.

[4] S. Kung, "VLSI Array processors" in IEEE ASSP Magazine, vol. 2, Pages 4-22, 1985.

[5] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning" in Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, Pages 269–284, 2014.

[6] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor" in 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Pages 92-104, 2015.

[7] Y. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks" in IEEE Journal of Solid-State Circuits, vol. 52, Pages 127-138, 2017.

[8] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. Keckler, and W. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks" in SIGARCH Comput. Archit. News, vol. 45, Pages 27–40, 2017.

[9] V. Sze, Y. Chen, T. Yang, and J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey" in Proceedings of the IEEE, vol. 105, Pages 2295-2329, 2017.

[10] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects" in SIGPLAN Not., vol. 53, Pages 461–475, 2018.

[11] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks" in IEEE International Symposium on High Performance Computer Architecture (HPCA), Pages 553-564, 2017.

[12] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks" in 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Pages 1-12, 2016.

[13] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-S: Addressing Irregularity in Sparse Neural Networks through A Cooperative Software/Hardware Approach" in 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Pages 15-28, 2018.

[14] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training" in IEEE International Symposium on High Performance Computer Architecture (HPCA), Pages 58-70, 2020.

[15] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer" in 47th Annual IEEE/ACM International Symposium on Microarchitecture, Pages 609-622, 2014.

[16] Y. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. Tell, Y. Zhang, W. Dally, J. Emer, C. Gray, B. Khailany, and S. Keckler, "Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture" in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, Pages 14–27, 2019.

[17] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "TANGRAM: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators" in Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, Pages 807–820, 2019.

[18] Online, "NVDLA", Available in https://nvdla.org/, 2024.

[19] Online, "CEREBRA's WSE-3", Available in https://cerebras.ai/product-chip/, 2024.

[20] Whitepaper, "Accelerated Computing with a Reconfigurable Dataflow Architecture" in https://sambanova.ai/, 2024.

[21] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. Reinhardt, A. Caulfield, E. Chung, and D. Burger, "A Configurable Cloud-Scale DNN Processor for Real-Time AI" in ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Pages 1-14, 2018.

[22] Online, Apple Neural Engine, Available in https://machinelearning.apple.com/research/neural-engine-transformers, 2024.

[23] R. Prabhakar, S. Jairath and J. L. Shin, "SambaNova SN10 RDU: A 7nm Dataflow Architecture to Accelerate Software 2.0," in IEEE International Solid-State Circuits Conference (ISSCC), pp. 350-352, 2022.

[24] D. Abts et al., "Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads," in ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 145-158, 2020.

[25] B. Hickmann, J. Chen, M. Rotzin, A. Yang, M. Urbanski and S. Avancha, "Intel Nervana Neural Network Processor-T (NNP-T) Fused Floating Point Many-Term Dot Product," in IEEE 27th Symposium on Computer Arithmetic (ARITH), pp. 133-136, 2020.

[26] E. Talpes, D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti, and G. Sachdev, "Compute Solution for Tesla's Full Self-Driving Computer" in IEEE Micro, vol. 40, Pages 25-35, 2020.

[27] A. Samajdar, J. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim" in IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Pages 58–68, 2020.

[28] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. Reinhardt, A. Caulfield, E. Chung, and D. Burger, "A Configurable Cloud-Scale DNN Processor for Real-Time AI" in ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Pages 1-14, 2018.

[29] H. Liao, J. Tu, J. Xia, H. Liu, X. Zhou, H. Yuan, and Y. Hu, "Ascend: a Scalable and Unified Architecture for Ubiquitous Deep Neural Network Computing : Industry Track Paper" in IEEE International Symposium on High-Performance Computer Architecture (HPCA), Pages 789-801, 2021.

[30] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "STONNE: Enabling Cycle-Level Microarchitectural Simulation for DNN Inference Accelerators" in IEEE International Symposium on Workload Characterization (IISWC), pp. 201-213, 2021.

[31] W. Won, T. Heo, S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-model Training at Scale" in IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), vol. , Pages 283-294, 2023.

[32] R. Venkatesan et al., "MAGNet: A Modular Accelerator Generator for Neural Networks," in IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1-8, 2019.

[33] Z. Wang et al., "AI Computing in Light of 2.5D Interconnect Roadmap: Big-Little Chiplets for In-memory Acceleration," in International Electron Devices Meeting (IEDM), pp. 23.6.1-23.6.4, 2022.

