

Energy-Efficient In-Memory Acceleration of Deep Neural Networks Through a
Hardware-Software Co-Design Approach

by

Gokul Krishnan

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved August 2022 by the
Graduate Supervisory Committee:

Yu Cao, Chair
Jae-sun Seo
Chaitali Chakrabarti
Umit Y. Ogras

ARIZONA STATE UNIVERSITY

December 2022

ABSTRACT

Deep neural networks (DNNs), as a main-stream algorithm for various AI tasks, achieve higher accuracy at the cost of increased computational complexity and model size, posing great challenges to hardware platforms. This dissertation first tackles the design challenges of resistive random-access-memory (RRAM) based in-memory computing (IMC) architectures. A new metric, model stability from the loss landscape, is proposed to help shed light on accuracy under variations and model compression and guide a novel variation-aware training (VAT) solution. The proposed method effectively improves post-mapping accuracy of multiple datasets. Next, a hybrid RRAM/SRAM IMC DNN inference accelerator is developed, that integrates an RRAM-based IMC macro, a reconfigurable SRAM-based multiply-accumulate (MAC) macro, and a programmable shifter. The hybrid IMC accelerator fully recovers the inference accuracy post the mapping. Furthermore, this dissertation researches on architectural optimizations for high IMC utilization, low on-chip communication cost, and low energy-delay product (EDP), including on-chip interconnect design, PE array utilization, and tile-to-router mapping and scheduling. The optimal choice of on-chip interconnect results in up to $6\times$ improvement in energy-delay-area product for RRAM IMC architectures. Furthermore, the PE and NoC optimizations show up to 62% improvement in PE utilization, 78% reduction in area, and 78% lower energy-area product for a wide range of modern DNNs. Finally, this dissertation proposes a novel chiplet-based IMC benchmarking simulator, SIAM, and a heterogeneous chiplet IMC architecture to address the limitations of a monolithic DNN accelerator. SIAM utilizes model-based and cycle-accurate simulation to provide a scalable and flexible architecture. SIAM is calibrated against a published silicon result, SIMBA, from Nvidia. The heterogeneous architecture utilizes a custom mapping with a bank of big and little

chipelets, and a hybrid network-on-package (NoP) to optimize the utilization, interconnect bandwidth, and energy efficiency. The proposed big-little chiplet-based RRAM IMC architecture significantly improves energy efficiency at lower area, compared to conventional GPUs. In summary, this dissertation comprehensively investigates novel methods that encompass device, circuits, architecture, packaging, and algorithm to design scalable high-performance and energy-efficient IMC architectures.

ACKNOWLEDGMENTS

I owe many thanks to my wife - Greeshma Sasikumar, my parents - Koumudi V.M and Krishnaprasad K, and my sister - Gayathri Krishnaprasad, for their unwavering support and love throughout my Ph.D. education. It is to them I dedicate this dissertation. I would like to sincerely thank my advisor and committee chair, Dr. Yu Cao, for developing my research skills, paper writing, time management and provide valuable guidance and encouragement throughout my Ph.D. studies. I appreciate his patience and assistance in guiding the research projects and giving his insightful recommendations and ideas at each step of the work. It has been such a pleasure and honor to be one of his students and learn from him.

I want to extend my gratitude to Dr. Jae-sun Seo, Dr. Chaitali Chakrabarti, and Dr. Umit Y. Ogras for their guidance in my research projects, comments and suggestions on my dissertation and for taking the time to serve as committee members. I thank Dr. Rajiv V. Joshi from IBM Corporation, Prof. Nathaniel C. Cady from SUNY Polytechnic, Prof. Deliang Fan from ASU and Michael Dreesen from Apple Inc for their collaborative research opportunity and invaluable inputs during my research work and summer internships.

I am indebted to my colleagues, Deepak Vinayak Kadetotad, Shihui Yin, Shreyas VK, Xiacong Du, Zheng Li, Gopikrishnan Nair, Kishore K, Abinash Mohanty, Yufei Ma, Sumit K. Mandal, Anish N.K, Zhenyu Wang, Ganapathi Bhat, and Injune Yeo for their invaluable contribution, discussions, and support throughout my study. I must also thank my graduate advisor Toni Mengert for her active help with all the administrative procedures. Thanks to my friends, Rohit K, Vasundhara D, Sooraj S, Sarath Omanakuttan, Bosco Paul, Gokul Krishnakumar, and many others, who were also crucial in the successful realization of this dissertation.

TABLE OF CONTENTS

	Page
LIST OF TABLES	x
LIST OF FIGURES	xiii
CHAPTER	
1 INTRODUCTION	1
1.1 Reliable In-Memory Computing	1
1.2 In-Memory Computing Architectures and Optimizations	4
1.3 IMC Chiplet Architecture and Benchmarking Simulator	6
1.4 Thesis organization	8
2 RELIABLE RRAM-BASED IN-MEMORY COMPUTING THROUGH MODEL STABILITY	10
2.1 Introduction	10
2.2 DNN Model Stability	14
2.2.1 Landscape visualization	15
2.2.2 Roughness Score	15
2.2.3 Roughness Score and DNN Accuracy	16
2.3 65nm 1T1R RRAM Device	17
2.4 Cross-Layer Simulation Framework	20
2.4.1 Device Models	20
2.4.2 Circuit Estimator	21
2.4.3 Architecture	21
2.4.4 Algorithm	22
2.4.4.1 Pruning	23
2.4.4.2 Quantization	23

CHAPTER	Page
2.5 Model Stability for RRAM-based IMC	24
2.6 Experiments and Results.....	26
2.6.1 Experimental Setup	26
2.6.2 Pruning and Quantization.....	27
2.6.2.1 Effect of Pruning and Quantization on RRAM IMC ..	27
2.6.3 System-Level IMC Analysis	29
2.6.3.1 Precision and Variation	29
2.6.3.2 Roofline Model	30
2.6.4 Model Stability-based Model Selection	31
2.6.5 Model Stability-based VAT.....	32
2.6.5.1 Overall Results with Proposed VAT.....	35
2.6.5.2 Comparison with Other Work	36
2.7 Conclusion.....	37
3 HYBRID RRAM/SRAM IN-MEMORY COMPUTING FOR ROBUST DNN ACCELERATION	38
3.1 Introduction	38
3.2 Hybrid IMC Architecture	42
3.2.1 RRAM IMC Macro.....	43
3.2.2 SRAM+MAC Engine Macro	44
3.2.3 Programmable Shifter	46
3.2.4 65nm Hybrid IMC Test Chip.....	47
3.2.4.1 RRAM IMC Macro	48
3.2.4.2 SRAM Macro	49
3.3 Hybrid IMC Training Framework.....	50

CHAPTER	Page
3.3.1 Statistical RRAM Device Models	51
3.3.2 Training for Hybrid IMC Architecture	53
3.3.2.1 RRAM Macro Training	53
3.3.2.2 Training SRAM Macro and Programmable Shifter ...	54
3.4 Experiments and Results.....	56
3.4.1 Effect of different Scales of SRAM Compensation	57
3.4.2 Optimal Shifter Configuration	58
3.4.3 Pruning Analysis of SRAM.....	61
3.4.4 Overall Accuracy Results	62
3.4.5 Evaluation with $2\times$ RRAM Variation	63
3.4.6 SRAM Macro and Shifter Overhead Analysis	64
3.4.7 Comparison with Other Work	68
3.5 Conclusion.....	69
4 IMPACT OF ON-CHIP INTERCONNECT ON IN-MEMORY ACCEL- ERATION OF DEEP NEURAL NETWORKS	72
4.1 Introduction	72
4.2 Interconnect Network	78
4.3 Simulation Framework	80
4.3.1 Circuit-level Simulator: Customized NeuroSim.....	80
4.3.2 Interconnect Simulator: Customized BookSim	82
4.4 Analytical Performance Models for NoCs in IMC Architecture	84
4.5 Connection-centric Architecture	87
4.5.1 Design Space Exploration	88
4.5.2 Hardware Architecture.....	90

CHAPTER	Page
4.6 Experiments and Results.....	91
4.6.1 Experimental Setup	91
4.6.2 Evaluation of NoC Analytical Model	92
4.6.3 Analysis on Traffic Congestion in NoC	94
4.6.4 Guidance on Optimal Choice of Interconnect	96
4.6.4.1 Empirical Analysis.....	96
4.6.4.2 Theoretical Analysis	100
4.6.5 Comparison with state-of-the-art architectures.....	102
4.6.6 Connection Density and Hardware Performance.....	103
4.7 Conclusion.....	104
5 INTERCONNECT-AWARE AREA AND ENERGY OPTIMIZATION FOR IN-MEMORY ACCELERATION OF DNNS	105
5.1 Introduction	105
5.2 Overview of the Proposed Methodology	108
5.3 Area and Energy Optimization methodology	111
5.4 Experimental Evaluation.....	115
5.4.1 Experimental Setup	115
5.4.2 Area-Aware Optimization for Heterogeneous Tiles.....	116
5.4.3 Energy-Aware NoC Optimization	117
5.4.4 Overall Improvement	119
5.4.5 Comparison with State-of-the-Art Architectures	119
5.5 Discussion and Conclusion	120
6 SIAM: CHIPLET-BASED SCALABLE IN-MEMORY ACCELERA- TION WITH MESH FOR DEEP NEURAL NETWORKS	122

CHAPTER	Page	
6.1	Chiplet-based IMC Architecture	127
6.2	SIAM Simulator	129
6.2.1	Overview	130
6.2.2	Partition and Mapping Engine	131
6.2.3	Circuit and NoC Engine	135
6.2.3.1	Circuit Estimator	135
6.2.3.2	NoC Estimator	137
6.2.4	NoP Engine	138
6.2.5	DRAM Engine	141
6.3	SIAM Dataflow	144
6.4	Experimental Evaluation	145
6.4.1	Experimental Setup	146
6.4.2	Custom and Homogeneous Chiplet-based IMC Design	147
6.4.2.1	IMC Crossbar Utilization	148
6.4.2.2	IMC Chiplet Performance Breakdown	148
6.4.2.3	NoP and NoC Performance Trade-offs	150
6.4.2.4	Overall Hardware EDAP and Area	152
6.4.3	Comparison between Monolithic and Chiplet-based IMC Architectures	153
6.4.4	Calibration with SIMBA	154
6.4.5	Comparison with GPUs	156
6.4.6	Simulation Time	157
6.5	Conclusion and Discussion	158

CHAPTER	Page
7 BIG-LITTLE CHIPLETS FOR IN-MEMORY ACCELERATION OF DNNs: A SCALABLE HETEROGENEOUS ARCHITECTURE	161
7.1 Big-Little Chiplet Architecture	165
7.2 Parameters of the Big-Little Architecture and Mapping	168
7.2.1 Configuration of the big-little chiplets	168
7.2.2 Configuration of the big-little NoP	170
7.2.3 Mapping a Previously Unseen DNN to a System on big-little Chiplets	173
7.3 Experimental Evaluation	175
7.3.1 Experimental Setup	175
7.3.2 Big-Little IMC Structure and NoP	176
7.3.3 Comparison with Baseline Architectures with Homogeneous Chiplets	179
7.3.4 Results with DRAM (DDR4)	181
7.3.5 Comparison with State-of-the-art Work	182
7.4 Conclusion	183
8 CONCLUSION	185
REFERENCES	189

LIST OF TABLES

Table	Page
1. Endurance Measurement up to 1 Billion Cycles.....	18
2. Device Models From 65nm 1T1R RRAM Device.....	19
3. Roughness Score and Post-Mapping Accuracy for Different DNN Models Under RRAM Variation and a 5-bit ADC. A More Stable Model (Lower Roughness Score) Effectively Improves the Accuracy, More Than That by Reducing RRAM Variation by 50% Only.	31
4. Post-Mapping Accuracy and Improvement for VGG-16 on CIFAR-10 (82.3% Sparsity and Ternary Precision). Conventional Method: Same Training and Testing Variation.....	34
5. Comprehensive Results Using the Proposed Model Stability-based VAT Method. Conventional Method: Same Training and Testing Variation	35
6. Post-Mapping Accuracy of VGG-16 on CIFAR-10	36
7. RRAM Device Variation for Different Bit	53
8. Comprehensive Evaluation of the Choice of Programmable Shifter Across Different DNNs for CIFAR-10 Dataset. (*At Same RRAM and Overall Activation Precision).VAT – Variation-Aware Training Long <i>et al.</i> (2019). ..	57
9. Comprehensive Evaluation of the Post-Mapping Accuracy Across Different DNNs and Datasets with the Proposed Hybrid IMC Architecture. SRAM is Pruned With a Group Size of 4. (*Top-1 Accuracy, **At Same RRAM Precision).....	59
10. Post-Mapping Accuracy and SRAM Macro Pruning Ratio for 1x and 2x RRAM Variation. We Use a Pruning Group Size of 4.	63

Table	Page
11. Comparison of Post-Mapping Accuracy with State-Of-The-Art Methods. Ours-A SRAM Macro Weights Pruned; Ours-B#: SRAM Macro Weights Not Pruned. (**Precision Not Reported In the Manuscript)	66
12. Summary Of Notations	76
13. Summary Of Design Parameters	92
14. Mean Absolute Percentage Deviation (<i>MAPD</i>) of Worst-Case NoC Latency From Average NoC For Different DNNs.	96
15. Inference Performance Results for VGG-19. *Reported in Qiao <i>et al.</i> (2018)	103
16. Summary Of Notation	110
17. Inference Performance Results for VGG-19. *Reported in Qiao <i>et al.</i> (2018)	120
18. Definition Of the User Inputs to SIAM	130
19. Simulation Time for SIAM	157
20. Set Of Configurations Considered to Determine Big-Little Chiplet and NoP Structure.	176
21. Performance Comparison of Each Component of a Homogeneous (Little Only, Big Only) Chiplet Architecture and the Heterogeneous Big-Little IMC Chiplet Architecture for VGG-19 on CIFAR-100.	177
22. Performance Comparison of a Homogeneous (Little Only, Big Only) Chiplet Architecture and the Heterogeneous Big-Little IMC Chiplet Architecture For Different DNNs.	179
23. Ratio Between DRAM Energy and Compute Energy for VGG-16 and VGG- 19 With Systems Having Different Number of Chiplets (**All Weights of VGG-19 Fit On Chip With This Configuration, Significantly Reducing the DRAM Energy).	182

Table	Page
24. Comparison With Other Platforms for ResNet-50 on ImageNet (*Reported in Shao <i>et al.</i> (2019)).	182

LIST OF FIGURES

Figure	Page
1. Post-Mapping Accuracy for ResNet-20 on CIFAR-10 (a) Across Different RRAM Levels and Average RRAM Variations. For a Given $R_{\text{off}}/R_{\text{on}}$ Ratio, Higher RRAM Levels Suffer From Variations and Thus Lower Accuracy. Simultaneously, a Lower Average RRAM Variation Results in Higher Accuracy, (b) At 8-bit and Ternary-Bit Precision, with 29% Sparsity Yang <i>et al.</i> (2020). Model is Trained and Tested with the Same Variation (σ) Long <i>et al.</i> (2019); Charan <i>et al.</i> (2020a). The 8-bit Model has More Accuracy Loss Than the Ternary Model As σ Increases.	11
2. A Lower Roughness Score Leads to a Smoother Loss Landscape, Higher Stability and Thus, Higher Model Accuracy.....	16
3. HRS (Left) and LRS (Right) Cycle-To-Cycle Switching Variation Across the 300mm Wafer at 65nm. HRS State has a Higher Variation Than LRS. .	17
4. Normalized Variation for Different $R_{\text{off}}/R_{\text{on}}$ Ratios for 1110 1T1R RRAM Devices at 65nm. A Higher $R_{\text{off}}/R_{\text{on}}$ Ratio Leads to Higher Variation.....	18
5. The 65nm RRAM Device Achieves Good Retention for Up to 10^5 Seconds. The Grey Region Shows the Dominating Effect of Static Variations for the 65nm 1T1R RRAM Device.	19
6. The Benchmarking Tool Incorporates Algorithm (Pruning and Quantization), RRAM IMC Architecture Properties, Circuit Models, and the 65nm 1T1R RRAM Device Models. The Tool Outputs Post-Mapping Accuracy, Hardware Performance, Roofline Model, and Model Stability.....	20
7. Normalized Energy-Delay Product (EDP) at Different Sparsity for ResNet-20 on CIFAR-10 at 8-bit and Ternary Precision.	27

Figure	Page
8. Loss Landscape for Floating-Point 32-bit, Pruned Only (29%), and Pruned and Quantized (29% and 8-bit) Variants of ResNet-20 on CIFAR-10. Model Pruning Results in a Model with Lower Stability and Accuracy, while Quantization to a Low-Precision Model Improves the Stability and the Roughness Score, Leading to Higher Accuracy.	28
9. ADC Dominates the Energy Consumption, Especially Under High ADC Precision. With Higher RRAM Levels, its Portion Reduces Due to Reduced Number of Crossbars and Associated Peripherals.....	29
10. ADC Precision and RRAM Device Variations Govern the Maximum and Realistic Post-Mapping Accuracy. A More Stable DNN Model Improves the Realistic Post-Mapping Accuracy.	31
11. Loss Landscape for ResNet-20 on CIFAR-10 Trained With Different Device Variations at 29% Sparsity and 8-bit Quantization, Tested at 0.1 Variation. The Optimal Scale of Training Variation (σ) of 0.15 has the Lowest Roughness Score and Smoother Loss Landscape and Hence, Higher Model Stability and Post-Mapping Accuracy.	33
12. Post-Mapping Accuracy for ResNet-20 on CIFAR-10 at 29% Sparsity and 8-bit Precision for Three Different RRAM Variations Under Test.The Optimal Scale of Training Variation (Green Circle) has the Lowest Roughness Score and Highest Post-Mapping Accuracy.	33
13. Post-Mapping Accuracy for VGG-16 on CIFAR-10 at 82.3% Sparsity and Ternary Precision for Three Different RRAM Testing Variations (σ). The Optimal Model (Green Circle) has the Lowest Roughness Score Resulting in Higher Accuracy.	34

Figure	Page
14. Accuracy with RRAM IMC Macro for Three Different DNNs for Both CIFAR-10 and ImageNet Datasets. The Baseline Model Deals with a 32-bit Floating-Point Model, Quantization Refers to Fixed-Point Precision for Activation and Weights (e.g., 3W3A Means 3-bit Weight and 3-bit Activation), and VAT Refers to Variation-Aware Training with the RRAM Variations Long <i>et al.</i> (2019).	39
15. Block Diagram of the Proposed Hybrid RRAM/SRAM IMC Architecture. Both the RRAM and SRAM Macros Compute in a Parallel Manner to Generate the Output. A Programmable Shifter Allows for Different Scales of Compensation Using the SRAM Macro. The Overall Output is an Ensemble of the RRAM and SRAM Macro Outputs.	43
16. Block diagram of the RRAM IMC Macro Within the Hybrid Architecture. The Macro Consists of a Crossbar Array of RRAM Cells, a Decoder, PMOS Headers, Column Multiplexers (MUX), BL and SL Mux, Shift and Add Circuit, and ADC.	44
17. Block diagram of the SRAM Macros Within the Hybrid Architecture. An SRAM Array Stores the Weights While an Array of Processing Elements (PE) Perform the MAC Operations.	45
18. Functioning of the Programmable Shifter Within the Hybrid IMC Architecture. (a) 1-bit SRAM Macro Output Compensates for the MSB in the Blue Region, LSB in the Grey Region (1-bit Shift), and Adds an Extra Bit for Increased Precision in the Purple Region. A Similar Operation is Performed for Both (b) 2-bit and (c) 3-bit SRAM Macro Outputs.	46

Figure	Page
19. Layout of the 65nm Test-Chip of the Proposed Hybrid IMC Architecture. The Prototype Consists of a 64×64 1T1R RRAM Crossbar Array and Associated Peripheral Circuits. In Addition, the Prototype Implements an SRAM Memory Array of Size 32x64 with a 16-Word Size and 16 MAC Engines that Utilize an Output Stationary Dataflow.	47
20. HRS (Left) and LRS (Right) Cycle-to-Cycle Switching Variation Across the 300mm Wafer at 65nm Liehr <i>et al.</i> (2020). HRS State has a Higher Variation Than LRS.	51
21. Box and Whisker Plot Showing the Eight Distinct Resistance Levels (6 LRS and 2 HRS) Within our 65nm 1T1R RRAM Device. Different Compliance Currents Lead to Different Resistance Levels.	52
22. Post-Mapping Accuracy with the Proposed Hybrid IMC Architecture for ResNet-20 on CIFAR-10. A Higher SRAM Macro Precision Leads to Better Compensation Across Different RRAM Precision.	56
23. Post-Mapping Accuracy for ResNet-20 on CIFAR-10 at Different RRAM Macro and SRAM Macro Precision. (a) Post-Mapping Accuracy with SRAM Macro at G_{prune} of 16 and (b) Pruning Ratio of the SRAM Macro Weights Across Different RRAM and SRAM Precision at G_{prune} of 16, (c) Post-Mapping Accuracy for a G_{prune} of 4 for the SRAM Macro Weights, and (d) Pruning Ratio of the SRAM Macro Weights at G_{prune} of 4. A Lower Group Size Leads to Higher Pruning at the Same Accuracy.	59

Figure	Page
24. (a) Training Time Overhead for the SRAM Macro and the Shifter. The Proposed Hybrid IMC Architecture Incurs at Most 25% (Compared to Time of RRAM Macro) Overhead in Training Time. (b) Memory Overhead for the SRAM Macro Compared to the RRAM Macro. The Hybrid IMC Architecture Incurs at Most 24% Overhead (Compared to the Memory of RRAM Macro).	64
25. (a) Area Overhead and (b) Power Overhead of the SRAM Macro as a Percentage of the Total RRAM Macro Area and Power. The Proposed Hybrid IMC Architecture Results in a Very Low Overhead with Up to 20% Area and 2.6% Power Overhead for State-of-the-Art Accuracy Across Different DNNs.	65
26. Connection Density of Different DNNs for Three Different Datasets. Each Output Feature Map (Convolution Layer) and Neural Unit (FC Layer) Represent a Neuron. Larger Markers Represent Higher Accuracy.	73
27. Different Types of DNN Structures and Their Representative Connection Density.	74
28. Contribution of Routing Latency to Total Latency for Different DNNs for a P2P-based IMC Architecture Chen <i>et al.</i> (2018). With Increase in Connection Density, Routing Latency Becomes the Bottleneck for Performance.	74
29. Multi-Tiled IMC Architecture with Routing Architectures Based on (a) P2P Network, (b) NoC-tree, (c) NoC-mesh. NoC-tree is a P2P Network with Routers at Junctions.	78

Figure	Page
30. Comparison of Average Latency Among P2P, NoC-tree, and NoC-mesh Interconnect for Different Injection Bandwidth Jiang <i>et al.</i> (2013). NoC Topologies Show Better Scalability Than P2P Interconnect.	78
31. Block-Level Representation of the Proposed Architecture Simulator.	81
32. Tile Numbering and Placement While Mapping the DNN to the IMC Architecture. The Red Arrows show the Flow of the Data Across the Tiles.	82
33. Throughput Comparison for Three Interconnect Topologies (P2P, NoC-tree, and NoC-mesh) for SRAM-based IMC Architecture, Normalized to P2P, for Different DNNs. NoC Shows Superior Performance and Scalability than P2P-Based Network.	87
34. Comparison of Energy-Delay-Area Product (EDAP) of NoC-tree, NoC-mesh, and C-mesh for Different DNNs.....	89
35. NoC-based Heterogeneous Interconnect IMC Architecture. A Three-Level Interconnect Scheme Consisting of NoC (Tree or Mesh) Between Tiles, P2P Network Between CEs, and Bus Between PEs.	90
36. Accuracy of NoC Analytical Model for NoC-mesh and NoC-tree With Respect to Cycle-Accurate Simulator Jiang <i>et al.</i> (2013).	93
37. Speed-up (in NoC Simulation) with NoC Analytical Models with Respect to Cycle-Accurate NoC Simulation for Different DNNs with Mesh-NoC.	93
38. Percentage of Queues with Zero Occupancy When a New Flit Arrives.	95
39. Average Occupancy of Queues with Non-Zero Length for (a) NiN, (b) VGG-19.	95
40. Comparison Between Average Latency and Worst-Case Latency for Source to Destination Pairs with Non-Zero Latency for (a) LeNet-5 and (b) NiN. ...	97

Figure	Page
41. (a) Normalized Throughput and (b) Normalized EDAP of NoC-tree and NoC-mesh-based On-Chip Interconnect for SRAM-based IMC Architecture for Different DNNs. Dense DNNs Favor NoC-mesh While NoC-tree is Sufficient for Shallow DNNs.	98
42. (a) Normalized Throughput and (b) Normalized EDAP of NoC-tree and NoC-mesh-based On-Chip Interconnect for ReRAM-based IMC Architecture for Different DNNs. Dense DNNs Favor NoC-mesh While NoC-tree is Sufficient for Shallow DNNs.	98
43. Assessment of (a) Throughput and (b) EDAP Between NoC-tree and NoC-mesh With Different Numbers of Virtual Channels for Different DNNs. The Throughput is Normalized to That of NoC-tree. The Preferred NoC Topology for Optimal Performance is Shown for the Regions Above and Below the Red Line.	99
44. Assessment of (a) Throughput and (b) EDAP Between NoC-tree and NoC-mesh With Different Bus Width for Different DNNs. The Throughput is Normalized to That of NoC-tree. The Preferred NoC Topology for Optimal Performance is Shown for the Regions Above and Below the Red Line.	100
45. Optimal NoC Topology for IMC Architectures for Different DNNs.	101
46. Appropriate Selection of NoC Topology Significantly Improves Performance for Both SRAM- and ReRAM-based IMC Architectures.	104

Figure	Page
47. Experiments Using NeuroSim Chen <i>et al.</i> (2018) for Different DNNs Using Homogeneous SRAM-based IMC Architecture With P2P Interconnect Show that (a) Less than 65% of the PEs are Utilized, Except for VGG Network, (b) up to 90% of the End-to-End Latency is Spent on On-Chip Communication, (c) Communication Energy Constitutes 20%–40% of the Total Energy with NoC Having One Node per Tile Chen <i>et al.</i> (2019); Kwon <i>et al.</i> (2017).	107
48. Overview of the Proposed Methodology to Obtain an Area- and Energy-Optimized IMC DNN Accelerator. (a) Shows DNNs with Various Connection Structures, (b) Shows the Joint Optimization Technique, and (c) Shows the Generated Heterogeneous IMC Architecture With Optimized Interconnect. Each PE Consists of the Crossbar of the Same Size, with Different Numbers of PEs Within Each Tile.	107
49. The Hierarchical Structure of the Generated Heterogeneous In-Memory Architecture with Optimized NoC by Utilizing the Proposed Area and Energy Optimization Methodology.	110
50. NoC Optimization Effectively Reduces the Power Consumption Because of its Non-Linear Dependence on the Mesh Size. We Obtain NoC Power Through BookSim Jiang <i>et al.</i> (2013) Simulations.	112
51. Improvement With Respect to the Baseline Architecture (SRAM) in (a) PE Utilization and (b) Total Area with the Proposed Area-Aware Optimization.	116
52. Layer-wise Improvement for NiN in (a) PE Utilization for Each Layer with SRAM-Based Heterogeneous Tile Architecture. The Tile Structure for Each Layer (c_k, p_k) is Shown on Top of Each Bar and (b) Communication Latency for Each Layer with Proposed NoC Optimization.	118

Figure	Page
53. Improvement in (a) Communication Energy of the Proposed Energy-Aware NoC Optimization with Respect to the Baseline (SRAM) and (b) Energy-Area Product of the Generated SRAM-based Architecture with Respect to the Baseline (SRAM).	119
54. (a) Total Chip Area and Fabrication Cost for a Monolithic RRAM-based IMC Architecture for Different DNNs Shafiee <i>et al.</i> (2016). Fabrication Cost Increases Exponentially with an Increase in Total Chip Area (Appendix A Details the Method to Calculate the Fabrication Cost). (b) 3-Dimensional Diagram Showing the Chiplet-based IMC Architecture. The Architecture Includes an Array of IMC Chiplets, Global Buffer, Global Accumulator, and DRAM Connected by an NoP. The Figure is for Representational Purposes and Not Drawn to Scale.	124
55. Chiplet-based IMC Architecture Utilized Within SIAM. The Architecture Consists of IMC Chiplets, Global Accumulator, Buffer, and DRAM Connected Using an NoP. SIAM Supports Both SRAM- and RRAM-based IMC Architectures. An NoP is Applied for Inter-Chiplet Communication and an NoC is Utilized Within the Chiplet for Intra-Chiplet Communication.	127
56. An Overview of the Proposed Chiplet-based IMC Benchmarking Simulator, SIAM. SIAM Incorporates Device, Circuits, Architecture, NoC Jiang <i>et al.</i> (2013), NoP, and DRAM Access Model Kim <i>et al.</i> (2015) Under a Single Roof for System-Level Analysis of Chiplet-based IMC Architectures.	129

Figure	Page
57. Representative Figure Showing the Two Generated Chiplet-based IMC Architectures for the Same DNN, Homogeneous (Left) and Custom (Right), from the Supported Partition Schemes in SIAM. Homogeneous Architecture is Generic, while Custom Architecture is DNN Specific. R Refers to the NoP Router.	134
58. Block Diagram of the Circuit and NoC Engine Within SIAM. The Engine Utilizes a Separate Circuit and NoC Simulators that Perform the Overall Hardware Performance Estimation.	136
59. (Top) Cross-Sectional Image of the NoP Interconnect. The NoP is Routed Within the Interposer Connecting Different Chiplets Across the Architecture. μ bumps Connect the Chiplets to the Interposer, (Bottom) Energy Per Bit for Different NoP Driver Circuit and Signaling Techniques Proposed in Prior Works.	140
60. (a) The Accuracy of EDP Prediction for Different Numbers of Instructions Processed to Represent 3,000 DRAM Instructions. Reduction in the Number of Instructions to Half Results in Less Than 2% EDP Accuracy Degradation for Half the Simulation Time, and (b) EDP of DRAM Transactions (DDR4) for Different DNNs. There is an Exponential Increase in DRAM Cost with an Increase in DNN Model Size.	142
61. Computation Dataflow Within the Chiplet-based IMC Architecture in SIAM. Two Cases Arise, (a) No Layer is Partitioned Across Two or More Chiplets, and (b) a Layer is Partitioned Across Two or More Chiplets.	143

Figure	Page
62. IMC Utilization for a Custom RRAM-based Chiplet IMC Architecture Across Different DNNs and Different Chiplet Configurations. The Mapping Strategy Adopted Within SIAM Ensures High Utilization Across all DNNs.	147
63. Breakdown of the Different Components Contributing to the Overall Area, Energy, and Latency Performance Metrics, for a Custom Design RRAM-based Chiplet IMC Architecture When Mapping ResNet-110 for CIFAR-10 Dataset.	148
64. NoP and NoC Trade-Off Analysis for ResNet-110 on CIFAR-10 Dataset. (a) Ratio of the Energy-Delay-Area Product (EDAP) of NoP to NoC for Both Homogeneous and Custom Chiplet-based IMC Architectures. The Increase in Tiles per Chiplet Reduces the NoP/NoC EDAP, (b) NoP and NoC Energy-Delay Product (EDP) for a 36 Chiplet Count Configuration of Homogeneous RRAM-based Chiplet IMC Architectures. An Increased Tiles per Chiplet Leads to Higher NoC Cost and Lower NoP Cost.	150
65. Energy-Delay-Area Product as the Metric. (a) Overall EDAP and (b) Total Area for the Homogeneous and Custom RRAM-based Chiplet IMC Architecture When Mapping ResNet-110 for CIFAR-10 Dataset. The Results Indicate that a Custom Architecture Outperforms a Homogeneous Architecture. The Increased Number of Tiles per Chiplet Provides Better Performance at the Cost of Increased Area for the Homogeneous Chiplet IMC Architecture, While Providing Better Performance and Lower Area for the Custom Chiplet IMC Architecture.	152

Figure	Page
66. Improvement in Fabrication Cost (\$) for the RRAM-based Chiplet IMC Architecture, (a) Custom and (b) Homogeneous, as Compared to a Monolithic RRAM-based IMC Architecture. Smaller DNNs Like ResNet-110 have Similar Cost for Both Architectures, while Larger DNNs Such as VGG-19 have up to 60% Improvement.	153
67. (a) Total Energy for DNN Inference Reduces with an Increase in the Number of Tiles per Chiplet (Chiplet Size), (b) Total Inference Latency and Throughput for ResNet-110 on CIFAR-10. Due to the Small Network Size, a Lower Number of Chiplets Provide Better Performance, (c) Normalized Latency for Two Layers Within ResNet-50 on ImageNet for SIAM RRAM-based Chiplet IMC Architecture and SIMBA Shao <i>et al.</i> (2019). The Decreasing Trend of Latency with Increasing Chiplet Count Exhibited by SIAM is Consistent with SIMBA, (d) Bandwidth Sensitivity for a Layer Within ResNet-50. The Decreasing Trend in the PE Cycles with Increasing NoP Speed-Up Similar to SIMBA.	155
68. Normalized Layer-Wise Activation/Weight Distribution for (a) ResNet-50 (ImageNet) and (b) VGG-19 (CIFAR-100). Initial/Latter Layers are Activation/Weight Dominated.	162
69. IMC Utilization for Different DNNs Using a Homogeneous Chiplet RRAM IMC Architecture Krishnan <i>et al.</i> (2021e) and the Proposed Heterogeneous Big-Little Chiplet Architecture. The Heterogeneous Big-Little Architecture Improves the IMC Utilization.	162

Figure	Page
70. Cross-Sectional View of the Big-Little Chiplet-based IMC Architecture. The Architecture Consists of a Little Chiplet Bank with Little Chiplets (Connected by an NoP Within the Interposer and a Big Chiplet Bank with Big Chiplets Connected by a Bridge NoP. NoP Properties: 1.5–8mm Length, 2–4.5 μ m Pitch, and 0.5–2 μ m Width.....	164
71. (a) Overview of the Big-Little Chiplet IMC Architecture. The Little Chiplet Bank Utilizes Smaller Chiplets Connected by a Interposer-based NoP While the Big Chiplet Bank Utilizes Bigger Chiplets Connected by a Bridge-based NoP. Each Chiplet Utilizes a Local DRAM, (b) IMC Chiplet Architecture (Big and Little). Each Chiplet Consists of an Array of IMC Tiles and a Dedicated NoP Transceiver and Router, (c) The Little Chiplet Bank Consists of Fewer and Smaller Tiles While the Big Chiplet Bank Consists of More Bigger Tiles. Both Chiplet Structures Utilize a Mesh-based NoC for On-chip Communication, and (D) Structure of Each Tile Within the Big and Little Chiplet. It Consists of an Array of IMC Crossbar Arrays and Associated Peripheral Circuits with an Interconnect Similar to That in Shafiee <i>et al.</i> (2016). The Little Chiplet Consists of Fewer and Smaller IMC Crossbars While the Big Chiplet has Larger and More IMC Crossbar Arrays.	166
72. IMC Utilizations for Different DNNs Across Different Big-Little Chiplet-based RRAM IMC Configurations for (a) ResNet-110, (b) ResNet-34, (c) VGG-19, (d) DenseNet-40. Based on the Utilization, We Choose Crossbar Size of Big Chiplet as 256 \times 256 and Crossbar Size of Little Chiplet as 64 \times 64 (256–64).	177

Figure	Page
73. Normalized NoP EDP for Different Bus-Widths for VGG-19 and ResNet-34. The NoP With Bus Width of 24 for Big and 32 for Little Chiplets (24–32) Shows Lowest EDP.	177
74. EDAP Comparison (Log-Scale) of the Big-Little Chiplet-based RRAM IMC Architecture to ‘Little Only’ and ‘Big Only’ Chiplet-based RRAM IMC Architectures. The Big-Little Architecture Achieves up to 329× Improvement Compared to ‘Little Only’ Architecture.	181

INTRODUCTION

1.1 Reliable In-Memory Computing

Deep neural networks (DNNs) have seen exceptional success in numerous cognitive applications such as image classification and object detection. Higher accuracy comes at the cost of increased computational complexity and model size, posing great challenges to traditional architectures Chen *et al.* (2019); Jouppi *et al.* (2017). Furthermore, DNN hardware accelerators incur a significant area overhead, especially when the DNN model size is rapidly increasing. Finally, limited on-chip memory capacity leads to a significant amount of communication with off-chip memory, whose energy consumption is $1,000\times$ higher than that of computations Horowitz (2014).

In-Memory computing (IMC) architecture provide a viable alternative to conventional architectures. They combine memory access and computation into a single unit through analog or digital domain computation. IMC architectures are designed using different types of IMC cells, RRAM, SRAM, FeFET, etc. First, we focus on an RRAM-based IMC architecture. RRAM-based IMC accelerators provide a dense and parallel structure to achieve high performance and energy efficiency Song *et al.* (2017); Krishnan *et al.* (2020b); Mandal *et al.* (2022); Krishnan *et al.* (2022a). RRAM memory cell stores the weights of the DNN. Furthermore, multi-level RRAM cells can store multi-bit weights in a single cell, thus increasing the density of the IMC architecture. Through this, a more dense computing structure of the RRAM-based IMC allows for more weights to be stored on-chip resulting in reduced off-chip memory

access. Prior works with RRAM-based crossbar architectures have shown up to $1,000\times$ improvement in energy efficiency as compared to CPUs/GPUs Song *et al.* (2017); Krishnan *et al.* (2020b); Shafiee *et al.* (2016); Mandal *et al.* (2020); Imani *et al.* (2019); Qiao *et al.* (2018); Mao *et al.* (2019). The increased energy-efficiency is attributed to a full-custom design following the assumption; all weights are stored on-chip Shafiee *et al.* (2016); Song *et al.* (2017); Krishnan *et al.* (2020b).

But, the increased model size of DNNs result in higher area overhead for RRAM-based IMC architecture. Hence, model compression (e.g. pruning and quantization) is necessary for RRAM-based in-memory acceleration of DNNs. Furthermore, in reality, RRAM suffers from statistical variations such as quantization error, device-to-device write variations, stuck-at-faults, and limited $R_{\text{off}}/R_{\text{on}}$ ratio, posing a significant challenge to designing reliable RRAM-based IMC architectures Chen *et al.* (2014); Chakraborty *et al.* (2020); Charan *et al.* (2020a). To mitigate the post-mapping accuracy loss in DNNs, variation-aware training (VAT) is employed Long *et al.* (2019); Charan *et al.* (2020a); Chakraborty *et al.* (2020); He *et al.* (2019b); Ma *et al.* (2020). But, conventional VAT methods perform training and testing at the same RRAM variations. Simultaneously, conventional VAT methods require precise knowledge of the RRAM variations from the fabricated cells. Finally, conventional VAT methods do not address effect of limited precision within the RRAM IMC architecture (ADC and accumulator) and effect of pruning and quantization for DNNs.

To address this, in this dissertation, we propose the following:

- A new metric, *model stability*, from the loss landscape to help shed light on accuracy under variations and model compression and guide an algorithmic solution that mitigates the loss. The model stability is visualized by the loss landscape and evaluated by the roughness score Du *et al.* (2020),

- To the best of our knowledge, this is the first time model stability and loss landscape are used for reliable RRAM-based IMC computing,
- We utilize model stability to select the more stable model that can withstand the variations better. The proposed model stability-based model selection effectively tolerates device variations and achieves a post-mapping accuracy higher than that with 50% lower RRAM variations,
- We show that pruning results in a less stable model, while quantization improves the model stability,
- We further propose a model stability-based VAT method for compressed DNNs, which searches the most stable model under variations to achieve the best post-mapping accuracy, without knowing the exact amount of RRAM testing variations upfront,
- Finally, we show that the model-stability-based VAT method achieves up to 19%, 21%, and 11% improvement in accuracy for compressed DNNs on CIFAR-10, CIFAR-100, and SVHN datasets, respectively.

Next, in this dissertation, we aim to bridge the gap between the floating-point software accuracy and the post-mapping accuracy of RRAM-based IMC architectures (after applying model stability-based VAT methods). To address this, we propose the following:

- We propose a novel hybrid IMC architecture that utilizes an RRAM-based IMC macro and a reconfigurable SRAM array and output stationary multiply-and-accumulate (MAC) macro,
- The RRAM macro consists of a IMC crossbar array with a column multiplexer, ADCs, a PMOS header circuit, and a shift and add circuit. At the same time, the SRAM macro consists of an SRAM memory array with N-bit word size and

N parallel MAC engines that utilize an output stationary architecture to reduce partial sum movement,

- A programmable shifter is implemented to allow for 1-bit, 2-bit, and 3-bit shift for the SRAM output to generate different levels of compensation for the RRAM macro. Finally, an adder is used to combine the RRAM and SRAM output to generate the overall output of the DNN layer,
- We designed a prototype of the hybrid IMC architecture with a fully integrated 1T1R RRAM structure and a custom SRAM module in the SUNY Polytechnic Institute’s 65nm process,
- Compared to state-of-the-art methods, the proposed hybrid IMC architecture achieves up to 21.9%, 12.65%, and 6.52% improvement in post-mapping accuracy with minimal overhead for ResNet-20 on CIFAR-10, VGG-16 on CIFAR-10, and ResNet-18 on ImageNet, respectively.

1.2 In-Memory Computing Architectures and Optimizations

In addition to the mitigation of the non-ideal effects of RRAM-based IMC architecture, this dissertation also presents two architectural optimizations (both RRAM- and SRAM-based IMC) that provides high IMC utilization, optimal on-chip communication cost, and reduced energy-delay product (EDP) for DNN inference. First we evaluate and give insights on the effect of the choice of the interconnect in IMC architectures, as shown below:

- First, we illustrate that the point-to-point (P2P)-based interconnect is incapable of handling a high volume of on-chip data movement for DNNs,
- We evaluate P2P and network-on-chip (NoC) interconnect (with a regular

topology such as a mesh) for SRAM- and ReRAM-based in-memory computing (IMC) architectures for a range of DNNs,

- Furthermore, we propose to use analytical models of NoC to evaluate end-to-end communication latency of any given DNN and determine the optimal choice of interconnect for any given DNN,
- We demonstrate that the interconnect optimization in the IMC architecture results in up to $6\times$ improvement in energy-delay-area product for VGG-19 inference compared to the state-of-the-art ReRAM-based IMC architectures.

Second, we identify the area and energy bottleneck in IMC architectures. We propose to utilize an area and energy optimization to improve the IMC utilization and reduce the interconnect energy, as shown below:

- We propose an area-aware optimization technique that improves the PE array utilization. This is achieved by generating a heterogeneous tile-based IMC architecture that consists of tiles of different sizes, i.e. with different numbers of PEs where each PE is of the same size,
- Further, we minimize the communication energy across a large number of tiles using an NoC architecture with optimized tile-to-router mapping and scheduling,
- Overall, our proposed area and energy optimization methodology generates a heterogeneous IMC architecture coupled with an optimized NoC for DNN acceleration,
- Experimental evaluations show up to 62% improvement in PE utilization, 78% reduction in area, and 78% lower energy-area product for a wide range of modern DNNs such as DenseNet (100,24), and ResNet-152.

1.3 IMC Chiplet Architecture and Benchmarking Simulator

Furthermore, in this dissertation, to address the limitations of a monolithic DNN accelerator, we propose a novel chiplet-based IMC benchmarking simulator and an heterogeneous chiplet IMC architecture. The following are the key contributions of this dissertation through the chiplet-based IMC benchmarking simulator, SIAM:

- We propose a novel benchmarking tool, SIAM, device, circuits, architecture, NoC, NoP, and DRAM access estimation under a single roof for design space exploration,
- SIAM supports both monolithic and chiplet-based IMC architectures,
- To the best of our knowledge, this is the first open-sourced architectural exploration tool for chiplet-based IMC architectures,
- SIAM has a flexible architecture to support multiple DNN to IMC chiplet and crossbar partition and mapping schemes, thus generating different types of chiplet-based IMC architectures,
- Furthermore, SIAM has a low simulation time ranging from a few minutes to a few hours (4.5Hrs for VGG-16 with 138M parameters) to support the fast design and benchmarking exploration,
- We calibrate SIAM against a published silicon result, SIMBA, a real-world chip from Nvidia,
- We demonstrate SIAM’s capabilities by conducting experiments on state-of-the-art DNNs such as ResNet-110 for CIFAR-10, VGG-19 for CIFAR-100, and ResNet-50 and VGG-16 for ImageNet datasets.

Finally, to optimize chiplet-based IMC architecture, we propose a heterogeneous chiplet-based IMC architecture with a custom mapping for scalable DNN acceleration.

- We propose a heterogeneous big-little chiplet-based IMC architecture that utilizes a big and little IMC-based chiplet bank coupled with an optimal NoP configuration (interposer and bridge),
- We develop an algorithm to determine the optimal configuration of the big-little IMC chiplet architecture. The little-chiplet bank consists of little chiplets interconnected by an interposer-based NoP (chiplets are placed closed to each other). Similarly, the big-chiplet bank consists of big chiplets interconnected by a bridge-based NoP. In addition, each chiplet (big/little) utilizes a local DRAM to store the weights of the DNN.
- We present a custom mapping strategy of DNNs onto the big-little chiplet IMC architecture that exploits the non-uniform distribution of weights and activations. The smaller structure of the weights in the early layers results in higher utilization within the little chiplet bank, while the larger layers towards the end of the DNN achieve high utilization on the big-chiplet bank. We also exploit the activation distribution by utilizing an interposer-based NoP with high bandwidth within the little chiplet bank, which houses the early layers with higher on-chip data movement. Simultaneously, the subsequent layers with lower on-chip data movement (fewer activations) utilize the bridge-based NoP with lower bandwidth within the big chiplet bank.
- Experimental evaluation of the proposed big-little chiplet-based RRAM IMC architecture on ResNet-50 on ImageNet shows up to $259\times$, $139\times$, and $48\times$ improvement in energy-efficiency and lower area compared to Nvidia V100 GPU, Nvidia T4 GPU, and SIMBA architecture, respectively.

1.4 Thesis organization

The outline of this thesis is as follows:

- **Chapter 2** presents model stability as a metric for reliable RRAM-based IMC acceleration. This chapter explains model stability, the methodology utilized to evaluate the metric and effect of pruning and quantization on model stability. Next, we discuss the proposed model stability-based model selection for reliable RRAM-based IMC accelerations. Finally, we extend model stability to propose a novel VAT method that searches for the most stable model through an optimal training variations scale for reliable RRAM computing.
- **Chapter 3** presents the hybrid RRAM/SRAM IMC architecture for robust RRAM-based IMC acceleration of DNNs. The chapter discussed a hybrid IMC architecture that utilizes an RRAM-based IMC macro and a SRAM memory and associated output stationary MAC engine. The architecture utilizes a programmable shifter to combine the outputs from the RRAM and SRAM macro where the SRAM output acts as a compensation with different scales through the shifter.
- **Chapter 4** presents a detailed study and insights into the effect of the choice of the interconnect for IMC architectures. We discuss P2P and network-on-chip (NoC) as a choice for the interconnect. Furthermore, we propose to use analytical models of NoC to evaluate end-to-end communication latency of any given DNN and determine the optimal choice of interconnect for any given DNN.
- **Chapter 5** presents an area and energy optimization to improve the utilization and energy cost for IMC architectures. We evaluate the methodology for both SRAM and RRAM IMC architectures. The chapter discusses the effect of the

optimizations individually and together for a wide-range of DNNs across different datasets.

- **Chapter 6** presents the first open-sourced architectural benchmarking simulator, SIAM, that supports both monolithic and chiplet-based IMC architectures. The chapter details each engine within SIAM, the supported architectures, underlying dataflow, working, and benchmarking with a real-world chip like SIMBA. Finally, the chapter discusses the capabilities of SIAM through experiments and the different grades of architectural exploration across different DNNs and datasets.
- **Chapter 7** presents the heterogeneous big-little chiplet architecture for In-Memory acceleration of DNNs. The architecture utilizes a big and little IMC-based chiplet bank coupled with an optimal NoP configuration (interposer and bridge). Furthermore, Each chiplet (big/little) utilizes a local DRAM to store the weights of the DNN. In addition, we develop an algorithm to determine the optimal configuration of the big-little IMC chiplet architecture. Finally, we present a custom mapping strategy of DNNs onto the big-little chiplet IMC architecture that exploits the non-uniform distribution of weights and activations for higher IMC utilization and NoP energy efficiency.
- **Chapter 8** concludes the dissertation.

RELIABLE RRAM-BASED IN-MEMORY COMPUTING THROUGH MODEL STABILITY

2.1 Introduction

Deep neural networks (DNNs) outperform humans for a variety of applications, such as computer vision and natural language processing. Higher accuracy comes at the cost of increased computational complexity and model size, posing great challenges to traditional architectures Chen *et al.* (2019). In addition, limited on-chip memory capacity leads to a significant amount of communication with off-chip memory, whose energy consumption is $1,000\times$ higher than that of computations Horowitz (2014).

RRAM-based IMC accelerators provide a dense and parallel structure to achieve high performance and energy efficiency Song *et al.* (2017); Krishnan *et al.* (2020b); Du *et al.* (2019). Prior works with RRAM-based crossbar architectures have shown up to $1,000\times$ improvement in energy efficiency as compared to CPUs/GPUs Song *et al.* (2017); Krishnan *et al.* (2020b); Shafiee *et al.* (2016); Mandal *et al.* (2020); Imani *et al.* (2019). The increased energy-efficiency is attributed to a full-custom design following the assumption; all weights are stored on-chip Shafiee *et al.* (2016); Song *et al.* (2017); Krishnan *et al.* (2020b). However, RRAM-based IMC architectures incur a significant area overhead, especially when the DNN model size is rapidly increasing. Hence, model compression (e.g. pruning and quantization) is necessary for RRAM-based in-memory acceleration of DNNs.

In reality, RRAM suffers from statistical variations such as quantization error,

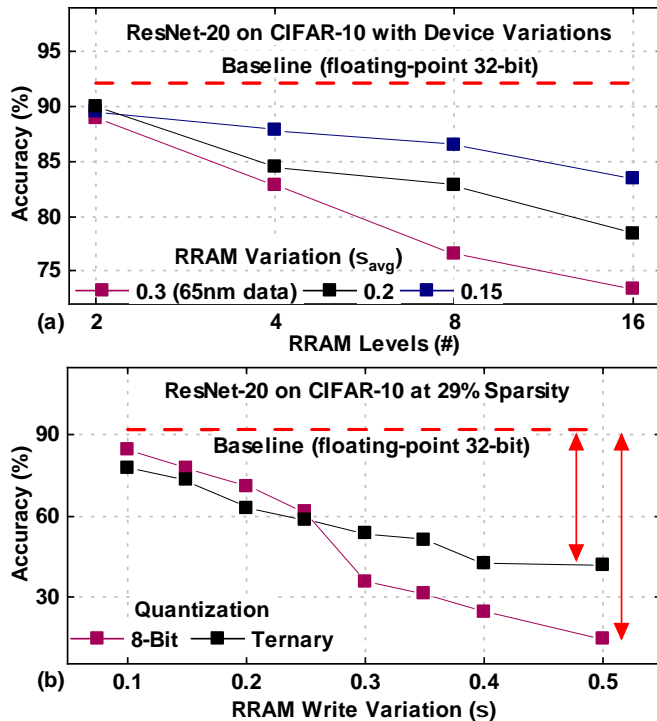


Figure 1. Post-Mapping Accuracy for ResNet-20 on CIFAR-10 (a) Across Different RRAM Levels and Average RRAM Variations. For a Given R_{off}/R_{on} Ratio, Higher RRAM Levels Suffer From Variations and Thus Lower Accuracy. Simultaneously, a Lower Average RRAM Variation Results in Higher Accuracy, (b) At 8-bit and Ternary-bit Precision, with 29% Sparsity Yang *et al.* (2020). Model is Trained and Tested with the Same Variation (σ) Long *et al.* (2019); Charan *et al.* (2020a). The 8-bit Model has More Accuracy Loss Than the Ternary Model As σ Increases.

device-to-device write variations, stuck-at-faults, and limited R_{off}/R_{on} ratio, posing a significant challenge to designing reliable RRAM-based IMC architectures Chen *et al.* (2014); Chakraborty *et al.* (2020); Charan *et al.* (2020a); Krishnan *et al.* (2021a); Charan *et al.* (2020b). The statistical variations in RRAM cause deviation in the programmed resistance leading to a significant loss in post-mapping accuracy (i.e., accuracy in the presence of RRAM variations) for DNNs. To mitigate the post-mapping accuracy loss in DNNs, variation-aware training (VAT) is employed Long *et al.* (2019); Charan *et al.* (2020a); Chakraborty *et al.* (2020); He *et al.* (2019b); Ma

et al. (2020); Krishnan *et al.* (2022b). VAT exploits the inherent redundancy in DNN by embedding the device variations (σ), based on a log-normal or normal distribution model, into the training process to achieve a variation-tolerant model, with no need of re-training for each individual RRAM chip Long *et al.* (2019); Charan *et al.* (2020a); He *et al.* (2019b). The conventional VAT techniques train and test the DNN model at the same level of variation.

Fig. 1(a) shows the post-mapping accuracy for ResNet-20 He *et al.* (2016) on CIFAR-10 dataset for different RRAM levels across various average RRAM variations. The baseline accuracy for the floating-point 32bit (FP-32) model is shown in red (dash line). The red curve shows the variation of pre-mapping accuracy with RRAM levels for our 65nm RRAM data with an average variation (σ_{avg}) of 0.3. For a given $R_{\text{off}}/R_{\text{on}}$ ratio, a higher number of RRAM levels leads to higher variation and lower accuracy, and vice-versa. A higher variation for a higher number of RRAM levels arises from the increased HRS state utilization. Further, we analyze the scenario with a reduced average variation of 0.2 and 0.15. A reduction in RRAM variation through improved process control improves the pre-mapping accuracy. But, though the reduction in RRAM variation improves the accuracy, it does not achieve the same accuracy as FP-32. Hence, we establish that the reduction in RRAM variation does not get the pre-mapping accuracy back to the FP-32 level (baseline).

Next, we analyze the effect of RRAM variations on a sparse and quantized DNN. Fig. 1(b) shows the accuracy of ResNet-20 for CIFAR-10 at 29% sparsity for different RRAM write variations and data precision using the conventional VAT method Long *et al.* (2019). Conventional VAT proves ineffective under pruning Krishnan *et al.* (2020a) and quantization, resulting in reduced post-mapping accuracy. Furthermore, a lower precision (ternary) helps improve the post-mapping accuracy, as shown in

Fig. 1(b). Hence, there is a need for a more systematic solution for reliable RRAM-based in-memory computing for dense, sparse, and quantized DNNs.

To address this, in this work¹, we propose a new metric, *model stability*, from the loss landscape to help shed light on accuracy under variations and model compression and guide an algorithmic solution that mitigates the loss. The model stability is visualized by the loss landscape and evaluated by the roughness score Du *et al.* (2020). A lower roughness score indicates a smoother loss landscape and a more stable model. Through this, we select the more stable model that can withstand the variations better. The proposed model stability-based model selection effectively tolerates device variations and achieves a post-mapping accuracy higher than that with 50% lower RRAM variations. Next, we propose a novel variation-aware training (VAT) method for best model stability in compressed DNNs. The proposed method utilizes VAT to train the compressed DNN with different scales of device variations (σ) to search for the most stable model and improve post-mapping accuracy. For a given DNN model, higher model stability implies better tolerance of variations and thus, higher post-mapping accuracy. We utilize a structured pruning method Yang *et al.* (2020) and model quantization He *et al.* (2019a); Zhou *et al.* (2016) to compress DNN. The pruning method considers the mapping of the DNN onto the RRAM crossbar for best IMC performance. We show that pruning results in a less stable model, while quantization improves the model stability. We demonstrate that the proposed method achieves up to 19%, 21%, and 11% improvement in post-mapping accuracy on CIFAR-10, CIFAR-100, and SVHN datasets, respectively. The major contributions of this work are as follows:

¹Work done in collaboration with Prof. Deliang Fan, Li Yang, and Jingbo Sun at ASU, and SUNY.

- We propose a new metric, *model stability*, using the loss landscape to mitigate the accuracy loss of dense and compressed DNNs in the presence of RRAM variations,
- Using model stability as a metric to choose the more stable model results in similar accuracy improvement as that for 50% lower RRAM variations through costly process control,
- We further propose a model stability-based VAT method for compressed DNNs, which searches the most stable model under variations to achieve the best post-mapping accuracy, without knowing the exact amount of RRAM testing variations upfront,
- Finally, we show that the model-stability-based VAT method achieves up to 19%, 21%, and 11% improvement in accuracy for compressed DNNs on CIFAR-10, CIFAR-100, and SVHN datasets, respectively.

2.2 DNN Model Stability

Given a trained DNN model, its model stability is an intrinsic property to withstand perturbations, such as variations in model weights and input noise. Model stability of a DNN, i.e., the DNN generalization capability, is directly related to the contour of the loss function Hochreiter and Schmidhuber (1997); Keskar *et al.* (2019); Li *et al.* (2018); Du *et al.* (2020). A flatter contour of the loss function leads to a larger region of acceptable minima, which allows the DNN model to better tolerate variations in both weights and inputs. Vice versa, a steeper contour of the loss function leads to a smaller region of acceptable minima Hochreiter and Schmidhuber (1997), which implies that any perturbations to the weights or inputs will lead to appreciable movement of the

minima point and thus, reduce model accuracy. In this work, we utilize DNN model stability as a metric to guide reliable RRAM-based IMC acceleration.

2.2.1 Landscape visualization

In order to quantitatively understand model stability, we utilize the landscape visualization method Li *et al.* (2018) to visualize the minima of the loss function. In Li *et al.* (2018), filter normalization is applied to remove the scaling effect of injected noise, and a 3-dimension matrix is generated with x, y, and z coordinates, where x and y represent the scale of two random perturbations injected into the model and z is the loss function. Essentially this matrix plots the fluctuation of the loss function under the local perturbation around the local minimum.

2.2.2 Roughness Score

We calculate the smoothness of the loss function, defined as roughness score, to quantify the loss landscape’s stability further. We fit the 3-dimensional data from the landscape using quadratic linear regression and obtain the mean square error (MSE) of the fitting model, as shown below:

$$\hat{z}_j = w_{j4}x_j^2 + w_{j3}y_j^2 + w_{j2}x_j + w_{j1}y_j + w_{j0}, \quad (2.1)$$

$$\hat{w} = \underset{w_j}{\operatorname{argmin}} \frac{1}{n} \sum_{j=0}^n (z_j - \hat{z}_j)^2 \quad (2.2)$$

where w_j represents the fitted coefficients. We denote the stability or roughness score of the DNN model as $\operatorname{MSE}(z; x^2, y^2, x, y; \hat{w})$. A smaller MSE arises from a flat and smooth landscape and vice-versa. Note that such a method was previously used to

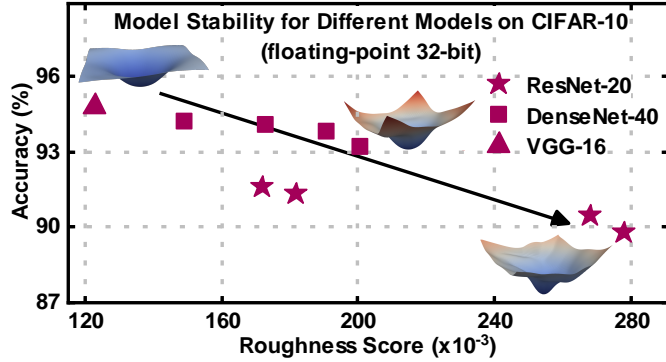


Figure 2. A Lower Roughness Score Leads to a Smoother Loss Landscape, Higher Stability and Thus, Higher Model Accuracy.

improve the accuracy in continual learning Du *et al.* (2020), while it did not consider device variations, sparsity, and quantization. *To the best of our knowledge, this is the first time that model stability has been employed to provide systematic guidance to improve the post-mapping accuracy for the acceleration of dense and compressed DNNs using RRAM-based IMC architectures.*

2.2.3 Roughness Score and DNN Accuracy

Fig. 2 shows the accuracy and roughness score for different DNNs. We generate different versions for a DNN model by utilizing different weight initialization, which leads to different roughness scores and corresponding accuracy. VGG-16 for CIFAR-10 achieves 94.2% accuracy and the lowest roughness score of 118×10^{-3} . At the same time, a ResNet-20 version achieves the lowest accuracy of 89.5% with a roughness score of 278×10^{-3} . To further understand the relationship between the roughness score, loss landscape, and DNN accuracy, we visualize the loss landscape using the method detailed in Li *et al.* (2018). VGG-16 has a shallow and smooth loss landscape while

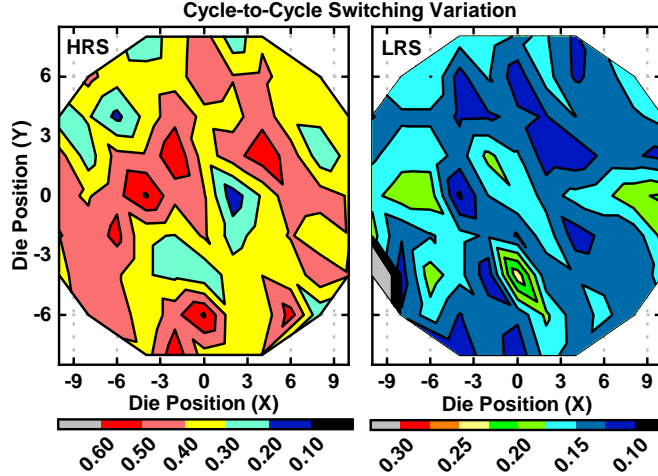


Figure 3. HRS (Left) and LRS (Right) Cycle-To-Cycle Switching Variation Across the 300mm Wafer at 65nm. HRS State has a Higher Variation Than LRS.

the lowest accuracy ResNet-20 variant has a rough loss landscape. Through these examples we establish that a lower roughness score leads to a smoother loss landscape, more acceptable local minima for the loss function, and a higher DNN accuracy.

2.3 65nm 1T1R RRAM Device

To accurately model the RRAM device properties, RRAM data is collected from a fully integrated 1T1R structure on a 300mm wafer, using a custom RRAM module within the SUNY Polytechnic Institute’s 65nm process. The size of each RRAM device is 100nmx100nm. The RRAM device stack is comprised of a 6nm HfO₂ mem-resistive switching layer, a 6nm PVD Ti oxygen exchange layer (OEL), and TiN electrodes (top and bottom). Electrical characterization is performed using a pulse-based approach having a magnitude 1V – 1.2V and width of 10 μ s for the set and reset operation of RRAM devices.

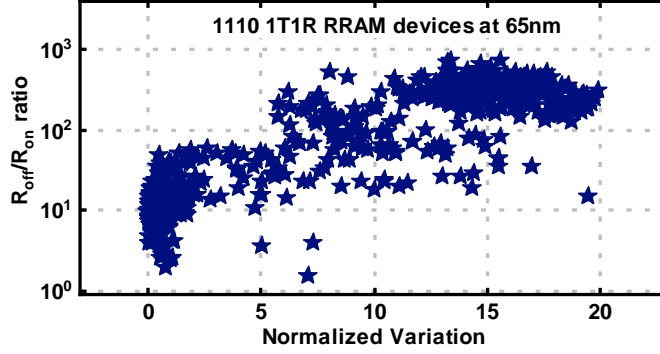


Figure 4. Normalized Variation for Different $R_{\text{off}}/R_{\text{on}}$ Ratios for 1110 1T1R RRAM Devices at 65nm. A Higher $R_{\text{off}}/R_{\text{on}}$ Ratio Leads to Higher Variation.

Table 1. Endurance Measurement up to 1 Billion Cycles

Cycles	10	10^2	10^3	10^5	10^7	10^9
LRS (K Ω)	5.7	5.5	5.7	5.6	5.8	5.9
HRS (K Ω)	109	231	157	43.3	113.8	30.8

Fig. 3 shows the wafer-level cycle-to-cycle switching variations for the 65nm RRAM device measured using the pulse-based switching technique. The high-resistance state (HRS) has a higher variation up to 0.6, while the low-resistance state (LRS) has a lower variation up to 0.1. The average variation (σ_{avg}) for the entire range of HRS and LRS amounts to 0.3.

Fig. 4 shows the normalized variation for different $R_{\text{off}}/R_{\text{on}}$ ratios across 1110 1T1R RRAM devices at 65nm. As the ratio of $R_{\text{off}}/R_{\text{on}}$ goes up its HRS state utilization increases. A higher HRS state utilization results in higher device variations and an increase in overall average variation, as shown in Fig. 3. Overall, a lower $R_{\text{off}}/R_{\text{on}}$ ratio results in lower average device variation at the cost of lower usable resistance levels and hardware performance, and vice-versa.

Fig. 5 shows the retention of both HRS and LRS for 10^5 seconds at 100°C . The

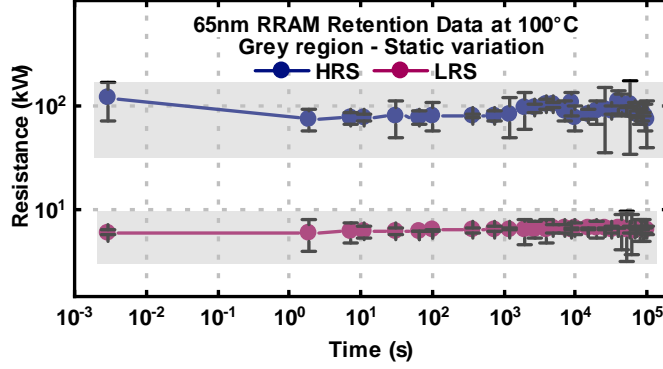


Figure 5. The 65nm RRAM Device Achieves Good Retention for Up to 10^5 Seconds. The Grey Region Shows the Dominating Effect of Static Variations for the 65nm 1T1R RRAM Device.

Table 2. Device Models From 65nm 1T1R RRAM Device

Parameter	Value
$R_{\text{off}}/R_{\text{on}}$	2-650
Write Variation ¹	$r' \leftarrow r.e^{\theta}; \theta \sim \mathcal{N}(0, \sigma^2)$
Average Variation (σ_{avg})	0.3
RRAM levels (#)	16
Aging Variation (σ_{age})	0.1, 0.02 (HRS, LRS)

¹ r is the ideal resistance to be programmed and r' is the real value in RRAM. $0.1 \leq \sigma \leq 0.5$.

grey region overlaid on the plot shows the static variation from the RRAM device. The 65nm 1T1R RRAM device shows low retention variation, as shown in Fig. 5. Table 1 shows the endurance data for both HRS and LRS states up to 1 billion cycles. Both the HRS and LRS states show high endurance with distinction between the two states up to 1 billion cycles. Since static write variations are dominant, in this work, we do not consider the effects of retention or endurance. Table 2 summarizes the device models used in the cross-layer simulation framework described in Section 2.4. In this work, we use the 65nm RRAM models for all our experiments to provide realistic results.

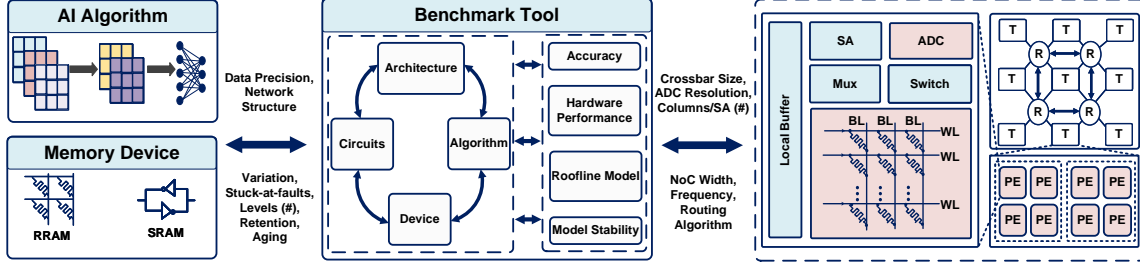


Figure 6. The Benchmarking Tool Incorporates Algorithm (Pruning and Quantization), RRAM IMC Architecture Properties, Circuit Models, and the 65nm 1T1R RRAM Device Models. The Tool Outputs Post-Mapping Accuracy, Hardware Performance, Roofline Model, and Model Stability.

2.4 Cross-Layer Simulation Framework

In this work, we develop an in-house simulator to perform system-level benchmarking of the RRAM-based IMC architecture. Fig. 6 shows the block diagram of the cross-layer benchmarking tool utilized.

The simulator incorporates device, circuits, architecture, and algorithm under a single roof to perform system-level benchmarking. The simulator provides post-mapping accuracy (hardware accuracy), the overall hardware performance, roofline model, and model stability. The inputs to the simulator include the DNN structure, data precision, target sparsity, technology node, bits per RRAM cell, IMC crossbar size, ADC resolution, read-out method, frequency, NoC topology, and NoC size, among others.

2.4.1 Device Models

The tool incorporates device models from the 65nm 1T1R RRAM device, as shown in Table 2. The RRAM device variations are modeled using the log-normal

distribution Krishnan *et al.* (2021f); Charan *et al.* (2020a). The $R_{\text{off}}/R_{\text{on}}$ ratio of the RRAM device ranges between 2 and 650. We assume that the discrete resistance levels used to represent the weights are within the limited $R_{\text{off}}/R_{\text{on}}$ ratio. The maximum number of resistance levels that a single RRAM cell can handle is 16, limiting the weights to be mapped to a single cell to 4-bits.

2.4.2 Circuit Estimator

The circuit estimator performs the estimation of the IMC, peripheral circuits, and digital modules within the architecture. We benchmark the overall circuit estimator with SIAM Krishnan *et al.* (2021e). The circuit estimator performs the mapping of the DNN onto the RRAM-based IMC crossbar architecture. The mapping utilized within the estimator follows that in Krishnan *et al.* (2021e). The IMC circuit components include the crossbar, wordline (WL) driver, level shifters, bitline (BL) and select-line (SL) multiplexers (MUX), column MUX, analog-to-digital converter (ADC), and shift and add circuit. In addition, the circuit estimator benchmarks the accumulator, pooling unit, and buffer circuits in the architecture. The estimator utilizes the device models and the transistor properties to perform the overall estimation.

2.4.3 Architecture

Fig. 6 shows the overall architecture utilized in this work. The architecture utilized is similar to that in Krishnan *et al.* (2020b) with a homogeneous crossbar size. The top-level consists of an array of IMC tiles interconnected by an NoC-mesh. Each tile consists of an array of PEs connected by an H-Tree interconnect. The PE consists

of an RRAM-based IMC crossbar and associated peripheral circuits. The peripheral circuits include an ADC, column MUX, switch matrix, WL decoder and driver, level shifters, and SL and BL MUX. In addition, the PE consists of a local buffer that is utilized for the movement of activations and partial sums into and out of the PE.

The architecture utilizes an NoC-mesh to perform the on-chip data movement at the tile-level. Each tile is associated with an NoC router that performs the packet scheduling and routing. The NoC utilizes an X-Y routing mechanism. To benchmark the NoC interconnect, we utilize a cycle-accurate simulator that is benchmarked against the NoC module within SIAM Krishnan *et al.* (2021e). To perform the estimation, we generate traces for the packets communicated between the tiles similar to that detailed in Krishnan *et al.* (2021e). These traces are then utilized as the input to the NoC estimator to evaluate the cost of on-chip communication.

2.4.4 Algorithm

The algorithm component of the simulator performs the DNN training, pruning, quantization, variation-aware-training (VAT), evaluation of model stability, and hardware-aware training. The VAT training performed in this work utilizes the device models detailed in Table 2. We utilize the log-normal distribution to add the variations for each weight. The variations are added such that the variations depend on the weight value, thus having one-to-one correlation to the real hardware. In addition, for the hardware-aware training we include the effect of the limited precision of the RRAM, the ADC, and the accumulator within the shift and add circuit. Finally, the algorithm component of the simulator evaluates the model stability of the DNN after training. To achieve this, we evaluate the roughness score of the loss function and

visualize the loss landscape as detailed Section 2.2. In the following sections we detail the pruning and quantization methodologies utilized in this work.

2.4.4.1 Pruning

In this work, we adopt the structured pruning method in Yang *et al.* (2020). It utilizes a weight-penalty clipping with a self-adapting threshold, as shown below:

$$\hat{\mathcal{L}} = \mathcal{L}(f(\mathbf{x}; \{\mathbf{W}_l\}_{l=1}^L), \mathbf{t}) + \lambda \sum_{l=1}^L \sum_{i=1}^{G_l} \min(\|W_{l,i}\|_2; \delta_l) \quad (2.3)$$

$$\delta_l = \alpha \cdot \frac{1}{G_l} \sum_{i=1}^{G_l} \|\mathbf{W}_{l,i}\|_2 \quad (2.4)$$

where δ_l denotes the layer-wise self-adapting clipping threshold, L is the number of layers, G_l is the number of groups in the l -th layer, λ is the hyper-parameter to be tuned based on the dataset, and α is the scaling coefficient. The pruning is conducted group-wise along the output channel dimension: for layer l with weight matrix $W_l \in \mathbb{R}^{N_{of} \times N_{if} \times K_x \times K_y}$, we choose a group of size N_g along the N_{if} dimension, where N_g is determined by the crossbar size. Groups of $K_x \times K_y \times N_g$ weights are pruned across output channels to favor the IMC.

2.4.4.2 Quantization

The pruned model is further compressed by applying quantization. For 4-bits or higher precision, we employ uniform in-training quantization Zhou *et al.* (2016). Furthermore, for ternary bit precision, we follow the ternarization method in He *et al.* (2019a). For both ternary and higher bit precision, we employ the straight-through-estimator (STE) method in the backward-propagation to counteract the

non-differential issues of the discrete quantization function. In this work, we focus on 8-bit and ternary weight quantization.

2.5 Model Stability for RRAM-based IMC

In this section, we detail the algorithm utilized to evaluate the model stability of DNN under the presence of RRAM variations, sparsity, and quantization. Algorithm 1

Algorithm 1: Model Stability

```

1 Input: DNN, weight precision ( $W$ ), activation precision ( $A$ ), RRAM training
   variations ( $\sigma_{train}$ ), RRAM testing variations ( $\sigma_{test}$ ), crossbar size, and ADC
   precision
2 Output: Roughness Score ( $R_s$ ) and loss landscape
3 for DNN Model do
4   Perform DNN training
   /* Model A */
5   if Quantize then
6     Perform DNN quantization (Section 2.4.4.2)
     /* Model B */
7   end
8   if Pruning then
9     Perform pruning for DNN Model as in Equation 3.1 and Equation 3.2
     /* Model C */
10  end
11  Add RRAM variations ( $\sigma_{train}$ ) and perform hardware-aware training
   /* Model D */
12  Plot loss landscape using tool in Li et al. (2018) for models A, B, C, and D
13  Calculate roughness score of loss landscape for models A, B, C, and D
14 end

```

details the methodology utilized in this work to evaluate the model stability. First, for each DNN model we perform training to generate the floating-point model A. We perform inference with model A to calculate the inference accuracy. Next, we quantize the DNN model to fixed-point weights and activations across all layers

of the DNN, to generate model B. Uniform quantization is employed to maximize the hardware performance for the RRAM-based IMC architecture. The quantized model is then pruned to generate the structured sparse DNN model C. Thereafter, we add the RRAM variations (σ_{train}) and perform hardware-aware training for the quantized model to generate model D. In addition to adding the RRAM variations, hardware-aware training involves breaking the convolution or fully-connected (FC) layer into partial convolutions and FC layer operations based on the size of the crossbar and adding the ADC quantization for the column sum from each crossbar. We then perform inference for the hardware-aware trained model D in the presence of the RRAM testing variations (σ_{test}) to generate the realistic hardware accuracy. Next, to evaluate the model stability for each model, we plot the loss landscape as defined in Section 2.2.1. Finally, we evaluate the roughness score of the loss landscape (models A, B, C, and D) to quantify the stability of the DNN model. In this work, we propose to use model stability as a metric for reliable RRAM-based IMC accelerations. To achieve this, we propose two directions, first, a model stability-based model selection, and second, a model stability-based VAT method.

Given a dataset, Fig. 2 illustrates that the choice of the DNN model significantly affects the accuracy. Based on this observation, we propose a novel loss landscape-based model selection for stability that tolerates RRAM device variations and achieves higher post-mapping accuracy. Such an observation is attributed to the DNN model stability. Model stability of a trained DNN is the intrinsic property to withstand perturbations such as variations and noise. A more stable model with higher model stability will be more robust under RRAM variations and have higher post-mapping accuracy.

Model stability-based model selection provides a viable solution when there is a

choice for the target DNN. But, if the DNN cannot be changed and is compressed, model selection cannot be utilized. To address this, in this work, we propose a novel model stability-based VAT to improve the post-mapping accuracy of DNNs under sparsity and quantization. Previous VAT approaches focus on non-pruned DNNs and require the precise knowledge of RRAM testing variations and apply that to the training Long *et al.* (2019); Charan *et al.* (2020a). Distinct from that, we first train the sparse and quantized DNN model with different scales of device variations (σ_{train}), without knowing the exact amount of RRAM testing variations. The range of σ_{train} is from 0.1 to 0.5, as suggested by the 65nm 1T1R RRAM data. Furthermore, we evaluate the loss landscape and the roughness score for each of the different VAT variants to help identify the optimal model with the highest model stability (Algorithm 1). A higher model stability from the optimal scale of training variation leads to higher post-mapping accuracy.

2.6 Experiments and Results

2.6.1 Experimental Setup

We evaluate the proposed model stability metric for reliable RRAM-based IMC acceleration using two main methods. First, we demonstrate a DNN model selection for higher model stability which improves the overall DNN accuracy. We evaluate the proposed method for ResNet-20 on CIFAR-10, DenseNet-40 for CIFAR-10 and CIFAR-100, and ResNet-32 for CIFAR-100. All experiments are done for a crossbar size of 256×256 with a 5-bit ADC at the periphery. We evaluate for different RRAM levels ranging from 2 to 16.

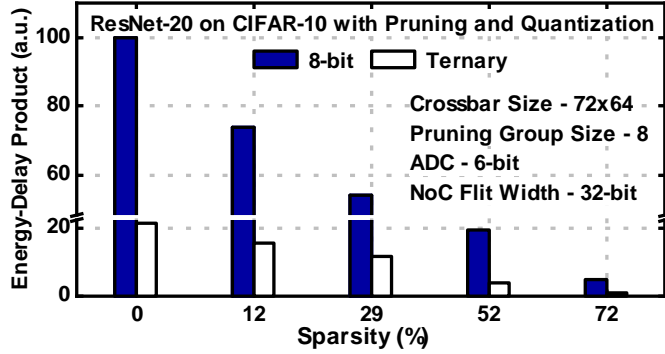


Figure 7. Normalized Energy-Delay Product (EDP) at Different Sparsity for ResNet-20 on CIFAR-10 at 8-bit and Ternary Precision.

Next, we extend the model stability metric to present a novel model stability-based VAT method to mitigate the accuracy degradation in RRAM-based IMC architectures. We evaluate the proposed VAT method for ResNet-20 on CIFAR-10, VGG-16 on CIFAR-10, ResNet-32 on SVHN, and ResNet-56 for CIFAR-100. We evaluate the VAT method in the presence of RRAM variations that range from 0.1 to 0.5, ternary and 8-bit quantization, and different levels of structured sparsity generated using pruning method detailed in Section 2.4.4.1. The pruning group size is chosen equal to the crossbar size for maximum hardware inference performance. We utilize the same crossbar size of 256×256 with a 5-bit ADC at the periphery.

2.6.2 Pruning and Quantization

2.6.2.1 Effect of Pruning and Quantization on RRAM IMC

We follow the mapping as in Krishnan *et al.* (2021e). In the pruning method, we set the group size in accordance with the number of rows of the crossbar. For example,

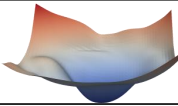
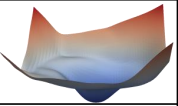
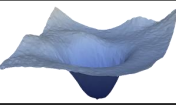
Model	Baseline (32-bit)	Pruning Only (29%)	Pruning (29%) & 8-bit
Landscape			
Roughness Score ($\times 10^{-3}$)	172	192	130
Accuracy (%)	92.35	92.16	92.62

Figure 8. Loss Landscape for Floating-Point 32-bit, Pruned Only (29%), and Pruned and Quantized (29% and 8-bit) Variants of ResNet-20 on CIFAR-10. Model Pruning Results in a Model with Lower Stability and Accuracy, while Quantization to a Low-Precision Model Improves the Stability and the Roughness Score, Leading to Higher Accuracy.

for a crossbar of size 72×64 and kernel size of 3×3 , we set the group size to be 8. Hence, we prune groups of $3 \times 3 \times 8$ weights along the output feature dimension. Therefore, we are able to skip the mapping of $3 \times 3 \times 8$ weights along the column dimension of the RRAM crossbar while maintaining high utilization. In this work, we set the group size to be 8 and the crossbar size as 72×64 . Fig. 7 shows the energy-delay product for ResNet-20 on CIFAR-10 with pruning and quantization. At higher rates of sparsity, the EDP reduces exponentially across different grades of quantization, thus increasing the hardware performance.

Fig. 8 shows the loss landscape, roughness score, and accuracy for the floating-point 32-bit (FP-32), pruning only (29%), and pruned and quantized (29% and 8-bit) versions of ResNet-20 on CIFAR-10. Pruning and quantization follow the methodology detailed in Sections 2.4.4.1 and 2.4.4.2, respectively. The pruned model has a roughness score higher than the FP-32 model, resulting in a rougher loss landscape, less stability, and lower accuracy. At the same time, the addition of quantization to the pruned model results in a reduced roughness score, making it more stable with a smoother loss landscape and a higher accuracy. Hence, we quantitatively establish through

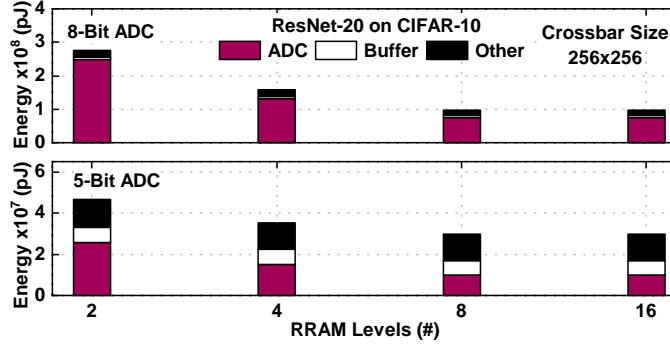


Figure 9. ADC Dominates the Energy Consumption, Especially Under High ADC Precision. With Higher RRAM Levels, its Portion Reduces Due to Reduced Number of Crossbars and Associated Peripherals.

the roughness score and loss landscape that quantization helps improve the model stability and is a necessary step for reliable RRAM-based IMC acceleration of sparse DNNs.

2.6.3 System-Level IMC Analysis

2.6.3.1 Precision and Variation

The inherent variations with the RRAM device result in significant accuracy degradation. Section 2.1 details the effect of RRAM variation for ResNet-20 on CIFAR-10 with full precision and pruned and quantized models (8-bit and ternary). Through this, we establish that higher RRAM levels lead to higher variation and degradation in post-mapping accuracy.

Next, we evaluate the effect of ADC precision on post-mapping accuracy. Fig. 9 shows the total inference energy breakdown for ResNet-20 on CIFAR-10 at two ADC precisions, 8-bit and 5-bit. We divide the total energy into ADC, buffer, and other

(accumulator, NoC, crossbar, ReLU, pooling, etc.) components. It can be seen that a higher ADC precision leads to higher energy dominated by the ADC and higher post-mapping accuracy. At the same time, a lower precision reduces the ADC component resulting in reduced total energy. At higher RRAM levels, the ADC cost reduces due to reduced crossbars and associated peripherals. Considering the dramatic design challenge and cost, a low-power and high throughput, and high-precision ADC may not be practical soon for RRAM IMC.

2.6.3.2 Roofline Model

In this section, we develop a roofline model that comprises of the number of RRAM levels, RRAM variation, stuck-at-faults, $R_{\text{off}}/R_{\text{on}}$ ratio, and ADC precision. We evaluate the post-mapping accuracy for two DNNs, DenseNet-40 and ResNet-32 for the CIFAR-100 dataset, for our fabricated HfO_2 based RRAM device.

Fig. 10 shows the roofline model for the two DNNs, DenseNet-40 and ResNet-32, on the CIFAR-100 dataset. The black curve shows the post-mapping accuracy for different RRAM levels with (black) and without considering the ADC precision (grey). A higher number of RRAM levels leads to lower post-mapping accuracy and vice-versa. A similar trend is seen for both ResNet-32 and DenseNet-40. Next, we consider the ADC precision in the accuracy estimation and evaluate the post-mapping accuracy. The red curve shows the maximum achievable post-mapping accuracy with a 5-bit ADC. Hence, the RRAM-based IMC accuracy at lower RRAM levels is limited by the ADC precision, while at higher RRAM levels, the RRAM device limits the accuracy. Finally, for our 16-level 65nm 1T1R RRAM devices, only 4-6 levels are useful to achieve

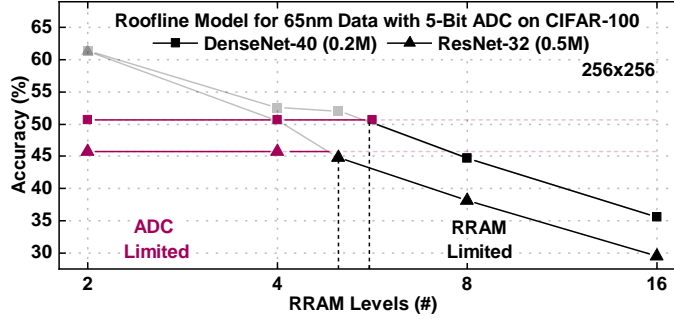


Figure 10. ADC Precision and RRAM Device Variations Govern the Maximum and Realistic Post-Mapping Accuracy. A More Stable DNN Model Improves the Realistic Post-Mapping Accuracy.

the best performance due to the ADC precision, RRAM variations, and algorithm limits.

2.6.4 Model Stability-based Model Selection

Table 3. Roughness Score and Post-Mapping Accuracy for Different DNN Models Under RRAM Variation and a 5-bit ADC. A More Stable Model (Lower Roughness Score) Effectively Improves the Accuracy, More Than That by Reducing RRAM Variation by 50% Only.

Model Size (Size)	Variation (σ)	Roughness Score ($\times 10^{-3}$)	Accuracy (%)
CIFAR-10			
ResNet-20 (0.3M)	0.3*	278	68.83
	0.15		72.04
DenseNet-40 (0.2M)	0.3*	173	72.34
CIFAR-100			
ResNet-32 (0.5M)	0.3*	130	36.76
	0.15		43.41
DenseNet-40 (0.2M)	0.3*	121	44.68

In this section, we show the efficacy of the model stability-based model selection

method. Here the training and testing RRAM variations are the same (σ) Charan *et al.* (2020a). Table 3 shows the post-mapping accuracy of various DNNs for CIFAR-10 and CIFAR-100 datasets. We evaluate ResNet-20 and DenseNet-40 for CIFAR-10. For our 65nm RRAM device with a device variation (σ) of 0.3, ResNet-20 achieves 68.83% accuracy. A 50% reduction in RRAM variation ($\sigma = 0.15$) through process control results in 72.04% accuracy, a 4% increment. At the same time, DenseNet-40, which has higher model stability, achieves a higher accuracy of 72.34% at a σ of 0.3, a 0.3% and 4% improvement over ResNet-20 with 0.15 and 0.3 σ , respectively. For CIFAR-100 dataset, we evaluate ResNet-32 and DenseNet-40. For our 65nm RRAM device, ResNet-32 achieves 36.76% and 43.41% post-mapping accuracy at σ of 0.3 and 0.15, respectively. DenseNet-40 (more stable model) achieves 44.68% post-mapping accuracy at a σ of 0.3, a 1.27% improvement over ResNet-32 at of 0.15 (7.9% higher than ResNet-32 at σ of 0.3). We note that the more stable model has a smaller model size and attributes to improved hardware performance. The improved accuracy is attributed to the lower roughness score and higher model stability from the proposed loss landscape-based model selection. Through this, we establish that a loss landscape-based model selection achieves higher post-mapping accuracy than a 50% reduction in RRAM device variation through process control.

2.6.5 Model Stability-based VAT

In this section, we detail the efficacy of the proposed model stability-based VAT method. Fig. 11 shows the loss landscape, roughness score, and post-mapping accuracy for ResNet-20 at 29% sparsity and 8-bit quantization for a testing variation (σ) of 0.1


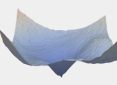
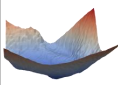
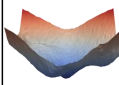
Training Variation (σ)	0.1	0.15	0.2	0.3
Landscape				
Roughness Score ($\times 10^{-3}$)	94	91	120	139
Post-mapping Accuracy (%)	84.7	86.0	72.3	62.8

Figure 11. Loss Landscape for ResNet-20 on CIFAR-10 Trained With Different Device Variations at 29% Sparsity and 8-bit Quantization, Tested at 0.1 Variation. The Optimal Scale of Training Variation (σ) of 0.15 has the Lowest Roughness Score and Smoother Loss Landscape and Hence, Higher Model Stability and Post-Mapping Accuracy.

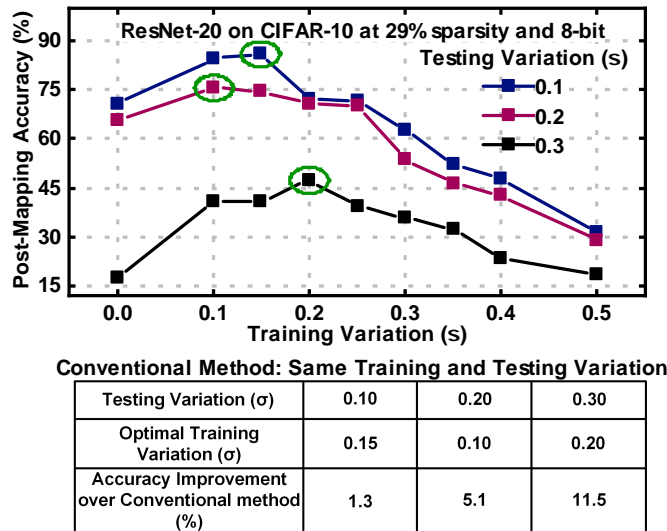
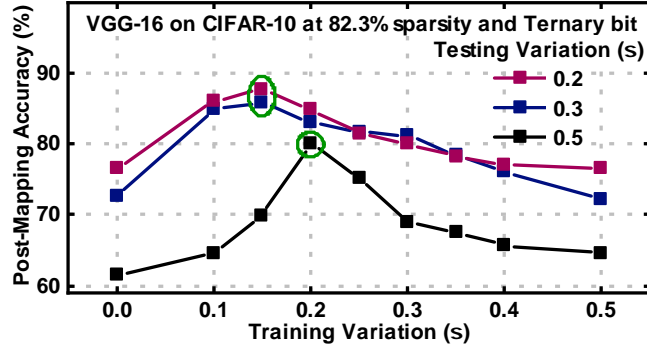


Figure 12. Post-Mapping Accuracy for ResNet-20 on CIFAR-10 at 29% Sparsity and 8-bit Precision for Three Different RRAM Variations Under Test. The Optimal Scale of Training Variation (Green Circle) has the Lowest Roughness Score and Highest Post-Mapping Accuracy.

on the CIFAR-10 dataset. The model is trained with different scales of variations (σ) from 0.1 to 0.3 to generate each of the VAT models. The most stable VAT model, σ equal to 0.15, has the lowest roughness score of 91×10^{-3} , resulting in a smoother loss landscape, higher model stability, and post-mapping accuracy. As the roughness score increases, the loss landscape becomes more rough, and the post-mapping accuracy



Conventional Method: Same Training and Testing Variation

Testing Variation (σ)	0.20	0.30	0.50
Optimal Training Variation (σ)	0.15	0.15	0.20
Roughness Score ($\times 10^{-3}$) (Conventional, Optimal)	(172, 147)	(584, 454)	(750, 665)
Accuracy Improvement over Conventional method (%)	2.9	4.6	15.4

Figure 13. Post-Mapping Accuracy for VGG-16 on CIFAR-10 at 82.3% Sparsity and Ternary Precision for Three Different RRAM Testing Variations (σ). The Optimal Model (Green Circle) has the Lowest Roughness Score Resulting in Higher Accuracy.

reduces. Fig. 12 shows the detailed result. The optimal scale of training variation is different for each testing variation and is chosen based on the model stability. *Thus, the proposed method is also applicable to situations with unknown precise RRAM testing variation.*

Table 4. Post-Mapping Accuracy and Improvement for VGG-16 on CIFAR-10 (82.3% Sparsity and Ternary Precision). Conventional Method: Same Training and Testing Variation

Testing Variation	Optimal Training Variation (σ)	Post-Mapping Accuracy (%)		Accuracy Improvement (%)
		Conventional Charan <i>et al.</i> (2020a)	Optimal	
0.1	0.15	87.9	88.5	0.6
0.2	0.15	84.8	87.7	2.9
0.3	0.15	81.2	85.8	4.6
0.4	0.10	64.9	83.9	19.0
0.5	0.20	64.6	80.0	15.4

Table 5. Comprehensive Results Using the Proposed Model Stability-based VAT Method. Conventional Method: Same Training and Testing Variation

Dataset	Network (Size)	Quantization	Sparsity (%)	Testing Variation	Optimal Training Variation (σ)	Post-Mapping Accuracy (%)		Accuracy Improvement (%)
						Conventional	Optimal	
CIFAR-10	ResNet-20 (0.27M)	8-Bit	29.0	0.30	0.20	35.7	47.2	11.5
	VGG-16 (15M)	Ternary	82.3	0.40	0.10	64.9	83.9	19.0
			88.3	0.20	0.10	84.7	86.9	2.2
SVHN	ResNet-32 (0.45M)	Ternary	48.4	0.15	0.20	91.0	91.6	0.6
			71.5	0.30	0.15	75.1	86.3	11.2
CIFAR-100	ResNet-56 (0.85M)	8-Bit	17.8	0.15	0.10	30.5	51.6	21.1

We repeat the same experiment for VGG-16 on CIFAR-10 with ternary quantization and 83% sparsity as shown in Fig. 13. Table 4 shows the detailed results for VGG-16 on CIFAR-10 across the entire range of testing variations. Conventional methods refer to using the same scale of variation for both training and testing Long *et al.* (2019); Charan *et al.* (2020a). In contrast, in this work, we show that an optimal scale for training variation results in higher model stability and post-mapping accuracy. Furthermore, at a higher range of testing variations, the proposed method provides greater improvement in post-mapping accuracy. Hence, the systematic model stability-based VAT method is effective in choosing the optimal VAT model at different precision and sparsity for best accuracy across a range of DNN models.

2.6.5.1 Overall Results with Proposed VAT

Table 5 shows the overall results across different models and datasets. All post-mapping accuracy is compared to that of conventional VAT, where the training and testing variations are the same. ResNet-20 on CIFAR-10 at 29% sparsity and 8-bit precision shows 11.5% improvement in post-mapping accuracy with the proposed method at 0.3 testing variation. At the same time, VGG-16 at 88.3% and 82.3% sparsity and ternary bit quantization achieve 2.2% and 19% improvement post-mapping

Table 6. Post-Mapping Accuracy of VGG-16 on CIFAR-10

Method	Sparsity (%)	Quantization	Accuracy (%)	
			Baseline	Post-Mapping (Average)
DFP+DVA Long <i>et al.</i> (2019)	-	8-Bit	93.85	80.1
Ours	83.2	Ternary		85.2

accuracy for 0.2 and 0.4 testing variations (σ), respectively. We evaluate ResNet-32 on SVHN dataset at ternary quantization and two sparsity (48.4% and 71.5%) and achieve up to 11.2% improvement in post-mapping accuracy. Finally, for ResNet-56 for CIFAR-100, at 17.8% sparsity, 8-bit quantization, and testing variation of 0.15, we achieve a 21.1% improvement in post-mapping accuracy.

2.6.5.2 Comparison with Other Work

We compare the proposed model-stability-based VAT method with the state-of-the-art method as proposed in Long *et al.* (2019). Table 6 shows the post-mapping accuracy for VGG-16 on CIFAR-10. The proposed method achieves a 5.1% higher average improvement (as defined in Long *et al.* (2019)) in post-mapping accuracy as compared to Long *et al.* (2019). Furthermore, the proposed method provides an improvement in the presence of structured sparsity, which reduces the model stability due to the presence of more sensitive weights. Finally, the proposed method does not require precise prior knowledge of the testing variations (instead requires only the expected range), hence providing a more generic solution for reliable RRAM-based IMC acceleration.

2.7 Conclusion

In this work, we explore the model stability of DNNs as a metric for reliable RRAM-based in-memory acceleration. Utilizing the loss landscape and roughness score, we show that a more stable model has a lower roughness score, a smoother loss landscape, and higher accuracy under variations. To provide realistic evaluation, we measured statistical variations from a 65nm 1T1R RRAM test chip and integrated them into a cross-layer benchmark tool to access model accuracy and other performance metrics under variations. Based on the model stability of DNNs, we propose two methods to achieve reliable RRAM-based in-memory acceleration. First, a novel model stability-based model selection that effectively tolerates RRAM device variations and achieves higher accuracy than that with 50% lower RRAM variations for both CIFAR-10 and CIFAR-100 datasets. Second, we propose a variation-aware training (VAT) method to mitigate the post-mapping accuracy loss in sparse and quantized DNNs. We conclude that quantization improves the stability under variations, leading to higher accuracy, but pruning reduces the model stability. The proposed VAT method searches for the most stable model to mitigate the post-mapping accuracy loss without pre-knowledge of testing RRAM variations and no re-training during mapping. Experimental evaluation shows up to 19%, 21%, and 11% improvement in post-mapping accuracy at different sparsity, quantization, and device variations on CIFAR-10, CIFAR-100, and SVHN datasets, respectively.

HYBRID RRAM/SRAM IN-MEMORY COMPUTING FOR
ROBUST DNN ACCELERATION

3.1 Introduction

RRAM device suffers from several non-idealities such as limited resistance levels, device-to-device write variations, stuck-at-faults, and limited $R_{\text{off}}/R_{\text{on}}$ ratio, posing a significant challenge to designing reliable RRAM-based IMC architectures Long *et al.* (2019); Ma *et al.* (2020); Chakraborty *et al.* (2020); Charan *et al.* (2020a); Sun *et al.* (2021); Krishnan *et al.* (2021f); Joshi *et al.* (2020); Yang *et al.* (2021). The non-idealities within the RRAM device results in a deviation of the programmed weight values (resistance value), causing a significant reduction in post-mapping accuracy for DNNs. Furthermore, the crossbar structure of the IMC, with its limited array size, requires splitting of the large convolution (conv) or fully-connected (FC) layers into partial operations. Such partial operation (conv/FC) results in further error due to the limited precision of the peripheral circuits of the RRAM-based IMC crossbar.

To mitigate the post-mapping accuracy loss in DNNs, variation-aware training (VAT) and special encoding schemes are employed Long *et al.* (2019); Charan *et al.* (2020a); Chakraborty *et al.* (2020); He *et al.* (2019b); Ma *et al.* (2020); Sun *et al.* (2021). VAT exploits the inherent DNN redundancy by embedding the device variations (σ), based on a log-normal or normal distribution model, into the training process to achieve a variation-tolerant model Long *et al.* (2019); Charan *et al.* (2020a); Chakraborty *et al.* (2020); He *et al.* (2019b); Ma *et al.* (2020); Sun *et al.* (2021). To further understand

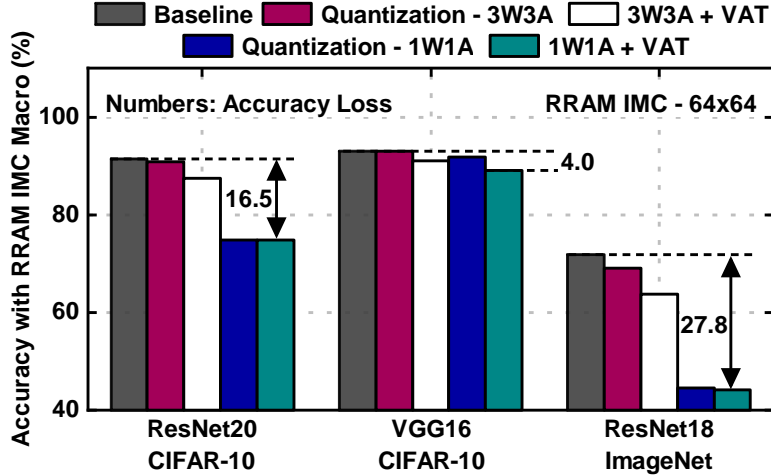


Figure 14. Accuracy with RRAM IMC Macro for Three Different DNNs for Both CIFAR-10 and ImageNet Datasets. The Baseline Model Deals with a 32-bit Floating-Point Model, Quantization Refers to Fixed-Point Precision for Activation and Weights (e.g., 3W3A Means 3-bit Weight and 3-bit Activation), and VAT Refers to Variation-Aware Training with the RRAM Variations Long *et al.* (2019).

VAT, we evaluate the post-mapping accuracy for three DNNs across CIFAR-10 and ImageNet datasets. We perform in-training quantization for both weights Zhou *et al.* (2016) and activations Choi *et al.* (2018). In addition, we extract RRAM device variation from our 65nm SUNY 1T1R device (average variation (σ_{avg}) of 0.3) to perform VAT. Finally, we perform a hardware-aware training for the DNN by splitting the conv and FC layers into partial operations based on the IMC crossbar size (we use 64×64 He *et al.* (2019b)). Fig. 14 shows the accuracy of the quantized VAT trained RRAM IMC macro. Although VAT improves the accuracy, it does not achieve the same accuracy as the baseline 32-bit floating-point (FP-32) model, with up to 27.8% accuracy loss for ImageNet class DNNs. Hence, there is an urgent need to bridge this gap in the accuracy for RRAM-based IMC acceleration of DNNs.