[34] Z. Tan, H. Cai, R. Dong and K. Ma, "NN-Baton: DNN Workload Orchestration and Chiplet Granularity Exploration for Multichip Accelerators," in ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), pp. 1013-1026, 2021.

[35] R. Hwang, T. Kim, Y. Kwon, and M. Rhu, "Centaur: a chiplet-based, hybrid sparse-dense accelerator for personalized recommendations" In Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA '20). IEEE Press, 968–981, 2020.

[36] Y. Li, A. Louri and A. Karanth, "Scaling Deep-Learning Inference with Chiplet-based Architecture and Photonic Interconnects," in 58th ACM/IEEE Design Automation Conference (DAC), pp. 931-936, 2021.

[37] K. Mishty and M. Sadi, "Chiplet-Gym: Optimizing Chiplet-based AI Accelerator Design with Reinforcement Learning," in IEEE Transactions on Computers, doi: 10.1109/TC.2024.3457740., 2024,

[38] M. Orenes-Vera, et al. "Massive data-centric parallelism in the chiplet era." in arXiv preprint arXiv:2304.09389, 2023.

[39] M. Odema et al. "SCAR: Scheduling Multi-Model AI Workloads on Heterogeneous Multi-Chiplet Module Accelerators." in arXiv preprint arXiv:2405.00790, 2024.

[40] H. Peng et al. "Chiplet cloud: Building ai supercomputers for serving large generative language models." in arXiv preprint arXiv:2307.02666, 2023.

[41] X. Hao, Z. Ding, J. Yin, Y. Wang and Y. Liang, "Monad: Towards Cost-Effective Specialization for Chiplet-Based Spatial Accelerators," in IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 1-9, 2023.

[42] A. Mirhoseini et al. "Chip placement with deep reinforcement learning" in arXiv preprint arXiv:2004.10746, 2020.

[43] H. Wang et al. "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning." in 57th ACM/IEEE Design Automation Conference (DAC) IEEE, pp. 1-6, 2020.

[44] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi and B. Nikolic, "AutoCkt: Deep Reinforcement Learning of Analog Circuit Designs," in Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 490-495, 2020.

[45] J. You, J.W. Chung, and M. Chowdhury. "Zeus: Understanding and optimizing GPUenergy consumption of DNNtraining." in 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). 2023.

[46] T. -R. Lin, D. Penney, M. Pedram and L. Chen, "A Deep Reinforcement Learning Framework for Architectural Exploration: A Routerless NoC Case Study," in IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 99-110, 2020.

[47] A. Yazdanbakhsh et al. "Apollo: Transferable architecture exploration." in arXiv preprint arXiv:2102.01723, 2021.

[48] S. Krishnan et al. "Multi-Agent Reinforcement Learning for Microprocessor Design Space Exploration." in arXiv preprint, 2022.

[49] S. C. Kao, G. Jeong and T. Krishna, "ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning," in 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 622-636, 2020.

[50] F. Ahmad et al. "Llmcarbon: Modeling the end-to-end carbon footprint of large language models." in arXiv preprint arXiv:2309.14393, 2023.

[51] E. Strubell, A. Ganesh, and A. McCallum. "Energy and policy considerations for modern deep learning research." in Proceedings of the AAAI conference on artificial intelligence. Vol. 34. No. 09. 2020.

[52] OpenAI's GPT-3: Technical Overview, in https://lambdalabs.com/blog/demystifying-gpt-3

[53] A. Parashar et al. "Timeloop: A systematic approach to DNN accelerator evaluation," in IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 304-315, 2019.

[54] Y. N. Wu, J. S. Emer, V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," in International Conference on Computer Aided Design (ICCAD), pp. 1-8, November 2019.

[55] K. He et al. "Deep residual learning for image recognition." in Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pp. 770-778, 2016.

[56] G. Batra et al., "Artificial-intelligence hardware: New opportunities for semiconductor companies" in McKinsey & Company, 2019.

[57] S. Moore, "Cerebras's Giant Chip Will Smash Deep Learning's Speed Barrier" in IEEE Spectrum, 2020.

[58] D. Shin, J. Lee, J. Lee, J. Lee, and H. Yoo, "DNPU: An Energy-Efficient Deep-Learning Processor with Heterogeneous Multi-Core Architecture" in IEEE Micro, vol. 38, no. 5, pp. 85-93, Sep./Oct. 2018.