To address this, in this work, we first propose a hybrid RRAM/SRAM IMC architecture for robust DNN acceleration. The proposed hybrid architecture utilizes

an RRAM-based IMC macro, an SRAM-based macro, and a programmable shifter. The RRAM macro consists of the RRAM IMC crossbar, decoder, and associated peripheral circuits. The SRAM macro consists of an SRAM memory array, buffers, and an output stationary CMOS-based multiply-and-accumulate (MAC) engine. The output from the SRAM macro (clean) is added to the noisy RRAM macro output to create an ensemble model and achieve bit-level compensation. Furthermore, the degree of compensation is controlled by utilizing a programmable shifter. Depending on the DNN, different scales of the shift operation are performed on the SRAM macro output to achieve varying degrees of compensation for the RRAM macro output. To illustrate the efficacy of the architecture, we design a test-chip in the 65nm SUNY process Liehr *et al.* (2020) to demonstrate the proposed hybrid RRAM/SRAM IMC architecture. We utilize an RRAM macro with a 64×64 IMC and associated peripheral circuits, and a dedicated control logic. Furthermore, we design an SRAM macro with a 32×64 SRAM memory array and an output stationary MAC engine Ma *et al.* (2017a). Finally, a custom control logic is designed to synchronize both RRAM and SRAM macro. The programmable shifter is implemented outside the chip for simplicity.

Next, on the algorithm side, we develop a framework for the training of the DNNs to support the hybrid IMC architecture. The proposed framework performs in-training quantization for both weights and activations following Zhou *et al.* (2016); Choi *et al.* (2018), structured pruning Yang *et al.* (2020), RRAM IMC-aware training, and support for different compensation scales through the programmable shifter. The parallel SRAM macro addition is performed in a layer-wise manner. In addition, structured pruning is performed on the SRAM macro weights to achieve minimal hardware overhead. For efficient hardware execution, the precision of the RRAM and SRAM macros, activations, and shift scale are kept constant across all layers of the DNN. To

accurately model the RRAM device in the framework, RRAM data is collected from a fully integrated 1T1R structure on a 300mm wafer, using a custom RRAM module within the SUNY 65nm process. We plan to open-source the algorithm framework upon acceptance of this work.

We perform extensive experiments on four DNNs across CIFAR-10 and ImageNet datasets at different precision (weights and activations). We show that at higher RRAM macro precision and SRAM macro precision, the proposed method achieves near FP-32 accuracy (at $1\times$ and $2\times$ RRAM variations of 65nm RRAM device). Furthermore, we show that the proposed hybrid IMC architecture with SRAM compensation opens up the opportunity for a realistic IMC architecture with multi-level RRAM cells (MLC) even though they suffer from high variations. Compared to state-of-the-art methods, the proposed hybrid IMC architecture achieves up to 21.9%, 12.65%, and 6.52% improvement in post-mapping accuracy with minimal overhead for ResNet-20 on CIFAR-10, VGG-16 on CIFAR-10, and ResNet-18 on ImageNet, respectively.

In addition, we analyze the overhead from the SRAM macro and programmable shifter. In terms of training time, the proposed method incurs up to a 25% increase in training time compared to the VAT method. We evaluate the hardware cost overhead for the SRAM macro by extracting the post-layout area and power measurements from the 65nm test-chip. The proposed hybrid IMC architecture achieves small overhead in terms of memory requirement (up to 24%), area (up to 20%), and power (up to 2.6%) across different DNNs and datasets. The major contributions of this work are as follows:

- We propose a novel hybrid RRAM/SRAM IMC architecture that utilizes an RRAM IMC macro with MLC cells, SRAM macro, and a programmable shifter for robust DNN acceleration,

- We develop a training framework to enable the hybrid IMC architecture that supports quantization, structured pruning, RRAM IMC-aware training, and different compensation scales through the programmable shifter,
- *Experimental evaluation of the hybrid IMC architecture shows that the SRAM compensation opens the opportunity for a realistic IMC architecture with multi-level RRAM cells.* Compared to state-of-the-art methods, the proposed hybrid IMC architecture achieves up to 21.9%, 12.65%, and 6.52% improvement in post-mapping accuracy with minimal overhead for ResNet-20 on CIFAR-10, VGG-16 on CIFAR-10, and ResNet-18 on ImageNet, respectively.
- Finally, we design a test-chip using the 65nm SUNY process to demonstrate the proposed hybrid IMC architecture and analyze the hardware performance.

3.2 Hybrid IMC Architecture

In this section, we detail the hybrid IMC architecture proposed in this work. Fig. 15 shows the overall block diagram of the proposed hybrid IMC architecture for one RRAM IMC and SRAM macro. The architecture consists of an RRAM IMC macro, an SRAM macro, a programmable shifter, adder, buffers, and associated control logic. The proposed hybrid IMC architecture utilizes an ensemble of the output from the RRAM IMC macro and the SRAM macro combined using a programmable shifter and an adder circuit. Each macro functions independently with the associated control logic. The control logic also handles the scale of the shifter to facilitate different compensation of the RRAM macro by the SRAM macro. Finally, each layer of the DNN utilizes a number of RRAM IMC and SRAM macros to perform the DNN acceleration.

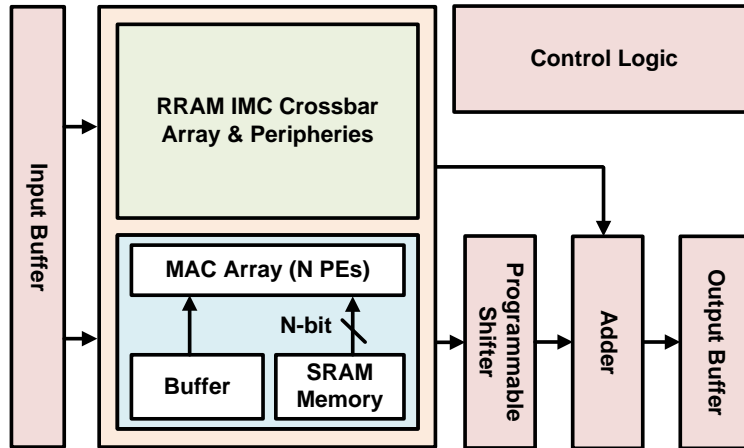


Figure 15. Block Diagram of the Proposed Hybrid RRAM/SRAM IMC Architecture. Both the RRAM and SRAM Macros Compute in a Parallel Manner to Generate the Output. A Programmable Shifter Allows for Different Scales of Compensation Using the SRAM Macro. The Overall Output is an Ensemble of the RRAM and SRAM Macro Outputs.

3.2.1 RRAM IMC Macro

Fig. 16 shows the architecture of the RRAM IMC macro. The RRAM macro consists of an RRAM-based IMC crossbar structure of a specific size. Each cross-point in the array consists of a 1T1R RRAM multi-level cell. The 1T1R cell connects the transistor (gate) to the wordline (WL), while the two terminals of the RRAM are connected to the select-line (SL) and the bitline (BL). The RRAM-based IMC crossbar utilizes analog domain computation to perform the MAC operation. Each IMC crossbar has associated peripheral circuits such as analog-to-digital converter (ADC), PMOS header, column multiplexer, BL and SL mux, WL driver and level shifters, and RRAM decoder. The column multiplexer is used to share the read-out circuit (ADC and PMOS header) across multiple columns of the IMC crossbar array. The ADC performs the conversion of the analog output to the digital domain. In

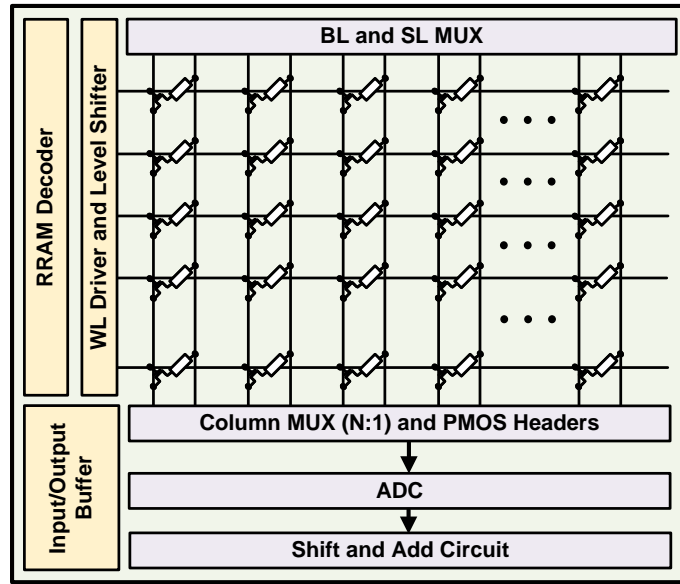


Figure 16. Block diagram of the RRAM IMC Macro Within the Hybrid Architecture. The Macro Consists of a Crossbar Array of RRAM Cells, a Decoder, PMOS Headers, Column Multiplexers (MUX), BL and SL Mux, Shift and Add Circuit, and ADC.

this work, instead of employing a digital-to-analog converter (DAC), we perform bit-serial computing over multiple cycles for multi-bit activations. The ADC output is accumulated based on the input bit significance using shifter and adder circuits to compute the MAC output. Finally, the overall result is generated by accumulating the outputs from each IMC crossbar array.

3.2.2 SRAM+MAC Engine Macro

Fig. 17 shows the block diagram of the SRAM macro within the hybrid IMC architecture. The SRAM macro consists of an array of processing elements (PE) of MAC engines, SRAM memory array, buffers, and associated control logic. The SRAM memory array stores the weights while the inputs are streamed into the PEs through

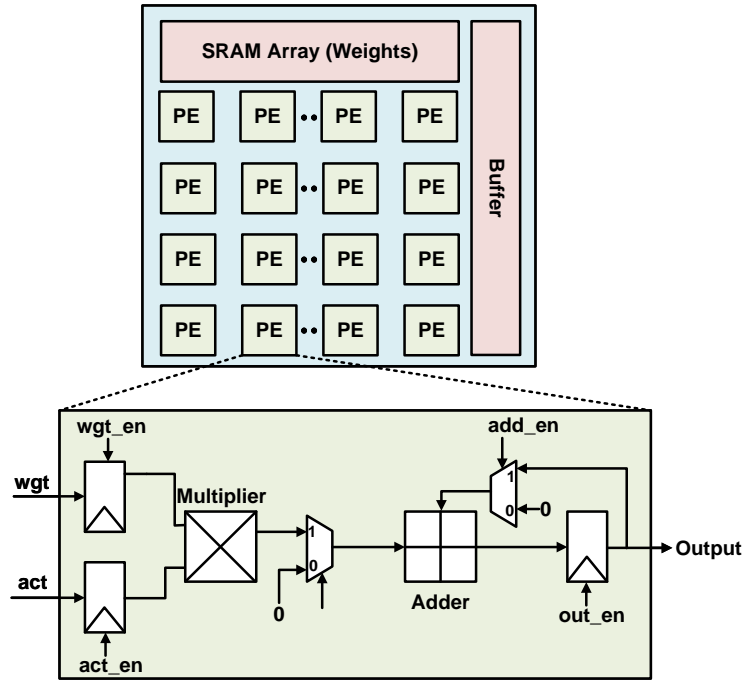


Figure 17. Block diagram of the SRAM Macros Within the Hybrid Architecture. An SRAM Array Stores the Weights While an Array of Processing Elements (PE) Perform the MAC Operations.

the buffer. Each PE utilizes an output stationary computation flow to reduce on-chip data movement. The multiplier and adder support the fixed-point MAC operations with the required precision for the SRAM macro output. The final output is obtained pixel-wise across feature maps and moved to the output buffer within the SRAM macro. Note that the read word size of the SRAM memory array is equal to the number of parallel PE to ensure maximum utilization and throughput. The SRAM macro performs the computations in parallel with the RRAM macro, thus avoiding any reduction in the overall hardware performance.

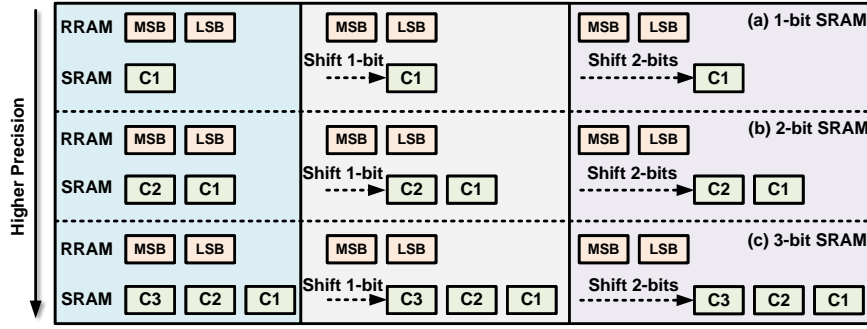


Figure 18. Functioning of the Programmable Shifter Within the Hybrid IMC Architecture. (a) 1-bit SRAM Macro Output Compensates for the MSB in the Blue Region, LSB in the Grey Region (1-bit Shift), and Adds an Extra Bit for Increased Precision in the Purple Region. A Similar Operation is Performed for Both (b) 2-bit and (c) 3-bit SRAM Macro Outputs.

3.2.3 Programmable Shifter

The programmable shifter performs the shift operation for the output from the SRAM macro, as shown in Fig. 15. The different scales of shifting allow for the compensation of different bits of the RRAM macro output. The shift operation follows a right shift within the hybrid IMC architecture. Fig. 18 shows an example of 1-bit, 2-bit, and 3-bit SRAM macro output compensation for a 2-bit RRAM macro output. We show three shift cases across all configurations. Fig. 18(a) shows RRAM macro output at 2-bit and SRAM macro output at 1-bit. First, the blue region shows the case when the SRAM macro output compensates only for the MSB of the RRAM macro output, i.e., no shift. Such compensation provides a larger impact as the position of compensation of the RRAM macro output has higher significance. Next, the grey region shows the case when the SRAM macro output compensates only the LSB of the RRAM macro output through a 1-bit shift. Finally, the purple region shows a 2-bit shift for the SRAM macro output, thus adding a higher precision for the RRAM

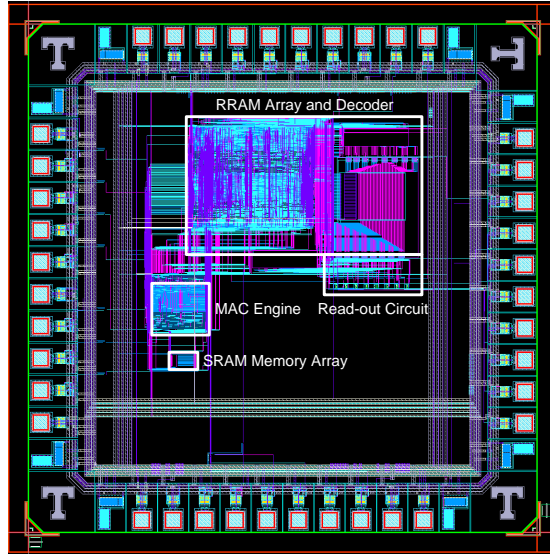


Figure 19. Layout of the 65nm Test-Chip of the Proposed Hybrid IMC Architecture. The Prototype Consists of a 64×64 1T1R RRAM Crossbar Array and Associated Peripheral Circuits. In Addition, the Prototype Implements an SRAM Memory Array of Size 32×64 with a 16-Word Size and 16 MAC Engines that Utilize an Output Stationary Dataflow.

macro output by adding an extra bit. In this case, neither of the RRAM output bits are individually compensated, while the extra bit provides an increased precision to the overall output. Similarly, Fig. 18(b) and Fig. 18(c) show the cases for 2-bit and 3-bit SRAM macro compensation. The optimal choice of the scale of shift depends on the DNN algorithm, dataset, the extent of RRAM device variations, and the hardware performance overhead.

3.2.4 65nm Hybrid IMC Test Chip

To demonstrate the efficacy of the proposed hybrid IMC architecture, we design a test-chip using a custom RRAM module within the 65nm SUNY process. Fig. 19

shows the layout of the 65nm test-chip of the hybrid IMC architecture. The test-chip consists of a RRAM macro, an SRAM macro, and associated testing structures such as scan chain. The shifter circuit is implemented outside the chip for higher testing flexibility. Finally, the test-chip is designed for an operating frequency of 100MHz.

3.2.4.1 RRAM IMC Macro

The RRAM macro consists of a 64×64 IMC crossbar array of 1T1R cells. In addition to the IMC crossbar array, peripheral circuits such as ADC, PMOS header, BL/SL/column multiplexers, WL driver and level shifter, and RRAM decoder are custom designed in the 65nm process. A 64-to-1 BL and SL one-hot multiplexer are utilized for the programming of RRAM cells. Furthermore, to share eight columns across each read-out circuit, an 8-to-1 column multiplexer is designed using a transmission gate and sized carefully to reduce the overall resistance of the circuit. The read-out circuit within the RRAM macro utilizes a flash ADC with a 3-bit resolution and a PMOS header. A single column (BL) combined with the PMOS header forms a resistance divider circuit, which converts the accumulated current to voltage. The voltage is then used as the input to the ADC, converting the analog output to the digital domain. Overall, eight read-out circuit instances are utilized across the 64 columns of the IMC crossbar array. Note that the PMOS header is sized appropriately to ensure that the resistance is much lower than the minimum resistance from the IMC crossbar array (with only one row turned on). Finally, input and output buffers are used to store the activations.

The RRAM decoder performs the overall control of the macro. The decoder utilizes a finite state machine (FSM) with three main states to generate the required

control signals. Furthermore, the decoder also performs the operation of driving the inputs to the WL through the driver and level shifters. During the write state, the RRAM cells can be programmed one-by-one by choosing a specific WL, BL, and SL. Next, the compute state performs the MAC operations in a parallel fashion across rows and columns. The sharing of the columns requires the column multiplexer to choose columns sequentially to generate the output. Hence, 8 cycles are required to perform the MAC operations with the 64×64 IMC crossbar array. Furthermore, during the compute state the decoder enables the ADC to perform the analog to digital conversion for the MAC output. The ADC output is then moved to the buffer that holds the value until it is moved outside the chip using the scan chain. Finally, the new input state is utilized within the decoder to accept the next stream of input activations.

3.2.4.2 SRAM Macro

The SRAM macro consists of an SRAM memory array, MAC engine array (PE array), and associated control logic. The SRAM memory consists of a 32×64 SRAM cells with a read-out word size of 16 bits. To match the SRAM memory read-out size, the MAC engine consists of an array of 16 PEs that implement an output stationary dataflow. The memory read out, and the number of computation units is matched to achieve the best performance and utilization. Next, a custom control logic is implemented utilizing an FSM. The FSM consists of 4 main states, namely, idle, load, compute, and finish. The idle state is the default state the SRAM macro assumes upon power-up of the chip. The load state is utilized to perform the loading of the weights and activations to the SRAM memory and local input buffer. The load state triggers

a counter that automatically moves the data from the scan chain to the corresponding memory/buffer. The SRAM memory with 64 columns is divided into 16 sets, with each column multiplexer servicing four columns of the array. Next, the compute state is utilized to perform the MAC operations. A counter is triggered such that it reads the weights from the SRAM memory and the input buffer to perform the MAC operation within the PEs. All PEs operate in a parallel fashion on different data, thus providing high throughput. Each PE performs the computations for a pixel within an output feature map. Through this, one pixel across 16 different output feature maps is generated. In addition, PE control is generated to enable the D flip-flops (DFF) at the input and the subsequent datapath to follow the output stationary dataflow. Finally, the finish state is utilized to denote the completion of the MAC operations and the transfer of the output data to the scan chain. We note that, for the increased flexibility of testing, we perform the computation of multi-bit inputs with bit-serial processing. The output is then post-processed outside the chip to obtain the final result.

3.3 Hybrid IMC Training Framework

In this section, we detail the algorithm framework developed for the hybrid RRAM/SRAM IMC architecture. The overall framework is developed using the Python programming language and the PyTorch deep learning framework.

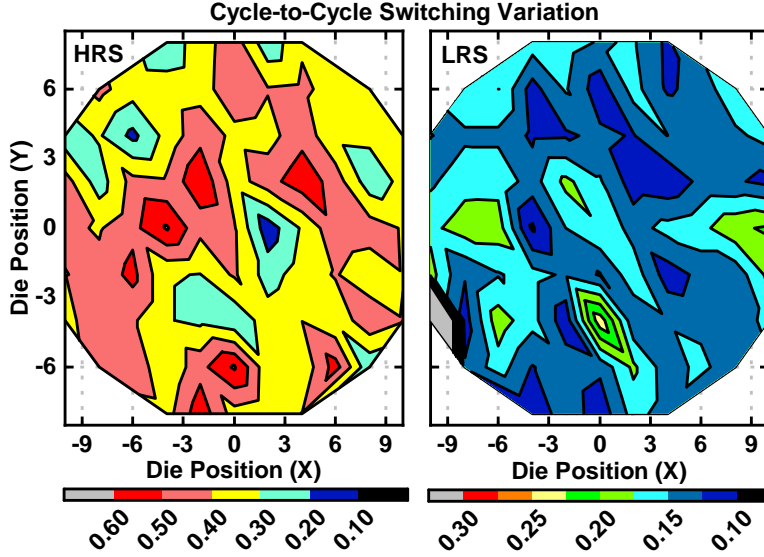


Figure 20. HRS (Left) and LRS (Right) Cycle-to-Cycle Switching Variation Across the 300mm Wafer at 65nm Liehr *et al.* (2020). HRS State has a Higher Variation Than LRS.

3.3.1 Statistical RRAM Device Models

To accurately model the RRAM device, data is collected from a fully integrated 1T1R RRAM structure on a 300mm wafer (at room temperature), using a custom RRAM module within the SUNY 65nm process Krishnan *et al.* (2021f). In this work, we focus on a multi-level RRAM device. Each RRAM device has a size of $100\text{nm} \times 100\text{nm}$ and utilizes a device stack comprised of a 6nm HfO_2 mem-resistive switching layer, a 6nm PVD Ti oxygen exchange layer (OEL), and TiN electrodes (top and bottom). Pulses with a magnitude of $1\text{V} - 1.2\text{V}$ and width of $10\mu\text{s}$ are used for the set/reset operation of RRAM devices.

Fig. 20 shows the wafer-level cycle-to-cycle switching variations for the 65nm RRAM device measured using the pulse-based switching technique. The high-resistance state

(HRS) has a higher variation up to 0.6σ , while the low-resistance state (LRS) has a lower variation up to 0.2σ . The average variation (σ_{avg}) for the entire range of HRS and LRS across the wafer amounts to 0.3. To further understand the RRAM variations, we analyze the distinct resistance levels that can be achieved for the 65nm 1T1R RRAM device. Fig. 21 shows the box and whisker plot of the measured resistance at different compliance currents for the multi-level 65nm RRAM device. The 65nm RRAM device achieves up to 8 distinct levels with 6 LRS and 2 HRS states. Furthermore, at the single device level, the LRS achieves a lower variation compared to HRS. Hence, the 65nm RRAM device can support up to 3-bit data.

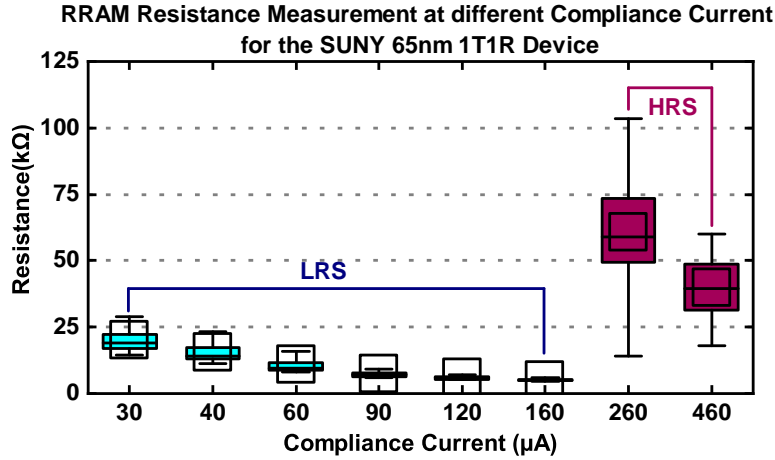


Figure 21. Box and Whisker Plot Showing the Eight Distinct Resistance Levels (6 LRS and 2 HRS) Within our 65nm 1T1R RRAM Device. Different Compliance Currents Lead to Different Resistance Levels.

Next, we analyze the variations for each bit when mapped to the 65nm 1T1R RRAM device. Table 7 summarizes the level-wise and bit-wise RRAM device variation up to 3-bits for the 65nm 1T1R RRAM device. For a 1-bit value, only two levels are needed to map the data to the RRAM device. Hence, the lowest two resistance levels (LRS) can be utilized, thus reducing the overall RRAM device variations.

Simultaneously, for a 2-bit value, four levels are required to map the data to the RRAM device. The first two levels utilize the lowest two resistance levels from the 1-bit case, while the third and fourth levels utilize higher resistance levels (LRS) with higher variation, as shown in Fig. 21. Finally, for a 3-bit data, eight resistance levels are required to map to the RRAM device. The first four levels utilize the same resistance levels as that for the 2-bit case. Meanwhile, the third bit further utilizes four resistance states of which two are LRS and two are HRS for the 65nm RRAM device. Hence, for accurate RRAM variation modeling, we utilize the bit-wise variation models within the hybrid IMC training framework.

Table 7. RRAM Device Variation for Different Bit

Level	0	1	2	3	4	5	6	7
RRAM State	LRS						HRS	
Average Variation (σ)	1-bit	0.1035	NA					
	2-bit	0.1035	0.2760	NA				
	3-bit	0.1035	0.2760	0.2259	0.3549			

3.3.2 Training for Hybrid IMC Architecture

Algorithm 2 details the methodology utilized to train the DNN for the hybrid RRAM/SRAM IMC architecture.

3.3.2.1 RRAM Macro Training

The training of the RRAM macro is performed in two stages. The first stage performs in-training quantization for the DNN model, while the second stage performs

the RRAM IMC-aware training in the presence of quantization. First, the DNN model weights are randomly initialized. Next, we perform in-training quantization for the weights with precision W_{RRAM} and RRAM macro output (A_{RRAM}) and train the DNN to generate Model A. In this work, we keep the weight and activation precision the same for the RRAM macro model (W_{RRAM} equal to A_{RRAM}). We utilize the quantization method in Zhou *et al.* (2016) for the weights and Choi *et al.* (2018) for the activations. Next, we split each conv and FC layer into partial MAC operations based on the IMC crossbar size. For our test-chip, we utilize a 64×64 IMC crossbar array. Finally, the output from each partial MAC operation is accumulated to generate layer-wise output activations. The quantized weights from Model A are then loaded to the partial conv/FC operations, and RRAM variations (σ_{bit}) are added using the log-normal distribution Charan *et al.* (2020a) (at each epoch) in a bit-wise manner following Table 7. Finally, the model is trained for M epochs to generate the RRAM IMC-aware trained model (Model B). Lines 3–13 of Algorithm 2 show the RRAM macro model training.

3.3.2.2 Training SRAM Macro and Programmable Shifter

Once the RRAM macro is trained (Model B), we perform the training for the SRAM macro with the programmable shifter. First, we add a parallel model (Model C) for the SRAM macro with the same size as that of RRAM macro model and randomly initialize weights in a layer-wise manner (100% size as that of the RRAM macro model). Next, we add the output from Model C to that of Model B in a layer-wise manner. The Model C output is shifted based on the scale of shifting (b_{shift}) utilized in the programmable shifter. We note that the scale of shift is kept constant

across all layers of the DNN. Through this, we generate the DNN structure that follows the hybrid IMC architecture (Model D). Thereafter, Model D is trained such that the weights of the SRAM Macro model (Model C) are quantized to W_{SRAM} and the activations to A_{SRAM} following the methodology in Zhou *et al.* (2016) and Choi *et al.* (2018), respectively. Furthermore, the output from each layer within Model D (after adding the Model B and shifted Model C outputs) is quantized to the overall layer-wise activation precision A . During the training of Model D, the weights of Model B are frozen without any update during backpropagation while the Model C weights are updated. Hence, the RRAM macro model serves as the backbone model while the SRAM macro model assists it.

Next, to reduce the overhead from the SRAM macro, we utilize group-wise pruning Yang *et al.* (2020) with a group-size of G_{prune} for the Model C weights. The pruning utilizes weight-penalty clipping with a self-adapting threshold Yang *et al.* (2020), as shown below:

$$\hat{\mathcal{L}} = \mathcal{L}(f(\mathbf{x}; \{\mathbf{W}_l\}_{l=1}^L), \mathbf{t}) + \lambda \sum_{l=1}^L \sum_{i=1}^{G_l} \min(\|\mathbf{W}_{l,i}\|_2; \delta_l) \quad (3.1)$$

$$\delta_l = \alpha \cdot \frac{1}{G_l} \sum_{i=1}^{G_l} \|\mathbf{W}_{l,i}\|_2 \quad (3.2)$$

where δ_l denotes the layer-wise self-adapting clipping threshold, L is the number of layers, G_l is the number of groups in the SRAM macro model for the l -th layer, λ is the hyper-parameter to be tuned based on the dataset, and α is the scaling coefficient. The pruning is performed group-wise along the output channel dimension: for layer l with SRAM macro weight matrix $W_l \in \mathbb{R}_{\mathbb{Q}}^{N_{of} \times N_{if} \times K_x \times K_y}$, we choose a group of size G_{prune} along the N_{if} dimension, where the maximum value of G_{prune} is determined by the number of PEs in the SRAM macro ($\mathbb{R}_{\mathbb{Q}}$ denotes quantized SRAM macro weights). The training for Model D is performed for N epochs, where N is less than M . Overall,

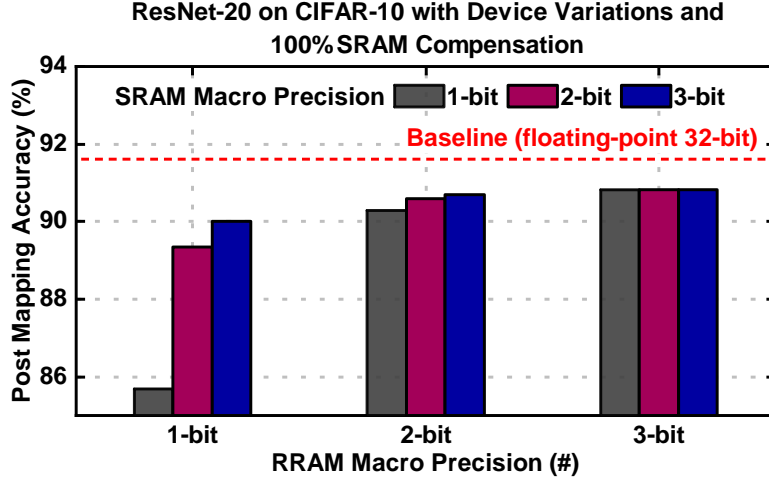


Figure 22. Post-Mapping Accuracy with the Proposed Hybrid IMC Architecture for ResNet-20 on CIFAR-10. A Higher SRAM Macro Precision Leads to Better Compensation Across Different RRAM Precision.

the trained model consists of an RRAM macro model with weight precision W_{RRAM} and output activation precision A_{RRAM} , a structured sparse SRAM macro model with weight precision W_{SRAM} output activation precision A_{SRAM} , overall layer-wise activation precision A , and a shift scale of b_{shift} . Lines 14–22 of Algorithm 2 show the SRAM macro with programmable shifter model training.

3.4 Experiments and Results

We perform extensive experiments to evaluate the proposed hybrid RRAM/SRAM IMC architecture from both an algorithm and a hardware standpoint. The algorithm experiments are performed on a Nvidia Quadro RTX 8000 GPU by utilizing the algorithm framework developed in this work (Section 3.3). We evaluate four different DNNs across two datasets: ResNet-20 on CIFAR-10 (0.27M), VGG-16 on CIFAR-10 (15M), ResNet-18 for ImageNet (11.5M), and MobileNet-v2 for ImageNet (3.4M). All

Table 8. Comprehensive Evaluation of the Choice of Programmable Shifter Across Different DNNs for CIFAR-10 Dataset. (*At Same RRAM and Overall Activation Precision). VAT – Variation-Aware Training Long *et al.* (2019).

Network	RRAM Macro Precision	Accuracy with VAT Only* (%)	SRAM Macro Precision	Activation Precision	Shifter Scale for SRAM Macro	Post-Mapping Accuracy - Ours (%)	Improvement in Accuracy over VAT only (%)
ResNet-20	2-bit	87.40	1-bit	3-bit	0-bit	90.28	2.88
					1-bit	90.33	2.93
					2-bit	89.68	2.28
	3-bit	87.45	3-bit	6-bit	0-bit	90.92	3.47
					1-bit	90.73	3.28
					2-bit	90.87	3.42
					3-bit	90.80	3.35
VGG-16	1-bit	89.02	2-bit	3-bit	0-bit	92.56	3.54
					1-bit	92.54	3.52
	3-bit	91.06	3-bit	6-bit	0-bit	92.97	1.91
					1-bit	92.96	1.90
					2-bit	92.92	1.86
					3-bit	92.90	1.84

experiments performed utilize the device models extracted (at room temperature) for the 65nm 1T1R RRAM device with up to 8 levels (Section 3.3.1). The weight and output activation precision values are kept the same within each macro ($W_{RRAM}=A_{RRAM}$ and $W_{SRAM}=A_{SRAM}$) throughout the DNN. The experiments utilize a RRAM crossbar size of 64×64 for consistency with the 65nm test-chip. Furthermore, we evaluate up to a 3-bit RRAM macro weight and activation precision, a 3-bit SRAM macro weight and activation precision, a 6-bit overall layer-wise activation precision, and a 3-bit shift scale. Finally, for the hardware performance, we utilize the post-layout performance of the 65nm test-chip at 100MHz. The results obtained through VAT refer to the variation-aware training method in Long *et al.* (2019).

3.4.1 Effect of different Scales of SRAM Compensation

We analyse the effect of different scales of SRAM compensation by varying the SRAM macro precision. Fig. 22 shows the post-mapping accuracy for ResNet-20 on CIFAR-10 across different RRAM macro and SRAM macro precisions. No pruning

is performed for the SRAM macro weights, and no shift is applied. The baseline FP-32 ResNet-20 model for CIFAR-10 dataset achieves 91.34% accuracy. Consider an RRAM macro precision of 1-bit. At an SRAM macro precision of 1-bit, the SRAM compensates entirely for the variations within the RRAM macro. Meanwhile, at a higher SRAM macro precision, in addition to the variation compensation the SRAM adds additional bits to increase the dynamic range and the precision of the RRAM macro output for the DNN layer. Hence, at higher SRAM macro precisions, higher accuracy is achieved with up to 90.2%.

Simultaneously, consider an RRAM macro precision of 3-bits. A 3-bit RRAM macro precision provides higher density and better hardware performance, but suffers from higher RRAM device variations. At an SRAM macro precision of 1-bit, the MSB of the RRAM macro is compensated, thus providing a high degree of compensation. Hence, the higher RRAM macro precision and the MSB compensation result in a higher post-mapping accuracy of 90.01%. At an SRAM macro precision of 2-bit, both the MSB and the second bit are compensated, thus providing a higher accuracy of 90.7%. Finally, at a 3-bit SRAM macro precision, all the three bits of the RRAM macro output are compensated, thus providing maximum accuracy of 90.92%. Hence, the SRAM macro compensates for the RRAM macro variations, thus achieving higher accuracy. Through this, the SRAM macro compensation, the hybrid IMC architecture helps exploit the advantage within MLC RRAM cells.

3.4.2 Optimal Shifter Configuration

In this section, we analyze the effect of the scale of shift utilized in the hybrid IMC architecture on the post-mapping accuracy. We note that no pruning is performed

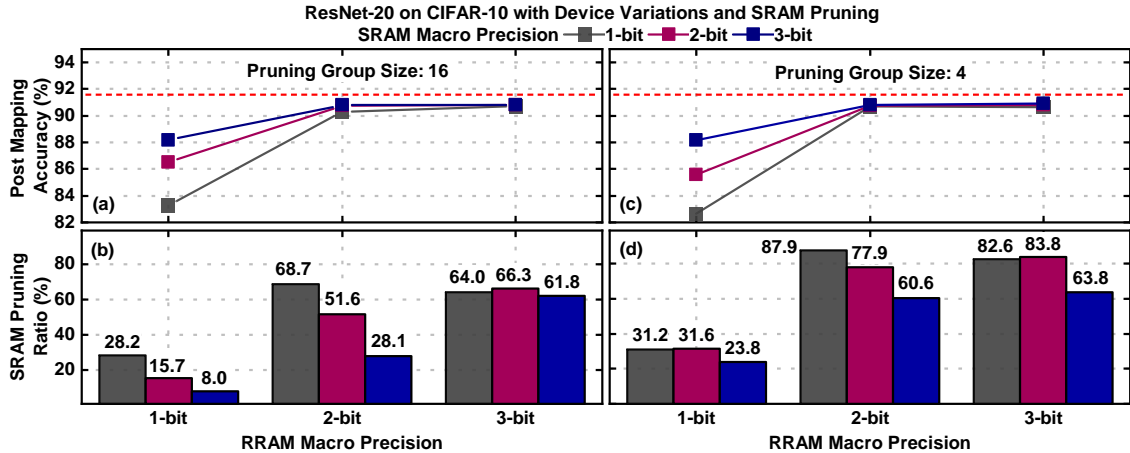


Figure 23. Post-Mapping Accuracy for ResNet-20 on CIFAR-10 at Different RRAM Macro and SRAM Macro Precision. (a) Post-Mapping Accuracy with SRAM Macro at G_{prune} of 16 and (b) Pruning Ratio of the SRAM Macro Weights Across Different RRAM and SRAM Precision at G_{prune} of 16, (c) Post-Mapping Accuracy for a G_{prune} of 4 for the SRAM Macro Weights, and (d) Pruning Ratio of the SRAM Macro Weights at G_{prune} of 4. A Lower Group Size Leads to Higher Pruning at the Same Accuracy.

Table 9. Comprehensive Evaluation of the Post-Mapping Accuracy Across Different DNNs and Datasets with the Proposed Hybrid IMC Architecture. SRAM is Pruned With a Group Size of 4. (*Top-1 Accuracy, **At Same RRAM Precision).

Network	Dataset	RRAM Precision	VAT Only Accuracy (%)**	SRAM Precision	Activation Precision	Shifter Scale	SRAM Pruning Ratio (%)	Post-Mapping Accuracy - Ours (%)	Accuracy Improvement over VAT only (%)
ResNet-20	CIFAR-10	2-bit	87.40	1-bit	3-bit	1-bit	87.8	90.73	3.3
VGG-16	CIFAR-10	3-bit	91.06	3-bit	6-bit	0-bit	98.9	92.75	1.7
ResNet-18	ImageNet	3-bit	63.79	2-bit	5-bit	2-bit	99.7	69.21*	5.4
MobileNet-v2	ImageNet	3-bit	36.60	2-bit	5-bit	2-bit	36.6	61.6*	25.0

on the SRAM macro in this experiment. Table 8 shows the post-mapping accuracy for different DNNs on CIFAR-10 dataset across varying RRAM and SRAM macro precisions at different shifting scales. Consider ResNet-20 on CIFAR-10 at 2-bit RRAM macro precision and 1-bit SRAM macro precision. We utilize a 3-bit overall activation precision across layers. At a 0-bit shift scale, the SRAM output compensates for the MSB of the RRAM macro output, achieving a post-mapping accuracy of 90.28%. At the same time, a 1-bit shift for the SRAM macro compensates the LSB of the RRAM

macro output, achieving an accuracy of 90.33% (within statistical error range). Finally, a 3-bit shift for the SRAM macro output results in the extension of the precision of the output with no compensation for the MSB and LSB of the RRAM macro output. Such a scale of shift results in reduced post-mapping accuracy (89.68%) due to the effect of the RRAM variations on the output. Similarly, we evaluate a 3-bit RRAM macro precision and a 3-bit SRAM macro precision. A 0-bit shift provides the best post-mapping accuracy as the SRAM macro compensates for all the bits of the RRAM macro output.

We repeat the same experiment with VGG-16 for CIFAR-10. For a 1-bit RRAM macro precision and a 2-bit SRAM macro precision, a 0-bit shift provides the highest accuracy. Similarly, a 0-bit shift provides the highest accuracy of 92.97% for a 3-bit RRAM and SRAM macro precision. We conclude that the optimal scale of shift depends on the DNN structure, RRAM and SRAM macro precisions, and the overall activation precision. To provide further context, we compare the post-mapping accuracy of the two DNNs to the FP-32 baseline accuracy. For ResNet-20 on CIFAR-10, the FP-32 model achieves 91.34% accuracy, while the best configuration hybrid IMC model achieves 90.92% accuracy. For VGG-16 on CIFAR-10, the hybrid IMC model achieves 92.97% accuracy compared to 93.04% for the FP-32 model. Finally, we compare the post-mapping accuracy with the hybrid IMC architecture to that with VAT only. The proposed method consistently outperforms the VAT method and achieves near baseline (Floating-point 32-bit) accuracy.

3.4.3 Pruning Analysis of SRAM

The addition of the SRAM macro and programmable shifter results in an overhead for the hardware architecture. Hence, to reduce the overhead, we utilize group-wise pruning of the weights within the SRAM macro, as detailed in Section 3.3.2. Fig. 23 shows the post-mapping accuracy and the SRAM macro pruning ratio for two different pruning group sizes for ResNet-20 on CIFAR-10. The pruning ratio gives the amount of sparsity achieved through pruning. We note that we perform the pruning for the optimal configurations of the shift scale. We explore 1-bit to 3-bit precisions for both the RRAM and SRAM macros. Fig. 23(a) shows the post-mapping accuracy across different configurations for a pruning group size of 16. A higher SRAM macro precision allows for higher compensation for the RRAM macro output in the presence of sparsity. Next, we analyze the pruning ratio achieved with pruning across the different configurations, as shown in Fig. 23(b). For the 1-bit RRAM macro precision, a lower pruning ratio (sparsity) is obtained for best post-mapping accuracy since the SRAM macro output provides both compensation and increased precision and range. Meanwhile, a higher precision for the RRAM macro weights results in a higher accuracy compared to a binary model. Therefore, for 2-bit and 3-bit RRAM macro precision, the SRAM macro compensation is easier, allowing the pruning method to generate a higher pruning ratio for the SRAM macro. We repeat the same experiment for a smaller group size of 4, as shown in Fig. 23(c) and (d). The hybrid IMC architecture achieves similar post-mapping accuracy as that for the 16 group size. In addition, a higher pruning ratio is achieved for the SRAM in all cases due to the ability of the pruning algorithm to remove smaller groups of weights compared to a group size of 16. Hence, we conclude that a smaller group size and a higher RRAM precision is

preferred as it provides higher sparsity (lower overhead), higher RRAM density, and similar post-mapping accuracy with the hybrid IMC architecture.

3.4.4 Overall Accuracy Results

Table 9 shows the overall comprehensive results for the hybrid IMC architecture. The baseline (floating-point 32-bit) accuracy for the DNNs are as follows; ResNet-20 - 91.32%, VGG-16 - 93.04%, ResNet-18 - 69.57%, and MobileNet-v2 - 71.87%. The choice of the optimal configuration is determined based on the post-mapping accuracy and the SRAM macro pruning ratio. For example, for VGG-16 on CIFAR-10 with 3-bit RRAM macro precision, an SRAM macro precision of 2-bit with a 2-bit shift achieves 92.76% post-mapping accuracy with a 95% SRAM pruning ratio. At the same time, a 3-bit SRAM macro precision with a 0-bit shift results in 92.75% accuracy at a 98.9% SRAM pruning ratio. Hence, we choose the 3-bit RRAM 3-bit SRAM configuration considering both the accuracy and SRAM macro pruning ratio.

We compare the post-mapping accuracy of the hybrid IMC architecture with that of the conventional VAT technique. For fair comparison, we utilize the same RRAM macro precision (weights and activation) for both approaches. For ResNet-20 on CIFAR-10, the hybrid IMC architecture achieves 3.3% higher accuracy, while for VGG-16 on CIFAR-10 the hybrid architecture achieves 1.7% improvement in post-mapping accuracy. At the same time, for ImageNet models ResNet-18 and MobileNet-v2, the proposed hybrid IMC architecture achieves 5.4% and 25% improvement, respectively. For all the networks except MobileNet-v2, the proposed hybrid IMC architecture achieves greater than 87% sparsity (pruning ratio) for the SRAM macro. MobileNet-v2 being a small model for the ImageNet dataset requires highly accurate weights in the

Table 10. Post-Mapping Accuracy and SRAM Macro Pruning Ratio for 1x and 2x RRAM Variation. We Use a Pruning Group Size of 4.

RRAM Precision	SRAM Precision	Post-Mapping Accuracy (%)		SRAM Macro Pruning Ratio (%)	
		1x Var	2x Var	1x Var	2x Var
1-bit	1-bit	82.68	83.50	31.20	29.10
	2-bit	85.59	85.37	31.63	31.10
	3-bit	88.18	88.00	27.78	27.30
2-bit	1-bit	90.73	90.11	87.88	67.11
	2-bit	90.78	90.41	77.90	73.70
	3-bit	90.84	90.39	60.52	50.60
3-bit	1-bit	90.65	90.59	82.62	54.80
	2-bit	90.83	90.75	83.79	74.11
	3-bit	90.90	90.90	63.81	44.90

RRAM macro, i.e., a higher degree of compensation to ensure high post-mapping accuracy.

3.4.5 Evaluation with 2× RRAM Variation

In this section, we evaluate the post-mapping accuracy for the hybrid IMC architecture with 2× the bit-wise variations (σ) in Table 7. Note that we perform shifting and pruning for the SRAM macro in this experiment. Table 10 shows the comparison of the post-mapping accuracy and SRAM pruning ratio for ResNet-20 on CIFAR-10 at 1× and 2× RRAM variations. For each RRAM macro precision, the proposed hybrid IMC architecture achieves similar post-mapping accuracy for both 1× and 2× RRAM variations. At the same time, a 1× RRAM variations results in a higher SRAM pruning ratio and a lower SRAM macro overhead. The increased overhead for 2× RRAM variations is attributed to the higher SRAM compensation needed for better

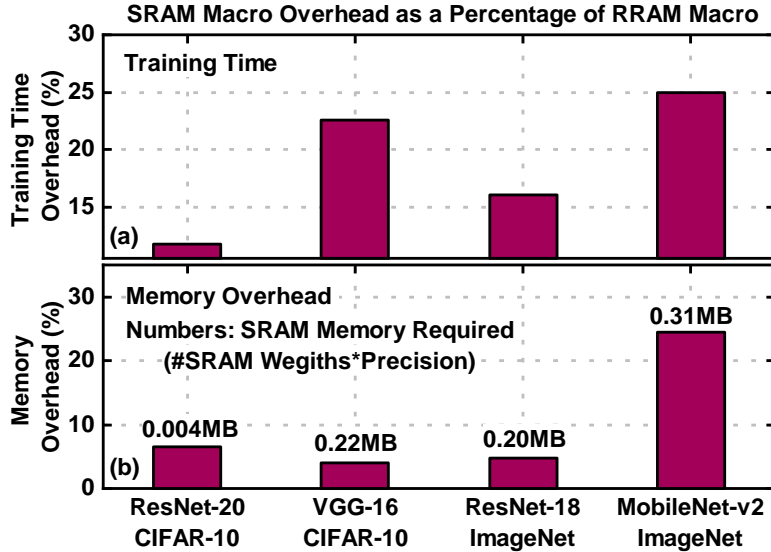


Figure 24. (a) Training Time Overhead for the SRAM Macro and the Shifter. The Proposed Hybrid IMC Architecture Incurs at Most 25% (Compared to Time of RRAM Macro) Overhead in Training Time. (b) Memory Overhead for the SRAM Macro Compared to the RRAM Macro. The Hybrid IMC Architecture Incurs at Most 24% Overhead (Compared to the Memory of RRAM Macro).

accuracy. Furthermore, for a given RRAM macro precision, the optimal SRAM macro precision results in higher accuracy for both $1\times$ and $2\times$ RRAM variations. Hence, the hybrid IMC architecture provides a scalable solution that opens the opportunity for multi-level RRAM devices for the acceleration of a wide range of DNNs across different datasets.

3.4.6 SRAM Macro and Shifter Overhead Analysis

We analyze the overhead for the SRAM and programmable shifter from an algorithm and hardware standpoint in the proposed hybrid IMC architecture. Note that we utilize the optimal hybrid IMC architecture model for each of the DNNs. Fig. 24(a)

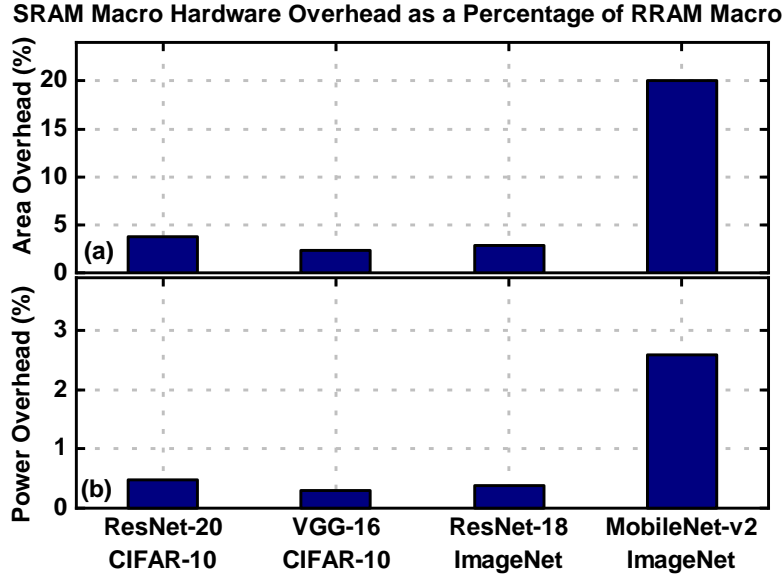


Figure 25. (a) Area Overhead and (b) Power Overhead of the SRAM Macro as a Percentage of the Total RRAM Macro Area and Power. The Proposed Hybrid IMC Architecture Results in a Very Low Overhead with Up to 20% Area and 2.6% Power Overhead for State-of-the-Art Accuracy Across Different DNNs.

shows the training time overhead for the SRAM macro as a percentage of the training time for the RRAM macro. For ResNet-20 and VGG-16 on CIFAR-10, the hybrid IMC architecture results in 12% and 23% overhead in training time. The increased overhead for VGG-16 is attributed to the larger model size compared to ResNet-20 (15M for VGG-16 compared to 0.27M for ResNet-20). At the same time, for ResNet-18 and MobileNet-v2, a 16% and 25% overhead in training time are incurred. Hence, the hybrid IMC architecture at most incurs a training time overhead of 25% of the RRAM macro training time. Next, we compare the overhead for the hybrid IMC training to the traditional read-verify-write (R-V-W) method utilized to achieve accurate RRAM device resistance levels. The R-V-W method requires time of the order of days to verify and write 100% of the RRAM cells Charan *et al.* (2020a). At the same time,

Table 11. Comparison of Post-Mapping Accuracy with State-Of-The-Art Methods. Ours-A SRAM Macro Weights Pruned; Ours-B[#]: SRAM Macro Weights Not Pruned. (**Precision Not Reported In the Manuscript)

Method	Weight Precision	Activation Precision	Post-Mapping Accuracy (%)	Accuracy Improvement by Our Work (%)
ResNet-20 on CIFAR-10				
Model Stability Krishnan <i>et al.</i> (2021f)	8-bit	8-bit	68.83	21.9
ReSNA Yang <i>et al.</i> (2021)	8-bit	8-bit	88.43	2.3
Ours-A	2-bit RRAM & 1-bit SRAM	3-bit	90.73	-
VGG-16 on CIFAR-10				
DFP+DVA Long <i>et al.</i> (2019)	8-bit	**	80.1	12.65
Go Unary Ma <i>et al.</i> (2020)	8-bit	**	87.94	4.84
KD+OSA Charan <i>et al.</i> (2020a)	4-bit	32-bit	92.57	0.18
Unary Opt Sun <i>et al.</i> (2021)	8-bit	**	92.77	0.20 (Compared to Ours-B)
Ours-A	3-bit RRAM & SRAM	6-bit	92.75	-
Ours-B[#]	3-bit RRAM & SRAM	6-bit	92.97	-
ResNet-18 on ImageNet (Top-1)				
Unary Opt Sun <i>et al.</i> (2021)	8-bit	**	62.69	6.52
Ours-A	3-bit RRAM & 2-bit SRAM	5-bit	69.21	-

the proposed hybrid IMC architecture requires time of the order of a couple of hours to achieve near baseline (FP-32) accuracy, thus, providing a scalable solution.