[59] Y. Jin, M. Shihab, and M. Jung, "Area, Power, and Latency Considerations of STT-MRAM to Substitute for Main Memory" in 41st International Symposium on Computer Architecture (ISCA-41), 2014.

[60] Q. Dong et al., "A 1Mb 28nm STT-MRAM with 2.8ns read access time at 1.2V VDD using single-cap offset-cancelled sense amplifier and in-situ self-write-termination" in IEEE ISSCC, pp. 480-482, 2018.

[61] Y. Chih et al., "A 22nm 32Mb Embedded STT-MRAM with 10ns Read Speed, 1M Cycle Write Endurance, 10 Years Retention at 150°C and High Immunity to Magnetic Field Interference" in IEEE ISSCC, pp. 222-224, 2020.

[62] L. Wei et al., "A 7Mb STT-MRAM in 22FFL FinFET Technology with 4ns Read Sensing Time at 0.9V Using Write-Verify-Write Scheme and Offset-Cancellation Sensing Technique" in IEEE ISSCC, pp. 214-216, 2019.

[63] A. Antonyan, S. Pyo, H. Jung, and T. Song, "Embedded MRAM Macro for eFlash Replacement" in IEEE ISCAS, pp. 1-4, 2018.

[64] A V Khvalkovskiy et al., "Basic principles of STT-MRAM cell operation in memory arrays," in Journal of Physics D: Applied Physics, 2013.

[65] Z. Diao et al., "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory" in Journal of Physics: Condensed Matter, 2007.

[66] A. Raychowdhury, D. Somasekhar, T. Karnik, and V. De, "Design space and scalability exploration of 1T-1STT MTJ memory arrays in the presence of variability and disturbances" in IEEE IEDM, pp. 1-4, 2009.

[67] E. Cheshmikhani, H. Farbeh, and H. Asadi, "A System-Level Framework for Analytical and Empirical Reliability Exploration of STT-MRAM Caches" in IEEE Transactions on Reliability, vol. 69, no. 2, pp. 594-610, 2020.

[68] T. Zheng, J. Park, M. Orshansky, and M. Erez, "Variable-energy write STT-RAM architecture with bit-wise write-completion monitoring" in ISLPED , pp. 229-234, 2013.

[69] S. Sakhare et al., "J_{SW} of 5.5 MA/cm^2 and RA of $5.2-\Omega.m^2$ STT- MRAM Technology for LLC Application," in IEEE Transactions on Electron Devices, vol. 67, no. 9, 2020, pp. 3618–3625.

[70] H. Li, M. Bhargav, P. N., and H. Philip, "On-Chip Memory Technology Design Space Explorations for Mobile Deep Neural Network Accelerators" in Design Automation Conference (DAC), pp. 1-6, 2019.

[71] I. Yoon, M. A., R. V., T. Rakshit, and A. Raychowdhury, "Hierarchical Memory System With STT-MRAM and SRAM to Support Transfer and Real-Time Reinforcement Learning in Autonomous Drones" in IEEE Journal on Emerging Topics, vol. 9, no. 3, pp. 485-497, Sept. 2019.

[72] C. W Smullen et al., "Relaxing non-volatility for fast and energy-efficient STT-RAM caches" in IEEE 17th International Symposium on High Performance Computer Architecture (HPCA), pp. 50-61, 2011.

[73] A. Ranjan, S. Venkataramani, Z. Pajouhi, R. Venkatesan, K. Roy, and A. Raghunathan, "STAxCache: An approximate, energy efficient STT-MRAM cache" in DATE, pp. 356-361, 2017.

[74] H.S.P. Wong et al., "Stanford Memory Trends" on https://nano.stanford.edu/downloads/stanford-memory-trends, 2020.

[75] G. Li et al., "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications" in ACM Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1-12, 2017.

[76] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K., N. Mulholland, D. Brooks, and G. Wei, "Ares: A framework for quantifying the resilience of deep neural networks" in Design Automation Conference (DAC), pp. 1-6, 2018.

[77] J. Park et al., "A novel integration of STT-MRAM for on-chip hybrid memory by utilizing non-volatility modulation" in IEEE International Electron Devices Meeting (IEDM), pp. 2.5.1-2.5.4, 2019.

[78] G. Hu et al., "Spin-transfer torque MRAM with reliable 2 ns writing for last level cache applications" in IEEE International Electron Devices Meeting (IEDM), pp. 2.6.1-2.6.4, 2019.

[79] J. Iwata-Harms et al., "High-temperature thermal stability driven by magnetization dilution in CoFeB free layers for spin-transfer-torque magnetic random access memory" in Nature Scientific Reports, 2018.

[80] J. G., "2 MB Array-Level Demonstration of STT-MRAM Process and Performance Towards L4 Cache Applications" in IEEE International Electron Devices Meeting (IEDM), pp. 2.4.1-2.4.4, 2019.

[81] M. Poremba, S. Mittal, D. Li, J. S., and Y. Xie, "DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches" in Design, Automation Test in Europe Conference Exhibition, pp. 1543-1546, 2015.

[82] S. Wang, and P. Kanwar, "BFloat16: The secret to high performance on Cloud TPUs" in Google Cloud Blog, 2019.

[83] D. Kalamkar et al., "A Study of BFLOAT16 for Deep Learning Training" in arXiv:1905.12322, 2019.

[84] Online, "PyTorch", Avalable in https://pytorch.org/, 2021.

[85] Online, "Synopsys Inc.", Available in https://www.synopsys.com/, 2021.

[86] B. Sun, D. Liu, L. Yu, J. Li, H. Liu, W. Zhang, and T. Torng, "Mram co-designed processing-in-memory cnn accelerator for mobile and iot applications" in arXiv preprint arXiv:1811.12179, 2018.

[87] H. Yan, H. Cherian, E. Ahn, X. Qian, and L. Duan, "iCELIA: a full-stack framework for STT-MRAM-based deep learning acceleration" in IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 2, pp. 408-422, 2019.

[88] A. Anwar, A. Raychowdhury, R. Hatcher, and T. Rakshit, "XBAROPT-Enabling ultra-pipelined, novel STT MRAM based processing-in-memory DNN accelerator" in IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp. 36-40, 2020.

[89] Y. Shi, S. Oh, Z. Huang, X. Lu, S. Kang, and D. Kuzum, "Performance prospects of deeply scaled spin-transfer torque magnetic random-access memory for in-memory computing" in IEEE Electron Device Letters, pp. 1126-1129, 2020.

[90] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic RAM" in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 3, pp. 470-483, 2017.

[91] S. Angizi, Z. He, A. Awad, and D. Fan, "MRIMA: An MRAM-Based In-Memory Accelerator" in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 5, pp. 1123-1136, 2020.

[92] M. Suri, A. Gupta, V. Parmar, and K. Lee, "Performance enhancement of edge-ai-inference using commodity MRAM: Iot case study" in 2019 IEEE 11th International Memory Workshop (IMW), pp. 1-4, 2019.

[93] N. Sayed, L. Mao, R. Bishnoi, and M. Tahoori, "Compiler-assisted and profiling-based analysis for fast and efficient STT-MRAM on-chip cache design" in ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 24, no. 41, Pages 1 - 25, 2019.

[94] N. Sayed, R. Bishnoi, and M. Tahoori. "Approximate spintronic memories." in ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 16, no. 43, Pages 1 - 22, 2020.

[95] Online, "NVIDIA Ampere100 GPU", Available in ttps://www.nvidia. com/en-us/data-center/ampere-architecture/, 2024.

[96] Q. Cao et al., "Are Mobile DNN Accelerators Accelerating DNNs?" in International Workshop on Embedded and Mobile Deep Learning, Pages 7 - 12, 2021.

[97] Y. Seo, K. Kwon, X. Fong, and K. Roy, "High Performance and Energy-Efficient On-Chip Cache Using Dual Port (1R/1W) Spin-Orbit Torque MRAM" in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, Pages 293-304, 2016.

[98] J. Park, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications" in arXiv preprint arXiv:1811.09886, 2018.

[99] K. Mishty, and M. Sadi, "Designing Efficient and High-Performance AI Accelerators With Customized STT-MRAM" in IEEE Transactions on Very Large Scale Integration Systems, vol. 29, Pages 1730-1742, 2021.

[100] T. Endoh, H. Honjo, K. Nishioka, and S. Ikeda, "Recent Progresses in STT-MRAM and SOT-MRAM for Next Generation MRAM" in 2020 IEEE Symposium on VLSI Technology, Pages 1-2, 2020.