Next, we evaluate the memory overhead as a percentage of the total RRAM memory requirement. Based on the SRAM pruning ratio, we evaluate the total number of non-zero bits that need to be stored within the SRAM macro. The structured pruning method employed allows the skipping of the zero weights in the SRAM macro. Fig. 24(b) shows the memory overhead for the hybrid IMC architecture. For ResNet-20 and VGG-16 on CIFAR-10, the hybrid IMC architecture incurs 6.5% (0.004MB SRAM to 0.067MB RRAM) and 4% (0.22MB SRAM to 5.63MB RRAM) overhead in the memory requirement. At the same time, for the ImageNet dataset, a 4.7% (0.2MB SRAM to 4.32MB RRAM) and 24% (0.31MB SRAM to 1.3MB RRAM) overhead in memory is incurred for ResNet-18 and MobileNet-v2, respectively. The

increased overhead for MobileNet-v2 is attributed to the lower SRAM pruning ratio due to the need for higher compensation.

Finally, we evaluate the overhead in terms of the hardware performance (area and power) for the hybrid IMC architecture. We utilize the post-layout area and power measurements at 100MHz from the designed 65nm test-chip. The area and power of the RRAM macro consists of the RRAM IMC crossbar array, WL driver and shifter, RRAM decoder, flash ADCs (3-bits), PMOS headers, and BL/SL/column multiplexers. The power is measured by evaluating the total current drawn by each supply voltage (1.2V and 3.3V) and taking the product of the voltage and current. The PMOS header and ADC account for 90% of the power of the RRAM macro. The SRAM macro consists of the SRAM memory array, PE array, and buffers. A similar area and power estimation as that of the RRAM macro is performed for the SRAM macro.

Fig. 25(a) and Fig. 25(b) show the area and power overhead for the SRAM macro as a percentage of the RRAM macro. For ResNet-20 on CIFAR-10, an area and power overhead of 3.7% and 0.5% are incurred for the SRAM macro, respectively. At the same time, for VGG-16 on CIFAR-10, an area and power overhead of 2.3% and 0.3% are incurred for the SRAM macro, respectively. Meanwhile, MobileNet-v2 incurs the highest area and power overhead of 20% and 2.6%, respectively. The higher overhead is attributed to the higher compensation requirement resulting in a lower SRAM macro pruning ratio.

3.4.7 Comparison with Other Work

We compare the post-mapping accuracy for the proposed hybrid RRAM/SRAM IMC architecture with state-of-the-art methods. Table 11 shows the comparison for ResNet-20 and VGG-16 on CIFAR-10, and ResNet-18 on the ImageNet dataset. For ResNet-20 on CIFAR-10, compared to the method in Krishnan *et al.* (2021f) and Yang *et al.* (2021), the proposed hybrid IMC architecture achieves 21.9% and 2.3% improvement in post-mapping accuracy at lower weight and activation precision, respectively.

Next, for VGG-16 on CIFAR-10, the proposed hybrid IMC architecture achieves 92.75% accuracy at 3-bit RRAM and SRAM macro precision with a 6-bit activation precision. Compared to Long *et al.* (2019); Ma *et al.* (2020); Charan *et al.* (2020a), the hybrid IMC architecture achieves 12.65%, 4.84%, and 0.18% improvement in post-mapping accuracy, respectively. Furthermore, if the SRAM macro is not pruned, the proposed method achieves 92.97% accuracy, a 0.2% improvement in post-mapping accuracy compared to Sun *et al.* (2021). For ResNet-18 on ImageNet, compared to Sun *et al.* (2021), the proposed hybrid IMC architecture achieves 6.52% higher post-mapping accuracy at 3-bit RRAM macro precision, 2-bit SRAM macro precision, and a 5-bit activation precision. Authors in Long *et al.* (2019); Ma *et al.* (2020); Sun *et al.* (2021); Charan *et al.* (2020a) do not discuss the quantization activation precision. The hybrid IMC architecture with a lower activation precision provides higher hardware performance and accuracy. Furthermore, Charan *et al.* (2020a) utilizes the larger ImageNet VGG-16 model with 3 FC layers for the CIFAR-10 dataset (134M parameters). In this work, we utilize the smaller CIFAR-10 model for VGG-16 with 15M parameters. Finally, both Ma *et al.* (2020) and Sun *et al.* (2021) utilize a

unary mapping scheme, thus requiring exact variation measurements for each cell in the RRAM IMC crossbar. A complete R-V-W (takes up to many days) needs to be performed to quantify the variations at the cell level. Hence, the proposed hybrid IMC architecture provides a more scalable solution for robust DNN acceleration. Overall, the improved accuracy is attributed to the hybrid IMC architecture with the optimal scale of SRAM compensation achieved through the programmable shifter. We carefully tune the precision (weights and activations), the sparsity for the SRAM MAC engine macro, and the scale of shift for the SRAM macro output to obtain the best accuracy.

3.5 Conclusion

In this work, we propose a novel hybrid RRAM/SRAM IMC architecture for robust DNN acceleration. The hybrid IMC architecture utilizes an RRAM IMC macro with MLC cells, an SRAM macro, and a programmable shifter. The output from the RRAM macro is compensated by the SRAM macro output to create an ensemble model and achieve bit-level compensation. The scale of compensation is controlled by using a programmable shifter for the SRAM macro output. Next, we develop a training framework to enable the hybrid IMC architecture that supports quantization, structured pruning, RRAM IMC-aware training, and different compensation scales through the programmable shifter. Finally, we design a test-chip using the 65nm SUNY process to demonstrate the efficacy of the proposed hybrid IMC architecture.

We perform detailed experiments across different DNNs and datasets to demonstrate the performance of the proposed hybrid IMC architecture. Compared to the conventional VAT method, the proposed hybrid IMC architecture achieves up to 25% improvement in post-mapping accuracy. In addition, compared to state-of-the-art

methods, the proposed hybrid IMC architecture provides a scalable solution that achieves up to 21.9%, 12.65%, and 6.52% improvement in post-mapping accuracy with minimal overhead for ResNet-20 on CIFAR-10, VGG-16 on CIFAR-10, and ResNet-18 on ImageNet, respectively. Finally, through the experimental evaluation of the hybrid IMC architecture, we show that the SRAM compensation opens the opportunity for a realistic IMC architecture with multi-level RRAM cells.

Algorithm 2: Training Methodology for the hybrid RRAM/SRAM IMC architecture

```
1 Input: DNN, RRAM weight precision ( $W_{RRAM}$ ), RRAM output precision
   ( $A_{RRAM}$ ), SRAM weight precision ( $W_{SRAM}$ ), SRAM output precision
   ( $A_{SRAM}$ ), overall activation precision ( $A$ ), shift scale ( $b_{shift}$ ), SRAM pruning
   group size ( $G_{prune}$ ), Training epochs M and N, and bit-wise RRAM variations
   ( $\sigma_{bit}$ )
2 Output: Trained Hybrid RRAM/SRAM IMC model
3 for RRAM Model do
4   Initialize DNN model randomly
5   Perform in-training quantization for weights ( $W_{RRAM}$ ) and activations
   ( $A_{RRAM}$ )
   /* Model A */
6   Layer-wise split conv and FC layer into partial conv/FC based on RRAM
   crossbar size
7   Load trained quantized weights from Model A
8   Add partial conv/FC outputs to generate final layer-wise output
9   for  $Epoch \leq M$  do
10    Add RRAM variations (bit-wise) to weights
11    Train DNN model
12  end
13 end
   /* Model B */
14 for SRAM Model do
15   Create parallel SRAM model layer-wise with size 100% of RRAM model
   and randomly initialize weights
   /* Model C */
16   Add Model B output layer-wise to the Model C output with  $b_{shift}$  shift
   /* Model D */
17   for  $Epoch \leq N$  do
18     Perform in-training quantization and group-wise pruning ( $G_{prune}$ ) for
     Model C weights ( $W_{SRAM}$ ) at activation precision  $A_{SRAM}$ 
19     Perform in-training quantization for overall layer-wise activation output
     to  $A$ 
20     Backpropagation: Freeze Model B weights with no update. Only
     update Model C weights
21   end
22 end
23 Save final trained Model D and perform inference
```

IMPACT OF ON-CHIP INTERCONNECT ON IN-MEMORY ACCELERATION
OF DEEP NEURAL NETWORKS

4.1 Introduction

DNNs have achieved high accuracy that exceeds human-level perception for a variety of applications such as computer vision, natural language processing, and medical imaging Krizhevsky *et al.* (2012); Deng *et al.* (2013); Litjens *et al.* (2017). The DNNs that achieve higher accuracy tend to consist of deeper and denser network structures. However, DNNs for edge devices tend to use smaller and shallower networks Bhat *et al.* (2018).

Figure 26 shows the trend in connection density for various DNNs in the literature, where *connection density* is defined as the average number of connections per neuron in DNNs. In the context of DNNs, a neuron is defined as an output feature of a convolution layer and every neural unit of the fully-connected (FC) layer. Three representative DNN structures and connection patterns are illustrated in Figure 27. Linear structures such as LeNet-5 LeCun *et al.* (1998) and VGG-19 Simonyan and Zisserman (2014) have a connection density of one owing to one connection per neuron. Since residual networks such as ResNet He *et al.* (2016) have residual skips, it has more connections than the number of neurons resulting in a connection density higher than one. Dense structures like DenseNet Huang *et al.* (2017) have multiple connections from each neuron, resulting in a higher connection density.

We observe two main trends by analyzing the connection density for different DNNs

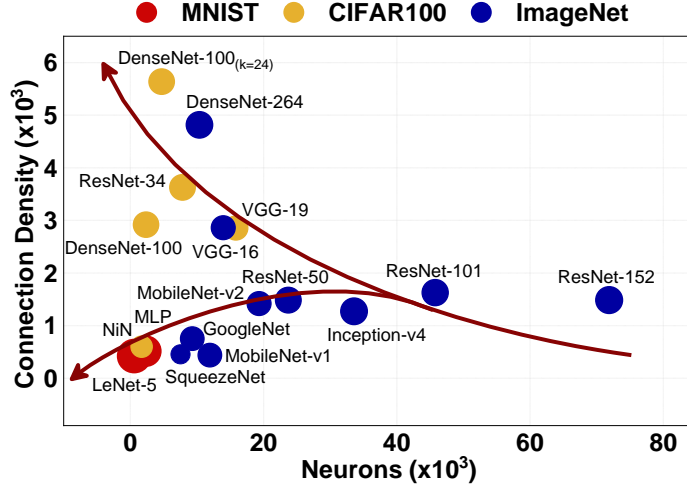


Figure 26. Connection Density of Different DNNs for Three Different Datasets. Each Output Feature Map (Convolution Layer) and Neural Unit (FC Layer) Represent a Neuron. Larger Markers Represent Higher Accuracy.

in Figure 26. First, increasing connection density provides higher accuracy, which is essential for cloud-based computing platforms. Second, lower connection density is observed for compact models, which is necessary for edge computing hardware. Both hardware platforms require the processing of large amounts of data with corresponding power and performance constraints. Hence, there is a need to design optimal hardware architectures with low power and high performance for DNNs with different connection densities.

With limited on-chip memory, conventional DNN architectures inevitably involve a significant amount of communication with off-chip memory resulting in increased energy consumption Chen *et al.* (2019). However, it has been reported that the energy consumption of off-chip communication is $1,000\times$ higher than the energy required to perform the computations Horowitz (2014).

Dense structures like DenseNet perform approximately 2.7×10^7 off-chip memory accesses to process a frame of an image Huang *et al.* (2017). As a result, off-chip memory access becomes the energy bottleneck for hardware architectures of dense

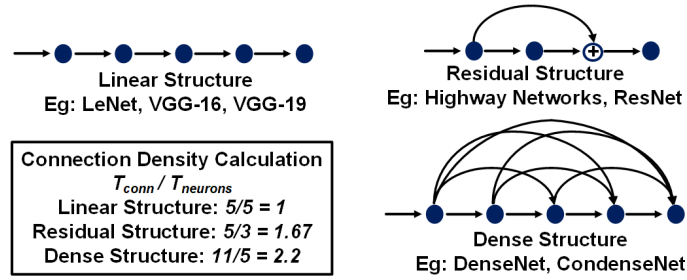


Figure 27. Different Types of DNN Structures and Their Representative Connection Density.

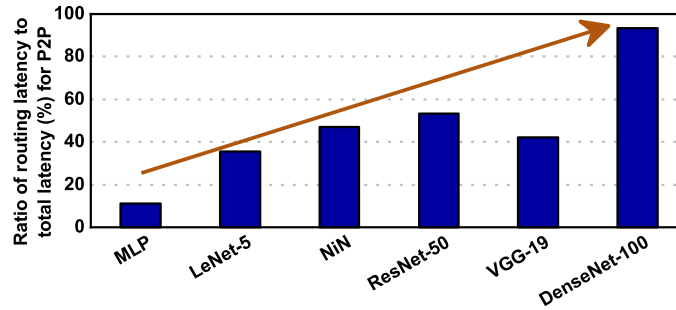


Figure 28. Contribution of Routing Latency to Total Latency for Different DNNs for a P2P-based IMC Architecture Chen *et al.* (2018). With Increase in Connection Density, Routing Latency Becomes the Bottleneck for Performance.

structures. Employing dense embedded non-volatile memory (NVM) such as ReRAM for in-memory computing (IMC) substantially reduces off-chip memory accesses Shafiee *et al.* (2016); Song *et al.* (2017).

On-chip interconnect is an integral part of hardware architectures that incorporate in-memory acceleration. Both point-to-point (P2P) interconnect Kwon *et al.* (2018); Venkataramani *et al.* (2017) and NoC-based interconnect Shafiee *et al.* (2016); Chen *et al.* (2019); Krishnan *et al.* (2020b) are used for on-chip communication in state-of-the-art DNN accelerators. Shafiee *et al.* Shafiee *et al.* (2016) utilizes a concentrated mesh for the interconnect, while Chen *et al.* Chen *et al.* (2019) employs three different NoCs that are used for on-chip data movement in the architecture. In contrast, Krishnan *et al.* Krishnan *et al.* (2020b) utilizes a custom mesh-NoC for on-chip

communication. The custom NoC derives the structure based on the on-chip traffic between different IMC processing elements (PEs), where each PE denotes the SRAM- or ReRAM-based IMC crossbar. A technique to construct custom NoC which provides minimum communication latency for a given DNN is proposed in Mandal *et al.* (2020). Since custom NoC requires alteration in hardware for different DNNs, our studies focus on regular NoC topologies. A more detailed survey on work which design efficient interconnect for DNN accelerators can be found in Nabavinejad *et al.* (2020).

To better understand the need for an NoC-based on-chip interconnect, we analyze the scalability of P2P interconnect in in-memory computing (IMC) architectures by evaluating the contribution of routing latency to end-to-end latency for different DNNs, as shown in Figure 28. The contribution of routing latency increases up to 94% with increasing connection density. The high routing latency is attributed to the increased connection density, which correlates to more on-chip data movement. VGG-19 shows a reduced contribution compared to lower connection density DNNs due to the high utilization of the IMC PEs or crossbars resulting in reduced on-chip data movement. Hence, P2P networks do not provide a scalable solution for high connection density DNNs. At the same time, NoC-based interconnects require higher area and energy for operation and can result in a significant overhead for low connection density DNNs. Furthermore, different NoC topologies, mesh, or tree, are appropriate for DNNs with varying connection densities. Therefore, a connection density-aware interconnect solution is critical to DNN acceleration.

In this work², we first perform an in-depth performance analysis of P2P interconnect-based in-memory computing (IMC) architectures Song *et al.* (2017). Through this analysis, we establish that P2P-based interconnects are incapable of

²Work done in collaboration with Sumit K. Mandal (UW– Madison)

Table 12. Summary Of Notations

Symbol	Definition	Symbol	Definition
N_L	Number of Layers	x_i, y_i	Input image size in i^{th} layer
T_i	Number of tiles in i^{th} layer	C_i	Input channels of i^{th} layer
A_i	Number of activations in i^{th} layer	$\lambda_{i,j,k}$	Injection rate from j^{th} tile of $(i-1)^{\text{th}}$ layer to k^{th} tile of i^{th} layer
d_i	Input activations in i^{th} layer	L_{comm}	Total communication latency

handling data communication for dense DNNs and that NoC-based interconnect is needed for IMC architectures. Next, we evaluate P2P-based and NoC-based SRAM and ReRAM IMC architectures for a range of DNNs. Further, we evaluate NoC-tree, NoC-mesh, and c-mesh topologies for the IMC architectures. A c-mesh NoC is used in Shafiee *et al.* (2016) at the tile-level to connect different tiles. C-mesh uses more number of links and routers, providing better performance in terms of communication latency. However, interconnect area and energy becomes exorbitantly high for c-mesh NoC. Therefore, the energy-delay-area product (EDAP) of c-mesh is higher than NoC-mesh. Hence, we restrict the detailed evaluations to NoC-mesh and NoC-tree. In these evaluations, we perform cycle-accurate NoC simulations through Booksim Jiang *et al.* (2013). However, cycle-accurate NoC simulations are very time consuming and consequently slow down the overall performance analysis of IMC architectures. Our experiment with different DNNs (the simulation framework is described in more detail in Section 4.3) shows that cycle-accurate NoC simulation takes up to 80% of the total simulation time for high connection density DNNs.

To accelerate the overall performance analysis of the IMC architecture, we propose analytical models to estimate the NoC performance of a given DNN. Specifically, we incorporate the analytical router modeling technique presented in Ogras *et al.*

(2010) to obtain the performance model for an NoC router. Then we extend the existing analytical model to get an estimation of end-to-end communication latency for NoC-tree and NoC-mesh for any given DNN as a function of the number of neurons and connection density. Through the analytical latency model, the variable communication patterns of different DNNs are incorporated using connection density and number of neurons. Leveraging this analysis and the analytical model, we conclude the importance of the optimal choice of interconnect at different hierarchies of the IMC architecture. Utilizing the same analysis, we provide guidance for the optimal choice of interconnect for IMC architectures. At the tile-level, NoC-mesh for high connection density DNNs and an NoC-tree for low connection density DNNs provide low power and high performance for IMC-based architectures. Leveraging this observation, we propose an NoC-based heterogeneous interconnect IMC architecture for DNN acceleration. We demonstrate that the NoC-based heterogeneous interconnect IMC architecture (ReRAM) achieves up to $6\times$ improvement in the energy-delay-area product (EDAP) for inference of VGG-19 when compared to state-of-the-art implementations. The following are key contributions of this work:

- An in-depth analysis of the shortcomings of P2P-based interconnect and the need for NoC in IMC architectures.
- Analytical and empirical analysis to guide the choice of optimal NoC topology for an NoC-based heterogeneous interconnect.
- Proposed heterogeneous interconnect IMC architecture achieves $6\times$ improvement in EDAP with respect to state-of-the-art ReRAM-based IMC accelerators.

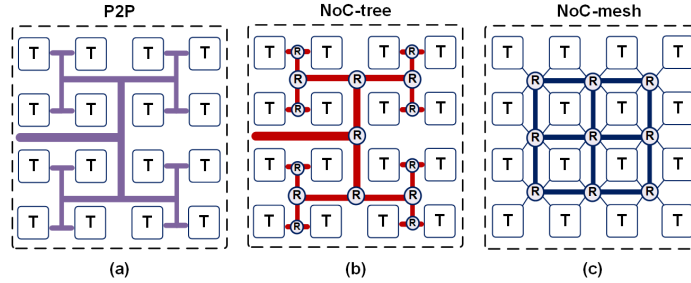


Figure 29. Multi-Tiled IMC Architecture with Routing Architectures Based on (a) P2P Network, (b) NoC-tree, (c) NoC-mesh. NoC-tree is a P2P Network with Routers at Junctions.

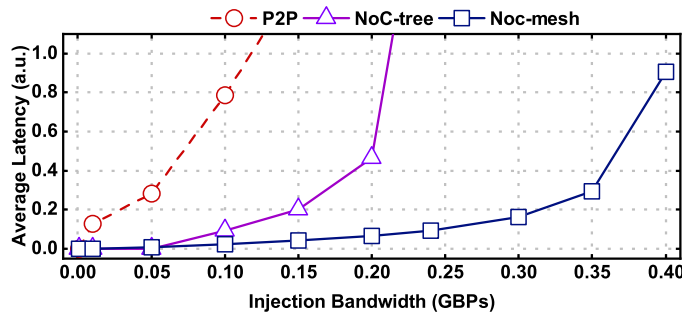


Figure 30. Comparison of Average Latency Among P2P, NoC-tree, and NoC-mesh Interconnect for Different Injection Bandwidth Jiang *et al.* (2013). NoC Topologies Show Better Scalability Than P2P Interconnect.

4.2 Interconnect Network

As discussed in Section 4.1, the on-chip interconnect is critical to the accelerator performance for DNN acceleration. There are multiple topologies for Network-on-Chip (NoC). The well-known topologies are mesh, tree, torus, hypercube, and concentrated mesh (c-mesh). NoC with torus topology shows better performance than mesh due to long links between the nodes located at the edges. However, the power consumption by torus is significantly higher than mesh, as shown in Mirza-Aghatabar *et al.* (2007). Hypercube and c-mesh have a similar disadvantage as a torus. Therefore, only NoC-

tree and NoC-mesh are considered in this work. Also, they are the industrial standard for SoCs used in heavy workloads Jeffers *et al.* (2016).

Figure 29 illustrates representative interconnect schemes of P2P, NoC-tree, and NoC-mesh for multi-tiled IMC architectures. Each tile consists of several crossbar sub-arrays which perform the IMC operation. Existing implementations of DNN accelerators use both P2P-based Venkataramani *et al.* (2017); Kwon *et al.* (2018) and NoC-based Shafiee *et al.* (2016); Krishnan *et al.* (2020b); Zhu *et al.* (2020) interconnect for on-chip communication. To better understand the performance of different interconnect architectures, we plot the average interconnect latency for a P2P network with 64 nodes, NoC-tree with 64 nodes, and an 8×8 NoC-mesh with X-Y routing as shown in Figure 30. The NoC utilizes one virtual channel, a buffer size (all input and output buffers) of eight, and three router pipeline stages. We observe that for lower injection rates, the performance is comparable for all topologies, while for higher injection rates, NoC performs better in terms of latency. Hence, NoC provides better scalability and performance compared to P2P interconnects. Moreover, with increasing connection density, injection bandwidth between layers increase due to increased on-chip data movement. Therefore, P2P interconnect performs poorly for DNNs with high connection density. Hence, there is a need for systematic guidance for choosing the optimal interconnect for in-memory acceleration of DNNs. Other works such as Chen *et al.* (2019) utilizes three separate NoC for weights, activations and partial sums. Such a design choice results in increased area and energy cost for the interconnect fabric. Furthermore, the three NoCs are under-utilized, resulting in a sub-optimal design choice for acceleration of DNNs.

4.3 Simulation Framework

There exist multiple simulators that evaluate the performance of DNNs on different hardware platforms Chen *et al.* (2018). These simulators consider different technologies, platforms, and peripheral circuit modeling while providing less consideration to interconnect. With the advent of dense DNN structures Xie *et al.* (2019), the importance of interconnect cost is higher, as discussed in Section 4.1. In this work, we develop an in-house simulator, where a circuit-level performance estimator of the computing fabric is combined with a cycle-accurate simulator for the interconnect. The simulator also aims at being versatile by supporting multiple DNN algorithms across different datasets, and various interconnect schemes.

Figure 31 shows a block-level representation of the simulator. The inputs of the simulator primarily include the DNN structure, technology node, and frequency of operation. In the proposed simulation framework, any circuit-level performance estimator Dong *et al.* (2012); Chen *et al.* (2018) and any interconnect simulator Jiang *et al.* (2013); Agarwal *et al.* (2009) can be plugged in to extract performance metrics such as area, energy, and latency, proving a common platform for system-level evaluation. In this work, we use customized versions of NeuroSim Chen *et al.* (2018) for circuit simulation and BookSim Jiang *et al.* (2013) for cycle-accurate NoC simulation.

4.3.1 Circuit-level Simulator: Customized NeuroSim

The inputs to NeuroSim include the DNN structure indicating the layer size and layer count along with technology node, the number of bits per in-memory compute cell, frequency of operation, read-out mode, etc. The simulator performs the mapping

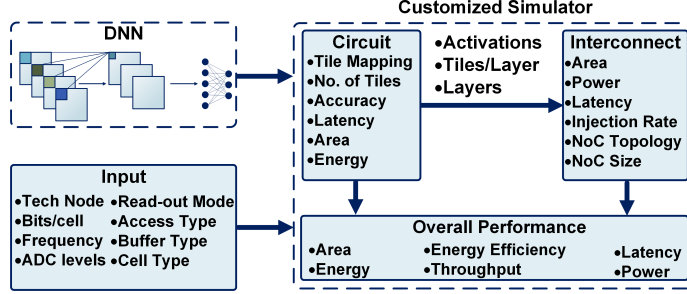


Figure 31. Block-Level Representation of the Proposed Architecture Simulator.

of the entire DNN to a multi-tiled cross-bar architecture by estimating the number of cross-bar arrays and the number of tiles per layer. Based on the size of the cross-bar PE_x and PE_y , the number of cross-bar arrays is determined by (4.1).

$$\text{No. crossbars} = \sum_{i=1}^{N_L} \left\lceil \frac{(Kx_i \times Ky_i \times C_i)}{(PE_x)_i} \right\rceil \times \left\lceil \frac{(C_{i+1}) \times N_{bits}}{(PE_y)_i} \right\rceil, \quad (4.1)$$

where N_{bits} is the precision of the weights. The total number of tiles is calculated as the ratio of the total number of crossbar arrays to the number of crossbar arrays per tile. Furthermore, the peripheral circuits are laid out, and the complete tile architecture is determined. The peripheral circuits include an ADC, sample and hold circuit, shift and add circuit, and a multiplexer circuit. However, NeuroSim lacks an accurate estimation of the interconnect cost in latency, energy, and area. Therefore, we replace the interconnect part of NeuroSim with customized BookSim. We also extract the performance metrics for tile-to-tile interconnect in NeuroSim and replace it with the BookSim tile-to-tile interconnect. With this customization, our circuit simulator only reports performance metrics, such as area, energy, and latency of the computing logic. It provides the number of tiles per layer, activations, and the number of layers to the interconnect simulator.

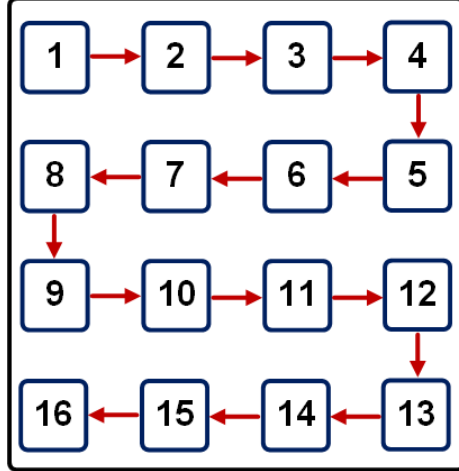


Figure 32. Tile Numbering and Placement While Mapping the DNN to the IMC Architecture. The Red Arrows show the Flow of the Data Across the Tiles.

4.3.2 Interconnect Simulator: Customized BookSim

Algorithm 3: Evaluation of interconnect latency through simulation

- 1 **Input:** Number of layers (N_L), Number of tiles in each layer (T_i), FPS (F),
Number of activation in each layer (A_i), interconnected topology
 - 2 **Output:** End-to-end interconnect latency ($L_{routing}$)
 - 3 **for** each layer i **do**
 - 4 **for** each tile j in layer $i - 1$ **do**
 - 5 **for** each tile k in layer i **do**
 - 6 **if** $i > 0$ **then**
 - 7 Compute $\lambda_{i,j,k}$ following Equation 4.2.
 - 8 **end**
 - 9 **end**
 - 10 **end**
 - 11 Simulate with interconnect topology and $\lambda_{i,j,k}$
 - 12 Obtain $(l_i)_{sim}$ from the simulator.
 - 13 Calculate l_i following Equation 4.3.
 - 14 **end**
 - 15 Calculate $L_{comm}^{sim} : L_{comm}^{sim} = \sum_{i=1}^{N_L} l_i$.
-

DNNs have varying structures resulting in different traffic loads and data-patterns

between the IMC PEs. To accurately capture the NoC traffic of a given DNN configuration, we customize BookSim to evaluate the area, energy, and latency for interconnect, as shown in Figure 31. We use the customized version of BookSim to simulate the network traffic using non-uniform injection rate. We compute the injection rates for each source-destination pair in the multi-tiled architecture. The placement of tiles and routers in the IMC architecture has a direct impact on the interconnect performance. In this work, we incorporate the impact of mapping into the injection matrix calculation. The mapping of the DNN is performed such that each tile can have at least one layer while no layer is divided between two tiles. Figure 32 shows a sixteen tile IMC architecture with the tiles numbered. The red arrows show the data flow in the IMC architecture. Next, while evaluating the interconnect latency, we create an injection matrix that incorporates the position of the tile into the calculation by calculating the number of hops for each source-destination pair. Hence, the injection matrix incorporates the tile placement into the NoC latency calculation. Overall, the proposed approach can be generalized to any tile placement.

Algorithm 3 describes the steps performed to compute injection rates and obtain the interconnect latency. Without loss of generality, we assume that the number of nodes required in the interconnect is equal to the total number of tiles across all layers.

The injection rate calculation is shown in lines 5–11 of Algorithm 3. The injection rate is expressed in (4.2) from each source to each destination in each layer.

$$\lambda_{i,j,k} = \frac{A_i \times N_{bits} \times FPS}{T_i \times T_{i-1} \times W \times freq} \quad (4.2)$$

where N_{bits} , W , and FPS represent data precision and bus width, and frames-per-second throughput, respectively. In the numerator of (4.2), we multiply the number of input activations (A_i) for i^{th} layer by N_{bits} to obtain the total number of bits to be transferred from $(i - 1)^{\text{th}}$ layer to i^{th} layer for one frame of an image. We further

multiply this term with FPS to obtain the total number of bits transferred between layers per second. Then, we divide this term by the operating frequency ($freq$) to obtain the total number of bits transferred between layers per cycle. We assume an equal injection rate between all tiles in two consecutive layers. Therefore, to get the number of bits transferred from one tile to another in two consecutive layers, the denominator in (4.2) includes a multiplication between T_i and T_{i-1} . Thus, we divide the expression obtained so far by W to obtain the injection rate ($\lambda_{i,j,k}$). The injection rate from every source to every destination is the input to the interconnect simulator. The interconnect simulator then provides average latency to complete all transactions from $(i-1)^{\text{th}}$ layer to i^{th} layer ($(l_i)_{sim}$ cycles). Next, we multiply this latency with the number of bits from one tile to the next tile to get the total number of cycles required to transfer all data between two consecutive layers. Then, the latency from one layer to the next layer (l_i) is given by:

$$l_i = \frac{(l_i)_{sim} \times A_i \times N_{bits} \times FPS}{freq} \quad (4.3)$$

Finally, we accumulate the latency of all layers to compute the end-to-end interconnect latency as

$$L_{comm}^{sim} = \sum_{i=1}^{N_L} l_i \quad (4.4)$$

4.4 Analytical Performance Models for NoCs in IMC Architecture

In this section, we discuss an analytical approach to estimate NoC performance for IMC architecture. The analytical performance model of NoCs is primarily useful to overcome longer simulation time incurred by cycle-accurate NoC simulators. Specifically, we utilize analytical performance models for NoCs to compare the performance of NoC-tree and NoC-mesh for a given DNN. The analytical model of an NoC router

is adopted from the work proposed in Ogras *et al.* (2010). We extend this router model for NoC-tree and NoC-mesh to obtain end-to-end communication latency for different DNNs. Algorithm 4 describes the technique to evaluate the communication latency through analytical models. There are two major steps involved in analyzing the performance of an NoC: 1) Computing injection rate and 2) Computing contention probability matrix.

Computing injection rate matrix (Λ): First, the injection rate from each source to each destination (λ_{sd}) for each layer of the DNN is computed through (4.2). We note that the injection rate calculation incorporates the tile placement as detailed in Section 4.3.2. Each NoC router has five ports: North (N), South (S), East (E), West (W), and Self (Se). The injection rate at each port p of every router r ($\lambda_p^r, p \in \{N, S, E, W, Se\}$) is computed as:

$$\lambda_p^r = \frac{A_p^r \times N_{bits} \times FPS}{T_l \times T_{l+1} \times W \times freq} \quad (4.5)$$

where T_l denotes the number of tiles in the l^{th} layer. λ_p^r is a function of the number of activations through each port p of router r (A_p^r). From λ_p^r , the injection rate matrix for router r (Λ^r) is computed (as shown in line 5–7 of Algorithm 4), where $\Lambda^r = \{\lambda_{ij}^r\}, 1 \leq i \leq 5, 1 \leq j \leq 5, \lambda_{ij}^r = 0 \forall i \neq j$.

Computing contention matrix (C): Each element of the contention matrix C (c_{ij}) denotes the contention between port i and port j . To compute the contention matrix of router r ($C^r = \{c_{ij}^r\}$), we first compute forwarding probability matrix $F^r = \{f_{ij}^r\}$. f_{ij}^r denotes the probability of a packet that arrived at the port i of the router r to be forwarded to the port j , and is computed as shown in (4.6) Ogras *et al.* (2010).

$$f_{ij}^r = \frac{\lambda_{ij}^r}{\sum_{k=1}^5 \lambda_{jk}^r} \quad (4.6)$$

Algorithm 4: End-to-end latency computation through analytical models

```
1 Input: Input activation, Number of routers in each layer  $l$  ( $R_l$ ), Number of
   layers ( $N_L$ )
2 Output: End-to-end communication latency ( $L_{comm}$ )
3 for  $l = 1: N_L-1$  do
4   for  $r = 1: R_l$  do
5     /* Computing injection rate matrix */
6     Compute  $A_p^r$ 
7     Compute  $\lambda_p^r$  using (4.5)
8     Construct  $\Lambda^r$ 
9     /* Computing contention matrix */
10    Compute forwarding probability matrix ( $F^r$ )
11    Compute contention matrix ( $C^r$ )
12    /* Computing average waiting time */
13    Compute average queue length ( $N^r$ ) using (4.7)
14    Compute average waiting time ( $W_{avg}^r$ ) using (4.8)
15  end
16  Compute average latency for the layer ( $L_{avg}^l$ ) using (4.9)
17 end
18  $L_{comm}^{ana} = \sum_{l=1}^{N_L} L_{avg}^l$ 
```

The contention probability between port i and port j of the router r is computed as $c_{ij}^r = \sum_{k=1}^5 f_{ik}^r f_{jk}^r$. Line 10-11 of Algorithm 4 shows the computation of the contention matrix.

Next, the average queue length of each port of the router r (N^r) is computed through the technique described in Ogras *et al.* (2010).

$$N^r = (I - t\Lambda^r C^r)^{-1} \Lambda^r R, \quad (4.7)$$

where t is the service time of the router, and we assume $t = 1$ for our evaluation. R is the average residual time and is calculated assuming that the packets arrive in discrete clock cycles Mandal *et al.* (2019). Waiting time of the packets at each port of the router r is computed as $W^r = N^r (\Lambda^r)^{-1}$. End-to-end average latency for each layer l (L_{avg}^l) is obtained by averaging the waiting time through all 5 ports (W_{avg}^r) of

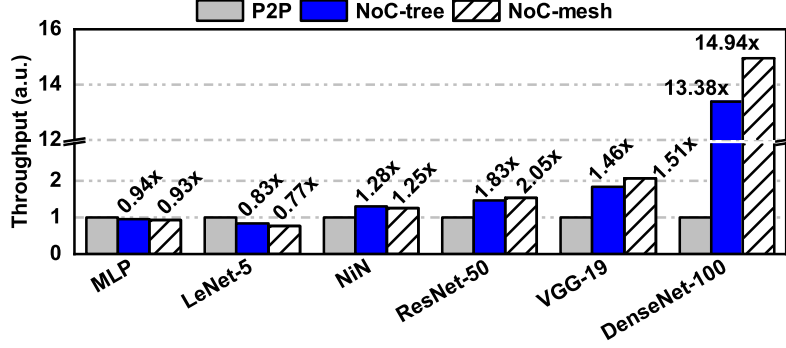


Figure 33. Throughput Comparison for Three Interconnect Topologies (P2P, NoC-tree, and NoC-mesh) for SRAM-based IMC Architecture, Normalized to P2P, for Different DNNs. NoC Shows Superior Performance and Scalability than P2P-Based Network.

router r and then adding across all routers, as shown in (4.8) and (4.9).

$$W_{avg}^r = \frac{1}{5} \sum_{p=1}^5 W_p^r \quad (4.8)$$

$$L_{avg}^l = \sum_{r=1}^R W_{avg}^r \quad (4.9)$$

Finally, total communication latency (L_{comm}^{ana}) is obtained by adding end-to-end average latency for each layer l as:

$$L_{comm}^{ana} = \sum_{l=1}^{N_L} L_{avg}^l \quad (4.10)$$

4.5 Connection-centric Architecture

In this section, we first discuss a multi-tiled SRAM-based IMC architecture with three different interconnect topologies, namely, P2P, NoC-tree, and NoC-mesh at the tile level. We perform a comprehensive analysis of these three interconnect-based SRAM IMC architectures for different DNNs using the simulation framework described in Section 4.3. Based on the analysis, we show the need for an NoC-based heterogeneous interconnect IMC architecture for efficient DNN acceleration. We assume all weights

are stored on-chip to avoid any DRAM access. The weights are loaded pre-execution and stored on-chip. The inputs are then loaded, and the computation is performed. There is no re-loading of intermediate results or weights from the off-chip memory during the execution of the DNN. The SRAM buffer is designed large enough to hold the intermediate results on-chip rather than moving them off-chip. Multiple inferences of the images can be performed using one pre-execution loading of the weights. Hence, we do not consider the initial loading of the weights into the energy calculation, consistent with prior work Shafiee *et al.* (2016); Song *et al.* (2017) compared in the manuscript. In addition, we adhere to layer-by-layer design instead of a layer-pipelined design, since a pipelined design introduces pipeline bubbles in the execution flow and complicates the control logic Qiao *et al.* (2018).

4.5.1 Design Space Exploration

We evaluate different performance metrics for a wide range of DNNs with P2P, NoC-tree, and NoC-mesh-based interconnect for SRAM-based IMC architectures. We consider routers with five ports, one virtual channel for NoCs and X-Y routing for NoC-mesh for this evaluation. To facilitate fair comparison, we normalize the throughput of the hardware architectures with three interconnect topologies to that of P2P interconnect.

Figure 33 shows the throughput comparison for different DNNs. For low connection density DNNs such as MLP and LeNet-5 LeCun *et al.* (1998), the choice of interconnect does not make a significant difference to the throughput, due to low data movement between different tiles of the IMC architecture. However, P2P interconnect results in $1.25\times$ and $2\times$ higher area cost than NoC-tree for MLP and LeNet-5, respectively.

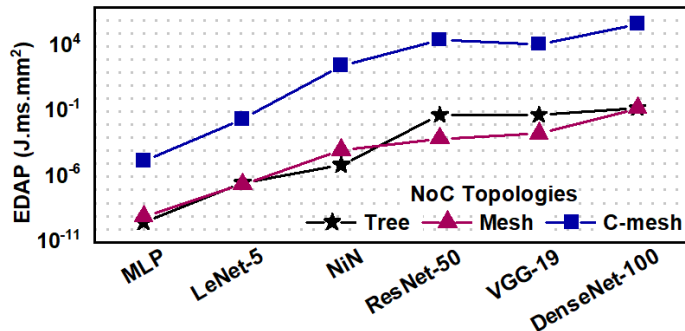


Figure 34. Comparison of Energy-Delay-Area Product (EDAP) of NoC-tree, NoC-mesh, and C-mesh for Different DNNs.

Hence, NoC-tree provides better overall performance than P2P for both MLP and LeNet-5. We further analyze dense DNNs such as NiN Lin *et al.* (2013), VGG-19, ResNet-50 He *et al.* (2016) and DenseNet-100 Huang *et al.* (2017). The performance comparison shows that the NoC-tree and NoC-mesh-based IMC architectures perform better than the P2P-based architectures (up to $15\times$ for DenseNet-100). Since higher connection density of the DNNs results in increased on-chip data movement, the routing latency dominates the end-to-end latency. We see a similar trend with ReRAM-based IMC architectures with similar throughput for MLP and $15\times$ improvement in throughput for DenseNet-100. Through this, we establish that the performance of the P2P-based IMC architecture (SRAM- or ReRAM-based) diminishes with increasing connection density. In contrast, the performance of the NoC-based (tree, mesh) IMC architecture scales better (Figure 33).

Exploration of other NoC topologies: Apart from tree and mesh, the other commonly known NoC topologies include c-mesh, hypercube, and torus. These topologies utilize more resources in terms of routers and links to reduce communication latency. However, the usage of more resources increases power consumption and the area of the NoC. For example, we performed experiments with c-mesh topology for

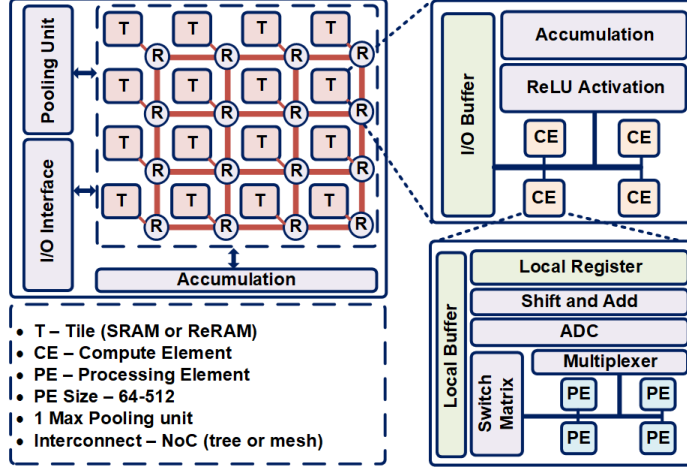


Figure 35. NoC-based Heterogeneous Interconnect IMC Architecture. A Three-Level Interconnect Scheme Consisting of NoC (Tree or Mesh) Between Tiles, P2P Network Between CEs, and Bus Between PEs.

different DNNs. Figure 34 compares energy-delay-area product (EDAP) of mesh-, tree- and c-mesh-based NoC for different NoC. We observe that while mesh- and tree-NoC provides comparable EDAP, the same for c-mesh is a minimum of five orders of magnitude higher than mesh- and tree-NoC.

4.5.2 Hardware Architecture

Based on the conclusions from Section 4.5.1, we derive an NoC-based heterogeneous interconnect IMC architecture for DNN acceleration. Figure 35 shows the hardware architecture which employs the heterogeneous interconnect system.

The proposed architecture is divided into a number of tiles, with each tile having a set of computing elements (CE). The tile architecture includes non-linear activation units, I/O buffer, and accumulators to manage data transfer efficiently. Each CE further consists of multiple processing elements (PE) or crossbar arrays, multiplexers,

buffers, a sense amplifier, and flash ADCs. The ADC precision is set to four bits such that there is minimum or no accuracy degradation for DNNs. In addition, the architecture does not utilize a digital-to-analog (DAC) converter; instead, it uses sequential signaling to represent multi-bit inputs Peng *et al.* (2019b). The proposed heterogeneous tile architecture can be used for both SRAM and ReRAM (1T1R) technologies. However, the peripheral circuits change based on the technology. In this work, we choose a homogeneous tile design consisting of four CEs and a CE structure consisting of four PEs. We evaluate both SRAM- and ReRAM-based IMC architectures for PE sizes varying from 64×64 to 512×512 . We sample 8 DNNs (LeNet, NiN, SqueezeNet, ResNet-152, ResNet-50, VGG-16, VGG-19, and DenseNet-100) and a crossbar size of 256×256 provides the lowest EDAP for 75% of the DNNs. Hence, in this work, we choose 256×256 as the crossbar size for both SRAM- and ReRAM-based IMC architectures. To maximize performance, the architecture uses heterogeneous interconnects. It employs the NoC-based interconnect on the global tile-level with a P2P interconnect (H-Tree) at the CE-level and bus at the PE-level due to significantly lower data volume. For low data volume, the NoC-based interconnect provides marginal performance gain while increasing energy consumption.

4.6 Experiments and Results

4.6.1 Experimental Setup

We consider an IMC architecture (Figure 35) with a homogeneous tile structure (SRAM, ReRAM) and one NoC router per tile. Table 13 summarizes the design parameters considered. We report the end-to-end latency, chip area, and total

energy obtained for a PE size of 256×256 for each of the DNNs using the simulation framework discussed in Section 4.3. We incorporate conventional mapping Shafiee *et al.* (2016), IMC SRAM bitcell/array design from Khwa *et al.* (2018) and 1T1R ReRAM bitcell/array properties from Chen *et al.* (2018). The IMC compute fabric utilizes a parallel read-out method. We utilize the same crossbar array size of 256×256 for both SRAM and ReRAM-based IMC architectures. All rows of the IMC crossbar are asserted together, analog MAC computation is performed along the bitline, and the analog voltage/current is digitized with a 4-bit flash ADC at the column periphery. We perform an extensive evaluation of the IMC architecture with both SRAM-based and ReRAM-based PE arrays for both NoC-tree and NoC-mesh. Unless specified, the NoC utilizes one virtual channel, a buffer size (all input and output buffers) of eight, and three router pipeline stages.

Table 13. Summary Of Design Parameters

PE array size	256×256	Read-out Method	Parallel
Technology node	32nm	Flash ADC resolution	4 bits
Cell levels	1 bit/cell	Operating frequency	1 GHz
Data precision	8 bits	NoC bus width	32

4.6.2 Evaluation of NoC Analytical Model

Figure 36 shows the accuracy of the analytical model (presented in Algorithm 4 in Section 4.4) to estimate the end-to-end communication latency with both NoC-tree and NoC-mesh. We observe that the accuracy is always more than 85% for different DNNs. On an average, the NoC analytical model achieves 93% accuracy with respect

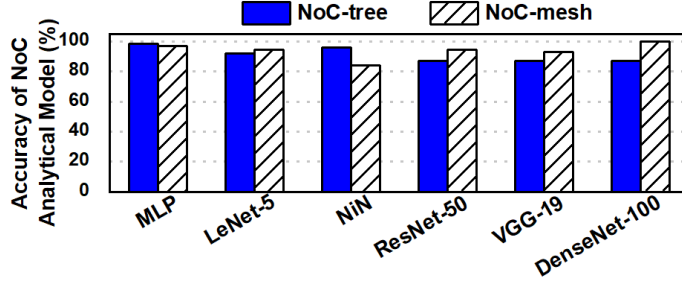


Figure 36. Accuracy of NoC Analytical Model for NoC-mesh and NoC-tree With Respect to Cycle-Accurate Simulator Jiang *et al.* (2013).

to cycle-accurate NoC simulation Jiang *et al.* (2013). Moreover, we achieve $100\times$ - $2000\times$ speed-up with the NoC analytical model with respect to cycle-accurate NoC simulation. Figure 37 shows the speed-up for different DNNs with mesh-NoC. This speed-up is useful to perform design space exploration by considering various sizes of PE arrays and other NoC topologies. Due to the high speed-up in NoC performance analysis, we achieve $8\times$ speed-up in overall performance analysis with respect to the framework which uses cycle-accurate NoC simulation.

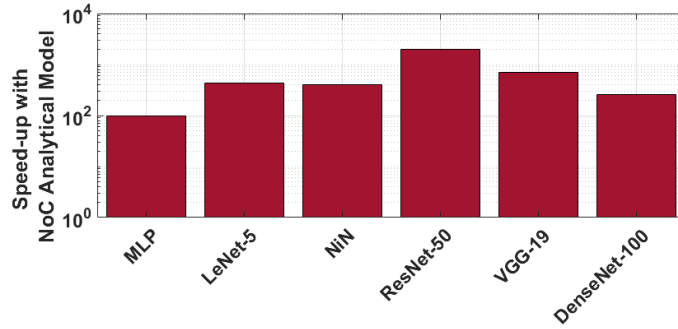


Figure 37. Speed-up (in NoC Simulation) with NoC Analytical Models with Respect to Cycle-Accurate NoC Simulation for Different DNNs with Mesh-NoC.

4.6.3 Analysis on Traffic Congestion in NoC

In this section, we present an analysis on traffic congestion in NoC for various DNNs. To this end we discuss about average queue length of different buffers in the NoC and worst case communication latency.

Analysis of the average queue length: Furthermore, we investigated the average queue length at different ports of different routers in the NoC through a cycle-accurate NoC simulator. We performed this experiment with mesh-NoC considering the configuration parameters shown in Table 13. Figure 38 shows that 64%-100% of the queues contain no flit when a new flit arrives for different DNNs. The percentage of queues with zero occupancy for LeNet-5 and NiN is 91% and 65%, respectively. These two DNNs utilize fewer number of routers, which results in less parallelism in data communication. However, we note that determining the optimal number of routers for a given DNN is not a scope of this work.

Figure 39 shows the average queue length for NiN and VGG-19 for the queues with non-zero length when a new flit arrives to the queues. We observe that the average queue length varies from 0.004-0.5 for these DNNs. Average queue length is very low in these cases since the injection rate to the queues are less, and NoC introduces a high degree of parallelism in data transmission between routers.