[101] F. Oboril, R. Bishnoi, M. Ebrahimi, and M. Tahoori, "Evaluation of Hybrid Memory Technologies Using SOT-MRAM for On-Chip Cache Hierarchy" in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, Pages 367-380, 2015.

[102] M. Natsui et al., "Dual-Port Field-Free SOT-MRAM Achieving 90-MHz Read and 60-MHz Write Operations under 55-nm CMOS Technology and 1.2-V Supply Voltage" in IEEE Symposium on VLSI Circuits, pp. 1-2, 2020.

[103] K. Garello, F. Yasin, and G. Kar, "Spin-Orbit Torque MRAM for ultrafast embedded memories: from fundamentals to large scale technology integration" in IEEE International Memory Workshop (IMW), pp. 1-4, 2019.

[104] S.Z. Rahaman et al., "Size-Dependent Switching Properties of Spin-Orbit Torque MRAM With Manufacturing-Friendly 8-Inch Wafer-Level Uniformity" in IEEE Journal of the Electron Devices Society, vol. 8, Pages 163-169, 2020.

[105] M. Gupta et al., "High-density SOT-MRAM technology and design specifications for the embedded domain at 5nm node" in IEEE International Electron Devices Meeting, Pages 24.5.1-24.5.4, 2020.

[106] H. Honjo et al., "First demonstration of field-free SOT-MRAM with 0.35 ns write speed and 70 thermal stability under 400°C thermal tolerance by canted SOT structure and its advanced patterning/SOT channel technology" in IEEE International Electron Devices Meeting (IEDM), pp. 28.5.1-28.5.4, 2019.

[107] M. Kazemi et al., "Compact Model for Spin–Orbit Magnetic Tunnel Junctions" in IEEE Transactions on Electron Devices, vol. 63, no. 2, pp. 848-855, 2016.

[108] Online, "Hugging Face", Available in https://huggingface.co/, 2024.

[109] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L. Chen, B. Zhang, and P. Deaville, "In-Memory Computing: Advances and Prospects" in IEEE Solid-State Circuits Magazine, vol. 11, Pages 43-55, 2019.

[110] A. Manchonet et al., "Current-induced spin-orbit torques in ferromagnetic and antiferromagnetic systems" in APS, vol. 91, Pages 035004, 2019.

[111] A. Vaswani et al., "Attention is all you need" in Advances in Neural Information Processing Systems, vol. 30, 2017.

[112] Y. Seo, K. Kwon, X. Fong, and K. Roy, "High Performance and Energy-Efficient On-Chip Cache Using Dual Port (1R/1W) Spin-Orbit Torque MRAM" in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, Pages 293-304, 2016.

[113] H. Wang et al., "A New MRAM-Based Process In-Memory Accelerator for Efficient Neural Network Training with Floating Point Precision" in IEEE International Symposium on Circuits and Systems, pp. 1-5, 2020.

[114] G. Yuan, X. Ma, S. Lin, Z. Li, and C. Ding, "A SOT-MRAM-based processing-in-memory engine for highly compressed DNN implementation" in arXiv preprint arXiv:1912.05416, 2019.

[115] Y. Luo et al., "Performance Benchmarking of Spin-Orbit Torque Magnetic RAM for Deep Neural Network (DNN) Accelerators" in IEEE International Memory Workshop (IMW), Pages 1-4, 2022.

[116] K. Lee, S. Lee, B. Min, and K. Lee, "Threshold current for switching of a perpendicular magnetic layer induced by spin Hall effect" in American Institute of Physics, vol. 102, 2013.

[117] N. Khang, Y. Ueda, and P. Hai, "A conductive topological insulator with large spin Hall effect for ultralow power spin–orbit torque switching" in Nature Publishing Group, vol. 17, Pages 808–813, 2018.

[118] B. Wu et al., "Field-Free 3T2SOT MRAM for Non-Volatile Cache Memories" in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 67, Pages 4660-4669, 2020.

[119] K. Tsunekawa et al., "Giant tunneling magnetoresistance effect in low-resistance CoFeB/ MgO (001)/ CoFeB magnetic tunnel junctions for read-head applications" in American Institute of Physics, vol. 87, 2005.

[120] K. Wang, J. Alzate, and P. Amiri, "Low-power non-volatile spintronic memory: STT-RAM and beyond" in IOP Publishing, vol. 46, 2013.

[121] K. Garello et al., "Ultrafast magnetization switching by spin-orbit torques" in AIP Publishing LLC, vol. 105, 2014.

[122] Y.C. Wu et al., "Voltage-gate-assisted spin-orbit-torque magnetic random-access memory for high-density and low-power embedded applications" in APS, vol. 15, 2021.

[123] K. Garello et al., "SOT-MRAM 300mm integration for low power and ultrafast embedded memories" in IEEE Symposium on VLSI Circuits, Pages 81-82, 2018.

[124] K. Garello et al., "Manufacturable 300mm platform solution for Field-Free Switching SOT-MRAM" in Symposium on VLSI Technology, pp. T194-T195, 2019.

[125] L. Chang, X. Ma, Z. Wang, Y. Zhang, Y. Xie, and W. Zhao, "PXNOR-BNN: In/With Spin-Orbit Torque MRAM Preset-XNOR Operation-Based Binary Neural Networks" in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, Pages 2668-2679, 2019.

[126] S. Angizi, Z. He, A. Rakin, and D. Fan, "CMP-PIM: An Energy-Efficient Comparator-based Processing-In-Memory Neural Network Accelerator" in 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp. 1-6, 2018.

[127] S. Naffziger et al., "2.2 AMD chiplet architecture for high-performance server and desktop products" in 2020 IEEE International Solid-State Circuits Conference-(ISSCC), Pages 44-45, 2020.

[128] M.-S. Lin et al., "A 7nm 4GHz Arm®-core-based CoWoS® Chiplet Design for High Performance Computing" in 2019 Symposium on VLSI Circuits, Pages C28-C29, 2019.

[129] C.-H.H. Kenny, "Designs of Communication Circuits for Side-by-Side and Stacked Chiplets" in ISSCC 2021 Forums, 2021.

[130] R. Mahajan et al., "Embedded Multi-die Interconnect Bridge (EMIB) – A High Density, High Bandwidth Packaging Interconnect" in IEEE 66th Electronic Components and Technology Conference, pp. 557-565, 2016.

[131] D. B. Ingerly et al., "Foveros: 3D Integration and the use of Face-to-Face Chip Stacking for Logic Devices" in IEEE International Electron Devices Meeting, pp. 19.6.1-19.6.4, 2019.

[132] F. J. C. Lee et al., "Heterogeneous System-Level Package Integration — Trends and Challenges" in IEEE Symposium on VLSI Technology, Pages 1-2, 2020.

[133] F.C. Chen et al., "System on integrated chips (SoIC^{TM}) for 3d heterogeneous integration," in IEEE 69th Electronic Components and Technology Conference, pp. 594-599, 2019.

[134] Online, "Heterogeneous Integration Roadmap", Available in https://eps.ieee.org/technology/heterogeneous-integration-roadmap/2021-edition.html, 2021.

[135] Online, "High-Bandwidth Memory(HBM)", Available in https://www.jedec.org/, 2023.

[136] Online, "TSMC", Available in https://3dfabric.tsmc.com/english/dedicatedFoundry/technology/cowos.htm, 2024.

[137] Online, "Intel's new 3D Foveros packaging tech: LEGO-like chiplets for CPUs", Available in https://www.tweaktown.com/news/index.html, 2022.

[138] S. Naffziger et al., "Ryzen" in 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture, vol. , Pages 57-70, 2021.

[139] W. Gomes et al., "8.1 Lakefield and Mobility Compute: A 3D Stacked 10nm and 22FFL Hybrid Processor System in $1212mm^2$, 1mm Package-on-Package" in 2020 IEEE International Solid- State Circuits Conference, Pages 144-146, 2020.

[140] Hong Jiang, "Intel's Ponte Vecchio GPU" in 2022 IEEE Hot Chips 34 Symposium, vol. , Pages 1-29, 2022.

[141] T. Wang et al., "Application Defined On-chip Networks for Heterogeneous Chiplets: An Implementation Perspective" in IEEE International Symposium on High-Performance Computer Architecture, Pages 1198-1210, 2022.

[142] Y. Wu et al., "Upward Packet Popup for Deadlock Freedom in Modular Chiplet-Based Systems" in IEEE International Symposium on High-Performance Computer Architecture, Pages 986-1000, 2022.