Analysis of the worst case latency: Furthermore, we extracted the worst-case latency (L_{max}) for different source to destination pairs of different DNNs with mesh-NoC. We compared L_{max} of each source to destination pair with corresponding average latency (L_{avg}). Then we compute mean absolute percentage deviation (MAPD) of

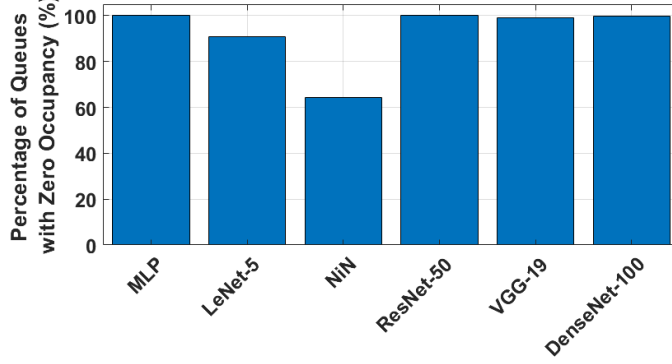


Figure 38. Percentage of Queues with Zero Occupancy When a New Flit Arrives.

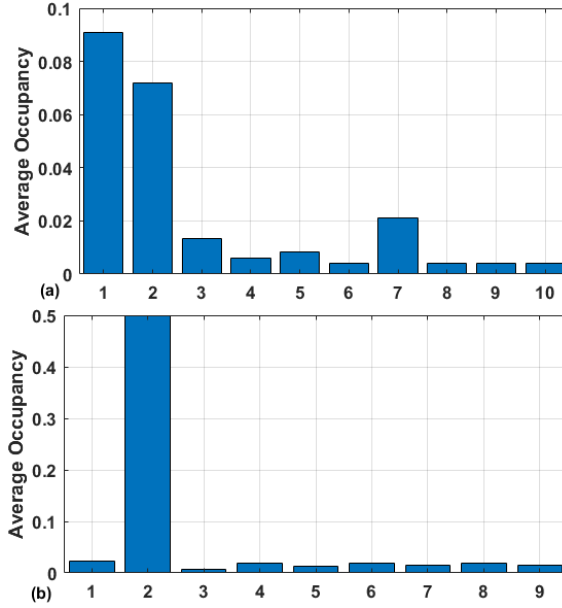


Figure 39. Average Occupancy of Queues with Non-Zero Length for (a) NiN, (b) VGG-19.

L_{max} from L_{avg} as the equation below.

$$MAPD = 100 \times \frac{1}{N} \sum_{i=1}^N \frac{(L_{max}^i - L_{avg}^i)}{L_{avg}^i} \quad (4.11)$$

Where N is the total number of source to destination pairs with non-zero average latency. L_{max}^i and L_{avg}^i are the worst-case latency and the average latency respectively of i^{th} source to destination pair. Table 14 shows the mean absolute percentage deviation for different DNNs. We observe that the deviation is insignificant, except

Table 14. Mean Absolute Percentage Deviation (*MAPD*) of Worst-Case NoC Latency From Average NoC For Different DNNs.

DNNs	MLP	LeNet-5	NiN	ResNet-50	VGG-19	DenseNet-100
MAPD(%)	0	9.13	20.76	0	0.14	0

for LeNet-5 and NiN. The deviations for these two networks are 9.13% and 20.76%, respectively.

Furthermore, in Figure 40 we show the absolute difference between the worst-case latency and the average latency for LeNet-5 and NiN for different source to destination pairs with non-zero latency. The maximum difference is 6 cycles both for LeNet-5 and NiN. This analysis shows that the worst-case latency has very less deviation from the average latency. Therefore, the studies of average queue length and worst-case latency confirm that there is no congestion in the NoC.

4.6.4 Guidance on Optimal Choice of Interconnect

4.6.4.1 Empirical Analysis

We compare the performance of the IMC architecture using both NoC-tree and NoC-mesh for both SRAM and ReRAM-based technologies. We perform the experiments for representative DNNs. MLP, LeNet-5, and NiN depict low connection density DNNs; ResNet-50, VGG-19, and DenseNet-100 depict high connection density DNNs. We report throughput and the product of energy consumption, end-to-end latency, and area (EDAP) of the IMC architectures. EDAP is used as the metric to guide the optimal choice for the interconnect for IMC architectures.

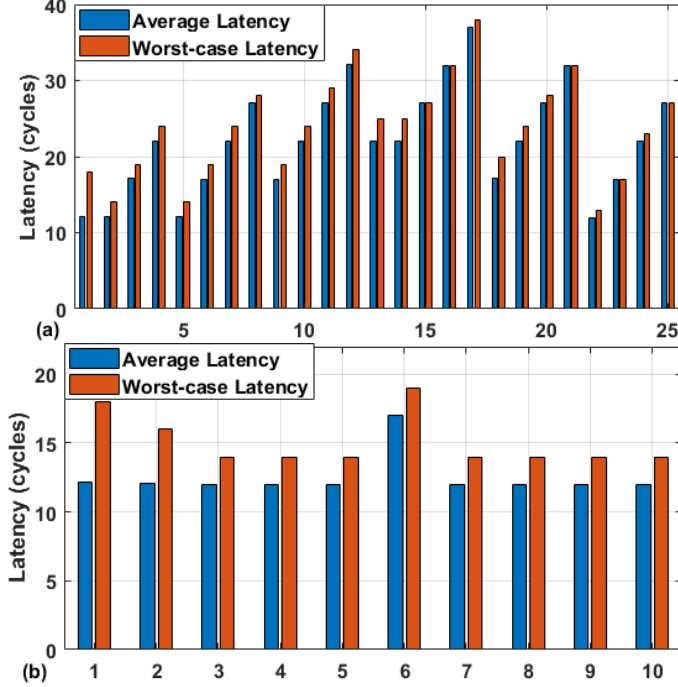


Figure 40. Comparison Between Average Latency and Worst-Case Latency for Source to Destination Pairs with Non-Zero Latency for (a) LeNet-5 and (b) NiN.

Figure 41(a) shows the ratio of the throughput of the SRAM-based IMC architecture using NoC-tree and NoC-mesh interconnect. We normalize the throughput values with respect to that of NoC-tree. NoC-tree performs better than the NoC-mesh for DNNs with low connection density. This is because of the reduced injection bandwidth into the interconnect. In addition, while NoC-mesh provides lower interconnect latency than NoC-tree, it comes at an increased area and energy cost. However, NoC-mesh performs better for DNNs with high connection density. The improved performance stems from the reduced interconnect latency for high injection rates of data into the interconnect. The reduction in latency is much higher than the additional overhead due to both area and energy of NoC-mesh.

To better understand the performance, we report the EDAP for the SRAM-

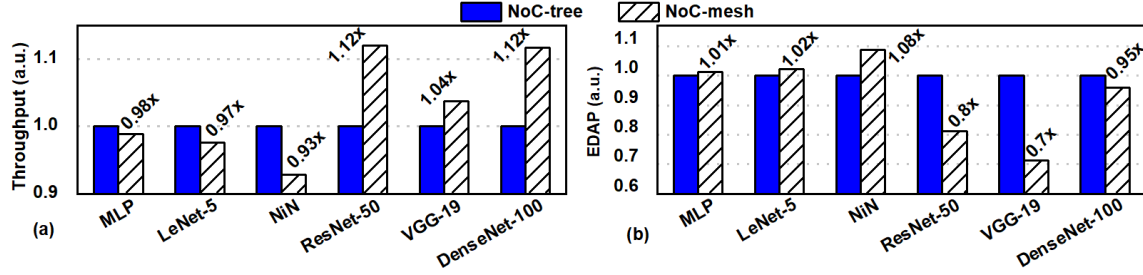


Figure 41. (a) Normalized Throughput and (b) Normalized EDAP of NoC-tree and NoC-mesh-based On-Chip Interconnect for SRAM-based IMC Architecture for Different DNNs. Dense DNNs Favor NoC-mesh While NoC-tree is Sufficient for Shallow DNNs.

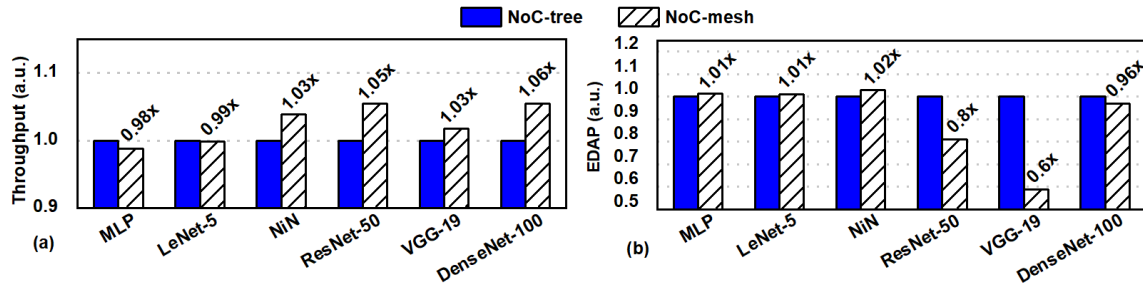


Figure 42. (a) Normalized Throughput and (b) Normalized EDAP of NoC-tree and NoC-mesh-based On-Chip Interconnect for ReRAM-based IMC Architecture for Different DNNs. Dense DNNs Favor NoC-mesh While NoC-tree is Sufficient for Shallow DNNs.

based IMC architecture. Figure 41(b) shows normalized EDAP of the NoC-tree and NoC-mesh for both low and high connection density DNNs. DNNs with low connection density have significantly lower EDAP for NoC-tree than that with NoC-mesh. Such an improved EDAP performance for NoC-tree complements the observation for throughput. At the same time, for DNNs with high connection density, the EDAP of NoC-mesh is lower than that of the NoC-tree for IMC architectures. A similar observation is seen for ReRAM-based IMC architectures as shown in Figure 42(a) and Figure 42(b). In contrast to the SRAM-based IMC architecture, NiN provides better performance in throughput for the NoC-mesh interconnect. At the same

time, NoC-tree provides better EDAP compared to NoC-mesh, similar to that of the SRAM-based IMC architecture.

Furthermore, we performed another two sets of experiments with NoC-mesh and NoC-tree by varying the number of virtual channels and bus-width. In this case, we consider ReRAM-based IMC architectures. Figure 43 shows the comparison with different numbers of virtual channels, and Figure 44 shows the comparison with different bus width of the NoC. We observe similar trends for different DNNs with a different NoC configurations.

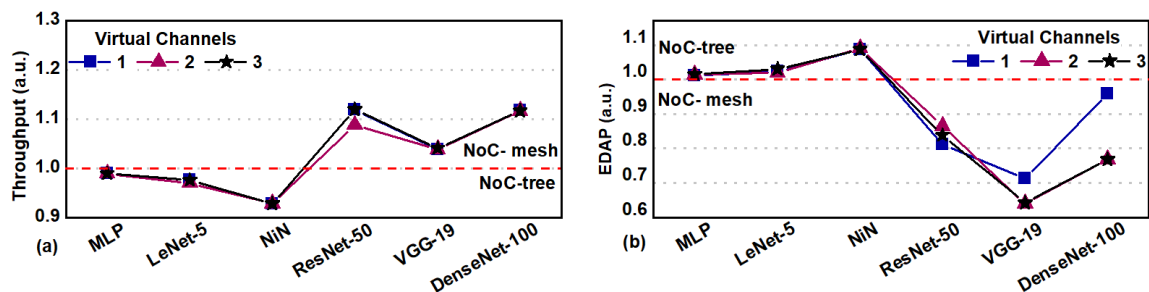


Figure 43. Assessment of (a) Throughput and (b) EDAP Between NoC-tree and NoC-mesh With Different Numbers of Virtual Channels for Different DNNs. The Throughput is Normalized to That of NoC-tree. The Preferred NoC Topology for Optimal Performance is Shown for the Regions Above and Below the Red Line.

Since the injection rate to the input buffer of the NoC is always low (less than one packet in 100 cycles), increasing the number of virtual channels does not alter the inference latency significantly. Therefore, throughput remains similar (for all DNNs) both for NoC-tree and NoC-mesh with an increasing number of virtual channels. However, the area and power of both NoC-mesh and NoC-tree increase proportionally with an increasing number of virtual channels. Therefore, the normalized EDAP (EDAP of mesh-NoC divided by the EDAP of tree-NoC) is similar for all DNNs with different numbers of virtual channels.

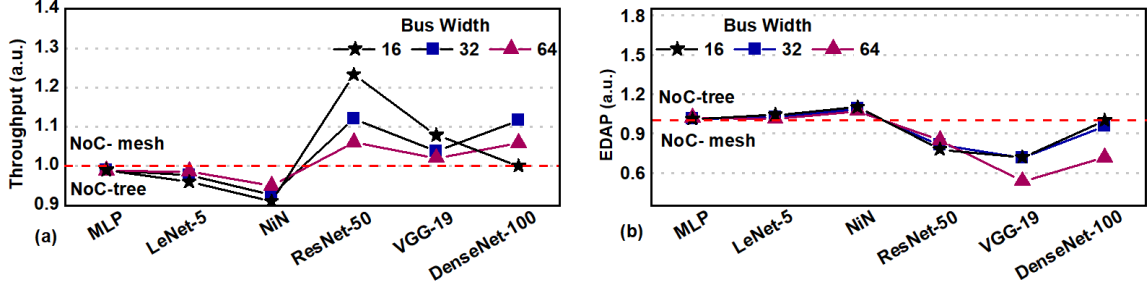


Figure 44. Assessment of (a) Throughput and (b) EDAP Between NoC-tree and NoC-mesh With Different Bus Width for Different DNNs. The Throughput is Normalized to That of NoC-tree. The Preferred NoC Topology for Optimal Performance is Shown for the Regions Above and Below the Red Line.

While we change the bus width of the NoC, the latency increases (decreases) with decreasing (increasing) bus width proportionally, i.e., the latency with a bus width of 32 is twice than the latency with bus width of 64. Moreover, the area and power of the NoC increases (decreases) with increasing (decreasing) bus width proportionally. Therefore, the normalized EDAP is similar for all DNNs with different NoC bus widths. Consequently, for all configurations, we obtain exactly the same guidance on the choice of NoC for different DNNs. Therefore, the guidance is consistent across different parameters of NoCs.

4.6.4.2 Theoretical Analysis

We utilize the analytical model in Section 4.4 and the experimental results described in Figure 41 and Figure 42 to provide guidance on the optimal choice of interconnect for IMC architectures. The injection rate at each port of an NoC router for each layer of the DNN is expressed in (4.5). The numerator of (4.5) denotes the total data volume between i^{th} layer and $(i + 1)^{\text{th}}$ layer for each port of the router per cycle. This is divided by $(T_i \times T_{i+1} \times W)$ to obtain the injection rate from for each port

of every router as detailed in Section 4.4. For a fixed NoC-based IMC architecture, target throughput (FPS), frequency of operation ($freq$), and bus width (W) remain constant. Hence, from (4.5) we obtain,

$$\lambda_i \propto \frac{A_i \times N_{bits}}{T_i \times T_{i+1}} \quad (4.12)$$

Let the connection density for i^{th} layer be ρ_i and the number of neurons be μ_i . Data volume between i^{th} and $(i + 1)^{\text{th}}$ layer is proportional to the product of ρ_i and μ_i , as shown in (4.13).

$$(A_i \times N_{bits}) \propto (\rho_i \times \mu_i) \quad (4.13)$$

Additionally, the number of tiles in i^{th} layer is directly proportional to μ_i . Hence, from (4.12) and (4.13) we get,

$$\lambda_i \propto \frac{\rho_i \times \mu_i}{\mu_i \times \mu_{i-1}} = \frac{\rho_i}{\mu_{i-1}} \quad (4.14)$$

Generalising (4.14), we obtain,

$$\lambda \propto \frac{\rho}{\mu} \quad (4.15)$$

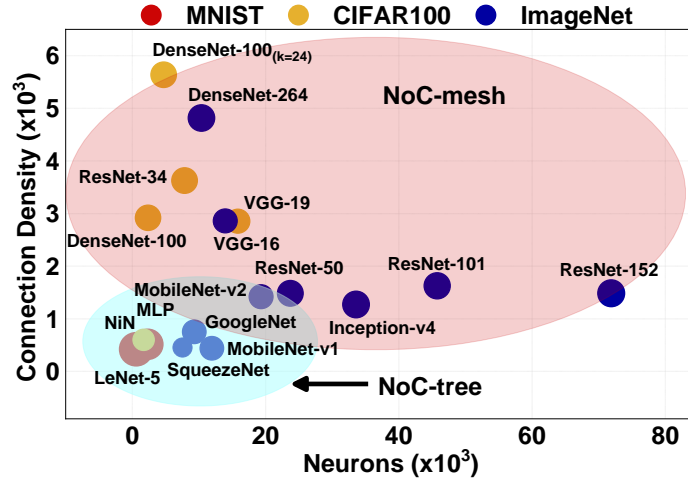


Figure 45. Optimal NoC Topology for IMC Architectures for Different DNNs.

Therefore, the injection rate is directly proportional to the connection density and inversely proportional to the number of neurons of the DNN. Figure 45 presents the preferred regions for NoC-tree and NoC-mesh for best throughput for different DNNs with IMC architectures. If the connection density of a DNN is more than 2×10^3 , then NoC-mesh is suitable. If the connection density is less than 1×10^3 , then NoC-tree is appropriate. Both NoC-tree and NoC-mesh are suitable for the DNNs with connection density in the range of 1×10^3 - 2×10^3 (the region where red and blue ovals overlap in Figure 45).

4.6.5 Comparison with state-of-the-art architectures

Table 15 compares the proposed architecture with state-of-the-art DNN accelerators. Prior works show the efficacy of their ReRAM-based IMC architectures for VGG-19 DNN Qiao *et al.* (2018); Shafiee *et al.* (2016); Song *et al.* (2017). Hence for comparison, we choose VGG-19 network as the representative DNN. Moreover, we compare the dynamic power consumption of the DNN hardware since prior work utilizes dynamic power in their results, hence making the comparison consistent. The inference latency of the proposed architecture with SRAM arrays is $2.2 \times$ lower than the architecture with ReRAM arrays. The proposed ReRAM-based architecture achieves $4.7 \times$ improvement in FPS and $6 \times$ improvement in EDAP than AtomLayer Qiao *et al.* (2018). The improvement in performance is attributed to the optimal choice of interconnect coupled with the absence of off-chip accesses. The proposed ReRAM-based architecture consumes $400 \times$ lower power per frame along with $1.74 \times$ improvement in FPS than PipeLayer Song *et al.* (2017). Moreover, there is a $5.4 \times$ improvement in

inference latency compared to ISAAC Shafiee *et al.* (2016), which is achieved by the heterogeneous interconnect structure.

Table 15. Inference Performance Results for VGG-19. *Reported in Qiao *et al.* (2018)

	Latency	Power/frame (W/frame)	FPS	EDAP (J.ms.mm²)
Proposed-SRAM	0.68	1.96	1458	0.46
Proposed-ReRAM	1.49	0.43	670	0.28
AtomLayer Qiao <i>et al.</i> (2018)	6.92	4.8	145	1.58
PipeLayer Song <i>et al.</i> (2017)	2.6*	168.6	385	94.17
ISAAC Shafiee <i>et al.</i> (2016)	8.0*	65.8	125	359.64

4.6.6 Connection Density and Hardware Performance

Figure 26 showed a trend of DNNs moving toward a high connection density structure for performance and low connection density structure for compact models. Figure 46 shows the performance for both P2P and NoC-based interconnect at the tile-level for IMC architecture for DNNs with different connection density. We observe a steep increase in total latency with a P2P interconnect. However, the IMC architecture with NoC interconnect shows a stable curve as we move towards high connection density DNNs. With the advent of neural architecture search (NAS) techniques Xie *et al.* (2019); Zoph *et al.* (2018), DNNs are moving towards a highly branched structure with very high connection densities. Hence, the NoC-based heterogeneous interconnect architecture provides a scalable and suitable platform for IMC acceleration of DNNs.

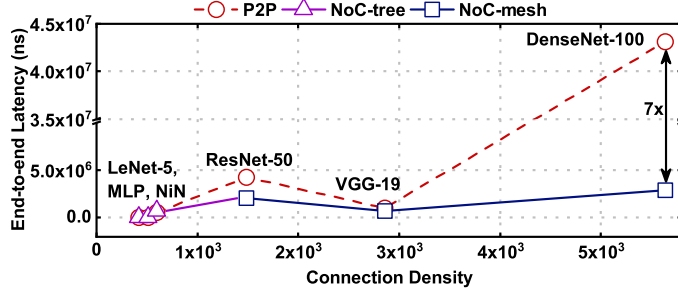


Figure 46. Appropriate Selection of NoC Topology Significantly Improves Performance for Both SRAM- and ReRAM-based IMC Architectures.

4.7 Conclusion

The trend of connection density in modern DNNs requires a re-evaluation of the underlying interconnect architecture. Through a comprehensive evaluation, we demonstrate that the P2P-based interconnect is incapable of handling the high volume of on-chip data movement of DNNs. Further, we provide guidance backed by empirical and analytical results to select the appropriate NoC topology as a function of the connection density and the number of neurons. We conclude that NoC-mesh is preferred for DNNs with high connection density, while NoC-tree is suitable for DNNs with low connection density. Finally, we show that the NoC-based heterogeneous interconnect IMC architecture achieves 6× lower EDAP than state-of-the-art ReRAM-based IMC accelerators.

INTERCONNECT-AWARE AREA AND ENERGY OPTIMIZATION FOR
IN-MEMORY ACCELERATION OF DNNs

5.1 Introduction

Deep neural networks (DNNs) achieve accuracy levels that exceed human-level perception for a variety of applications, such as computer vision, natural language processing, and medical imaging. Higher accuracy comes with increased computational complexity and model size which in turn require more memory to store both the weights and activations. Due to limited on-chip memory capacity, this leads to a significant amount of communication with off-chip memory Chen *et al.* (2019), whose energy is $1,000\times$ higher than the energy required to perform computations. Therefore, there is a strong need to minimize energy cost related to memory access.

In-memory computing (IMC) has emerged as a promising method to address the memory access bottleneck. Both SRAM and nanoscale non-volatile memory (e.g. resistive RAM or ReRAM) based IMC hardware architectures provide a dense and parallel structure to achieve high performance and energy efficiency Shafiee *et al.* (2016); Song *et al.* (2017); Valavi *et al.* (2019). However, state-of-the-art IMC architectures which employ an array of homogeneous tiles (tiles having the same size), have severely underutilized crossbar arrays or processing element (PE). For example, Figure 47(a) shows that most commonly used DNNs utilize less than 65% of PEs in an SRAM-based homogeneous IMC architecture with PE size of 256×256 Chen *et al.* (2018); the exceptions are the two VGG networks where input features in most layers

are multiples of 256. The low utilization of PE arrays occurs due to the non-uniform distribution of weights across different layers Ma *et al.* (2018). Reduced utilization results in increased area, leading to additional on-chip interconnect and higher energy consumption.

Communication latency between different tiles also plays a crucial role in overall hardware performance. Conventional hardware architectures use H-Tree, point-to-point (P2P), and bus architectures Chen *et al.* (2018) for on-chip communication, resulting in lower performance. Figure 47(b) shows that up to 90% of the total latency is spent on communication in a bus-based H-Tree interconnect Chen *et al.* (2018). In contrast, the NoC-based interconnect provides lower communication latency for DNN accelerators, as demonstrated in Chen *et al.* (2019); Shafiee *et al.* (2016).

State-of-the-art hardware architectures typically implement multiple DNNs on the same NoC Shafiee *et al.* (2016); Chen *et al.* (2019). Small DNNs, like NiN, under-utilize the NoC, while large DNNs, such as DenseNet, lead to congestion. Moreover, if one NoC node is employed per tile as in Chen *et al.* (2019); Kwon *et al.* (2017), communication energy constitutes 20%–40% of the total energy, as shown in Figure 47(c). Here the global interconnect energy from BookSim Jiang *et al.* (2013) is combined with the local interconnect energy (within tile) from NeuroSim Chen *et al.* (2018). As we show later in Section V-C, the initial layers account for a large portion of the total number of packets. Thus, if a tile is serviced by a dedicated router, the number of packets per router is very high for the initial layers. This increases the total communication latency and in-turn reduces energy efficiency.

In this work³, we first propose an area-aware optimization technique that improves

³Work done in collaboration with Sumit K. Mandal (UW–Madison)

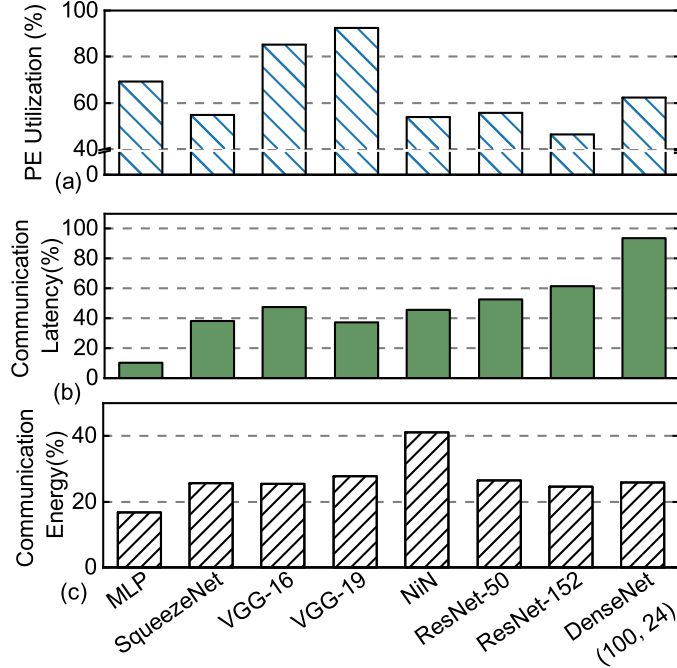


Figure 47. Experiments Using NeuroSim Chen *et al.* (2018) for Different DNNs Using Homogeneous SRAM-based IMC Architecture With P2P Interconnect Show that (a) Less than 65% of the PEs are Utilized, Except for VGG Network, (b) up to 90% of the End-to-End Latency is Spent on On-Chip Communication, (c) Communication Energy Constitutes 20%–40% of the Total Energy with NoC Having One Node per Tile Chen *et al.* (2019); Kwon *et al.* (2017).

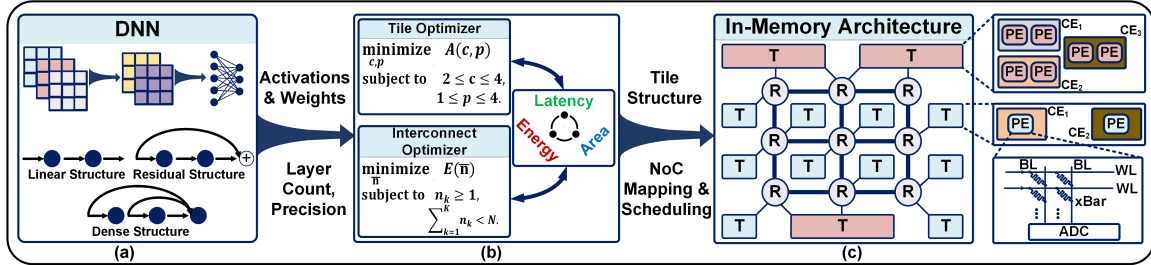


Figure 48. Overview of the Proposed Methodology to Obtain an Area- and Energy-Optimized IMC DNN Accelerator. (a) Shows DNNs with Various Connection Structures, (b) Shows the Joint Optimization Technique, and (c) Shows the Generated Heterogeneous IMC Architecture With Optimized Interconnect. Each PE Consists of the Crossbar of the Same Size, with Different Numbers of PEs Within Each Tile.

the PE array utilization. This is achieved by generating a heterogeneous tile-based IMC architecture that consists of tiles of different sizes, i.e. with different numbers

of PEs where each PE is of the same size. Second, we minimize the communication energy across a large number of tiles using an NoC architecture with optimized tile-to-router mapping and scheduling. Overall, our proposed area and energy optimization methodology generates a heterogeneous IMC architecture coupled with an optimized NoC for DNN acceleration. Thorough experimental evaluations show up to 62% improvement in PE utilization, 78% reduction in area, and 78% lower energy-area product for a wide range of modern DNNs such as DenseNet (100,24), and ResNet-152. The major contributions of this paper are as follows:

- An area-aware optimization technique to maximize the PE array utilization to reduce area,
- An energy-aware NoC mapping and scheduling technique to minimize communication latency and energy,
- Experimental demonstration of the proposed methodology showing up to 78% reduction in energy-area product with respect to conventional IMC architectures.

5.2 Overview of the Proposed Methodology

Our methodology to generate an area and energy-optimized IMC architecture for a given DNN is described in Figure 48. It targets the following three objectives.

- Increased PE array utilization for overall area reduction,
- Optimized interconnect for energy-efficient on-chip communication,
- Integration of area-aware and energy-aware optimization to generate the heterogeneous IMC architecture.

The skeleton architecture consists of multiple tiles where each tile is built with several

compute elements (CEs), and each CE consists of multiple PEs. Each PE is a crossbar array and all PEs are of the same size. The proposed hierarchical architecture enables efficient data transfer, reduced accumulator size, and distributed buffer structure. Figure 48(c) shows the skeleton architecture.

The methodology takes inputs such as the network structure and precision of the data for the target DNN, as illustrated in Figure 48(a). Additional inputs include a maximum number of routers and minimum/maximum number of PEs per CE, and CEs per tile. A joint optimization is performed on these inputs as shown in Figure 48(b). First, an area-aware optimization is done to improve the utilization of the PE arrays to produce the heterogeneous tile architecture. Second, an energy-aware NoC optimization is used to produce the optimal distribution of routers across different layers of DNNs on this architecture. Our optimization methodology also includes a scheduling technique to avoid congestion in the NoC, resulting in reduction of NoC energy and end-to-end latency. The proposed method supports different IMC technologies, such as SRAM, ReRAM, and PCM.

Overview of Generated Architecture: The heterogeneous IMC architecture generated by the proposed methodology is shown in Figure 49. It consists of tiles along with non-linear activation units, I/O buffer, and accumulators to manage data transfer efficiently. Each tile consists of a different number of CEs, buffers, and an accumulator. Each CE consists of several PE arrays, multiplexers, buffers, ADCs, and sample and hold circuits. In addition, the architecture does not use a DAC and assumes all weights are stored on-chip similar to Shafiee *et al.* (2016); Song *et al.* (2017). Such an assumption results in a large chip area for high-precision DNNs, but the chip area can be reduced by exploiting low-precision quantization and slim DNN models.

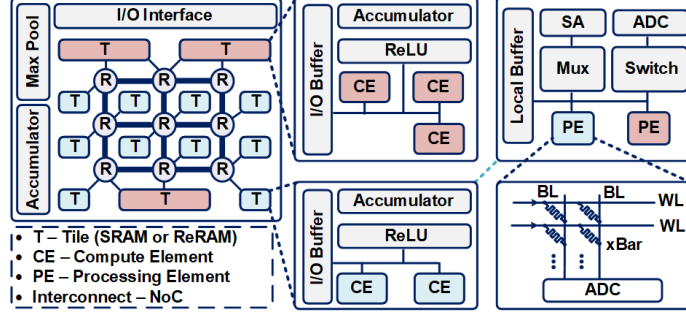


Figure 49. The Hierarchical Structure of the Generated Heterogeneous In-Memory Architecture with Optimized NoC by Utilizing the Proposed Area and Energy Optimization Methodology.

The proposed architecture employs NoC-based interconnect at the global tile level with an H-Tree interconnect at the CE level, and bus interconnect at the PE level. Since the injection rate is much lower at the CE and PE level than that at the tile level, H-Tree and bus interconnect provide ample performance. At the tile level, the proposed energy-optimized mesh NoC with X–Y routing algorithm is used. We consider mesh NoC as the interconnect topology since mesh NoC is the state-of-the-art interconnect topology both in the realm of computer architecture Jeffers *et al.* (2016) and DNN accelerators Chen *et al.* (2019); Shafiee *et al.* (2016).

Table 16. Summary Of Notation

Symbol	Definition	Symbol	Definition
T_k	Number of tiles in k^{th} layer	N_{bits}	Weight precision
Kx_k, Ky_k	Kernel size	c_k	Number of CEs in T_k
$(PE_x)_k, (PE_y)_k$	Size of the PE array	p_k	Number of PEs in c_k
I_k	Input activations in k^{th} layer	N_k^{if}, N_k^{of}	Number of i/p and o/p features

5.3 Area and Energy Optimization methodology

In this section, we describe our proposed methodology that performs area and energy optimizations to construct an optimized architecture for a given DNN.

Area-Aware Tile Optimization: Mapping different DNNs to a homogeneous tile-based hardware architecture results in considerable under-utilization of the PE arrays as shown in Figure 1(a). We propose an area-aware optimization to generate a heterogeneous tile-based (varying tile sizes) hardware architecture that increases PE array utilization and in-turn decreases the total area. Using the weight mapping scheme in Shafiee *et al.* (2016), we calculate the number of rows N_k^r in (5.1) and columns N_k^c in (5.2) of PEs required for the k^{th} layer of the DNN with K layers.

$$N_k^r = \left\lceil \frac{Kx_k \times Ky_k \times N_k^{if}}{(PE_x)_k} \right\rceil \quad (5.1)$$

$$N_k^c = \left\lceil \frac{N_k^{of} \times N_{bits}}{(PE_y)_k} \right\rceil \quad (5.2)$$

N_{bits} , N_k^{if} , N_k^{of} , $(PE_x)_k$, and $(PE_y)_k$ are defined in Table 12. The proposed area optimization applies to any other crossbar mapping method by changing the calculation of N_k^r and N_k^c . Next, we calculate the number of tiles T_k required for the k^{th} layer of the DNN, as shown in (5.3):

$$T_k = \left\lceil \frac{N_k^r \times N_k^c}{c_k \times p_k} \right\rceil \quad (5.3)$$

The parameters c_k and p_k are unknown and are found by the proposed optimization. To generate an optimal tile architecture, we propose an objective function that minimizes the residual area to increase PE utilization:

$$A_k(c_k, p_k) = (c_k \times p_k \times T_k - N_k^r \times N_k^c) \times T_k^2 \quad (5.4)$$

where the product of c_k , p_k , and T_k represents the number of PEs used to map the k^{th} layer of the DNN. The product of N_k^r and N_k^c provides the actual number of PEs

required to map the k^{th} layer of the DNN. The difference between the two products signifies the residual area. Additionally, the total area and energy of the chip are directly proportional to the number of tiles required to realize the k^{th} layer of the DNN. Hence, we incorporate the area and energy cost of the hardware by multiplying the residual area by the square of T_k . Finally, we minimize the objective function with an upper and lower bound on c_k and p_k for all layers of the DNN as shown in (5.5):

$$\begin{aligned}
 & \underset{c_k, p_k}{\text{minimize}} && A_k(c_k, p_k); \quad k = 1, \dots, K, \\
 & \text{subject to} && c_{\min} \leq c_k \leq c_{\max}, \\
 & && p_{\min} \leq p_k \leq p_{\max}.
 \end{aligned} \tag{5.5}$$

where $c_{\min}, c_{\max}, p_{\min}$, and p_{\max} are user-defined constraints that are an input to the optimization engine.

By solving the problem in (5.5), we obtain the optimal number of CEs in a tile (c_k) and the number of PEs in each CE (p_k) for each layer of the DNN. This results in a heterogeneous tiled IMC architecture with high PE array utilization and low area.

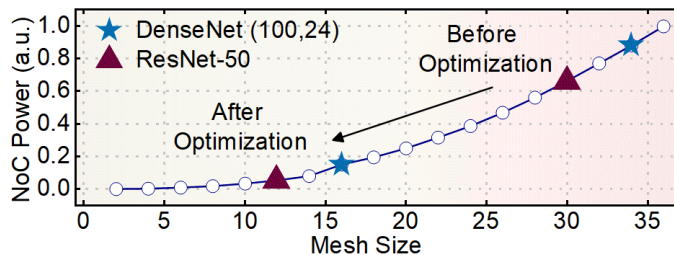


Figure 50. NoC Optimization Effectively Reduces the Power Consumption Because of its Non-Linear Dependence on the Mesh Size. We Obtain NoC Power Through BookSim Jiang *et al.* (2013) Simulations.

Energy-Aware Optimization for NoC: As a result of the proposed area-aware optimization, the total number of tiles in an IMC architecture can be very high. For example, DenseNet (100,24) requires 1,088 tiles Chen *et al.* (2018). For such an

architecture, one-to-one mapping of a router to tile Kwon *et al.* (2017) will require a large number of NoC routers and consume high power, as shown in Figure 50. Therefore, in our framework we introduce an energy-aware optimization for the NoC. Mapping Tiles to Routers: We first construct an objective function that represents the NoC energy consumption. Let n_k be the number of routers required for the k^{th} layer of the DNN. The number of activations communicated between n_k and n_{k+1} routers is I_k . Hence, the number of activations between each source-destination pair is given by $I_k/(n_k \times n_{k+1})$. The total amount of communication volume can be found by adding this across all K layers and routers:

$$E(\bar{\mathbf{n}}) = \left(\sum_{k=1}^{K-1} \frac{I_k}{n_k n_{k+1}} \right) \left(\sum_{k=1}^K n_k \right) \quad (5.6)$$

$E(\bar{\mathbf{n}})$ is proportional to the total communication energy of the DNN assuming that all transactions have a uniform size. We minimize this objective function with an upper bound on the total number of routers, N as:

$$\begin{aligned} & \underset{\bar{\mathbf{n}}}{\text{minimize}} && E(\bar{\mathbf{n}}) \\ & \text{subject to} && n_k \geq 1; \quad k = 1, \dots, K, \\ & && \sum_{k=1}^K n_k < N. \end{aligned} \quad (5.7)$$

where the first constraint ensures that each layer of the DNN is associated with at least one router. N is a user-defined constraint (input to the optimization framework) that represents the maximum number of routers in the IMC architecture. At the end of this optimization, we obtain the number of routers needed for each layer (n_k) of the DNN.

Packet Scheduling in NoC: If the activations of a layer are injected into the NoC in the order of computation, there is a high possibility of congestion resulting in high

communication latency in the NoC. Therefore, we propose a scheduling technique for the NoC to schedule the activations between two layers of the DNN. The scheduling technique is applied on top of the optimal tile-to-router mapping for the NoC. This scheduling technique provides a starting time for activations from each source to destination pair in the NoC. Without loss of generality, we assume that all activations for a particular source-destination pair can be injected back-to-back.

Using the NoC topology and the routing algorithm, we first find the source-destination pairs which contend for the same link in the NoC. We model each source-destination pair (sd) as an individual task. The start time of the task corresponding to the pair sd is denoted by t_{sd} and the duration of the task equals to the number of packets for that pair (n_{sd}). Next, we put constraints on the start time of each task so that there is no contention between two transactions for the same link. The set of all tasks is denoted by \mathcal{T} and the set of all non-overlapping tasks is denoted by \mathcal{C} . (5.8) shows the formulation of the non-overlap constraint, where the start time of two tasks is separated by the duration. Furthermore, the start time of all tasks are integers and greater than zero. We add one terminal task with the constraint that the start time of the terminal task ($t_{terminal}$) is greater than the start time of any of the source-destination pairs. We minimize $t_{terminal}$ to obtain the optimal schedule for all source-destination pairs.

$$\begin{aligned}
& \text{minimize} && t_{terminal} \\
& \text{subject to} && t_{mn} > t_{pq} + n_{pq} \vee t_{pq} > t_{mn} + n_{pq}, \\
& && \forall t_{mn}, t_{pq} \in \mathcal{C}, \\
& && t_{xy} \geq 0, \forall t_{xy} \in \mathcal{T} \\
& && t_{terminal} > t_{xy} + n_{xy}, \forall t_{xy} \in \mathcal{T}.
\end{aligned} \tag{5.8}$$

5.4 Experimental Evaluation

5.4.1 Experimental Setup

We evaluate the proposed methodology for a wide range of DNNs. An in-house simulator is developed to analyze the performance of the generated architecture for different DNNs. The inputs of the simulator primarily include the DNN structure, technology node, number of tiles, configuration of each tile, type of in-memory technology (ReRAM, SRAM, etc), number of bits per cell, and frequency of operation. The circuit part and interconnect part of the simulator are calibrated with NeuroSim Chen *et al.* (2018) and BookSim Jiang *et al.* (2013), respectively. The simulator performs the mapping of the entire DNN to a multi-tiled IMC architecture Shafiee *et al.* (2016) based on the output from the area-aware optimization. The number of tiles and configuration of each tile is taken as input to perform the DNN mapping. The circuit simulator reports performance metrics, such as area, energy, and latency of the computing logic. The interconnect performance is evaluated using the cycle-accurate NoC simulator. The circuit simulator provides the number of tiles per layer, activations, and the number of layers as output, which are taken as input by the NoC simulator. The NoC simulator computes the area, energy, and latency based on the number of routers and the computed schedules through our proposed approach. The overall performance of the architecture is calculated by combining the circuit-level and interconnect-level performance results.

Finally, to evaluate the effectiveness of the proposed methodology, we compare the generated IMC architecture using 1T1R ReRAM bitcell/array Chen *et al.* (2018) with state-of-the-art ReRAM-based IMC architectures.

Baseline Architecture: We incorporate IMC SRAM bitcell/array design Chen *et al.* (2018) for the baseline architecture. We use 256×256 crossbar array with 32nm technology Shafiee *et al.* (2016); all 256 rows are asserted together, analog MAC computation is performed along the bitline, and the analog voltage/current is digitized with a 4-bit flash ADC at the column periphery. The frequency of operation is 1GHz. Parallel read-out is assumed for the crossbar with a 4-bit flash ADC at the column periphery Chen *et al.* (2018). We consider NoC bus width of 32. It should be noted that our proposed methodology applies to other values of these parameters.

5.4.2 Area-Aware Optimization for Heterogeneous Tiles

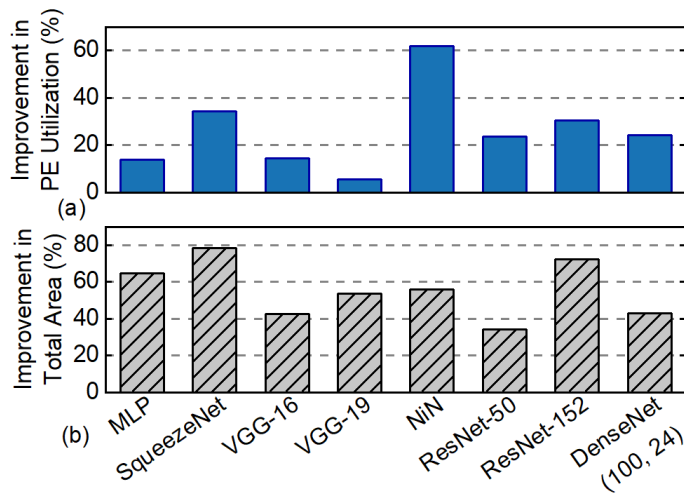


Figure 51. Improvement With Respect to the Baseline Architecture (SRAM) in (a) PE Utilization and (b) Total Area with the Proposed Area-Aware Optimization.

We compare both PE array utilization and the total chip area against the baseline architecture. The area-optimal tile architecture is obtained by following the methodology described in Section 5.3. For our evaluation, we consider $c_{min} = 2$, $c_{max} = 4$, $p_{min} = 1$ and $p_{max} = 4$. Increasing the p_{max} beyond 4 results in very low PE

array utilization. The optimal value of c_k and p_k is always less than 5, otherwise, the utilization starts reducing drastically. At the same time, the number of CEs is always more than 1 to keep the tile count reasonable, to limit the energy consumption.

Figure 51(a) shows the improvement in crossbar utilization with heterogeneous tile architecture for a range of DNNs. The improvement in utilization is the highest (62%) for NiN, and the least for VGG-19 (6%). The low improvement for VGG-19 is attributed to the baseline utilization being as high as 93%. This is because the number of input features in most layers are multiples of 256. The increase in PE utilization results in chip area reduction as shown in Figure 51(b). Compared to the homogeneous tile structure, we achieve a 79% reduction in area for SqueezeNet and 57% for NiN. For VGG-19, a higher area improvement is observed due to the reduction in both the number of PE arrays and associated peripheral circuits.

To better understand the efficacy of the proposed method, we analyze the layer-by-layer improvement in utilization for NiN in Figure 52(a). Two configurations of tile structures are obtained; (1) $c_k = 2$, $p_k = 1$ and (2) $c_k = 3$, $p_k = 2$. We note that, even with a high degree of freedom (15 possible combinations) for the optimization, only two configurations are chosen across the range of DNNs evaluated in this work.

5.4.3 Energy-Aware NoC Optimization

The proposed methodology includes an energy-aware tile-to-router mapping and scheduling technique for the NoC. The upper bound on the number of routers is set as three times the number of DNN layers to balance energy and performance. Figure 52(b) shows the improvement in latency for each layer of NiN due to the proposed NoC optimization. The proposed NoC mapping reduces the communication

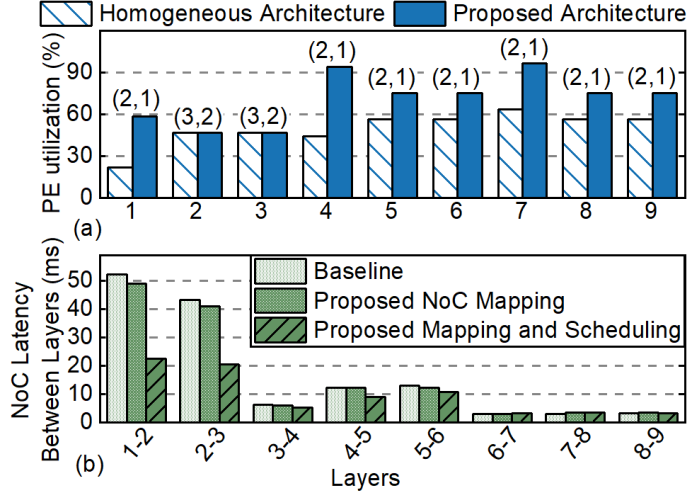


Figure 52. Layer-wise Improvement for NiN in (a) PE Utilization for Each Layer with SRAM-Based Heterogeneous Tile Architecture. The Tile Structure for Each Layer (c_k, p_k) is Shown on Top of Each Bar and (b) Communication Latency for Each Layer with Proposed NoC Optimization.

latency between layers 1 and 2 from 51ms to 47ms. As we integrate the NoC mapping with the scheduling technique, latency reduces further to 22ms. The first three layers of NiN contain more than 50% of the total number of activations. Therefore, the proposed NoC mapping reserves more routers for the first three layers, resulting in a significant reduction in latency for those layers. Additionally, the total number of routers is reduced which reduces the NoC area. A direct consequence of both latency and area reduction is lower communication energy, as shown in Figure 53(a) with an average reduction of 74%. The energy reduction is the highest for the case of VGG networks – 97%/98% for VGG-16/VGG-19. For ResNet-152, energy reduction is the lowest (15%), since the tiles are well distributed across layers for the baseline architecture, leaving less room for improvement.

5.4.4 Overall Improvement

We compare the energy-area product of the generated architecture against the baseline to assess the overall improvement. The proposed approach achieves up to 78% improvement in energy-area product, as shown in Figure 53(b). This improvement is a direct consequence of the heterogeneous tile architecture (with different sizes of tiles) and optimized NoC. The improvement for ResNet-50 is 10% since the initial mapping result is already area- and energy-efficient.

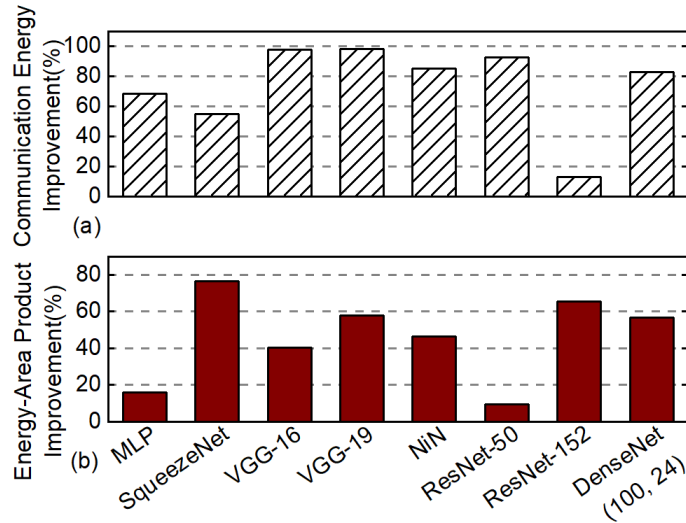


Figure 53. Improvement in (a) Communication Energy of the Proposed Energy-Aware NoC Optimization with Respect to the Baseline (SRAM) and (b) Energy-Area Product of the Generated SRAM-based Architecture with Respect to the Baseline (SRAM).

5.4.5 Comparison with State-of-the-Art Architectures

Table 17 compares the ReRAM-based architecture generated by the proposed methodology with state-of-the-art works using the VGG-19 network as a representative

example. The architectures have the following assumptions: crossbar of size 128×128 , 2 bits/cell, 32nm technology node, NoC flit width of 32 bits, and 16-bit precision for activations and weights. The generated ReRAM-based architecture achieves $2.56 \times$ improvement in frames per second (FPS) and $7.6 \times$ improvement in the energy-delay-area product (EDAP) than those of AtomLayer Qiao *et al.* (2018). It consumes $40 \times$ lower power per frame along with $452 \times$ improvement in EDAP than PipeLayer Song *et al.* (2017) for comparable throughput. Moreover, there is a $3 \times$ improvement in inference latency compared to ISAAC Shafiee *et al.* (2016). The gain in performance is attributed to the high utilization of the crossbar arrays and the efficient tile-to-router mapping and scheduling for the NoC. Overall, the proposed area and energy optimization methodology generates a heterogeneous IMC architecture (ReRAM) with an optimized NoC that has lower EDAP and power-per-frame than prior works for better or comparable throughput.

Table 17. Inference Performance Results for VGG-19. *Reported in Qiao *et al.* (2018)

	Latency (ms)	Power/frame (W/frame)	FPS	EDAP ($\text{mm}^2 \cdot \text{ms} \cdot \text{J}$)
Proposed Approach-ReRAM	2.69	4.2	372	0.208
AtomLayer Qiao <i>et al.</i> (2018)	6.92	4.8	145	1.58
PipeLayer Song <i>et al.</i> (2017)	2.6*	168.6	385	94.17
ISAAC Shafiee <i>et al.</i> (2016)	8.0*	65.8	125	359.64

5.5 Discussion and Conclusion

This work presents an area and energy optimization methodology to generate a heterogeneous IMC architecture with optimized NoC for a given DNN. Unlike conventional DNN architectures that use homogeneous tiles, the architecture derived

using the proposed area-aware optimization technique results in a heterogeneous architecture, where the tiles are of different sizes. The high energy efficiency is achieved through the proposed energy-aware optimization of the NoC architecture along with an associated scheduling technique. We show the efficacy of our proposed methodology for a wide range of DNN models from MLP to DenseNet, and depths up to 152 layers (ResNet-152). We observe that the proposed methodology has an execution overhead of 12 seconds for small DNNs to 150 seconds for large DNNs. This shows that the proposed methodology is scalable with the size and depth of DNNs. The proposed methodology also supports emerging technologies (such as SRAM, ReRAM, PCM, etc.) and applies to any mapping of weights on the crossbar. An evaluation of the architecture generated by the proposed methodology shows that our architecture achieves up to $7\times$ improvement in the energy-delay-area product compared to state-of-the-art DNN accelerators. For future work, we will build upon the optimization techniques proposed in this work to develop a run-time reconfigurable architecture that utilizes the reuse of IMC crossbar arrays.