[143] A. Coskun et al., "Cross-Layer Co-Optimization of Network Design" in IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, vol. 39, no. 12, pp. 5183-5196, 2022.

[144] T.-R. Lin et al., "A Deep Reinforcement Learning Framework for Architectural Exploration: A Routerless NoC Case Study," in IEEE International Symposium on High Performance Computer Architecture, pp. 99–110, 2020.

[145] A. Kumar et al., "Data-driven offline optimization for architecting hardware accelerators." arXiv preprint arXiv:2110.11346, 2021.

[146] Dan et al. Zhang, "A Full-Stack Search Technique for Domain Optimized Deep Learning Accelerators" in ASPLOS, Pages 27-42, 2022.

[147] S. Krishnan et al., "Multi-Agent Reinforcement Learning for Microprocessor Design Space Exploration" in arXiv preprint arXiv:2211.16385, 2022.

[148] J.A. Cunningham, "The use and evaluation of yield models in integrated circuit manufacturing" in IEEE Transactions on Semiconductor Manufacturing, vol. 3, Pages 60-71, 1990.

[149] Y. Feng, and K. Ma, "Chiplet Actuary: A Quantitative Cost Model and Multi-Chiplet Architecture Exploration" in Proceedings of the 59th ACM/IEEE Design Automation Conference, Pages 121-126, 2022.

[150] V. Sze et. al., "How to Evaluate Deep Neural Network Processors: TOPS/W (Alone) Considered Harmful" in IEEE Solid-State Circuits Magazine, vol. 12, Pages 28-41, 2020.

[151] S. Bharadwaj et al., "Kite: A Family of Heterogeneous Interposer Topologies Enabled via Accurate Interconnect Modeling" in 57th ACM/IEEE Design Automation Conference (DAC), Pages 1-6, 2020.

[152] S. Pal et al., "Designing a 2048-Chiplet, 14336-Core Waferscale Processor" in 58th ACM/IEEE Design Automation Conference (DAC), pp. 1183-1188, 2021.

[153] T. Tang, and Y. Xie, "Cost-Aware Exploration for Chiplet-Based Architecture with Advanced Packaging Technologies" in arXiv preprint arXiv:2206.07308, 2022.

[154] R. Sutton, and A. Barto, "Reinforcement learning: An introduction" in MIT press, 2018.

[155] N. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit" in Proceedings of the 44th annual international symposium on computer architecture, Pages 1-12, 2017.

[156] R. Mathur et al., "Thermal-aware design space exploration of 3-D systolic ML accelerators," in IEEE Journal on Exploratory Solid-State Computational Devices and Circuits, vol. 7, no. 1, pp. 70–78, 2021.

[157] H. T. Kung et al., "Systolic Building Block for Logic-on-Logic 3D-IC Implementations of Convolutional Neural Networks" in IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, 2019.

[158] G. Loh, and R. Swaminathan, "The Next Era for Chiplet Innovation" in Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1-6, 2023.

[159] P. Shukla et al., "Temperature-Aware Sizing of Multi-Chip Module Accelerators for Multi-DNN Workloads" in Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1-6, 2023.

[160] P. Vivet et al., "IntAct: A 96-Core Processor With Six Chiplets 3D-Stacked on an Active Interposer With Distributed Interconnects and Integrated Power Management" in IEEE Journal of Solid-State Circuits, vol. 56, Pages 79-97, 2021.

[161] D. C. Price e. al., "Optimizing performance-per-watt on GPUs in high performance computing: Temperature, frequency and voltage effects" in Springer, vol. 31, Pages 185–193, 2016.

[162] J. Schulman et al., "Proximal policy optimization algorithms" in arXiv preprint arXiv:1707.06347, 2017.

[163] Online, "Stable-Baselines3", Available in https://stable-baselines3.readthedocs.io/en/master/, 2024.

[164] N. Jouppi et al., "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings" in Proceedings of the 50th Annual International Symposium on Computer Architecture, Pages 1-14, 2023.

[165] Online, "AMD 3D V-Cache Technology", Available in https://www.amd.com/, 2024.

[166] Online, "OpenAI Gym", Available in https://github.com/openai/gym, 2024.

[167] V. Mnih, A. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning" in International conference on machine learning, Pages 1928–1937, 2016.

[168] Dzmitry Bahdanau, "The FLOPs Calculus of Language Model Training", Availble in https://medium.com/, 2022.

[169] Online, "MLPerf Benchmark", Available in https://www.nvidia.com/en-us/data-center/resources/mlperf-benchmarks/, 2024.

[170] A. Sangiovanni-Vincentelli et al., "Automated Design of Chiplets" in International Symposium on Physical Design, Pages 1-8, 2023.

[171] K. Hazelwood et al., "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective" in IEEE International Symposium on High Performance Computer Architecture (HPCA), Pages 620-629, 2018.

[172] Huwan et al. Peng, "Chiplet Cloud: Building AI Supercomputers for Serving Large Generative Language Models" in arXiv preprint arXiv:2307.02666, 2023.

[173] S. Chen et al., "Floorplet: Performance-Aware Floorplan Framework for Chiplet Integration," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 43, no. 6, pp. 1638-1649, June 2024.

[174] Y. Chang et al., "A Survey on Evaluation of Large Language Models" in ACM Trans. Intell. Syst. Technol., vol. 15, Pages 1-45, 2024.

[175] Y. Kim, and C. Wu, "Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning" in 53rd Annual IEEE/ACM international symposium on microarchitecture (MICRO), Pages 1082–1096, 2020.

[176] K. Settaluri et al., "AutoCkt: Deep reinforcement learning of analog circuit designs" in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Pages 490–495, 2020.

[177] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation" in International Conference on Machine Learning, vpp. 8821-8831, 2021.

[178] K. Mishty and M. Sadi, "System and Design Technology Co-Optimization of SOT-MRAM for High-Performance AI Accelerator Memory System," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 43, no. 4, pp. 1065-1078, April 2024.

[179] K. Mishty, and M. Sadi, "System and Design Technology Co-optimization of Chiplet-based AI Accelerator with Machine Learning" in Proceedings of the Great Lakes Symposium on VLSI, Pages 697 - 702, 2023.

[180] E. Marinissen, T. McLaurin, and H. Jiao, "IEEE Std P1838: DfT standard-under-development for 2.5D-, 3D-, and 5.5D-SICs" in 21th IEEE European Test Symposium (ETS), pp. 1-10, 2016.

[181] J. M. Joseph et al., "Architecture, dataflow and physical design implications of 3D-ICs for DNN-accelerators" in 22nd International Symposium on Quality Electronic Design , pp. 60-66, 2021.

[182] Online, "AMD MI300", Available in Available: https://www.amd.com/en.html, 2024.

[183] J. Kim et al., "Architecture, Chip, and Package Codesign Flow for Interposer-Based 2.5-D Chiplet Integration Enabling Heterogeneous IP Reuse" in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, Pages 2424-2437, 2020.

[184] S. W. Liang ett al., "High Performance and Energy Efficient Computing with Advanced SoIC™ Scaling" in 2022 IEEE 72nd Electronic Components and Technology Conference (ECTC), Pages 1090-1094, 2022.

[185] Q. Xu and L. Jiang, "On Effective Through-Silicon Via Repair for 3-D-Stacked ICs" in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 4, pp. 559-571, April 2013.

[186] H. Kwon et al., "MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings" in IEEE Micro, vol. 40, Pages 20-29, 2020.

Publications

- K. Mishty and M. Sadi, "Chiplet-Gym: Optimizing Chiplet-based AI Accelerator Design with Reinforcement Learning," in IEEE Transactions on Computers, 2024. doi: 10.1109/TC.2024.3457740
- K. Mishty and M. Sadi, "System and Design Technology Co-Optimization of SOT-MRAM for High-Performance AI Accelerator Memory System," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 43, no. 4, pp. 1065-1078, April 2024
- K. Mishty and M. Sadi, "System and Design Technology Co-optimization of Chiplet-based AI Accelerator with Machine Learning," In Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23). Association for Computing Machinery, 697–702, 2023.
- K. Mishty and M. Sadi, "Designing Efficient and High-Performance AI Accelerators With Customized STT-MRAM," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 10, pp. 1730-1742, Oct. 2021
- Knipper, R. Alexander, K. Mishty, M. Sadi, SKK Santu, "SNNLP: energy-efficient natural language processing using spiking neural networks." arXiv preprint arXiv:2401.17911 (2024).
- M. Sadi, B.M.S.B. Talukder, K. Mishty, M.T. Rahman, "Attacking Deep Learning AI Hardware with Universal Adversarial Perturbation," in Information 2023, 14, 516.