SIAM: CHIPLET-BASED SCALABLE IN-MEMORY ACCELERATION WITH
MESH FOR DEEP NEURAL NETWORKS

State-of-the-art deep neural networks (DNNs) have become more complex with wider, deeper, and more branched structures to cater to the needs of various applications Huang *et al.* (2017); Howard *et al.* (2019). For instance, network architecture search (NAS) methods generate highly branched and complex DNNs, which increase compute and memory requirements Xie *et al.* (2019); Zoph and Le (2016). In-memory computing (IMC)-based architectures can support these network models because of their ability to embed deep learning operations in the memory array, achieving massively parallel computing with high storage density. Prior studies demonstrated that crossbar-based IMC architectures with RRAM or SRAM significantly improved throughput and energy-efficiency for DNN accelerators Krishnan *et al.* (2020b); Mandal *et al.* (2020); Shafiee *et al.* (2016); Imani *et al.* (2019); Song *et al.* (2017); Krishnan *et al.* (2021d,c,b); Mandal *et al.* (2020). Such architectures usually assume all DNN weights are stored on a monolithic chip to minimize DRAM access and maximize parallel IMC computing. However, as the DNN model size becomes larger and larger, this approach leads to increased chip area and on-chip memory.

Figure 54(a) shows the total chip area for a monolithic RRAM-based IMC architecture across different DNNs Krishnan *et al.* (2020b). Larger and branched DNNs like DenseNet-110 Huang *et al.* (2017) result in a chip area of up to 1,200mm². The increased area is attributed to the larger model size and the branched structure in DNNs. For example, ResNet-50 has 23M parameters and residual connections

(branched connections). For an 8-bit precision, the mapping scheme in Shafiee *et al.* (2016), and a crossbar size of 128x128, results in 802 tiles. Here each tile consists of 16 IMC crossbar arrays. In addition, the presence of the residual connections results in increased buffer cost due to the need to store the activations of previous layers to perform residual addition operations in the ResNet DNN. For the same hardware configuration, LeNet-5 with 0.43M parameters requires 43 tiles, while DenseNet-110 with 28.1M parameters requires 2184 tiles for the same hardware configuration. Hence, the DNN size and structure influence the overall area and, in turn, fabrication cost.

Furthermore, higher chip area further causes lower yield and higher defects across the wafer Kannan *et al.* (2015), resulting in wasted area and a higher fabrication cost. Figure 54(a) also shows the fabrication cost of the monolithic RRAM-based IMC architecture for different DNNs. We see that the fabrication cost increases exponentially with increased chip area (note that the cost is shown in the logarithmic scale), thus making the monolithic IMC architecture much less cost-efficient if all weights are stored on a single chip. Hence, there is an urgent need to address the increased fabrication cost of IMC-based DNN accelerators Krishnan *et al.* (2020b); Song *et al.* (2017); Shafiee *et al.* (2016).

2.5D integration or chiplet-based architectures Shao *et al.* (2019); Beck *et al.* (2018); Zimmer *et al.* (2019); Lin *et al.* (2020) provide a promising alternative to monolithic hardware architectures. They integrate multiple chiplets through silicon interposers or organic substrates Turner *et al.* (2018); Greenhill *et al.* (2017). Compared to the monolithic chip, the smaller chiplet size helps improve the design effort, yield, reduces defect ratio, and reduces fabrication cost. Figure 54(b) shows a representative 3-dimensional diagram of a chiplet-based IMC architecture. Chiplets comprise of memory units, computation blocks, and DRAM allowing for large-scale system integration.

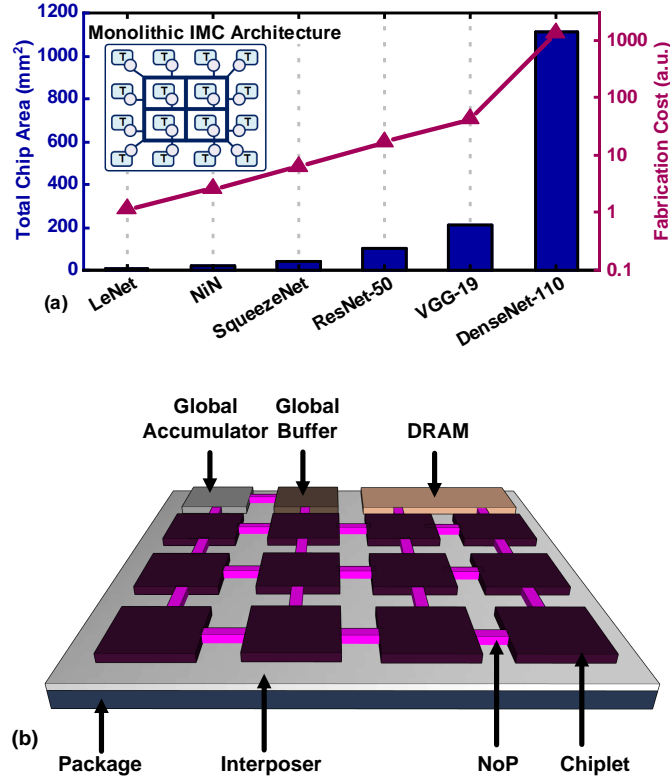


Figure 54. (a) Total Chip Area and Fabrication Cost for a Monolithic RRAM-based IMC Architecture for Different DNNs Shafiee *et al.* (2016). Fabrication Cost Increases Exponentially with an Increase in Total Chip Area (Appendix A Details the Method to Calculate the Fabrication Cost). (b) 3-Dimensional Diagram Showing the Chiplet-based IMC Architecture. The Architecture Includes an Array of IMC Chiplets, Global Buffer, Global Accumulator, and DRAM Connected by an NoP. The Figure is for Representational Purposes and Not Drawn to Scale.

The interposer acts as an additional routing layer (network-on-package or NoP) that utilizes package-level signaling to connect different chiplets. Recent advances in package-level signaling have enabled a $2\times$ improvement in bandwidth over board-level interconnections with $8\times$ lower energy-per-bit Turner *et al.* (2018); Lin *et al.* (2020). In a chiplet-based architecture, the design space parameters primarily include IMC crossbar size, the number of crossbars per chiplet, the number of chiplets, network-on-chip (NoC), NoP, and the DNN structure. These parameters need to be optimized

systematically in order to exploit the potential provided by this new architecture. For example, SIMBA Shao *et al.* (2019), a chiplet-based accelerator developed by Nvidia, utilizes 36 chiplets, with each chiplet consisting of 16 processing elements (PEs) Shao *et al.* (2019). While the underlying architecture ensures correct general-purpose functionality, it does not guarantee optimal performance for various DNN applications. Therefore, an extensive design space exploration is required to identify the optimal chiplet-based IMC architecture for DNN inference.

In this work⁴, we propose a novel chiplet-based IMC architecture simulator, SIAM, that integrates device, circuits, architecture, NoC, NoP, and DRAM access estimation under a single roof for design space exploration. We plan to open-source SIAM upon acceptance of this work. To the best of our knowledge, SIAM will be the first open-sourced chiplet-based IMC architecture simulator to promote architectural research in this emerging domain. SIAM includes four main components: partition and mapping engine, circuit and NoC engine, NoP engine, and DRAM engine. A Python wrapper is used to interface each engine with one another. The wrapper also interfaces SIAM with popular deep learning frameworks such as TensorFlow and PyTorch.

SIAM provides a scalable solution that utilizes model-based and cycle-accurate simulation components, allowing for performance evaluation of a wide range of DNNs across multiple datasets. It has a flexible architecture to support multiple DNN to IMC chiplet and crossbar partition and mapping schemes, thus generating different types of chiplet-based IMC architectures. Thus, SIAM provides a platform to enable comparisons across different chiplet-based IMC architectures and also between chiplet-based and monolithic IMC architectures. Furthermore, SIAM has a low simulation time to support the fast design and benchmarking exploration. For example, ResNet-

⁴Work done in collaboration with Sumit K. Mandal (UW–Madison)

110 with 1.7M parameters takes 12 minutes, while VGG-16 with 138M parameters takes 4.26 hours for benchmarking.

We demonstrate SIAM’s capabilities by conducting experiments on state-of-the-art DNNs such as ResNet-110 He *et al.* (2016) for CIFAR-10, VGG-19 Simonyan and Zisserman (2014) for CIFAR-100, and ResNet-50 He *et al.* (2016) and VGG-16 Simonyan and Zisserman (2014) for ImageNet datasets. Furthermore, to evaluate SIAM at the system level, we calibrate SIAM against a published silicon result, SIMBA Shao *et al.* (2019), especially the scaling trend with the number of chiplets.

The major contributions of this work are three-fold:

- We propose a complete framework, SIAM, that combines IMC circuit, NoC, NoP, and DRAM performance evaluation under a single roof. *SIAM is the first simulator to provide support for hardware performance evaluation of chiplet-based IMC architectures.* We carefully model the components of architecture like the IMC circuit, NoC, network-on-package (NoP), and DRAM.
- We provide a high degree of freedom to the user through different mapping schemes and customizable architectural parameters for IMC circuit, NoC, NoP, and DRAM components. We demonstrate different architectural design space exploration experiments that can be performed using SIAM.
- Extensive experimental evaluation of the SIAM simulator for different DNNs across CIFAR-10, CIFAR-100, and ImageNet datasets. Furthermore, we perform detailed experiments to calibrate the simulator to a real-design, SIMBA Shao *et al.* (2019), making SIAM a reliable and accurate simulator. For ResNet-50 on ImageNet, the generated chiplet-based IMC architecture achieves 130× and 72× improvement in energy-efficiency compared to Nvidia V100 and T4 GPUs.

6.1 Chiplet-based IMC Architecture

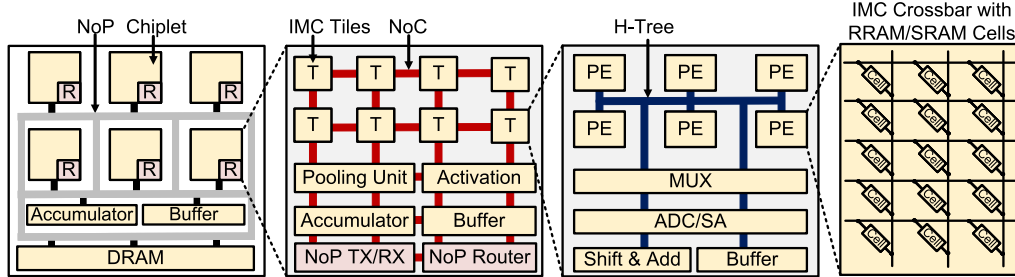


Figure 55. Chiplet-based IMC Architecture Utilized Within SIAM. The Architecture Consists of IMC Chiplets, Global Accumulator, Buffer, and DRAM Connected Using an NoP. SIAM Supports Both SRAM- and RRAM-based IMC Architectures. An NoP is Applied for Inter-Chiplet Communication and an NoC is Utilized Within the Chiplet for Intra-Chiplet Communication.

Overview: This section presents the underlying chiplet-based IMC architectures supported by SIAM, for *homogeneous (generic)* and *custom designs*. In a homogeneous architecture, the number of chiplets is fixed and is determined by the user. A custom architecture consists of the required number of chiplets to map the DNN under consideration. In both cases, the chiplet structure has a fixed number of IMC crossbar arrays inside (user-defined).

Figure 55 shows a homogeneous chiplet-based IMC architecture utilized by SIAM. The entire architecture consists of an array of chiplets that include IMC compute units, a global accumulator, a global buffer, and a DRAM chiplet. The chiplets are connected using an NoP fabric. The global accumulator and buffers are used to perform the accumulation across chiplets, and the DRAM chiplet is used to store the pre-trained DNN weights. In this work, we assume that all weights are transferred to the IMC chiplets from the DRAM chiplet before performing the DNN inference,

consistent with prior works Krishnan *et al.* (2020b); Shafiee *et al.* (2016); Imani *et al.* (2019).

Intra-Chiplet IMC Architecture: Each IMC chiplet consists of an array of IMC tiles connected using an NoC. The IMC tiles consist of an array of processing elements (PEs) or crossbar arrays. SIAM currently supports both RRAM- and SRAM-based IMC crossbar architectures. The IMC crossbars utilize analog computation to perform the multiply-and-accumulate (MAC) operation. Each IMC crossbar has associated peripheral circuitry (e.g. column multiplexers, analog-to-digital converter (ADC), shift and add circuits, etc.). A column multiplexer is used to share a flash ADC or sense amplifier (SA) across multiple columns of the IMC crossbar. The ADC converts the analog output from the IMC crossbar to the digital domain. Next, the ADC output is accumulated based on the bit significance using shifter and adder circuits to extract the computed MAC output. Finally, the overall result is generated by accumulating the outputs from each IMC crossbar across the entire input. Note that our architecture does not use a digital-to-analog converter (DAC), and it instead employs sequential bit-serial computing for multi-bit inputs. Furthermore, each chiplet consists of a pooling and activation unit. The pooling unit supports both max and average pooling operations, while the activation unit supports ReLU and sigmoid functions.

Interconnect: The IMC chiplets are connected at the tile-level (within chiplet) using an NoC. A point-to-point (P2P) interconnect, such as H-Tree, is used for communication at the PE-level. Each tile within the chiplet has a five-port router that performs the data scheduling and X–Y routing through the NoC. The NoC can be configured for multiple flit width and operating frequencies by the user. The array of chiplets are interconnected using an NoP that utilizes the interposer for routing.

A passive interposer is implemented within SIAM where the interposer does not contain any active elements like repeaters. Each chiplet consists of a dedicated NoP transmitter and receiver (TX/RX) circuit and an NoP router. The router performs the packet scheduling and utilizes a dedicated port to transmit data to the TX/RX circuit. The custom TX/RX circuit can be configured for a given signaling technique to achieve data transfer across the NoP Turner *et al.* (2018); Lin *et al.* (2020). The architecture also includes a clocking circuit (e.g.: LC-PLL Poulton *et al.* (2013)) for the TX/RX circuit. The NoP interconnect properties such as wire resistance, capacitance, and inductance are carefully modeled by utilizing the PTM models Sinha *et al.* (2012) following prior works Turner *et al.* (2018); Lin *et al.* (2020). Section 6.2.4 details the modeling of both the NoP interconnect and the driver.

6.2 SIAM Simulator

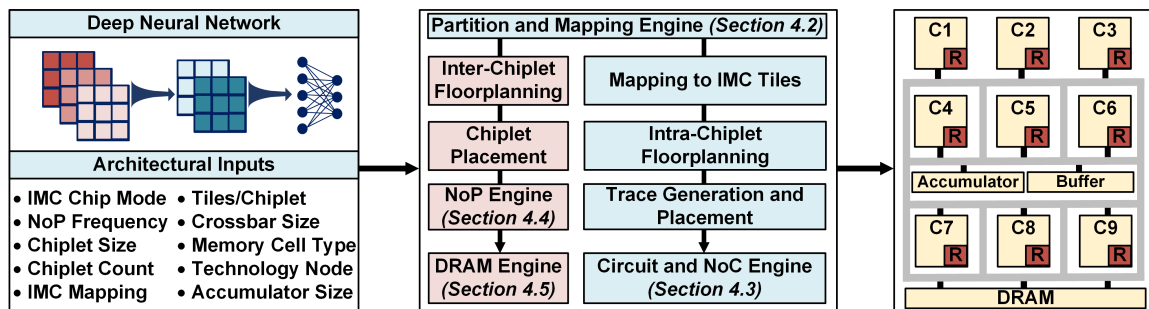


Figure 56. An Overview of the Proposed Chiplet-based IMC Benchmarking Simulator, SIAM. SIAM Incorporates Device, Circuits, Architecture, NoC Jiang *et al.* (2013), NoP, and DRAM Access Model Kim *et al.* (2015) Under a Single Roof for System-Level Analysis of Chiplet-based IMC Architectures.

6.2.1 Overview

SIAM provides a unified framework for performance benchmarking of chiplet-based IMC architectures, as shown in Figure 56. SIAM operates on user inputs to generate the chiplet-based IMC architecture and to benchmark the corresponding hardware performance. The hardware performance metrics include area, energy, latency, energy efficiency, power, leakage energy, and IMC utilization. The overall simulator is developed using Python and C++ programming languages. A top-level Python wrapper is built to combine the different components within the simulator. Furthermore, SIAM interfaces with popular deep learning frameworks such as PyTorch and Tensorflow. Thus, SIAM supports multiple network structures in current literature (as shown in Section 6.4) and can be used for exploring NAS techniques. Table 18 shows the user inputs and associated descriptions of the SIAM benchmarking tool.

Table 18. Definition Of the User Inputs to SIAM

User Input	Description	User Input	Description
DNN Algorithm		Device and Technology	
Network Structure	DNN network structure information	Tech Node	Technology node for fabrication
Data Precision	Weights and activation precision	Memory Cell	RRAM or SRAM
Sparsity	DNN layer-wise sparsity	Bits/Cell	Number of levels in RRAM
Intra-Chiplet Architecture		Inter-Chiplet Architecture	
Crossbar Size	IMC crossbar array size	Chip Mode	Monolithic or chiplet-based IMC architecture
Buffer Type	SRAM or Register File	Chiplet Structure	Homogeneous or custom chiplet structure
ADC Resolution	Bit-precision of flash ADC	Chiplet Size	Number of IMC tiles within each chiplet
Read-out Method	Sequential or Parallel	Total Chiplet Count	Fixed count or DNN specific custom count
NoC Topology	Mesh or Tree	Global Accumulator Size	Size of global accumulator
NoC Width	Number of channels in the NoC	NoP Frequency	Frequency of the NoP driver and interconnect
Frequency	Frequency of operation	NoP Channel Width	Number of parallel links for TX and RX

SIAM consists of four engines:

- Partition and mapping engine (Python)
- Circuit and NoC engine (C++)
- NoP engine (Python and C++)

- DRAM engine (Python and C++)

Each engine functions independently on a subset of the user inputs, while communicating with each other using the top-level Python wrapper. To further understand the framework, we detail the simulation flow used for SIAM in Figure 56. *First*, SIAM takes the user inputs and performs the layer partition and mapping onto the chiplets and IMC crossbars using the partition and mapping engine. The outputs include the structure of IMC architecture, the number of chiplets and IMC tiles required per layer, utilization of the IMC architecture, intra-chiplet and inter-chiplet data movement volume, and the number of global accumulator accesses. Next, the intra-chiplet and global circuit performance are evaluated using the circuit and NoC engine. The engine provides the hardware performance metrics such as area, energy, and latency for the intra-chiplet and global circuit operations across all chiplets. Simultaneously, the NoP engine evaluates the cost of the interconnect, router, and driver associated with the chiplet-to-chiplet data movement. Finally, the DRAM engine determines the cost of the memory accesses and provides the energy and latency performance metrics. All engines except the partition and mapping engine work simultaneously, thus reducing the total simulation time. We note that SIAM also supports benchmarking of conventional monolithic IMC architectures. In the following sections, we detail the four engines that represent the core functionality within SIAM.

6.2.2 Partition and Mapping Engine

Algorithm 5 describes the step-by-step operation of the partition and mapping engine. The engine performs the partition of DNN layers to the IMC chiplets and the corresponding mapping to the IMC crossbar arrays. The partition and mapping is

Algorithm 5: Partition and Mapping of DNN layers

```

1 Input: DNN structure, chiplet count ( $C$ ), chiplet size ( $S$ ), the number of DNN layers ( $N_l$ )
2 Output: Layer-wise chiplet partition ( $\mathcal{L} \rightarrow \mathcal{P}$ ) and layer to chiplet mapping ( $\mathcal{L} \rightarrow \mathcal{C}$ )
3 for  $i = 1 : N_l$  do
4    $N^{Chiplet} = 0, N_i^{Chiplet} = 0, N_i^{Total} = 0$  /* Initialize variables */
5   /* Layer-wise Mapping ( $\mathcal{L} \rightarrow \mathcal{C}$ ) */
6   Calculate number of rows of IMC crossbars ( $N_i^r$ ) to map layer  $i$  (Equation 6.1)
7   Calculate number of columns of IMC crossbars ( $N_i^c$ ) to map layer  $i$  (Equation 6.1)
8    $N_i^{Total} = N_i^r \times N_i^c$  /* Total number of IMC crossbars for layer  $i$  */
9   /* Layer-wise Partitioning ( $\mathcal{L} \rightarrow \mathcal{P}$ ) */
10   $N_i^{Chiplet} = \left\lceil \frac{N_i^{Total}}{S} \right\rceil$  /* Calculate the number of chiplets for layer  $i$  */
11   $N^{Chiplet+} = N_i^{Chiplet}$  /* Increment total chiplets in the architecture */
12  if Homogeneous Mapping then
13    if  $N^{Chiplet} > C$  then
14      exit() /* Error: Exceeded the maximum number of chiplets */
15    end
16  end
17  /* Partition and Mapping completed for layer  $i$  */
18 end

```

performed layer-wise for the entire DNN. The engine utilizes user inputs such as the DNN structure, DNN weight precision, IMC chiplet mapping scheme, size of the IMC chiplet, and the IMC crossbar size, among others.

We first discuss the IMC mapping scheme utilized in SIAM. For a given layer i , let the weight matrix be W_i represented by $Kx_i \times Ky_i \times Nif_i \times Nof_i$, where Kx and Ky represent the kernel size, Nif the number of input features, and Nof the number of output features. We adopt the following mapping scheme, similar to that in Krishnan *et al.* (2020b); Shafiee *et al.* (2016):

$$N_i^r = \left\lceil \frac{Kx_i \times Ky_i \times Nif_i}{(PE_x)} \right\rceil; \quad N_i^c = \left\lceil \frac{Nof_i \times N_{bits}}{(PE_y)} \right\rceil \quad (6.1)$$

In the above equation, N_i^r and N_i^c are the required number of rows and columns of IMC crossbars needed to map the layer i of the DNN. N_{bits} , PE_x and PE_y represent the DNN weight precision, the number of rows and columns in the IMC crossbar array,

respectively. The product of N_i^r and N_i^c is the total number of required IMC crossbar arrays N_i^{Total} to map layer i of the DNN (line 7 of Algorithm 5).

SIAM can generate (a) *homogeneous* and (b) *custom* chiplet-based IMC architectures using two types of chiplet partitions. Figure 57 shows the two generated architectures based on the homogeneous and custom chiplet partitioning. The partition and mapping engine assumes that DNN layers cannot be partitioned across multiple chiplets, and a single chiplet can support multiple layers to achieve high chiplet utilization (shown in Section 6.4). Since each layer of the DNN contains a large number of multi-bit weights, multiple chiplets consisting of IMC crossbar arrays are required to map the whole layer. If one layer is distributed across chiplets, it increases the overhead in terms of the control logic for routing the inputs to the respective chiplets, an increased volume of inter-chiplet data communication, and a higher chiplet-to-chiplet communication energy and latency. During the partition of layers across multiple chiplets, the engine divides the layer uniformly across the chiplets, thus avoiding the workload imbalance issue. Based on the total number of required IMC crossbar arrays, N_i^{Total} , the engine determines the number of chiplets necessary to map the layer i of the DNN as: $N_i^{Chiplet} = \left\lceil \frac{N_i^{Total}}{S} \right\rceil$, where S denotes the total number of IMC crossbar arrays within a chiplet (size of the chiplet).

Next, the total number of chiplets in the architecture ($N^{Chiplet}$) is determined (line 9 of Algorithm 5). In the *homogeneous chiplet partition* scheme, a fixed number of chiplets (user input) is used to map the DNN. Hence, the engine compares the total number of chiplets in the architecture ($N^{Chiplet}$) with the maximum available chiplets (C). If greater, then the engine throws an error and requests for an increase in the number of available chiplets in the architecture. If lesser, the engine continues the partition and mapping for the subsequent layers in the DNN.

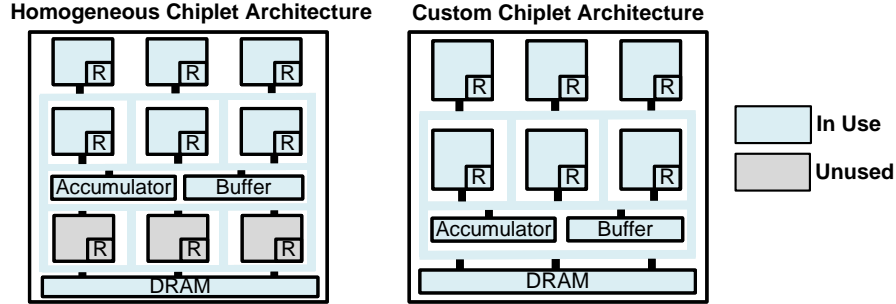


Figure 57. Representative Figure Showing the Two Generated Chiplet-based IMC Architectures for the Same DNN, Homogeneous (Left) and Custom (Right), from the Supported Partition Schemes in SIAM. Homogeneous Architecture is Generic, while Custom Architecture is DNN Specific. R Refers to the NoP Router.

In the *custom partition scheme*, the architecture consists of the required number of chiplets to map the DNN. Hence, there is no maximum limit in the number of available chiplets within the architecture. Such a design results in a fully-custom architecture specific to the DNN under consideration. Each chiplet has the same structure with a fixed number of IMC tiles, where each tile consists of IMC crossbar arrays and associated peripheral circuitry. Thus, SIAM provides a platform to perform comparison between homogeneous (generic) and custom-designed chiplet-based IMC architectures.

After partitioning and mapping layers onto the IMC chiplets, the engine determines the total volume of data communicated within the chiplet and across chiplets. Simultaneously, when a layer is partitioned across chiplets, the global accumulator is used to generate the overall layer output. The engine determines the number of additions performed by the global accumulator and the number of global buffer accesses. Overall, the engine provides the layer partition across chiplets, the number of required chiplets and IMC crossbars, IMC crossbar utilization, volume of intra- and inter-chiplet data movement, and the number of the global accumulator and buffer

accesses. The other engines (circuit, NoC and NoP) then utilize these outputs to evaluate the hardware performance of the chiplet-based IMC architecture.

6.2.3 Circuit and NoC Engine

After completing the partition and mapping of the DNN, SIAM performs the inter- and intra-chiplet floorplanning and placement, thus determining the entire chiplet-based IMC architecture. Thereafter, the circuit and NoC engine estimates the hardware performance. Figure 58 shows the block diagram of the circuit and NoC engine. The engine employs a model-based estimator for the circuit part, and a trace-based estimator for the interconnect part.

6.2.3.1 Circuit Estimator

The circuit estimator evaluates the overall hardware performance of each chiplet, global accumulator, and global buffer in the overall architecture. The inputs to the engine include the intra- and inter-chiplet placement, per layer chiplet and IMC crossbar count, layer-wise IMC utilization, technology node, frequency of operation, IMC cell type, the number of bits per cell, read-out mode (row-by-row or parallel), and ADC precision, among others. The intra-chiplet circuits include the IMC crossbar array and associated peripheral circuits, buffer, accumulator, activation unit, and the pooling unit. The peripheral circuits include the ADC, multiplexer circuit, shift and add circuit, and decoders. We calibrate the circuit estimator with NeuroSim Peng *et al.* (2019a).

The circuit estimator evaluates the performance of the entire chiplet-based IMC

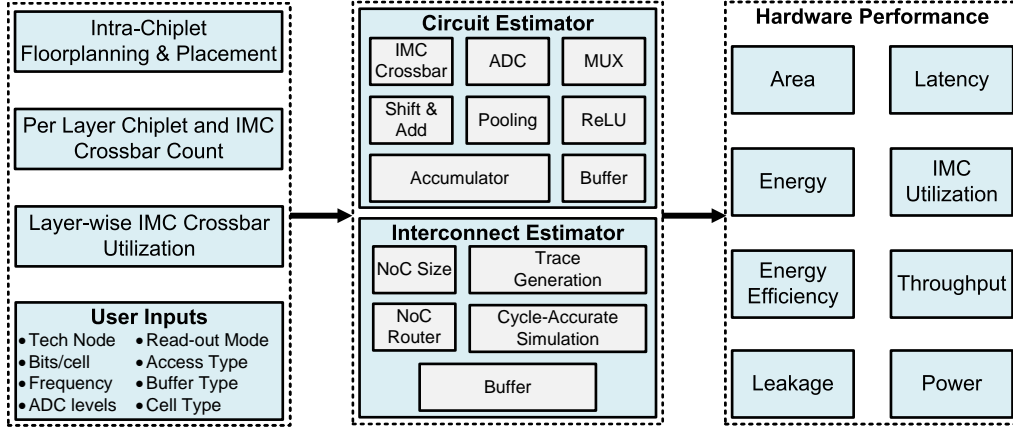


Figure 58. Block Diagram of the Circuit and NoC Engine Within SIAM. The Engine Utilizes a Separate Circuit and NoC Simulators that Perform the Overall Hardware Performance Estimation.

architecture in a layer-wise manner. Each chiplet performs the computations of a subset of layers in the DNN. For a given DNN layer i , the chiplet count per layer, the IMC crossbar count per layer, and the IMC utilization values are taken from the partition and mapping engine. Area, energy, and latency are estimated in a bottom-up manner, i.e., the estimation starts from the device level and moves up to the circuit level and, finally, the architecture level. Based on user inputs such as technology node, IMC cell type, IMC crossbar size, IMC utilization, ADC precision, and read-out mode, the estimator evaluates the cost of a single crossbar and associated peripheral circuits. The estimator repeats the process for all IMC crossbars within the chiplet for the given layer i in the DNN. Next, the estimator evaluates the buffer cost, shift and adder circuitry, and the accumulator within the chiplet. After that, the pooling and activation units are evaluated to obtain the total area, energy, and latency of the IMC chiplet. At the chiplet-level, the global accumulator and global buffer accumulate the partial sum of a layer across chiplets. The circuit estimator utilizes the number of additions performed, the data volume from each chiplet, and the accumulator size

Algorithm 6: NoC (or NoP) Trace Generation

```
1 Input: Number of tiles for each layer (for each chiplet in case of NoP) ( $|\mathcal{T}|$ ), Number of
   input activations for each layer ( $A$ ), Number of chiplets ( $\mathcal{C}$ ), Layer to chiplet mapping
   ( $\mathcal{L} \rightarrow \mathcal{C}$ ), Quantization bit ( $Q$ ), Bus width ( $W$ )
2 Output: Trace file for each chiplet ( $tr^c$ )
3 for  $c = 1 : |\mathcal{C}|$  do
4   Find index of the first layer ( $L_c^S$ ) and the last layer ( $L_c^E$ ) in the chiplet from  $\mathcal{L} \rightarrow \mathcal{C}$ 
5   for  $l = L_c^S : L_c^E$  do
6      $k = 0$  /* Initialize timestamp */
7     Find index of first source tile ( $T_l^S$ ) and last source tile ( $T_l^E$ )
8     Find index of first destination tile ( $T_{l+}^S$ ) and last destination tile ( $T_{l+}^E$ )
9      $N_p = \lceil \frac{A(l)Q}{W} \rceil$  /* Number of packets */
10    for  $n = 1 : N_p$  do
11      for  $s = T_l^S : T_l^E$  do
12        for  $d = T_{l+}^S : T_{l+}^E$  do
13           $tr^c \leftarrow [tr^c; [s, d, k]]$ 
14           $k \leftarrow k + 1$  /* Increment timestamp */
15        end
16       $k \leftarrow k + 1$  /* Increment timestamp */
17    end
18  end
19 end
20 end
```

(user input) to determine the area, energy, and latency of the global accumulator and buffer. Finally, based on the number of chiplets required for layer i of the DNN, the circuit estimator repeats the estimation for all chiplets to determine the overall hardware performance.

6.2.3.2 NoC Estimator

Communication plays a crucial role in the hardware performance of DNN accelerators Mandal *et al.* (2020). A detailed description of communication-centric DNN accelerators can be found in Nabavinejad *et al.* (2020). Each layer within the DNN sends a significant amount of data to other layers. Authors in Krishnan *et al.* (2020b) show that communication alone incurs up to 90% of the total inference latency for

DNNs. Therefore, designing an efficient communication protocol for DNNs is of supreme importance. Hence, we carefully incorporate the cost of communication between multiple layers within a chiplet. We consider an NoC for intra-chiplet communication since NoC is the standard interconnect fabric used in the SoC-domain Jeffers *et al.* (2016). We customize a cycle-accurate NoC simulator, BookSim Jiang *et al.* (2013), to evaluate the NoC performance. First, a trace file is generated for each chiplet following Algorithm 6. The algorithm takes the number of tiles for each layer, the number of input activations for each layer, number of chiplets, layer to chiplet mapping, quantization bit-precision, and bus width. From these inputs, we find the indices of the first and the last layer of each chiplet. Next, for each layer in each chiplet, we find the source and destination tile information as shown in lines 7–8 of Algorithm 6. The number of packets for each source-destination pair is then calculated. After that we iterate over the number of packets, the number of source tiles, and the number of destination tiles to obtain a trace in the form of a tuple consisting of the source tile ID, destination tile ID, and the timestamp. The timestamp variable is reset to zero after generating trace for each pair of layers. Then, the trace file is simulated using BookSim to obtain the area, energy, and latency for on-chip communication within each chiplet.

6.2.4 NoP Engine

The NoP connects different chiplets through a silicon interposer or organic substrate. It performs the on-chip data movement using special signaling techniques and driver circuits, as shown in Poulton *et al.* (2013); Lin *et al.* (2020). Figure 59 (left) shows the cross-sectional image of a 2.5D integration with chiplets and an interposer. Modeling

the NoP performance has many challenges due to the complex interconnect structure, specialized driver architectures, and the corresponding signaling techniques.

To this end, our NoP engine models each component in the NoP for accurate performance estimation. Figure 59 (right) shows various NoP implementations with the corresponding energy-per-bit (E_{bit}) proposed in prior works. There are two main components of the NoP performance evaluation: 1) NoP latency estimation and 2) NoP area and power estimation.

NoP latency estimation: The engine utilizes a cycle-accurate simulator to perform the interconnect evaluation. First, based on the chiplet-to-chiplet data volume generated by the partition and mapping engine, the NoP engine utilizes Algorithm 6 (same as the algorithm for NoC) to generate the trace for the NoP. These traces are simulated using a cycle-accurate simulator or the NoP estimator (a customized version of BookSim to incorporate a trace-based simulation) to obtain the latency of the NoP interconnect.

NoP area and power estimation: To estimate the area and power consumption of the NoP, we first obtain the interconnect parameters for the NoP, which include wire length, pitch, width, and stack-up. We use these parameters to determine the interconnect capacitance and resistance using the PTM interconnect models Sinha *et al.* (2012). Next, based on the capacitance and resistance, the timing parameters for the interconnect are generated and compared with the target bandwidth. If the timing parameters do not satisfy the bandwidth, the NoP engine chooses the maximum allowable bandwidth.

Next, the engine evaluates the NoP transmitter/receiver (TX/RX) circuits, including the clocking circuitry. The engine utilizes E_{bit} , number of TX/RX channels, bandwidth, chiplet-to-chiplet data volume, and operation frequency to generate the

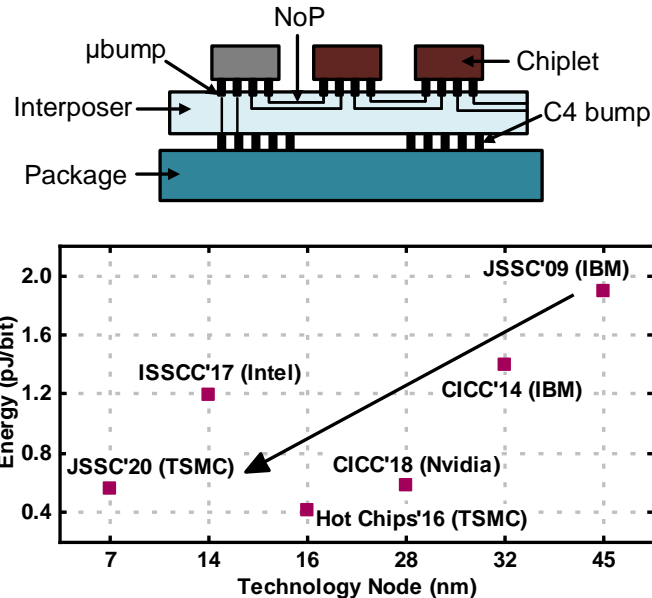


Figure 59. (Top) Cross-Sectional Image of the NoP Interconnect. The NoP is Routed Within the Interposer Connecting Different Chiplets Across the Architecture. μ bumps Connect the Chiplets to the Interposer, (Bottom) Energy Per Bit for Different NoP Driver Circuit and Signaling Techniques Proposed in Prior Works.

Algorithm 7: Computation of NoP driver energy

- 1 **Input:** DNN structure, Chiplet count (C), Number of activations per layer (A), NoP bus width (W), Quantization bit (Q_{bit}), Energy per bit (E_{bit})
 - 2 **Output:** Energy for NoP driver (E_D)
 - 3 **Initialize:** $E_D \leftarrow 0$
 - 4 **for** $c = 1 : |C|$ **do**
 - 5 Find index of source layer (l)
 - 6 Find index of destination layer ($l + 1$)
 - 7 /* Number of packets between two consecutive chiplets */
 - 8 $N_p = \lceil \frac{A^{(l)}Q}{W} \rceil$
 - 9 $N_{bits} = N_p \times Q_{bit}$
 - 10 $E_D = E_D + N_{bits} \times E_{bit}$
 - 10 **end**
-

energy and latency cost of the TX/RX circuits. Algorithm 7 details the energy calculation for the NoP driver. We compute the total number of bits between chiplets. Furthermore, we obtain the energy per bit (E_{bit}) from prior works, as shown in Fig-

ure 59(right). We multiply the number of bits and energy per bit to obtain the total energy for TX/RX channel, as shown in line 9 of Algorithm 7. Next, the TX/RX circuit area from prior implementations (Figure 59) is utilized to obtain the NoP driver area cost. Finally, the NoP engine combines the performance metrics for the interconnect and the driver to generate the overall NoP performance. We summarize the functional flow of the NoP engine:

- NoP trace generation based on the inter-chiplet layer partition, chiplet placement, and inter-chiplet data transfer volume.
- NoP interconnect evaluation using a cycle-accurate simulator to generate area, energy, and latency metrics.
- NoP TX/RX driver and router modeling based on real measurements. Finally, the NoP engine combines the interconnect and NoP driver metrics to generate the overall NoP performance.

6.2.5 DRAM Engine

The chiplet-based IMC architecture consists of a DRAM chiplet that acts as the external memory for the IMC chiplets. The DRAM engine performs the external memory access estimation for the chiplet-based IMC architecture. In this work, we assume that the DRAM only transfers the entire set of weights to the chiplet one time before the inference task is performed. Hence, it remains constant for a given DNN across different architectural configurations and inference runs.

The engine consists of a DRAM request generator, RAMULATOR Kim *et al.* (2015) for estimating the latency for the DRAM transactions, and VAMPIRE Ghose

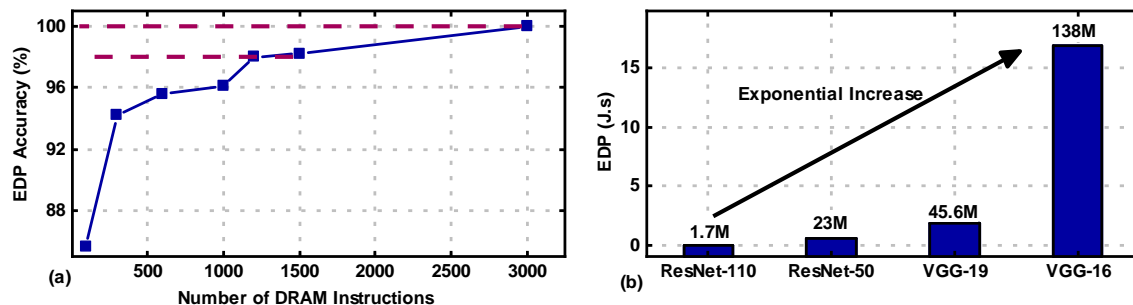


Figure 60. (a) The Accuracy of EDP Prediction for Different Numbers of Instructions Processed to Represent 3,000 DRAM Instructions. Reduction in the Number of Instructions to Half Results in Less Than 2% EDP Accuracy Degradation for Half the Simulation Time, and (b) EDP of DRAM Transactions (DDR4) for Different DNNs. There is an Exponential Increase in DRAM Cost with an Increase in DNN Model Size.

et al. (2018) to estimate the DRAM transaction power. First, the choice of DRAM is determined based on the user input. Currently, SIAM supports both DDR3 and DDR4. For DDR3 and DDR4, we incorporate the DRAM models detailed in MICRON (2011, 2014). Next, for a given DNN model, the model size and data precision are determined from the user inputs. Furthermore, the DRAM engine generates the required traces and memory requests with time stamps. The requests include the location within the DRAM memory and the operation.

SIAM utilizes a customized version of the cycle-accurate simulator RAMULATOR Kim *et al.* (2015) and the model-based power analysis tool VAMPIRE Ghose *et al.* (2018). The customization includes the addition of support for larger DNNs, different data precision, and the addition of custom DDR3/DDR4 models. Furthermore, to reduce the simulation time for large DNNs such as VGG-16 (138M parameters), the DRAM engine breaks down the total number of instructions into smaller sets. The engine then performs the estimation for one set of instructions and multiplies it by the total number of sets required to represent all the weights in the DNN. To calibrate the method, we perform an experiment for 3,000 instructions broken down into a number

of smaller sets of instructions. Figure 60(a) shows the corresponding energy-delay-product (EDP) accuracy for different sizes of instruction sets. A reduction in 50% of DRAM instructions to the engine results in less than 2% EDP accuracy degradation than that at 100% instructions. Furthermore, the reduced number of instructions allows for reduced simulation time for the DRAM engine. We establish that, through this method, the DRAM engine performs fast and accurate estimation of external memory access for the entire range of DNNs. Figure 60(b) shows the overall EDP for different networks across different datasets for DDR4. There is an exponential increase in EDP with the increase in the model size of the DNN.

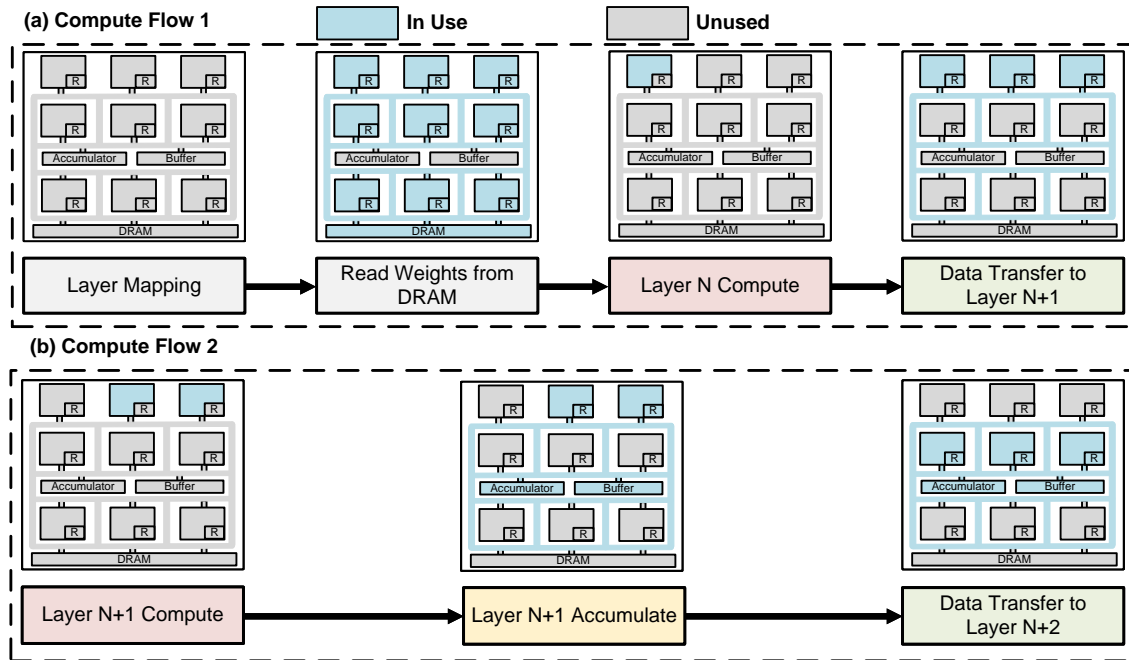


Figure 61. Computation Dataflow Within the Chiplet-based IMC Architecture in SIAM. Two Cases Arise, (a) No Layer is Partitioned Across Two or More Chiplets, and (b) a Layer is Partitioned Across Two or More Chiplets.

To summarize, the following are the key steps in the execution of the DRAM engine:

- Generate DRAM requests based on the DNN model size and data precision.
- Calculate DRAM transaction latency cost using a customized version of RAMULATOR, and calculate the power consumption using a customized version of VAMPIRE.
- Combine the outputs to generate the overall DRAM access cost.

6.3 SIAM Dataflow

This section presents the default dataflow in the generated chiplet-based IMC architecture. Figure 61 shows an example of the computation dataflow within the SIAM architecture. Before performing the inference task, the weights are retrieved from the DRAM and mapped to the IMC chiplets based on the output from the partition and mapping engine (detailed in Section 6.2.2), as shown in Figure 61(a).

Algorithm 8: Dataflow for SIAM IMC Chiplet Architecture

```

1 Input: Weights, input features, and total number of layers of the DNN
2 Output: Execution flow of the DNN on SIAM chiplet-based IMC architecture
3 while  $i < Total\ number\ of\ layers$  do
4     /* Partitioning and mapping of the DNN */
5     Calculate number of chiplets required for  $i^{th}$  layer
6     Perform computation for  $i^{th}$  layer
7     /* Check if layer is distributed across chiplets */
8     if  $Number\ of\ chiplets > 1$  then
9         Data transfer to accumulator
10        Partial sum accumulation for  $i^{th}$  layer
11    end
12    Data transfer to chiplets of  $(i + 1)^{th}$  layer
13     $i \leftarrow i + 1$ 
14 end

```

Two cases can arise during the partitioning: first, no layer is distributed across two or more chiplets; second, a layer is distributed across two or more chiplets. The two cases result in two different scenarios within the execution dataflow. Consider

that layer N of the DNN is mapped onto the first chiplet in the architecture, as shown in Figure 61(a). During the computation, the entire layer is consumed within one chiplet, producing the computed output activations from layer N . Both the global accumulator and buffer are not utilized in the process and are turned off. After the computation, the output activations are transferred to the chiplets that implement layer $N+1$. For layer $N+1$, let's assume that two chiplets are required to map the weights. Hence, the NoP transfers the output activation from layer N to both chiplets housing layer $N+1$, as shown in Figure 61(a). Figure 61(b) shows the computation flow for layer $N+1$. Both chiplets perform the computation in a parallel manner. The mapping ensures that the same number of weights are mapped to each chiplet, thus avoiding the workload imbalance issue. After completion of the computation, the generated partial sums are accumulated using the global accumulator and buffer. Then, the accumulated outputs from layer $N+1$ are transferred to the chiplets housing weights of layer $N+2$. The process is repeated until all the layers are completed and the final output is obtained. Algorithm 8 details the algorithmic implementation of the dataflow utilized in the SIAM IMC chiplet architecture.

6.4 Experimental Evaluation

We perform a wide range of experiments to demonstrate the effectiveness of the proposed SIAM simulator. These include detailed analysis of homogeneous and custom IMC chiplet architectures, comparison between monolithic and chiplet IMC architectures, calibration with real silicon data from SIMBA Shao *et al.* (2019), comparison of the performance with GPUs, and evaluation of the SIAM's simulation

time. We also illustrate the three characteristics of SIAM, flexibility, scalability, and simulation speed as shown below:

- *Flexibility and scalability:* Supporting different DNNs across datasets, two types of DNN partition to the IMC chiplets, and support for different IMC tile and chiplet configurations (Section 6.4.2, Section 6.4.3).
- *Simulation speed:* Fast design space exploration of SIAM is demonstrated for different DNNs (Section 6.4.6).

6.4.1 Experimental Setup

The DNNs that we evaluated include ResNet-110 (1.7M) on CIFAR-10, VGG-19 (45.6M) on CIFAR-100, ResNet-50 (23M) on ImageNet, and VGG-16 (138M) on ImageNet. We use 8-bit quantization for the weights and activations and a 32nm CMOS technology node for the hardware. We perform the experiments on an Intel Xeon CPU platform. The mapping of DNNs onto the IMC crossbars follows prior works Krishnan *et al.* (2020b); Shafiee *et al.* (2016). Unless specified otherwise, all the experiments are performed based on the assumptions detailed next. The chiplets are placed to achieve the least Manhattan distance. All results shown are for RRAM-based IMC architectures with the following parameters: one bit per RRAM cell, a R_{off}/R_{on} ratio of 100, 16 tiles per chiplet, IMC crossbar size of 128×128 , ADC resolution of 4-bits with 8 columns multiplexed, operating frequency of 1GHz Imani *et al.* (2019); Shafiee *et al.* (2016), and a parallel read-out method. We note that, the experiments do not consider the non-ideal effects within the RRAM-based IMC architecture. The NoP parameters include a bandwidth of 250MHz, E_{bit} of 0.54pJ/bit Poulton *et al.* (2013), interconnect parameters such as width, thickness, and pitch from Poulton *et al.*

(2013), NoP TX/RX area of $5,304 \mu\text{m}^2$ Poulton *et al.* (2013), NoP clocking circuit area of $10,609 \mu\text{m}^2$ Poulton *et al.* (2013), and 32 channels (channel width). We note that SIAM can support any NoP performance estimation as long as the NoP wiring parameters, bandwidth, channel width, TX/RX circuit area, clocking circuit area, and E_{bit} are provided by the user. Finally, the reported results do not include the RRAM write, and DRAM read energy and latency. Since we focus on DNN inference, the RRAM write and DRAM read operations are applied offline before performing the inference. They do not involve in the inference runs and remain constant across IMC chiplet architectural configurations.

6.4.2 Custom and Homogeneous Chiplet-based IMC Design

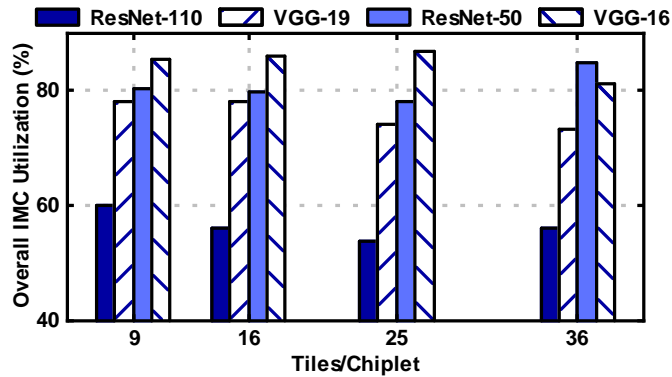


Figure 62. IMC Utilization for a Custom RRAM-based Chiplet IMC Architecture Across Different DNNs and Different Chiplet Configurations. The Mapping Strategy Adopted Within SIAM Ensures High Utilization Across all DNNs.

6.4.2.1 IMC Crossbar Utilization

Figure 62 shows the overall IMC crossbar utilization for a custom RRAM-based chiplet IMC architecture across different tiles per chiplet and DNNs. We see that SIAM consistently achieves high (>50%) IMC crossbar utilization. The high IMC utilization indicates that mapping within SIAM generates chiplet-based IMC architectures that are area-efficient. ResNet-110 has the lowest utilization due to the small network structure with fewer input and output features. At the same time, ResNet-50, VGG-19, and VGG-16 achieve >75% utilization across the entire chiplet-based IMC architecture. Hence, the partition and mapping engine within SIAM provides a flexible and efficient platform to generate chiplet-based IMC architectures with multiple configurations for design space exploration.

6.4.2.2 IMC Chiplet Performance Breakdown

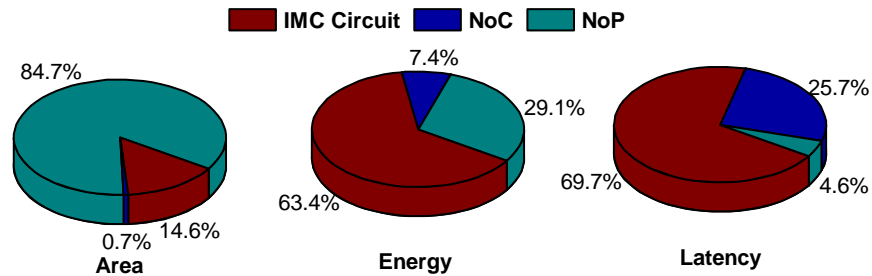


Figure 63. Breakdown of the Different Components Contributing to the Overall Area, Energy, and Latency Performance Metrics, for a Custom Design RRAM-based Chiplet IMC Architecture When Mapping ResNet-110 for CIFAR-10 Dataset.

We analyze the breakdown of different components for the area, energy, and latency metrics for the RRAM-based chiplet IMC architecture. Figure 63 shows the breakdown

for the implementation of ResNet-110 on CIFAR-10. We divide each metric into three main components, IMC circuit, NoC, and NoP. The IMC circuit component consists of the IMC crossbar array and associated peripherals, buffers (global and within chiplet), accumulators (global and within chiplet), pooling unit, and the activation unit. At the same time, the NoP component consists of the NoP interconnect, NoP router, and the NoP driver and clocking circuit. Finally, the NoC component deals with the intra-chiplet interconnect and the NoC routers.

We first analyze the area metric. The NoP dominates the overall area with 84.7%, while the NoC contributes the least to the area. The NoP drivers are designed such that differential signaling is utilized to avoid common-mode noise along with a clocking circuit for every N lanes. This results in increased circuitry for the TX-RX driver pairs and associated clocking circuitry for the 32 NoP channels. For example, Shao *et al.* (2019) utilizes one clocking lane per 4 data lanes. The NoP router area depends on the technology node of the chiplet and the number of ports (default ports is set to 5). Simultaneously, the NoP link area depends on the wire properties. The NoP wire width and length are designed to maintain signal integrity at the specified frequency of operation. The wire for NoP link requires shielding on both sides of the signal, thus resulting in an increased pitch Poulton *et al.* (2013). This results in a significant increase in the wiring area. We note that the NoP wire has a 56x larger metal pitch than that for the wires within the chiplet. Furthermore, an increased NoP channel width is needed for higher performance at the cost of increased area. For energy and latency, the IMC circuit component dominates with 63.4% and 69.7% contributions, respectively. The NoP contributes the second highest to energy, while the NoC contributes the least. Simultaneously, NoC contributes the second highest to

latency, while NoP contributes the least. Overall, the area is dominated by the NoP, and the energy and latency are dominated by the IMC circuit.

6.4.2.3 NoP and NoC Performance Trade-offs

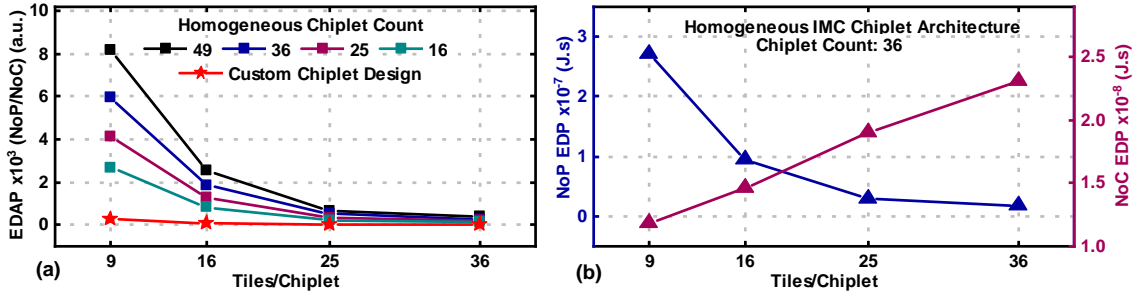


Figure 64. NoP and NoC Trade-Off Analysis for ResNet-110 on CIFAR-10 Dataset. (a) Ratio of the Energy-Delay-Area Product (EDAP) of NoP to NoC for Both Homogeneous and Custom Chiplet-based IMC Architectures. The Increase in Tiles per Chiplet Reduces the NoP/NoC EDAP, (b) NoP and NoC Energy-Delay Product (EDP) for a 36 Chiplet Count Configuration of Homogeneous RRAM-based Chiplet IMC Architectures. An Increased Tiles per Chiplet Leads to Higher NoC Cost and Lower NoP Cost.

We compare the EDAP for the NoC and NoP interconnect. Figure 64(a) shows the ratio of the EDAP of the NoP to that of NoC for ResNet-110 on CIFAR-10, for both homogeneous and custom RRAM-based chiplet IMC architectures. When there are fewer number of tiles per chiplet, more IMC chiplets are used to map the DNN to the IMC crossbars, resulting in distributed computing. This results in an increase in the data transfer volume across chiplets and higher NoP EDAP compared to that of NoC. Furthermore, at higher chiplet counts, the NoP is much larger and results in increased area, thus increasing the EDAP. As we increase the number of tiles per chiplet, computations are more localized, leading to reduced volume of data transfer

across chiplets. This reduces the ratio of NoP EDAP to NoC EDAP. The custom chiplet-based IMC architecture consists of the required number of chiplets to map the DNN under consideration. In addition, the custom chiplet architecture is designed specific to a DNN, resulting in a highly localized computing platform with a smaller NoP. Hence, the ratio of NoP EDAP to NoC EDAP is very small and is relatively insensitive to the change in tiles per chiplet.

To further understand the trade-off between NoC and NoP, we evaluate the energy-delay product (EDP) of NoC and NoP separately. Figure 64(b) shows the energy-delay product (EDP) for the NoP and NoC for a 36 chiplet count configuration of homogeneous chiplet IMC architecture. The x-axis shows the number of tiles in each chiplet. The EDP of NoP reduces with the increasing number of tiles in each chiplet. The reduced EDP of NoP is achieved due to the highly localized computing resulting in lesser inter chiplet communication data volume. At the same time, the NoC EDP increases with an increase in tiles per chiplet. The increased EDP is due to the larger NoC size (3x3 for 9 tiles per chiplet compared to 4x4 for 16 tiles per chiplet) and the increased intra-chiplet communication volume. Hence, a balance between the NoP and NoC cost is essential for optimal DNN inference performance with chiplet-based IMC architectures. We note that a similar trend is seen for other chiplet count configurations. From this experiment (ResNet-110 on CIFAR-10), we can conclude that the design with 16 tiles per chiplet provides a good balance in communication volume between NoC and NoP.

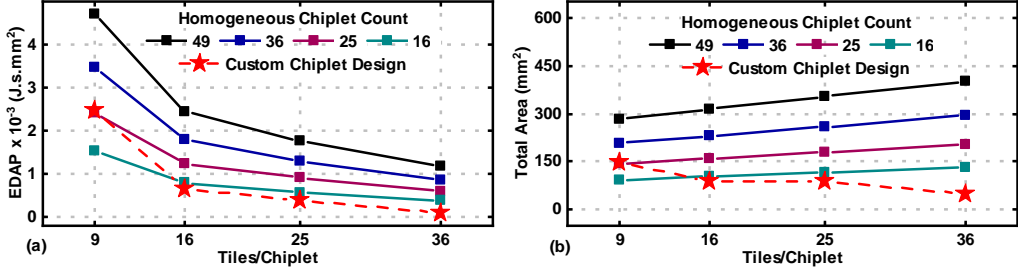


Figure 65. Energy-Delay-Area Product as the Metric. (a) Overall EDAP and (b) Total Area for the Homogeneous and Custom RRAM-based Chiplet IMC Architecture When Mapping ResNet-110 for CIFAR-10 Dataset. The Results Indicate that a Custom Architecture Outperforms a Homogeneous Architecture. The Increased Number of Tiles per Chiplet Provides Better Performance at the Cost of Increased Area for the Homogeneous Chiplet IMC Architecture, While Providing Better Performance and Lower Area for the Custom Chiplet IMC Architecture.

6.4.2.4 Overall Hardware EDAP and Area

Figure 65(a) shows the overall performance (EDAP) of the RRAM-based chiplet IMC architecture for ResNet-110 on CIFAR-10. For a homogeneous chiplet-based IMC architecture, the EDAP increases with higher chiplet counts (lower tiles per chiplet), resulting in higher chip area. Furthermore, with higher tiles per chiplet, the total energy contribution from the NoP reduces due to highly localized computing from the larger chiplet size and a lower NoP data volume. Overall, higher tiles per chiplet and lower chiplet count allow for a reduced EDAP in homogeneous RRAM-based chiplet IMC architectures. Simultaneously, the custom chiplet architecture has better performance than the homogeneous architecture due to the reduced NoP size and a customized architecture for the given DNN. A similar outcome arises for the custom design on increasing the tiles per chiplet, with the reduction in the EDAP.

Figure 65(b) shows the overall area of the chiplet-based IMC architecture with different tiles per chiplet and different chiplet counts (homogeneous) and the custom

chiplet architecture. The increase in the number of tiles per chiplet results in a higher area for the homogeneous chiplet architecture. The increase in area is due to the larger chiplet size while keeping the total chiplet count fixed. For example, the area for a 36 chiplet count and 16 tiles per chiplet architecture is larger than that of the 36 chiplet count and 9 tiles per chiplet architecture. Furthermore, the custom chiplet IMC architecture utilizes the required number of chiplet to map the whole DNN. Such an architecture benefits from increasing the tiles per chiplet since fewer chiplets are required to map the DNN. This results in lower NoP area and chiplet area (IMC circuit and NoC). Hence, with the increase in the number of tiles per chiplet the total area is reduced for the custom chiplet IMC architecture.

6.4.3 Comparison between Monolithic and Chiplet-based IMC Architectures

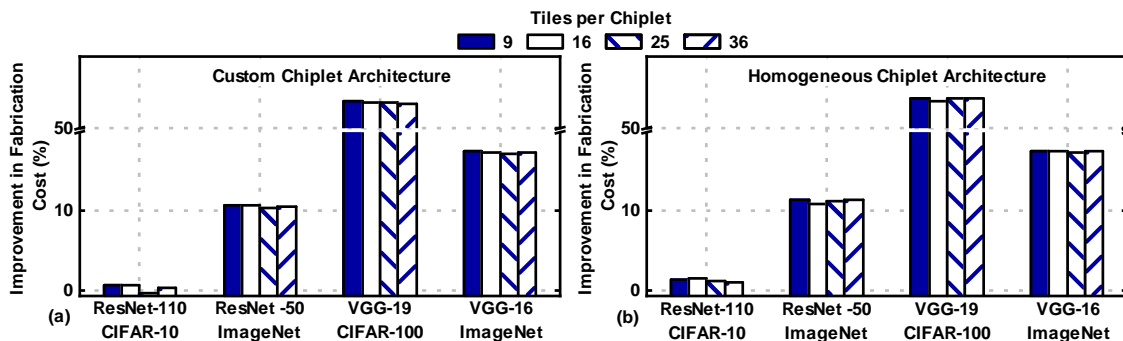


Figure 66. Improvement in Fabrication Cost (\$) for the RRAM-based Chiplet IMC Architecture, (a) Custom and (b) Homogeneous, as Compared to a Monolithic RRAM-based IMC Architecture. Smaller DNNs Like ResNet-110 have Similar Cost for Both Architectures, while Larger DNNs Such as VGG-19 have up to 60% Improvement.

We perform a comparison between a custom monolithic RRAM-based architecture and both homogeneous and custom RRAM-based chiplet IMC architectures. Due to

increased area, a monolithic IMC architecture suffers from increased defect ratio and lower yield. Consequently, a large monolithic IMC architecture experiences a very high fabrication cost, as shown in Figure 54(a). Figure 66 shows the improvement in fabrication cost of (a) custom and (b) homogeneous RRAM-based chiplet IMC architectures, compared to that of a monolithic RRAM-based IMC architecture. We observe that the improvement is similar for different number of tiles per chiplet for a particular DNN. The increase in the number of tiles per chiplet results in a reduction in the total used chiplets, while keeping the same utilization across the used chiplets. Moreover, the improvement is similar for both custom and homogeneous chiplet architecture for a particular DNN. The improvement in fabrication cost is a strong function of the DNN structures. DNNs with fewer parameters exhibit less improvement. For example, ResNet-110 with 1.7M number of parameters shows up to 0.57% improvement in fabrication cost. At the same time, VGG-19 with 45.6M number of parameters show more than 50% improvement in the fabrication cost. The reduced fabrication cost is attributed to lower defect ratio and increased yield achieved through smaller chiplets connected together to form a large system. Hence, larger and branched DNNs benefit significantly from chiplet-based IMC architectures.

6.4.4 Calibration with SIMBA

This section presents the calibration results for the SIAM RRAM-based chiplet IMC architecture compared to published silicon data from SIMBA Shao *et al.* (2019). We note that, there is no prior work that reports real silicon data for chiplet-based IMC architectures. Therefore, we choose SIMBA, the work that has the most resemblance to that of SIAM. We utilize the NoP driver circuit and the signaling technique similar to

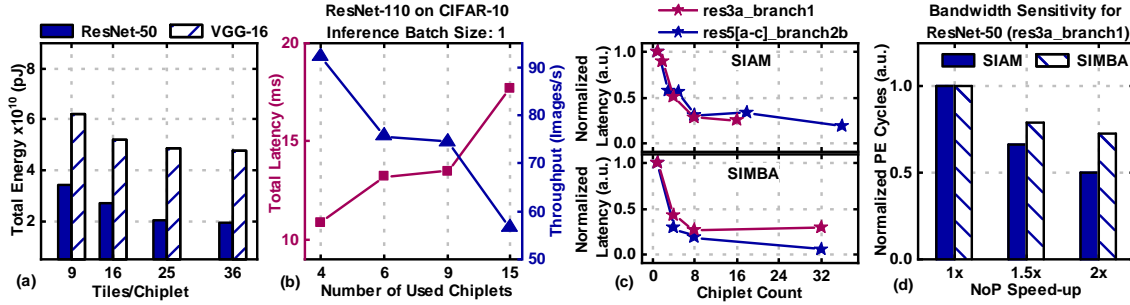


Figure 67. (a) Total Energy for DNN Inference Reduces with an Increase in the Number of Tiles per Chiplet (Chiplet Size), (b) Total Inference Latency and Throughput for ResNet-110 on CIFAR-10. Due to the Small Network Size, a Lower Number of Chiplets Provide Better Performance, (c) Normalized Latency for Two Layers Within ResNet-50 on ImageNet for SIAM RRAM-based Chiplet IMC Architecture and SIMBA Shao *et al.* (2019). The Decreasing Trend of Latency with Increasing Chiplet Count Exhibited by SIAM is Consistent with SIMBA, (d) Bandwidth Sensitivity for a Layer Within ResNet-50. The Decreasing Trend in the PE Cycles with Increasing NoP Speed-Up Similar to SIMBA.

those in SIMBA for our experiments. Furthermore, the NoP interconnect parameters utilized in SIAM are closed to that in SIMBA.

Total Energy: Figure 67(a) shows the total energy for inference across different number of tiles per chiplet for both ResNet-50 and VGG-16 on the ImageNet dataset. The increase in the number of tiles per chiplet results in a reduction in the total number of chiplets used to map the DNN and, in turn, a reduction in the total inference energy. The same trend is reported in SIMBA (ResNet-50).

Total Latency: We evaluate the effect of chiplet scaling or, in other words, the number of chiplets used to map a small DNN. Figure 67(b) shows the total inference latency and throughput for ResNet-110 on CIFAR-10 dataset. Since ResNet-110 is a small DNN, distributing the computation across more chiplets results in a sub-optimal configuration. A similar trend is shown in SIMBA for a small DNN, DriveNet Bojarski *et al.* (2017).

Layer Sensitivity: We consider two representative layers in ResNet-50,

res3a_branch1 and res5[a-c]_branch2b , same as that shown in SIMBA. We analyze the latency (normalized with the latency of the design consisting of only one chiplet) by varying the number of chiplets used to map the DNN layer. We note that the chiplet count to map the DNN is different from SIMBA due to the difference in the computation element in SIAM (IMC crossbars) and SIMBA (MAC arrays). The analysis shown in Figure 67(c) (top) reveals that there is a decreasing trend in latency with increasing number of chiplets. For res3a_branch1 the latency reduces initially with the increase in chiplet count and finally increases slightly (with chiplet count of 16). The increase in latency is due to a higher NoP latency from more distributed computation. res5[a-c]_branch2b shows a consistent decrease in the latency with the increase in chiplet count. These trends are consistent with that reported in SIMBA, as shown in Figure 67(c) (bottom).

PE cycles vs NoP speed-up: In this experiment, we vary NoP frequency and analyze the variation in PE cycles of res3a_branch1 layer in ResNet-50. We normalize it to the $1\times$ case to be consistent with SIMBA. The SIAM chiplet IMC architecture shows decreasing PE latency with increasing NoP bandwidth, which conforms SIMBA, as shown in Figure 67(d).

In summary, these comparisons confirm that, during the scaling of chiplet parameters, such as the number of chiplets and their utilization, SIAM predicts similar trends as the measured results from real silicon.

6.4.5 Comparison with GPUs

We compare the performance of the chiplet-based IMC architecture generated using SIAM with state-of-the-art GPUs such as Nvidia V100 and T4. Unlike GPUs, SIAM

generates architectures for energy-efficient inference using small batch sizes. All GPU hardware performance numbers have been adopted from those reported in Shao *et al.* (2019). We compare the performance of the architecture for inference with a batch size of one. For ResNet-50 on ImageNet dataset, the architecture generated using SIAM (36 tiles per chiplet) results in a total area of 273 mm² as compared to 525 mm² for T4 and 815 mm² for V100. The reduced area is attributed to the high compute density achieved using an IMC design and the support for a wide range of computations within the GPU. We also compare the energy-efficiency of SIAM IMC architecture to that of the GPUs for ResNet-50 on ImageNet. SIAM achieves 130× and 72× higher energy-efficiency as compared to V100 and T4 GPUs, respectively. The chiplet-based IMC architecture has higher performance since IMC architectures have all weights on chip, thus avoiding the external memory access. Furthermore, IMC utilizes analog domain computation within the crossbar arrays that are more energy-efficient than regular multiply-and-accumulate (MAC) units Shafiee *et al.* (2016).

6.4.6 Simulation Time

Table 19. Simulation Time for SIAM

Network	Dataset	Model Size (M)	Simulation Time (Hours)
ResNet-110	CIFAR-10	1.7	0.2
VGG-19	CIFAR-100	45.6	0.36
ResNet-50	ImageNet	23	1.26
VGG-16	ImageNet	138	4.26

Table 19 shows the simulation time for the proposed chiplet-based IMC simulator, SIAM, for different DNNs across different datasets. The simulation time is extracted by

running SIAM on an Intel Xeon W-2133 CPU platform with 12 cores and 32GB RAM. The range of the simulation times varies from a couple of minutes for small DNNs to a few hours for large DNNs. For a fair analysis, we report the overall simulation time that includes the partitioning and mapping, circuit and NoC simulation, NoP estimation, and DRAM access estimation. For example, ResNet-110 with 1.7M parameters for CIFAR-10 dataset takes 12 minutes (0.2 hours) for SIAM to perform the performance benchmarking. A large DNN such as VGG-16 with 138M parameters on the ImageNet dataset takes 4.26 hours for SIAM to perform the benchmarking.

Finally, we perform a comparison between SIAM and NeuroSim Peng *et al.* (2019a) in terms of simulation for benchmarking a RRAM-based monolithic IMC architecture. We note that we choose a monolithic IMC architecture as no other simulator supports chiplet-based IMC architecture benchmarking. We perform the comparison for four networks namely, ResNet-110 (1.7M), VGG-19 (45.6M), ResNet-50 (23M), and VGG-16 (138M). For ResNet-110 SIAM requires 60s while NeuroSim takes 30s while for VGG-19, SIAM takes 86s and NeuroSim takes 46s. Furthermore, for ResNet-50 SIAM takes 818s while NeuroSim takes 276s and for VGG-16 SIAM takes 3110s while NeuroSim takes 1110s. We note that the simulation times for SIAM are in the same range as that of NeuroSim rather than being orders of magnitude higher. The increased simulation times are due to the additional functionality that SIAM provides in NoC and DRAM performance estimation.

6.5 Conclusion and Discussion

This work presents SIAM, a novel performance benchmarking tool for chiplet-based IMC architectures. To the best of our knowledge, this will be the first benchmarking

tool for design space exploration of chiplet-based IMC architectures. SIAM integrates device, circuits, architecture, NoC, NoP, and DRAM estimation into an end-to-end system. It supports two types of chiplet architectures, homogeneous and custom, generated using different partition schemes. In addition, SIAM supports different DNNs across different datasets and various IMC chiplet configurations. Through this study, we establish the scalability and flexibility features of SIAM. We demonstrate the speed of SIAM by evaluating the simulation time of SIAM for different DNNs and comparing it against state-of-the-art chiplet simulators like NeuroSim for monolithic IMC architectures. Next, we calibrate SIAM with respect to published silicon result, SIMBA, which confirms that the trends projected by SIAM match the measured results from real silicon. Finally, we compare the performance of the generated chiplet-based IMC architecture using SIAM to that of state-of-the-art GPUs. The SIAM chiplet architecture achieves $130\times$ and $72\times$ improvement in energy-efficiency compared to Nvidia V100 and T4 GPUs, respectively.

Appendix A

Estimating manufacturing cost of a chip: Let us consider a reference chip with area A_{ref} and N_{ref} chips per wafer. The cost of the reference chip (C_{ref}) is expressed as shown in Equation 6.2.

$$C_{ref} = \frac{C_{Total}}{\eta_{ref}N_{ref}} \quad (6.2)$$

where, η_{ref} is the yield of the wafer. Number of chips per wafer (N_{ref}) is expressed in 6.3 AnySilicon (2011).

$$N_{ref} = D\pi\left(\frac{D}{4A_{ref}} - \frac{1}{\sqrt{2A_{ref}}}\right) \quad (6.3)$$

where D is the wafer diameter. The cost of a target (C_{target}) and normalized cost (C_{norm}) are:

$$C_{target} = \frac{C_{Total}}{\eta_{target}N_{target}}, \quad C_{norm} = \frac{C_{target}}{C_{ref}} = \frac{N_{ref}\eta_{ref}}{N_{target}\eta_{target}} \quad (6.4)$$

Assuming poisson defect model, $\eta = e^{-D_0A}$, where D_0 is the defect density. Replacing the expression of η in expression for C_{norm} , we obtain:

$$C_{norm} = \frac{N_{ref}e^{-D_0A_{ref}}}{N_{target}e^{-D_0A_{target}}} = \frac{N_{ref}}{N_{target}}e^{-D_0(A_{ref}-A_{target})} \quad (6.5)$$

Verification of chip cost estimation: To verify the chip cost estimation, we assume $A_{ref} = 296mm^2$, $D_0 = 0.012/mm^2$ and $D = 152.4mm$. The comparison reveals that the estimation is 98% accurate with respect to the real chip cost of commercial processors Tam *et al.* (2018).

BIG-LITTLE CHIPLETS FOR IN-MEMORY ACCELERATION OF DNNs: A
SCALABLE HETEROGENEOUS ARCHITECTURE

Prior studies have demonstrated chiplet-based architectures based on both IMC and conventional multiply-and-accumulate (MAC) engines for DNN acceleration Krishnan *et al.* (2021e); Tan *et al.* (2021); Shao *et al.* (2019); Pal *et al.* (2021); Wang *et al.* (2021); Kwon *et al.* (2021); Kim *et al.* (2020); Vivet *et al.* (2020); Zheng *et al.* (2020); Hwang *et al.* (2020); Li *et al.* (2021, 2022a,b). However, existing schemes do not consider the non-uniform distribution of weights and activations within DNNs while designing the chiplet-based architecture.

Figure 68(a) and Figure 68(b) show the distribution of activations and weights (normalized) across all layers of ResNet-50 on ImageNet and VGG-19 on CIFAR-100. The initial layers have more activations between layers but have fewer weights. A larger number of activations lead to more on-chip data movement, while fewer weights imply reduced computations. In contrast, the latter layers have more weights and fewer activations, resulting in increased computations and reduced data movement. Hence, the chiplet-based IMC architectures should be optimized to match the non-uniform algorithm structure and maximize the efficiency of computation and data movement across the DNN layers.

Figure 69(a) shows the IMC utilization of four different DNNs using a homogeneous chiplet-based RRAM IMC architecture. The architecture utilizes chiplets with 16 tiles, where each tile consists of an array of 16 IMC crossbar arrays of size 256×256 Krishnan

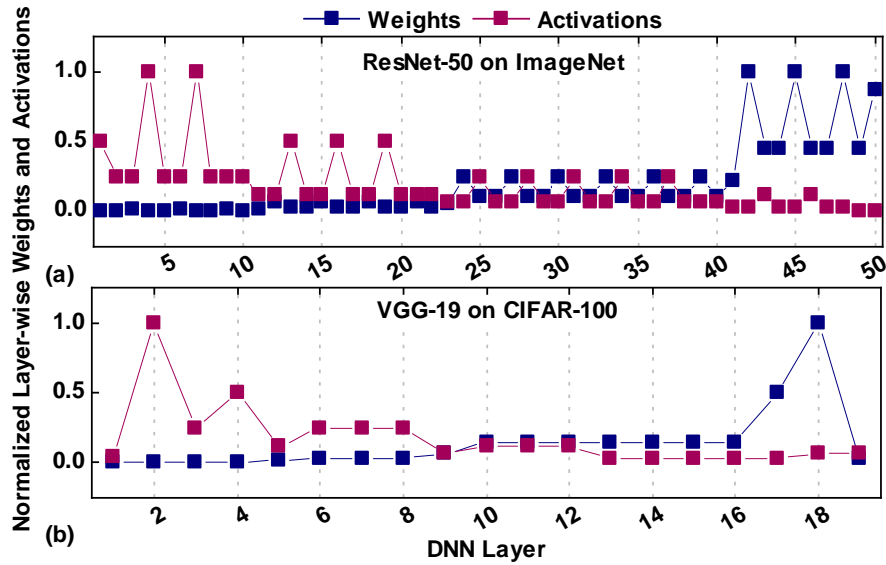


Figure 68. Normalized Layer-Wise Activation/Weight Distribution for (a) ResNet-50 (ImageNet) and (b) VGG-19 (CIFAR-100). Initial/Latter Layers are Activation/Weight Dominated.

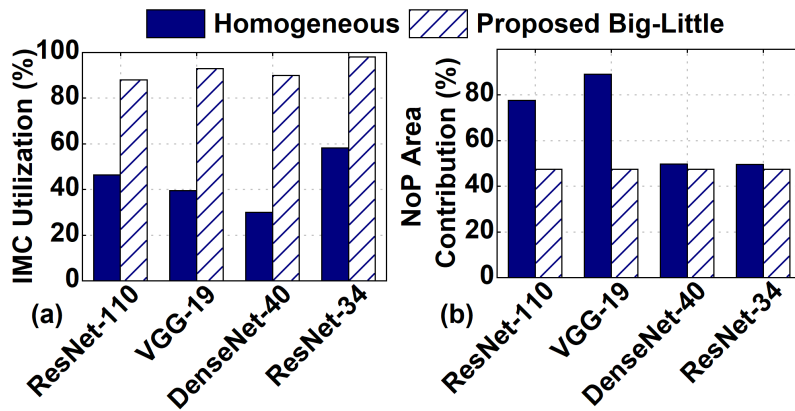


Figure 69. IMC Utilization for Different DNNs Using a Homogeneous Chiplet RRAM IMC Architecture Krishnan *et al.* (2021e) and the Proposed Heterogeneous Big-Little Chiplet Architecture. The Heterogeneous Big-Little Architecture Improves the IMC Utilization.

et al. (2021e). The chiplets are interconnected by a 32-bit wide NoP operating at 250MHz, having the signaling scheme in Turner *et al.* (2018). Smaller DNNs like

DenseNet-40 on CIFAR-10 have 29% IMC utilization, while larger DNNs like VGG-19 on CIFAR-100 achieve 40% IMC utilization.

A lower IMC utilization leads to increased IMC array arrays and in turn, higher energy and latency. Furthermore, a single NoP structure results in significant area overhead due to the large NoP driver and interconnect cost. Figure 69(b) shows that for the homogeneous structure, the NoP accounts for 90% and 50% of the total area for VGG-19 on CIFAR-100 and DenseNet-40 on CIFAR-10, respectively. In addition, the increased NoP bus width leads to higher NoP energy with up to $53.75\times$ higher cost relative to an 8-bit multiply-and-accumulate (MAC) operation in 16nm technology node Tan *et al.* (2021).

This work⁵ addresses the inefficiency of homogeneous chiplet-based IMC architectures that fail to exploit the underlying distribution of weights and activations within DNNs. To this end, we propose a heterogeneous chiplet-based IMC architecture that integrates big and little-chiplet banks, as illustrated in Figure 70. Specifically, we develop an algorithm to determine the optimal configuration of the big-little IMC chiplet architecture. The little-chiplet bank consists of little chiplets interconnected by an interposer-based NoP (chiplets are placed closed to each other) Turner *et al.* (2018). Similarly, the big-chiplet bank consists of big chiplets interconnected by a bridge-based NoP Mahajan *et al.* (2016). Little chiplets consist of fewer/smaller IMC crossbars or processing element (PE) arrays, while the big chiplets have more/larger IMC crossbars or PE arrays. In addition, each chiplet (big/little) utilizes a local DRAM to store the weights of the DNN.

In addition to the hardware architecture, we also propose a new technique to map DNNs onto the big-little chiplet-based IMC architecture. Taking a cue from the

⁵Work done in collaboration with Sumit K. Mandal (UW– Madison)

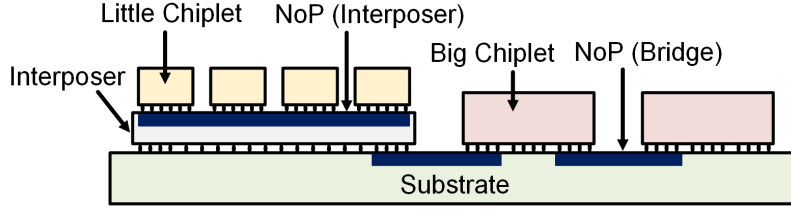


Figure 70. Cross-Sectional View of the Big-Little Chiplet-based IMC Architecture. The Architecture Consists of a Little Chiplet Bank with Little Chiplets (Connected by an NoP Within the Interposer) and a Big Chiplet Bank with Big Chiplets Connected by a Bridge NoP. NoP Properties: 1.5–8mm Length, 2–4.5 μ m Pitch, and 0.5–2 μ m Width.

non-uniform distribution of the weights and activations within the DNN, we propose to map the early layers within a DNN onto the little chiplet bank and the subsequent layers onto the big chiplet bank. The smaller structure of the weights in the early layers results in higher utilization within the little chiplet bank, while the larger layers towards the end of the DNN achieve high utilization on the big-chiplet bank. To achieve this, we develop a custom mapping algorithm that performs the mapping of the DNN on to the big-little architecture. We note that, the algorithm is universal and applies to the case when the resource in a given big-little chiplet is not enough to store all DNN weights. We exploit the activation distribution by utilizing an interposer-based NoP with high bandwidth within the little chiplet bank, which houses the early layers with higher on-chip data movement. Simultaneously, the subsequent layers with lower on-chip data movement (fewer activations) utilize the bridge-based NoP with lower bandwidth within the big chiplet bank. Experimental evaluation of the proposed big-little chiplet-based RRAM IMC architecture on ResNet-50 on ImageNet shows up to 259 \times , 139 \times , and 48 \times improvement in energy-efficiency with lower area compared to Nvidia V100 GPU, Nvidia T4 GPU, and SIMBA Shao *et al.* (2019) architecture, respectively.

The major contributions of this work are as follows:

- We propose a heterogeneous big-little chiplet-based IMC architecture that utilizes a big and little IMC-based chiplet compute structure coupled with an optimal NoP configuration (interposer and bridge),
- We present a custom mapping strategy of DNNs onto the big-little chiplet IMC architecture that exploits the non-uniform distribution of weights and activations,
- Our experiments of the proposed big-little chiplet-based RRAM IMC architecture on ResNet-50 on ImageNet achieve up to $259\times$, $139\times$, and $48\times$ improvement in energy-efficiency and lower area compared to Nvidia V100 GPU, Nvidia T4 GPU, and SIMBA Shao *et al.* (2019) architecture, respectively.

7.1 Big-Little Chiplet Architecture

Figure 71(a) shows the top-level block diagram of the heterogeneous big-little chiplet IMC architecture. The architecture consists of two banks of IMC chiplets, a little bank (shown in yellow color) and a big bank (shown in light red color). The little IMC chiplet bank consists of chiplets with smaller and fewer IMC crossbar arrays compared to the big chiplets. It is placed on an interposer that houses the NoP. The NoP provides high bandwidth and a compact structure for on-package communication within the little chiplet bank. At the same time, the increased size and count within the big chiplet bank allow for higher computation capability. The big chiplets are directly connected to the substrate using micro-bumps. A bridge-based NoP is utilized within the big chiplet bank for on-package communication. Long

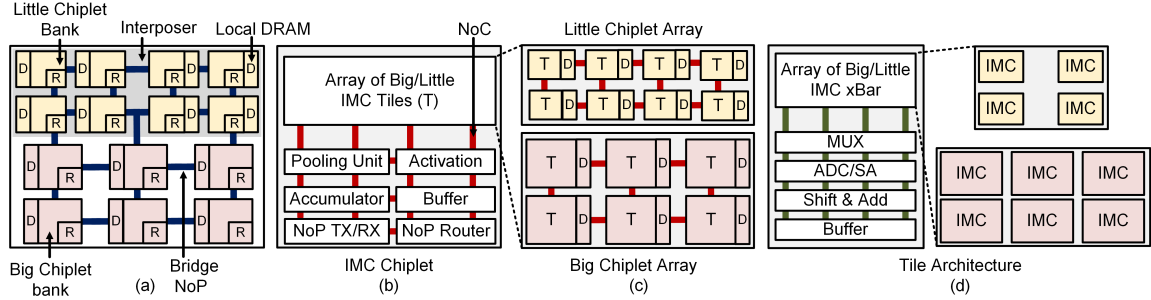


Figure 71. (a) Overview of the Big-Little Chiplet IMC Architecture. The Little Chiplet Bank Utilizes Smaller Chiplets Connected by a Interposer-based NoP While the Big Chiplet Bank Utilizes Bigger Chiplets Connected by a Bridge-based NoP. Each Chiplet Utilizes a Local DRAM, (b) IMC Chiplet Architecture (Big and Little). Each Chiplet Consists of an Array of IMC Tiles and a Dedicated NoP Transceiver and Router, (c) The Little Chiplet Bank Consists of Fewer and Smaller Tiles While the Big Chiplet Bank Consists of More Bigger Tiles. Both Chiplet Structures Utilize a Mesh-based NoC for On-chip Communication, and (d) Structure of Each Tile Within the Big and Little Chiplet. It Consists of an Array of IMC Crossbar Arrays and Associated Peripheral Circuits with an Interconnect Similar to That in Shafiee *et al.* (2016). The Little Chiplet Consists of Fewer and Smaller IMC Crossbars While the Big Chiplet has Larger and More IMC Crossbar Arrays.

wires of the bridge NoP allow easy integration of the big chiplets. We utilize the Y–X routing methodology for the NoP. Each chiplet (big and little) consists of a local DRAM (DDR4 in this work) that stores the weights required for the IMC crossbar arrays.

Figure 71(b) shows the structure of a IMC chiplet. Each chiplet utilizes a hierarchical structure that consists of an array of big (bottom of Figure 71(c)) or little IMC tiles (top of Figure 71(c)) and each tile consists of an array of IMC crossbars or PEs. In addition, the chiplet contains a pooling unit, non-linear activation unit, accumulator, and buffer. The accumulator is used for the partial sum accumulation across different tiles within the chiplet. Furthermore, the buffers allow for efficient data movement in and out of the chiplet. Each IMC chiplet consists of a dedicated NoP transceiver used for the transmission and reception of packets across the NoP. In

this work, we adopt the NoP transceiver from Turner *et al.* (2018). Each transceiver consists of a local PLL circuit that provides the clock for the transceiver. A five-port router is utilized for routing of the data across the NoP.

Each IMC chiplet utilizes a local DRAM to store the weights. The local DRAM allows for external memory access, thus making our proposed big-little architecture a generic platform. If a DNN does not fit on the entire chip, the DRAM stores all the weights necessary for each chiplet. First, the DRAM loads the necessary weights into the IMC crossbar arrays. Next, while the computation is performed, the DRAM loads the next set of weights of the DNN. The buffer is designed to support a ping-pong operation Ma *et al.* (2017b). The weights from the DRAM are loaded into the first buffer stage (ping) and then moved to the second buffer stage (pong). Therefore, the big-little IMC chiplet architecture masks the DRAM latency with the computation latency, achieving high throughput.

Finally, Figure 71(d) shows the structure of an IMC tile. Each array in the crossbar consists of PEs that perform the computations. In this work, we focus on a resistive random-access-memory (RRAM) based IMC crossbar array due to its superior energy-efficiency Shafiee *et al.* (2016). The computations are performed in the analog domain by turning on all wordlines (WL) together and performing accumulation along the bitline (BL). The inputs are given through the WL while the weights are stored within the RRAM cells. Each IMC array consists of specialized peripheral circuitry that assists the computation. The peripheral circuitry includes a column multiplexer (mux), an analog-to-digital converter (ADC), a shift and add circuit, and a buffer. The column mux is used to share the ADC across columns of the IMC array. The ADC converts the MAC output in the analog domain across each column into the digital domain. The big-little IMC architecture does not utilize a digital-to-analog converter

(DAC) by employing bit-serial computing. The shift and add circuit handles the positional value of each bit within the multi-bit input activations that are computed using the IMC arrays. The buffers within the tile are utilized for storing the partial sums and the input activations.

7.2 Parameters of the Big-Little Architecture and Mapping

The underlying non-uniform distribution of weights and activations within a DNN results in an increased number of activations in the early layers and larger number of weights in the subsequent layers (Figure 68). This non-uniform weight distribution leads to under-utilization of chiplets in the early layers, thus a lower overall IMC utilization. To improve the IMC utilization, crossbar arrays with smaller size (e.g. 32×32 instead of 128×128) can be used everywhere. In turn, larger number of chiplets in the system increases the area as well as energy consumption (due to higher relative area and energy of the peripheral circuits) masking the benefit of using chiplet-based system. Therefore, a balance between crossbar array size and number of chiplets in the system is necessary. To this end, we propose a technique to optimize the big-little chiplet configuration as discussed next.

7.2.1 Configuration of the big-little chiplets

We first determine the configuration of big-little chiplets by computing the tile utilization with different big-little chiplet configurations for a given DNN. Algorithm 11 shows our proposed technique to find the utilization. The inputs to the algorithm are

1. the set of crossbar sizes for the little chiplets (\mathcal{X}_L) and the big chiplets (\mathcal{X}_B),

2. set of number of tiles in the little chiplets ($\mathcal{T}_{\mathcal{L}}$) and the big chiplets ($\mathcal{T}_{\mathcal{B}}$),
3. number of little chiplets ($\mathcal{N}_{\mathcal{L}}$) and big chiplets ($\mathcal{N}_{\mathcal{B}}$),
4. the DNN structure,
5. the total number of chiplets in the system.

We note that the initial layers of the DNN are mapped on to little chiplets since there are fewer weights in the initial layers. A DNN layer is mapped on to a chiplet when number of tiles required for that layer is less than the number of remaining tiles in the chiplet, i.e., the available resource on the chiplet is sufficient for the layer (as shown in line 13–17 of Algorithm 11). Once a layer (layer- j) is mapped on to a chiplet, the tile utilization is computed as:

$$\begin{aligned}
 IMC_j &= \left\lceil \frac{K_j^x \times k_j^y \times N_j^{if}}{x} \right\rceil \times \left\lceil \frac{N_j^{of} \times Q}{x} \right\rceil \\
 u_j &= 100 \times \frac{K_j^x \times k_j^y \times N_j^{if} \times N_j^{of} \times Q}{IMC_j \times x \times x}
 \end{aligned} \tag{7.1}$$

where K_j^x and K_j^y are the kernel sizes of layer- j , N_j^{if} and N_j^{of} are the number of i/p and o/p features for layer- j , Q is the quantization precision, IMC_j is the number of IMC crossbars required for layer- j and x is the IMC crossbar size ($x \times x$). Once the resources of a chiplet are exhausted, the next chiplet is considered for mapping. This process continues until no chiplet (little/big) is available.

In the proposed method, for each chiplet configuration, we obtain the average utilization for a particular DNN after each layer is mapped (line 36 of Algorithm 11). Then we sort (in descending order) the configurations based on the utilization and save the top K configurations. The above procedure is repeated for M different DNNs and the configuration with highest utilization which is common for all DNNs is considered as the final configuration for the big-little chiplet system. We note that

K and M are user-defined parameters and our proposed technique is independent of these parameters.

Algorithm 9: Determining Big-Little NoP Configuration

```

1 Input: DNN structure, number of chiplets ( $N_C$ ), set of NoP bus widths for
   the little chiplets ( $\mathcal{W}_L$ ) and the big chiplets ( $\mathcal{W}_B$ ); set of NoP frequency for
   the little chiplets ( $\mathcal{F}_L$ ) and the big chiplets ( $\mathcal{F}_B$ ), mapping of layers to the
   big-little chiplet ( $\mathcal{L} \rightarrow \mathcal{C}$ )
2 Output: NoP EDP for each configuration- $i$  ( $E_i$ )
3  $N_{cfg} \leftarrow$  number of configurations in the set containing all possible
   combinations of the elements in  $\mathcal{W}_L, \mathcal{W}_B, \mathcal{F}_L, \mathcal{F}_B$ 
4  $L \leftarrow$  number of DNN layers
5  $n_l =$  Number of little chiplets
6  $n_b =$  Number of big chiplets
7 for  $i = 1 : N_{cfg}$  do
8    $w_l =$  Bus-width of little chiplets in Config- $i$ 
9    $w_b =$  Bus-width of big chiplets in Config- $i$ 
10   $f_l =$  NoP frequency of little chiplets in Config- $i$ 
11   $f_b =$  NoP frequency of big chiplets in Config- $i$ 
12   $E_i \leftarrow 0$  // Initializing EDP of Config- $i$ 
13  for  $j = 1 : n_l$  do
14    Compute  $edp_j$  by from Equation 7.2
15     $E_i = E_i + edp_j$  // Communication EDP
16  end
17  for  $k = 1 : (n_b - 1)$  do
18    Compute  $edp_k$  from Equation 7.2
19     $E_i = E_i + edp_k$  // Communication EDP
20  end
21 end

```

7.2.2 Configuration of the big-little NoP

The heterogeneous chiplet configuration (discussed in Section 7.2.1) improves the overall chiplet utilization by using smaller chiplets that match well to the early layers with fewer weights. However, the initial DNN layers produce higher number

Algorithm 10: Mapping DNN Layers to Big-Little Chiplets

```
1 Input: DNN layers ( $\mathcal{L}$ ), IMC crossbar size in Big chiplet ( $x_b$ ), IMC crossbar
   size in little chiplet ( $x_l$ ), number of tiles in big chiplets ( $t_b$ ), number of tiles in
   little chiplets ( $t_l$ ), number of available big chiplets ( $n_b$ ), number of available
   little chiplets ( $n_l$ )
2 Output: Mapping of layers to of big-little chiplet ( $\mathcal{L} \rightarrow \mathcal{C}$ )
3 Compute  $S_B$  by following Equation 7.3
4 Compute  $Pr$  by following Equation 7.4
5 for  $j = 1 : Pr$  do
6    $\mathcal{L}_j \rightarrow$  DNN layers for partition- $j$ ;  $\mathcal{L}_j \in \mathcal{L}$ 
7   for  $i = 1 : |\mathcal{L}_j|$  do
8      $a_L \rightarrow 1$  // Number of little chiplets used
9     Compute utilization of  $i^{\text{th}}$  ( $U_B^i$ ) layer on a big chiplet using  $x_b, t_b$ 
10    Compute utilization of  $i^{\text{th}}$  ( $U_L^i$ ) layer on a little chiplet using  $x_l, t_l$ 
11    if  $((U_B^i < U_L^i) \& (a_L \leq A_L))$  then
12      Map  $i^{\text{th}}$  layer to little chiplet.
13      if Resource in  $a_L$  is exhausted then
14         $a_L \rightarrow a_L + 1$ 
15      end
16    end
17    else
18      Compute number of big chiplets ( $a_B$ ) required to map layer- $i$  –
        layer- $|\mathcal{L}_j|$ 
19       $\text{assert}((a_B \leq A_B), \text{'Error'})$ 
20      for  $k = i : |\mathcal{L}_j|$  do
21        Map  $k^{\text{th}}$  layer to big chiplet.
22      end
23      break;
24    end
25  end
26 end
```

of activations compared to later layers. Therefore, the volume of traffic between little chiplets (used for initial DNN layers) is higher than the traffic volume between big chiplets (used for later DNN layers). Hence, the network-on-package (NoP) configuration between little chiplets needs to be different than that of the big chiplets. To this end, we propose a technique to determine optimal NoP configuration for a

system with big-little chiplet targeted for a particular DNN. Algorithm 9 shows the technique to determine NoP configuration for a particular DNN. The inputs to the algorithm are:

1. big-little chiplet configuration obtained from Algorithm 11,
2. set of NoP bus width for the little chiplets (\mathcal{W}_L) and the big chiplets (\mathcal{W}_B),
3. set of NoP frequency for the little chiplets (\mathcal{F}_L) and the big chiplets (\mathcal{F}_B),
4. the DNN structure.

We evaluate the energy-delay product of communication for each NoP configuration in the set of configurations. An analytical expression based evaluation is incorporated to perform fast exploration in the NoP configuration space. First, we evaluate communication volume of each NoP configuration given a particular DNN. The communication volume is equivalent to the number of packets transferred between two chiplets, and the number of packets (P) is expressed as $P = \frac{b}{w}$, where b is the number of bits to be communicated and w is the NoP bus width. We divide the number of packets by NoP frequency (f) to obtain an approximation of NoP latency $d = \frac{P}{f} = \frac{b}{w \times f}$. Next, we compute NoP power consumption by assuming that it is proportional to cube of NoP frequency Mudge (2001); $p = f^3$. Then the approximate energy consumption (e) is computed by multiplying communication latency and communication power; $e = d \times p$. Finally, communication EDP between each pair of chiplet (edp) is computed as:

$$edp = e \times d = d \times p \times d = d^2 \times f^3 = \left(\frac{b}{w \times f}\right)^2 \times f^3 = \frac{b^2 \times f}{w^2} \quad (7.2)$$

The total communication EDP for each NoP configuration for a particular DNN is obtained by adding the communication EDP between each pair of chiplets. A total of K NoP configurations with lower EDP are saved and the above procedure is repeated

for M different DNNs. The configuration with lowest cost which is common for all DNNs is considered as the final NoP configuration for the big-little chiplet system. Similar to the technique of selecting big-little chiplet configuration (described in Section 7.2.1), K and M are the user defined parameter and our proposed technique is independent of these parameters.

7.2.3 Mapping a Previously Unseen DNN to a System on big-little Chiplets

So far, we described our proposed technique to determine the optimal configuration of big-little chiplet and the NoP. The optimal configuration is determined by performing design space exploration with several DNNs. However, an unknown DNN (not seen before) may be encountered at runtime. Moreover, there is no guarantee that all the weights of a given DNN will fit in the on-chiplet resources since the number of DNN parameters seem to be continuously growing. In these cases, we need to divide the entire DNN into multiple parts and load the weights of each part from DRAM before executing. Algorithm 10 shows the DNN partitioning as well as the mapping technique. The input to the algorithm is the DNN structure, big-little chiplet configuration and big-little NoP configuration. First, we compute the number of in-memory computing bits available on the system (S_B). Specifically, for each type (little/big) of chiplets, we multiply the number of available chiplets (n_l/n_b), the number of tiles in each chiplet (t_l/t_b), the number of crossbar array in each tile (16), and the size of IMC crossbar array for big and little chiplets (x_l/x_b). Then we add the product for big and little chiplets to obtain the total number of in-memory computing bits available on the system (S_B):

$$S_B = (n_l \times t_l \times 16 \times x_l \times x_l) + (n_b \times t_b \times 16 \times x_b \times x_b) \quad (7.3)$$

Next, we compute the number of bits required to store all the weights of the DNN (D_B). Assuming average utilization of $u(0 < u \leq 1)$, the total number of partitions (Pr) required for the DNN is computed by taking the ceiling of the quotient obtained by dividing the required number of bits to store all weights (D_B) by the available number of in-memory bits on the system (S_B):

$$Pr = \left\lceil \frac{D_B}{S_B \times u} \right\rceil \quad (7.4)$$

For each partition, first, we compute the utilization of i^{th} layer on a big chiplet (U_B^i) as well as on a little chiplet (U_L^i). We compute U_B^i and U_L^i using Equation 7.1. If the big chiplet utilization (U_B^i) is less than the little chiplet utilization (U_L^i) and the little chiplet bank is not exhausted, then the layer is mapped onto a little chiplet, as shown in lines 7–12 of Algorithm 10. Otherwise, we compute the number of big chiplets required (a_B) to map the rest of the layers. If a_B is less than or equal to the number of available big chiplets (A_B), then we map the rest of the layers to the big chiplet bank, else the algorithm throws an error since the resource requirement exceeds the available capacity (shown in line 18–23 of Algorithm 10). Thus, we ensure that the initial layers with fewer weights are mapped into little chiplets and the latter layers with higher number of weights are mapped onto big chiplets with more computation resources. Therefore, our proposed custom mapping of the DNN onto the big-little chiplet architecture ensures high IMC utilization.

7.3 Experimental Evaluation

7.3.1 Experimental Setup

Evaluation platform: To evaluate the proposed heterogeneous big-little IMC chiplet architecture, we use a customized version of the open-sourced tool SIAM Krishnan *et al.* (2021e). The customization includes the addition of the custom mapping scheme detailed in Section 7.2. In addition, we handle the big-little chiplet IMC architecture by adding the number of each type (big/little) of chiplets, the number of tiles inside big and little chiplets, and the big-little IMC structure. Furthermore, we also assume that each type of chiplet can use different NoP width. The simulator performs the mapping of a given DNN onto the big-little IMC chiplet architecture. The outputs include area, energy, latency, throughput, energy efficiency, and IMC utilization (for all individual components in the architecture). Finally, we add support for intermediate DRAM access (DDR4 MICRON (2014)) for each chiplet to handle the case where all weights do not fit on the system at once. We plan to open-source the tool and optimization methodology upon acceptance of the paper.

DNN algorithms and architectural parameters: We evaluate the proposed heterogeneous chiplet architecture with DenseNet-40 (0.26M) on CIFAR-10, ResNet-110 (1.7M) on CIFAR-10, VGG-19 (45.6M) on CIFAR-100, ResNet-34 (21.5M) and ResNet-50 (23M) on ImageNet. We utilize an RRAM-based IMC structure for DNN inference with the following parameters: one bit per RRAM cell, a R_{off}/R_{on} ratio of 100, ADC resolution of 4-bits with 8 columns multiplexed, operating frequency of 1GHz Imani *et al.* (2019); Shafiee *et al.* (2016), and a parallel read-out method. We use 8-bit quantization for the weights and activations, and a 32nm CMOS technology

Table 20. Set Of Configurations Considered to Determine Big-Little Chiplet and NoP Structure.

Chiplet Configuration		NoP Configuration	
Parameter	Values in the Set	Parameter	Values in the Set
$\mathcal{X}_{\mathcal{L}}$	{32, 64}	$\mathcal{W}_{\mathcal{L}}$	{16, 32, 64}
$\mathcal{X}_{\mathcal{B}}$	{128, 256, 512}	$\mathcal{W}_{\mathcal{B}}$	{4, 8, 12, 16, 20, 24}
$\mathcal{T}_{\mathcal{L}}$	{9, 16, 25}	$\mathcal{F}_{\mathcal{L}}$	{600, 1000, 1400, 1800} MHz
$\mathcal{T}_{\mathcal{B}}$	{36, 49}	$\mathcal{F}_{\mathcal{B}}$	{600, 800, 1000} MHz

node. The chiplets are placed to achieve the least Manhattan distance. The NoP parameters include E_{bit} of 0.54pJ/bit Turner *et al.* (2018), interconnect parameters width, length, and pitch for the interposer-based NoP from Turner *et al.* (2018) and for bridge-based NoP from Greenhill *et al.* (2017) (Figure 70), per lane NoP TX/RX area of $5,304 \mu\text{m}^2$, and NoP clocking circuit area of $10,609 \mu\text{m}^2$ Poulton *et al.* (2013). In addition, we also model the μbump for both the interposer Su *et al.* (2016) and bridge-based Liu *et al.* (2021) NoP by utilizing the PTM models Sinha *et al.* (2012).

7.3.2 Big-Little IMC Structure and NoP

This section demonstrates the parameters related to big-little IMC structure and big-little NoP. Specifically, we consider four DNNs (mentioned in Section 7.3.1) and execute Algorithm 11 to determine the top 10 ($K=10$) configurations with highest utilization for each DNN. We consider a system with 36 chiplets to limit the total area and power consumption of the system. Table 20 shows the input parameters ($\mathcal{X}_{\mathcal{L}}, \mathcal{X}_{\mathcal{B}}, \mathcal{T}_{\mathcal{L}}, \mathcal{T}_{\mathcal{B}}$) to the algorithm. We vary the number of little chiplets from 1 to 35 while maintaining the total number of chiplets to be 36. Then, we choose the best

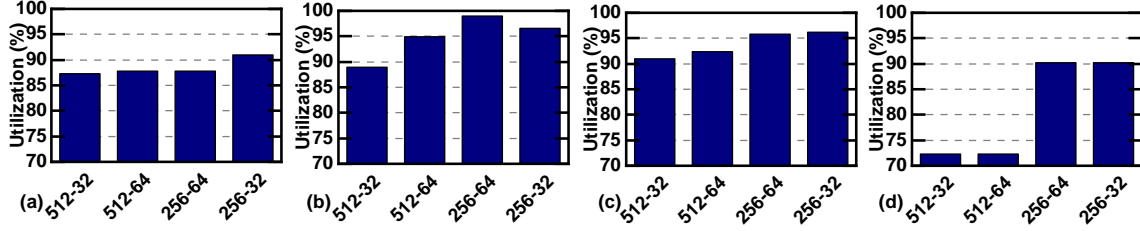


Figure 72. IMC Utilizations for Different DNNs Across Different Big-Little Chiplet-based RRAM IMC Configurations for (a) ResNet-110, (b) ResNet-34, (c) VGG-19, (d) DenseNet-40. Based on the Utilization, We Choose Crossbar Size of Big Chiplet as 256×256 and Crossbar Size of Little Chiplet as 64×64 (256-64).

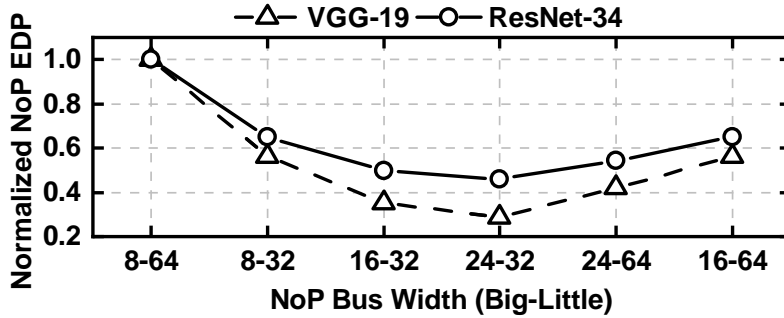


Figure 73. Normalized NoP EDP for Different Bus-Widths for VGG-19 and ResNet-34. The NoP With Bus Width of 24 for Big and 32 for Little Chiplets (24-32) Shows Lowest EDP.

Table 21. Performance Comparison of Each Component of a Homogeneous (Little Only, Big Only) Chiplet Architecture and the Heterogeneous Big-Little IMC Chiplet Architecture for VGG-19 on CIFAR-100.

Configuration	Area					Energy					Latency				
	IMC (%)	NoP (%)	NoC (%)	Total (mm ²)	Normalized to big-little (x)	IMC (%)	NoP (%)	NoC (%)	Total (mJ)	Normalized to big-little (x)	IMC (%)	NoP (%)	NoC (%)	Total (ms)	Normalized to big-little (x)
Little only	11.9	88.0	0.1	952.1	10.9	99.7	0.2	0.1	1.3	4.1	99.7	0.1	0.2	1.6	1.3
Big only	44.0	55.5	0.5	597.2	6.8	78.6	11.0	10.4	0.43	1.3	99.6	0.1	0.3	3.2	2.7
Big-Little (this work)	52.4	47.4	0.2	87.4	1.0	99.8	0.1	0.1	0.32	1.0	99.2	0.3	0.5	1.2	1.0

configuration which is common for all four DNNs. We observe that a system with *25 little chiplets* with a 64×64 IMC crossbar and *25 tiles per chiplet*, and *11 big chiplets* with a 256×256 IMC crossbar and *36 tiles per chiplet* provides best utilization across all four DNNs.

Figure 72 shows the utilization for a system with 25 little and 11 big chiplets with varying size of crossbars (both for big and little chiplet) for all four DNNs. In this case, we also fixed the number of tiles per chiplet to 25 for the little chiplets and 36 for the big chiplets. Figure 72 reveals that the configuration where the crossbar size of the big chiplets is 256×256 and the crossbar size of the little chiplets is 64×64 (256–64, 256 denotes crossbar size of big chiplets and 64 denotes crossbar size of little chiplets) shows higher utilization than other configurations for three out of four DNNs. Only in the case of ResNet-110, the configuration 256–32 shows higher utilization than 256–64. However, we choose 256–64 over 256–32 since it provides more on-chip resources, lower area and energy efficiency for the IMC crossbar array (due to peripheral circuits).

Similarly, we execute Algorithm 9 for four DNNs to obtain the NoP configuration. Table 20 shows the set of different NoP parameters ($\mathcal{W}_{\mathcal{L}}, \mathcal{W}_{\mathcal{B}}, \mathcal{F}_{\mathcal{L}}, \mathcal{F}_{\mathcal{B}}$ used as inputs to Algorithm 9). The parameters are adopted from CHIPS Alliance (INTEL) (2021). EDP for NoP is obtained for all NoP configurations for the four DNNs. Then, the NoP configuration having the lowest EDP for all four DNNs is chosen. Based on the EDP results, the big NoP frequency and the little NoP frequency is set to 600 MHz and 1 GHz, respectively; the big NoP bus width and the little NoP bus width is set to 24 and 32, respectively. Figure 73 shows the normalized NoP EDP for different combination of bus width for big and little chiplets. For illustration purpose, we show VGG-19 and ResNet-34 since these two DNNs utilize more than 34 out of 36 chiplets. From Figure 73, it is observed that the configuration with big NoP bus width of 24 and little NoP bus width of 32 shows the lowest EDP. Since little chiplets produce higher number of activations than the big chiplets, it is intuitive that little NoP are wider (larger bus width) than the big NoP.

Table 22. Performance Comparison of a Homogeneous (Little Only, Big Only) Chiplet Architecture and the Heterogeneous Big-Little IMC Chiplet Architecture For Different DNNs.

Configuration	Utilization (%)				Area (mm ²)				Energy (mJ)				Latency (ms)			
	Res-110	VGG-19	Dense-40	Res-34	Res-110	VGG-19	Dense-40	Res-34	Res-110	VGG-19	Dense-40	Res-34	Res-110	VGG-19	Dense-40	Res-34
Little only	69	92	58	93	171.7	952.1	71.5	657.8	1.4	1.3	0.22	41.1	23.0	1.6	1.6	13.1
Big only	44	59	32	82	220.0	597.2	220.2	595.9	0.28	0.43	0.11	3.7	1.1	3.2	0.02	20.2
Big-Little (this work)	88	93	90	98	87.4	87.4	87.4	87.4	0.18	0.32	0.06	8.2	1.1	1.2	0.03	48.6

7.3.3 Comparison with Baseline Architectures with Homogeneous Chiplets

We compare the performance of our proposed big-little chiplet architecture with respect to two baseline architectures with homogeneous chiplets Krishnan *et al.* (2021e).

1) Little only: In this configuration, we consider a system where the configuration of all chiplets as well as the NoP is same as that of the little chiplets. **2) Big only:** In this configuration, we consider a system where the configuration of all chiplets as well as the NoP is same as that of the big chiplets. We note that, the total number of chiplets with ‘Little only’ and ‘Big only’ configurations vary for different DNNs.

Table 21 shows the performance comparison for ‘Little only’, ‘Big only’ and the proposed big-little architectures for VGG-19 on CIFAR-100. In this table, the performance of each component of the architecture, i.e. IMC, NoP and NoC is shown. Our proposed big-little chiplet architecture results in a balanced distribution of the area among the circuit and NoP components, while the NoC accounts for a minimal portion (0.2%) of the total area. In ‘Little-only’ architecture, NoP becomes the bottleneck for area since the chiplets have smaller size, hence more number of chiplets are required which increases the NoP. In ‘Big only’ architecture, NoP consumes more energy due to higher volume of data movement between each pair of chiplets. In contrast, the proposed big-little architecture with its high IMC utilization and reduced on-chip communication as well as on-package data movement results in less total

energy consumption and less inference latency. Overall, the proposed heterogeneous big-little architecture achieves up to $10.9\times$ lower area, $4.1\times$ lower energy, and $2.7\times$ lower latency than ‘Little only’ and ‘Big only’ architectures.

Next, we compare the IMC utilization and the performance (area, energy and latency) for ResNet-110, VGG-19, DenseNet-40, and ResNet-34 against ‘little only’ and ‘big only’ architecture. For VGG-19, our proposed big-little architecture achieves the highest IMC utilization of 93% compared to 92% and 59% for ‘Little only’ and ‘Big only’, respectively. Similarly, the big-little architecture achieves 88%, 90%, and 98% IMC utilization for ResNet-110, DenseNet-40, and ResNet-34, respectively, up to $2.8\times$ greater than ‘Little only’ and ‘Big only’ architectures. We observe that the big-little architecture provides *up to $7.8\times$ improvement in energy and up to $21\times$ improvement* in inference latency with respect to baseline homogeneous architectures. ‘Big only’ architecture consumes less energy and less latency than big-little architecture for ResNet-34, but in this case, the area of ‘Big only’ is $6.8\times$ higher than big-little architecture. To better analyze the performance comparison, we plot the energy-delay-area product (EDAP) for all DNNs, as shown in Figure 74. The big-little chiplet architecture provides up to $329\times$ lower EDAP than the ‘Little only’ and ‘Big only’ architectures across all four DNNs. Although ‘Big only’ architecture shows improvement in energy consumption and inference latency with respect to big-little for ResNet-34, the EDAP with ‘Big only’ is $1.3\times$ higher than big-little architecture in this case. Hence, the proposed big-little IMC architecture achieves optimal performance through reduced EDAP at higher IMC utilization across different DNNs.

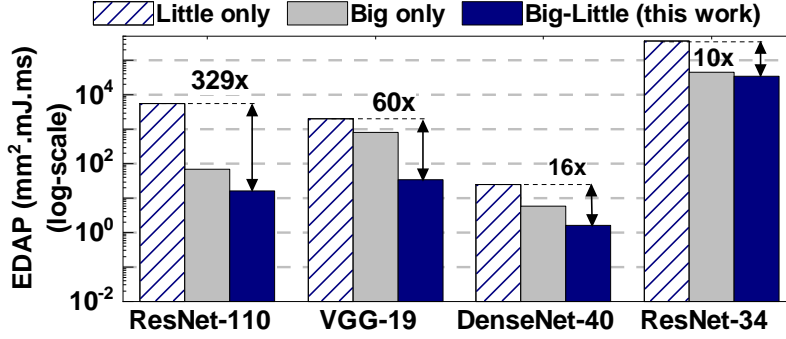


Figure 74. EDAP Comparison (Log-Scale) of the Big-Little Chiplet-based RRAM IMC Architecture to ‘Little Only’ and ‘Big Only’ Chiplet-based RRAM IMC Architectures. The Big-Little Architecture Achieves up to 329× Improvement Compared to ‘Little Only’ Architecture.

7.3.4 Results with DRAM (DDR4)

In this section, we show the performance results when the resource on a big-little chiplet-based system is not sufficient to store all the weights of a given DNN. In that case, the DNN is divided into multiple partitions. One partition is mapped on to the big-little chiplets at a time. While the computations of a partition of the DNN are performed, the weights corresponding to the next partition are loaded from the DRAM into the ping-pong buffer. The additional DRAM accesses result in increased energy. At the same time, the impact on latency is reduced through the ping-pong buffers Ma *et al.* (2017b). Table 23 shows the ratio between DRAM energy and compute energy for VGG-16 and VGG-19 with systems having different number of chiplets. We observe that, the ratio of DRAM energy to computation energy increases with reduction in the system sizes for both the DNNs. With decreasing system size, more weights need to be stored and loaded from DRAM, thereby increasing DRAM energy.

Table 23. Ratio Between DRAM Energy and Compute Energy for VGG-16 and VGG-19 With Systems Having Different Number of Chiplets (**All Weights of VGG-19 Fit On Chip With This Configuration, Significantly Reducing the DRAM Energy).

# Chiplets	VGG-16		VGG-19	
	#partitions	Ratio	#partitions	Ratio
36	2	1.1	1	0.08**
25	2	2.1	2	131
16	3	3.6	2	161

Table 24. Comparison With Other Platforms for ResNet-50 on ImageNet (*Reported in Shao *et al.* (2019)).

Platform	Area (mm ²)	Energy Efficiency (Images/s/W)
Nvidia V100 GPU*	815	8.3
Nvidia T4 GPU*	525	15.5
SIMBA Shao <i>et al.</i> (2019)	215	45
Big-Little (this work)	85	827

7.3.5 Comparison with State-of-the-art Work

Table 24 shows the comparison of the proposed heterogeneous big-little RRAM IMC chiplet architecture with an Nvidia T4 and V100 GPU, and SIMBA Shao *et al.* (2019). The big-little chiplet architecture achieves a lower area for the architecture due to the custom RRAM-based IMC and the optimized NoP structure. Compared to the Nvidia V100, Nvidia T4, and SIMBA architecture, the big-little IMC architecture achieves $9.6\times$, $6.2\times$, and $2.5\times$ area improvement and $99.6\times$, $53.4\times$, and $18.4\times$ energy-efficiency improvement, respectively. The improved energy efficiency is attributed to the higher IMC utilization, analog computation within the RRAM-based IMC,

reduced NoP data movement and bus width, and the absence of intermediate DRAM transactions for weights and partial sums.

7.4 Conclusion

We proposed a heterogeneous big-little chiplet-based IMC architecture driven by the non-uniform nature of DNN layers. To the best of our knowledge, this is the first heterogeneous chiplet-based IMC architecture that leverages different IMC compute structures coupled with a heterogeneous NoP. We show that mapping the early layers to the little chiplet bank and the subsequent layers to the big chiplet bank, achieve up to $2.8\times$ higher IMC utilization and up to $329\times$ improvement in energy-delay-area product compared to homogeneous chiplet IMC architectures. Experimental evaluation of the proposed big-little chiplet-based RRAM IMC architecture for ResNet-50 on ImageNet shows $18.4\times$ energy-efficiency improvement compared to SOTA chiplet-based architecture SIMBA.

Algorithm 11: Determining Big-Little Chiplet Configuration

```
1 Input: DNN structure, number of chiplets ( $N_C$ ), set of crossbars sizes for the
   little chiplets ( $\mathcal{X}_L$ ) and the big chiplets ( $\mathcal{X}_B$ ); set of number of tiles in the
   little chiplets ( $\mathcal{T}_L$ ) and the big chiplets ( $\mathcal{T}_B$ ); number of little chiplets ( $\mathcal{N}_L$ )
   and number of big chiplets ( $\mathcal{N}_B$ )
2 Output: Tile utilization for each configuration  $i$  ( $U_i$ )
3  $N_{cfg} \leftarrow$  number of configurations in the set containing all possible
   combinations of the elements in  $\mathcal{X}_B, \mathcal{X}_L, \mathcal{T}_L, \mathcal{T}_B, \mathcal{N}_L, \mathcal{N}_B$ 
4  $L \leftarrow$  number of DNN layers
5 for  $i = 1 : N_{cfg}$  do
6    $n_l =$  Number of little chiplets in Config- $i$ 
7    $n_b =$  Number of big chiplets in Config- $i$ 
8    $j \leftarrow 0$  // Number of layers already mapped
9    $U \leftarrow 0$  // Sum of utilization
10   $n_l^u \leftarrow 0$  // Number of little chiplets used
11  while  $n_l^u \leq n_l$  and  $j < L$  do
12     $j_t \leftarrow$  Number of tiles required for layer- $j$ 
13     $r_t^l \leftarrow$  Number of remaining tiles in the little chiplet
14    if  $j_t < r_t^l$  then
15      Map layer- $j$  to the little chiplet
16       $u_j \leftarrow$  Tiles utilization for layer- $j$  (Eq. 7.1)
17       $U = U + u_j; j = j + 1$ 
18    end
19    else
20       $n_l^u = n_l^u + 1$ 
21    end
22  end
23   $n_b^u \leftarrow 0$  // Number of big chiplets used
24  while  $n_b^u \leq n_b$  and  $j < L$  do
25     $j_t \leftarrow$  Number of tiles required for layer- $j$ 
26     $r_t^b \leftarrow$  Number of remaining tiles in the big chiplet
27    if  $j_t < r_t^b$  then
28      Map layer- $j$  to the big chiplet
29       $u_j \leftarrow$  Tiles utilization for layer- $j$  (Eq. 7.1)
30       $U = U + u_j; j = j + 1$ 
31    end
32    else
33       $n_b^u = n_b^u + 1$ 
34    end
35  end
36   $U_i = \frac{U}{L}$ 
37 end
```

CONCLUSION

In this dissertation, first, we presented an algorithm solution through a novel metric, model stability, to achieve reliable RRAM-based IMC acceleration of DNNs. The proposed metric, model stability, from the loss landscape helps shed light on accuracy under variations and model compression and guide an algorithmic solution that mitigates the loss. We utilize model stability to select the more stable model that can withstand the RRAM variations better. The proposed model stability-based model selection effectively tolerates device variations and achieves a post-mapping accuracy higher than that with 50% lower RRAM variations. Next, we show that pruning results in a less stable model, while quantization improves the model stability. Finally, we propose a model stability-based VAT method for compressed DNNs, which searches the most stable model under variations to achieve the best post-mapping accuracy, without knowing the exact amount of RRAM testing variations upfront. We show that the model-stability-based VAT method achieves up to 19%, 21%, and 11% improvement in accuracy for compressed DNNs on CIFAR-10, CIFAR-100, and SVHN datasets, respectively.

Next, we bridge the gap between the floating-point software accuracy and the post-mapping accuracy of RRAM-based IMC architectures (after applying model stability-based VAT methods). To achieve this, we propose a novel hybrid RRAM/SRAM IMC architecture that utilizes an RRAM-based IMC macro and a reconfigurable SRAM array and output stationary multiply-and-accumulate (MAC) macro. The hybrid IMC architecture utilizes an RRAM IMC macro with MLC cells, an SRAM macro, and

a programmable shifter. The output from the RRAM macro is compensated by the SRAM macro output to create an ensemble model and achieve bit-level compensation. The scale of compensation is controlled by using a programmable shifter for the SRAM macro output. Next, we develop a training framework to enable the hybrid IMC architecture that supports quantization, structured pruning, RRAM IMC-aware training, and different compensation scales through the programmable shifter. Finally, we design a test-chip using the 65nm SUNY process to demonstrate the efficacy of the proposed hybrid IMC architecture. Compared to the conventional VAT method, the proposed hybrid IMC architecture achieves up to 25% improvement in post-mapping accuracy. In addition, compared to state-of-the-art methods, the proposed hybrid IMC architecture provides a scalable solution that achieves up to 21.9%, 12.65%, and 6.52% improvement in post-mapping accuracy with minimal overhead for ResNet-20 on CIFAR-10, VGG-16 on CIFAR-10, and ResNet-18 on ImageNet, respectively.

Next, in this dissertation, we propose architectural optimizations to guide the optimal choice of the on-chip interconnect and generate an area and energy optimized IMC architecture. First, we provide quantitative insights into the effect of the choice of the interconnect in IMC architectures. We illustrate that the P2P-based interconnect is incapable of handling a high volume of on-chip data movement for DNNs. Furthermore, we propose to use analytical models of NoC to evaluate end-to-end communication latency of any given DNN and determine the optimal choice of interconnect for any given DNN. We demonstrate that the optimal choice of interconnect in the IMC architecture results in up to $6\times$ improvement in energy-delay-area product for VGG-19 inference compared to the state-of-the-art ReRAM-based IMC architectures. Second, we propose an interconnect-aware area and energy optimization for efficient IMC acceleration of DNNs. To achieve this, we propose an area-aware optimization

technique that improves the PE array utilization. This is achieved by generating a heterogeneous tile-based IMC architecture that consists of tiles of different sizes, i.e. with different numbers of PEs where each PE is of the same size. Further, we minimize the communication energy across a large number of tiles using an NoC architecture with optimized tile-to-router mapping and scheduling. Experimental evaluations show up to 62% improvement in PE utilization, 78% reduction in area, and 78% lower energy-area product for a wide range of modern DNNs such as DenseNet (100,24), and ResNet-152.

Furthermore, this dissertation proposes a novel chiplet-based IMC benchmarking simulator, SIAM, and an heterogeneous chiplet IMC architecture to address the limitations of a monolithic DNN accelerator. SIAM bridges the gap between architectural optimizations and benchmarking of such optimizations for IMC-based architectures. SIAM integrates device, circuits, architecture, NoC, NoP, and DRAM access estimation under a single roof for design space exploration. To the best of our knowledge, this is the first open-sourced architectural exploration tool for chiplet-based IMC architectures. SIAM has a flexible architecture to support multiple DNN to IMC chiplet and crossbar partition and mapping schemes, thus generating different types of chiplet-based IMC architectures. Furthermore, SIAM has a low simulation time ranging from a few minutes to a few hours (4.5Hrs for VGG-16 with 138M parameters) to support the fast design and benchmarking exploration. We calibrate SIAM against a published silicon result, SIMBA, a real-world chip from Nvidia. We demonstrate SIAM’s capabilities by conducting experiments on state-of-the-art DNNs such as ResNet-110 for CIFAR-10, VGG-19 for CIFAR-100, and ResNet-50 and VGG-16 for ImageNet datasets.

Finally, this dissertation proposes a heterogeneous big-little chiplet-based IMC

architecture driven by the non-uniform nature of DNN layers. The proposed architecture utilizes a heterogeneous IMC architecture with big-little chiplets and a hybrid network-on-package (NoP) to optimize the utilization, interconnect bandwidth, and energy efficiency. We also develop a custom methodology to map the model onto the big-little architecture such that the early layers in the DNN are mapped to the little chiplets with higher NoP bandwidth and the subsequent layers are mapped to the big chiplets with lower NoP bandwidth. Furthermore, we achieve a scalable solution by incorporating a DRAM into each chiplet to support a wide range of DNNs beyond the area limit. Experimental evaluation of the proposed big-little chiplet-based RRAM IMC architecture for ResNet-50 on ImageNet shows $259\times$, $139\times$, and $48\times$ improvement in energy-efficiency at lower area compared to Nvidia V100 GPU, Nvidia T4 GPU, and SIMBA architecture, respectively.

In summary, this dissertation comprehensively discusses techniques to achieve scalable high performance and energy-efficient in-memory acceleration of DNNs through a hardware-software co-design approach. We propose novel methods from both an algorithm and architecture end to achieve reliable RRAM-based IMC acceleration. Furthermore, the dissertation details architectural optimizations for energy-efficient IMC architecture design. Finally, the dissertation proposed a chiplet-based IMC benchmarking simulator and a heterogeneous big-little chiplet-based IMC architecture for scalable DNN acceleration.

REFERENCES

- Agarwal, N., T. Krishna, L.-S. Peh and N. K. Jha, “GARNET: A Detailed On-Chip Network Model Inside a Full-System Simulator”, in “2009 IEEE ISPASS”, pp. 33–42 (IEEE, 2009).
- AnySilicon, “Fabrication Cost”, <https://anysilicon.com/die-per-wafer-formula-free-calculators/> Accessed 29 Mar. 2021 (2011).
- Beck, N., S. White, M. Paraschou and S. Naffziger, “‘Zeppelin’: An SoC for Multichip Architectures”, in “2018 IEEE ISSCC”, pp. 40–42 (IEEE, 2018).
- Bhat, G., R. Deb, V. V. Chaurasia, H. Shill and U. Y. Ogras, “Online Human Activity Recognition using Low-power Wearable Devices”, in “2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)”, pp. 1–8 (2018).
- Bojarski, M., P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel and U. Muller, “Explaining how a deep neural network trained with end-to-end learning steers a car”, arXiv preprint arXiv:1704.07911 (2017).
- Chakraborty, I., M. F. Ali, D. E. Kim, A. Ankit and K. Roy, “GENIEx: A Generalized Approach to Emulating Non-Ideality in Memristive Xbars using Neural Networks”, in “DAC”, (IEEE, 2020).
- Charan, G., J. Hazra, K. Beckmann, X. Du, G. Krishnan, R. V. Joshi, N. C. Cady and Y. Cao, “Accurate Inference with Inaccurate RRAM Devices: Statistical Data, Model Transfer, and On-line Adaptation”, in “DAC”, (IEEE, 2020a).
- Charan, G., A. Mohanty, X. Du, G. Krishnan, R. V. Joshi and Y. Cao, “Accurate inference with inaccurate rram devices: A joint algorithm-design solution”, *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* **6**, 1, 27–35 (2020b).
- Chen, C.-Y., H.-C. Shih, C.-W. Wu, C.-H. Lin, P.-F. Chiu, S.-S. Sheu and F. T. Chen, “RRAM Defect Modeling and Failure Analysis Based on March Test and a Novel Squeeze-search Scheme”, *IEEE Transactions on Computers* (2014).
- Chen, P.-Y., X. Peng and S. Yu, “NeuroSim: A Circuit-level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **37**, 12, 3067–3080 (2018).
- Chen, Y.-H., T.-J. Yang, J. Emer and V. Sze, “Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **9**, 2, 292–308 (2019).

- CHIPS Alliance (INTEL), “EMIB PHY RTL”, <https://github.com/chipsalliance/aib-phy-hardware>, [Online; Accessed 20-April-2022] (2021).
- Choi, J., Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan and K. Gopalakrishnan, “Pact: Parameterized clipping activation for quantized neural networks”, arXiv preprint arXiv:1805.06085 (2018).
- Deng, L., G. Hinton and B. Kingsbury, “New Types of Deep Neural Network Learning for Speech Recognition and Related Applications: An Overview”, in “2013 IEEE international conference on acoustics, speech and signal processing”, pp. 8599–8603 (IEEE, 2013).
- Dong, X., C. Xu, Y. Xie and N. P. Jouppi, “Nvsim: A Circuit-level Performance, Energy, and Area Model for Emerging Nonvolatile Memory”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **31**, 7, 994–1007 (2012).
- Du, X., G. Krishnan, A. Mohanty, Z. Li, G. Charan and Y. Cao, “Towards efficient neural networks on-a-chip: Joint hardware-algorithm approaches”, in “2019 China Semiconductor Technology International Conference (CSTIC)”, pp. 1–5 (IEEE, 2019).
- Du, X., Z. Li, J.-s. Seo, F. Liu and Y. Cao, “Noise-Based Selection of Robust Inherited Model for Accurate Continual Learning”, in “CVPR Workshops”, (2020).
- Ghose, S., A. G. Yaglikçi, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal *et al.*, “What your dram power models are not telling you: Lessons from a detailed experimental study”, *Proceedings of the ACM on Measurement and Analysis of Computing Systems* **2**, 3, 1–41 (2018).
- Greenhill, D., R. Ho, D. Lewis, H. Schmit, K. H. Chan, A. Tong, S. Atsatt, D. How, P. McElheny, K. Duwel *et al.*, “3.3 a 14nm 1ghz fpga with 2.5 d transceiver integration”, in “2017 IEEE International Solid-State Circuits Conference (ISSCC)”, pp. 54–55 (IEEE, 2017).
- He, K., X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition”, in “Proceedings of the IEEE conference on computer vision and pattern recognition”, pp. 770–778 (2016).
- He, Z., B. Gong and D. Fan, “Optimize Deep Convolutional Neural Network with Ternarized Weights and High Accuracy”, in “WACV”, (IEEE, 2019a).
- He, Z. *et al.*, “Noise Injection Adaption: End-to-end Reram Crossbar Non-Ideal Effect Adaption for Neural Network Mapping”, in “DAC”, (2019b).
- Hochreiter, S. and J. Schmidhuber, “Flat minima”, *Neural computation* **9**, 1, 1–42 (1997).

- Horowitz, M., “1.1 computing’s energy problem (and what we can do about it)”, in “2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)”, pp. 10–14 (IEEE, 2014).
- Howard, A., M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, “Searching for mobilenetv3”, in “Proceedings of the IEEE/CVF International Conference on Computer Vision”, pp. 1314–1324 (2019).
- Huang, G., Z. Liu, L. Van Der Maaten and K. Q. Weinberger, “Densely Connected Convolutional Networks”, in “IEEE Conference on Computer Vision and Pattern Recognition”, pp. 4700–4708 (2017).
- Hwang, R., T. Kim, Y. Kwon and M. Rhu, “Centaur: A Chiplet-based, Hybrid Sparse-Dense Accelerator for Personalized Recommendations”, in “2020 ACM/IEEE 47th Annual ISCA”, pp. 968–981 (IEEE, 2020).
- Imani, M., S. Gupta, Y. Kim and T. Rosing, “Floatpim: In-memory Acceleration of Deep Neural Network Training with High Precision”, in “Proceedings of the 46th Annual ISCA”, pp. 802–815 (2019).
- Jeffers, J., J. Reinders and A. Sodani, *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition* (Morgan Kaufmann, 2016).
- Jiang, N., D. U. Becker, G. Micheliogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim and W. J. Dally, “A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator”, in “IEEE ISPASS”, (2013).
- Joshi, V., M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian and E. Eleftheriou, “Accurate Deep Neural Network Inference using Computational Phase-Change Memory”, *Nature communications* (2020).
- Jouppi, N. P., C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, “In-datacenter performance analysis of a tensor processing unit”, in “Proceedings of the 44th annual international symposium on computer architecture”, pp. 1–12 (2017).
- Kannan, A., N. E. Jerger and G. H. Loh, “Enabling Interposer-based Disintegration of Multi-core Processors”, in “2015 48th Annual IEEE/ACM MICRO”, pp. 546–558 (IEEE, 2015).
- Keskar, N. S., J. Nocedal, P. T. P. Tang, D. Mudigere and M. Smelyanskiy, “On large-batch training for deep learning: Generalization gap and sharp minima”, in “5th International Conference on Learning Representations, ICLR 2017”, (2019).

- Khwa, W.-S., J.-J. Chen, J.-F. Li, X. Si, E.-Y. Yang, X. Sun, R. Liu, P.-Y. Chen, Q. Li, S. Yu *et al.*, “A 65nm 4kb algorithm-dependent computing-in-memory sram unit-macro with 2.3 ns and 55.8 tops/w fully parallel product-sum operation for binary dnn edge processors”, in “2018 IEEE International Solid-State Circuits Conference-(ISSCC)”, pp. 496–498 (IEEE, 2018).
- Kim, G., Jinwoo Murali, H. Park, E. Qin, H. Kwon, V. C. K. Chekuri, N. M. Rahman, N. Dasari, A. Singh, M. Lee *et al.*, “Architecture, Chip, and Package Codesign Flow for Interposer-Based 2.5D Chiplet Integration Enabling Heterogeneous IP Reuse”, IEEE TVLSI (2020).
- Kim, Y., W. Yang and O. Mutlu, “RAMULATOR: A Fast and Extensible DRAM Simulator”, IEEE Computer architecture letters **15**, 1, 45–49 (2015).
- Krishnan, G., J. Hazra, M. Liehr, X. Du, K. Beckmann, R. V. Joshi, N. C. Cady and Y. Cao, “Design limits of in-memory computing: Beyond the crossbar”, in “2021 5th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)”, pp. 1–3 (IEEE, 2021a).
- Krishnan, G., Y. Ma and Y. Cao, “Small-world-based structural pruning for efficient fpga inference of deep neural networks”, in “2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)”, pp. 1–5 (IEEE, 2020a).
- Krishnan, G., S. K. Mandal, C. Chakrabarti, J.-s. Seo, U. Y. Ogras and Y. Cao, “Interconnect-aware Area and Energy Optimization for In-Memory Acceleration of DNNs”, IEEE Design & Test (2020b).
- Krishnan, G., S. K. Mandal, C. Chakrabarti, J.-S. Seo, U. Y. Ogras and Y. Cao, “Impact of on-chip interconnect on in-memory acceleration of deep neural networks”, ACM Journal on Emerging Technologies in Computing Systems (JETC) **18**, 2, 1–22 (2021b).
- Krishnan, G., S. K. Mandal, C. Chakrabarti, J.-s. Seo, U. Y. Ogras and Y. Cao, “Interconnect-Centric Benchmarking of In-Memory Acceleration for DNNs”, in “2021 China Semiconductor Technology International Conference (CSTIC)”, pp. 1–4 (IEEE, 2021c).
- Krishnan, G., S. K. Mandal, C. Chakrabarti, J.-S. Seo, U. Y. Ogras and Y. Cao, “System-level benchmarking of chiplet-based imc architectures for deep neural network acceleration”, in “2021 IEEE 14th International Conference on ASIC (ASICON)”, pp. 1–4 (IEEE, 2021d).
- Krishnan, G., S. K. Mandal, M. Pannala, C. Chakrabarti, J.-S. Seo, U. Y. Ogras and Y. Cao, “SIAM: Chiplet-based Scalable In-Memory Acceleration with Mesh for Deep Neural Networks”, ACM Transactions on Embedded Computing Systems (TECS) **20**, 5s, 1–24 (2021e).

- Krishnan, G., J. Sun, J. Hazra, X. Du, M. Liehr, Z. Li, K. Beckmann, R. V. Joshi, N. C. Cady and Y. Cao, “Robust RRAM-based In-Memory Computing in Light of Model Stability”, in “2021 IEEE International Reliability Physics Symposium (IRPS)”, pp. 1–5 (IEEE, 2021f).
- Krishnan, G., Z. Wang, I. Yeo, L. Yang, J. Meng, M. Liehr, R. V. Joshi, N. C. Cady, D. Fan, J.-s. Seo *et al.*, “Hybrid rram/sram in-memory computing for robust dnn acceleration”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2022a).
- Krishnan, G., L. Yang, J. Sun, J. Hazra, X. Du, M. Liehr, Z. Li, K. Beckmann, R. Joshi, N. C. Cady *et al.*, “Exploring model stability of deep neural networks for reliable rram-based in-memory acceleration”, IEEE Transactions on Computers (2022b).
- Krizhevsky, A., I. Sutskever and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, in “Advances in Neural Information Processing Systems”, pp. 1097–1105 (2012).
- Kwon, H., A. Samajdar and T. Krishna, “Rethinking Nocs for Spatial Neural Network Accelerators”, in “2017 Eleventh IEEE/ACM Intl. Symp. on NOCS”, pp. 1–8 (2017).
- Kwon, H., A. Samajdar and T. Krishna, “Maeri: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects”, in “ACM SIGPLAN Notices”, vol. 53, pp. 461–475 (2018).
- Kwon, L., Hyoukjun Lai, M. Pellauer, T. Krishna, Y.-H. Chen and V. Chandra, “Heterogeneous Dataflow Accelerators for Multi-DNN Workloads”, in “IEEE HPCA”, (2021).
- LeCun, Y., L. Bottou, Y. Bengio, P. Haffner *et al.*, “Gradient-based learning applied to document recognition”, Proc. of the IEEE **86**, 11, 2278–2324 (1998).
- Li, H., Z. Xu, G. Taylor, C. Studer and T. Goldstein, “Visualizing the Loss Landscape of Neural Nets”, in “Advances in Neural Information Processing Systems”, (2018).
- Li, Y., A. Louri and A. Karanth, “Scaling deep-learning inference with chiplet-based architecture and photonic interconnects”, in “2021 58th ACM/IEEE Design Automation Conference (DAC)”, pp. 931–936 (IEEE, 2021).
- Li, Y., A. Louri and A. Karanth, “Spacx: Silicon photonics-based scalable chiplet accelerator for dnn inference”, in “Proc. IEEE Int. Symp. High-Perform. Comput. Archit.”, pp. 1–13 (2022a).

- Li, Y., K. Wang, H. Zheng, A. Louri and A. Karanth, “Ascend: A scalable and energy-efficient deep neural network accelerator with photonic interconnects”, *IEEE Transactions on Circuits and Systems I: Regular Papers* (2022b).
- Liehr, M., J. Hazra, K. Beckmann, S. Rafiq and N. Cady, “Impact of Switching Variability of 65nm CMOS Integrated Hafnium Dioxide-based ReRAM Devices on Distinct Level Operations”, in “IIRW”, (IEEE, 2020).
- Lin, M., Q. Chen and S. Yan, “Network in Network”, arXiv preprint arXiv:1312.4400 (2013).
- Lin, M.-S., T.-C. Huang, C.-C. Tsai, K.-H. Tam, K. C.-H. Hsieh, C.-F. Chen, W.-H. Huang, C.-W. Hu, Y.-C. Chen, S. K. Goel *et al.*, “A 7-nm 4-ghz arm¹-core-based cowos¹ chiplet design for high-performance computing”, *IEEE Journal of Solid-State Circuits* **55**, 4, 956–966 (2020).
- Litjens, G., T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken and C. I. Sánchez, “A Survey on Deep Learning in Medical Image Analysis”, *Medical image analysis* **42**, 60–88 (2017).
- Liu, C., J. Botimer and Z. Zhang, “A 256gb/s/mm-shoreline aib-compatible 16nm finfet cmos chiplet for 2.5 d integration with stratix 10 fpga on emib and tiling on silicon interposer”, in “2021 IEEE Custom Integrated Circuits Conference (CICC)”, pp. 1–2 (IEEE, 2021).
- Long, Y., X. She and S. Mukhopadhyay, “Design of Reliable DNN Accelerator with Un-Reliable ReRAM”, in “DATE”, (IEEE, 2019).
- Ma, C., Y. Sun, W. Qian, Z. Meng, R. Yang and L. Jiang, “Go Unary: A Novel Synapse Coding and Mapping Scheme for Reliable Reram-based Neuromorphic Computing”, in “DATE”, (IEEE, 2020).
- Ma, Y., Y. Cao, S. Vrudhula and J. Seo, “Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks”, in “ISFPGA”, (2017a).
- Ma, Y., Y. Cao, S. Vrudhula and J.-s. Seo, “Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks”, in “Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays”, *FPGA '17*, pp. 45–54 (ACM, New York, NY, USA, 2017b), URL <http://doi.acm.org/10.1145/3020078.3021736>.
- Ma, Y., Y. Cao, S. Vrudhula and J.-s. Seo, “Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA”, *IEEE Trans. on VLSI Systems* **26**, 7 (2018).

- Mahajan, R., Ravi Sankman, N. Patel, D.-W. Kim, K. Aygun, Z. Qian, Y. Mekonnen, I. Salama, S. Sharan, D. Iyengar *et al.*, “Embedded Multi-Die Interconnect Bridge (EMIB)—A High Density, High Bandwidth Packaging Interconnect”, in “IEEE ECTC”, (2016).
- Mandal, S. K., R. Ayoub, M. Kishinevsky and U. Y. Ogras, “Analytical Performance Models for NoCs with Multiple Priority Traffic Classes”, *ACM Transactions on Embedded Computing Systems (TECS)* **18**, 5s, 1–21 (2019).
- Mandal, S. K., G. Krishnan, C. Chakrabarti, J.-S. Seo, Y. Cao and U. Y. Ogras, “A Latency-Optimized Reconfigurable NoC for In-Memory Acceleration of DNNs”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **10**, 3, 362–375 (2020).
- Mandal, S. K., G. Krishnan, A. A. Goksoy, G. R. Nair, Y. Cao and U. Y. Ogras, “COIN: Communication-Aware In-Memory Acceleration for Graph Convolutional Networks”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2022).
- Mao, M., X. Peng, R. Liu, J. Li, S. Yu and C. Chakrabarti, “MAX2: An ReRAM-based Neural Network Accelerator that Maximizes Data Reuse and Area Utilization”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2019).
- MICRON, “Datasheet for DDR3 Model”, https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr3/2gb_ddr3l-rs.pdf?rev=f43686e89394458caff410138d9d2152 Accessed 29 Mar. 2021 (2011).
- MICRON, “Datasheet for DDR4 Model”, https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/4gb_ddr4_dram_2e0d.pdf Accessed 29 Mar. 2021 (2014).
- Mirza-Aghatabar, M., S. Koochi, S. Hessabi and M. Pedram, “An Empirical Investigation of Mesh and Torus NoC Topologies under Different Routing algorithms and Traffic Models”, in “10th Euromicro conference on digital system design architectures, methods and tools (DSD 2007)”, pp. 19–26 (2007).
- Mudge, T., “Power: A First-class Architectural Design Constraint”, *Computer* **34**, 4, 52–58 (2001).
- Nabavinejad, S. M., M. Baharloo, K.-C. Chen, M. Palesi, T. Kogel and M. Ebrahimi, “An Overview of Efficient Interconnection Networks for Deep Neural Network Accelerators”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **10**, 3, 268–282 (2020).

- Ogras, U. Y., P. Bogdan and R. Marculescu, “An Analytical Approach for Network-on-Chip Performance Analysis”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **29**, 12, 2001–2013 (2010).
- Pal, J., Saptadeep Liu, I. Alam, N. Cebry, H. Suhail, S. Bu, S. S. Iyer, S. Pamarti, R. Kumar and P. Gupta, “Designing a 2048-Chiplet, 14336-Core Waferscale Processor”, in “ACM/IEEE DAC”, (2021).
- Peng, X., S. Huang, Y. Luo, X. Sun and S. Yu, “DNN+ NeuroSim: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators with Versatile Device Technologies”, in “IEDM”, (IEEE, 2019a).
- Peng, X., M. Kim, X. Sun, S. Yin, T. Rakshit, R. M. Hatcher, J. A. Kittl, J.-s. Seo and S. Yu, “Inference Engine Benchmarking Across Technological Platforms from CMOS to RRAM”, in “Proceedings of the International Symposium on Memory Systems”, pp. 471–479 (2019b).
- Poulton, J. W., W. J. Dally, X. Chen, J. G. Eyles, T. H. Greer, S. G. Tell and C. T. Gray, “A 0.54 pj/b 20gb/s ground-referenced single-ended short-haul serial link in 28nm cmos for advanced packaging applications”, in “2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers”, pp. 404–405 (IEEE, 2013).
- Qiao, X., X. Cao, H. Yang, L. Song and H. Li, “Atomlayer: a universal reram-based cnn accelerator with atomic layer computation”, in “Proceedings of the 55th Annual Design Automation Conference”, pp. 1–6 (2018).
- Shafiee, A., A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams and V. Srikumar, “ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars”, *ACM/IEEE ISCA* (2016).
- Shao, Y. S., J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, “Simba: Scaling deep-learning inference with multi-chip-module-based architecture”, in “Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture”, pp. 14–27 (2019).
- Simonyan, K. and A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, *arXiv preprint arXiv:1409.1556* (2014).
- Sinha, S., G. Yeric, V. Chandra, B. Cline and Y. Cao, “Exploring Sub-20nm FinFET Design with Predictive Technology Models”, in “DAC 2012”, pp. 283–288 (IEEE, 2012).
- Song, L., X. Qian, H. Li and Y. Chen, “Pipelayer: A Pipelined Reram-Based Accelerator for Deep Learning”, in “HPCA”, (IEEE, 2017).

- Su, M., B. Black, Y.-H. Hsiao, C.-L. Changchien, C.-C. Lee and H.-J. Chang, “2.5 d ic micro-bump materials characterization and imcs evolution under reliability stress conditions”, in “2016 IEEE 66th Electronic Components and Technology Conference (ECTC)”, pp. 322–328 (IEEE, 2016).
- Sun, Y., C. Ma, Z. Li, Y. Zhao, J. Jiang, W. Qian, R. Yang, Z. He and L. Jiang, “Unary Coding and Variation-Aware Optimal Mapping Scheme for Reliable ReRAM-based Neuromorphic Computing”, IEEE TCAD (2021).
- Tam, S. M., H. Muljono, M. Huang, S. Iyer, K. Royneogi, N. Satti, R. Qureshi, W. Chen, T. Wang and H. Hsieh, “SkyLake-SP: A 14nm 28-Core Xeon® Processor”, in “2018 IEEE ISSCC”, pp. 34–36 (2018).
- Tan, H., Zhanhong Cai, R. Dong and K. Ma, “NN-Baton: DNN Workload Orchestration and Chiplet Granularity Exploration for Multichip Accelerators”, in “ACM/IEEE ISCA”, (2021).
- Turner, W. J., J. W. Poulton, J. M. Wilson, X. Chen, S. G. Tell, M. Fojtik, T. H. Greer, B. Zimmer, S. Song, N. Nedovic *et al.*, “Ground-referenced signaling for intra-chip and short-reach chip-to-chip interconnects”, in “2018 IEEE Custom Integrated Circuits Conference (CICC)”, pp. 1–8 (IEEE, 2018).
- Valavi, H. *et al.*, “A 64-Tile 2.4-Mb In-Memory-Computing CNN Accelerator Employing Charge-Domain Compute”, IEEE JSSC (2019).
- Venkataramani, S., A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey *et al.*, “Scaleddeep: A Scalable Compute Architecture for Learning and Evaluating Deep Networks”, ACM SIGARCH Computer Architecture News **45**, 2 (2017).
- Vivet, E., Pascal Guthmuller, Y. Thonnart, G. Pillonnet, C. Fuguet, I. Miro-Panades, G. Moritz, J. Durupt, C. Bernard, D. Varreau *et al.*, “IntAct: A 96-core Processor with Six Chiplets 3D-stacked on an Active Interposer with Distributed Interconnects and Integrated Power Management”, IEEE JSSC (2020).
- Wang, M., Y. Wang, C. Liu and L. Zhang, “Network-on-Interposer Design for Agile Neural-Network Processor Chip Customization”, in “ACM/IEEE DAC”, (2021).
- Xie, S., A. Kirillov, R. Girshick and K. He, “Exploring Randomly Wired Neural Networks for Image Recognition”, in “Proceedings of the IEEE International Conference on Computer Vision”, pp. 1284–1293 (2019).
- Yang, L., Z. He and D. Fan, “Harmonious coexistence of structured weight pruning and ternarization for deep neural networks”, in “Proceedings of the AAAI Conference on Artificial Intelligence”, vol. 34, pp. 6623–6630 (2020).

- Yang, X., S. Belakaria, B. K. Joardar, H. Yang, J. R. Doppa, P. P. Pande, K. Chakrabarty and H. H. Li, “Multi-Objective Optimization of ReRAM Crossbars for Robust DNN Inferencing under Stochastic Noise”, in “ICCAD”, (IEEE/ACM, 2021).
- Zheng, H., K. Wang and A. Louri, “A Versatile and Flexible Chiplet-based System Design for Heterogeneous Manycore Architectures”, in “ACM/IEEE DAC”, (2020).
- Zhou, S., Y. Wu, Z. Ni, X. Zhou, H. Wen and Y. Zou, “Dorefa-net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients”, arXiv preprint arXiv:1606.06160 (2016).
- Zhu, Z., H. Sun, K. Qiu, L. Xia, G. Krishnan, G. Dai, D. Niu, X. Chen, X. S. Hu, Y. Cao *et al.*, “Mnsim 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems”, in “Proceedings of the 2020 on Great Lakes Symposium on VLSI”, pp. 83–88 (2020).
- Zimmer, B., R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, “A 0.11 pj/op, 0.32-128 tops, scalable multi-chip-module-based deep neural network accelerator with ground-reference signaling in 16nm”, in “2019 Symposium on VLSI Circuits”, pp. C300–C301 (IEEE, 2019).
- Zoph, B. and Q. V. Le, “Neural Architecture Search with Reinforcement Learning”, arXiv preprint arXiv:1611.01578 (2016).
- Zoph, B., V. Vasudevan, J. Shlens and Q. V. Le, “Learning transferable architectures for scalable image recognition”, in “Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition”, pp. 8697–8710 (2018).