DOMAIN-AWARE GENETIC ALGORITHMS FOR HARDWARE AND MAPPING OPTIMIZATION FOR EFFICIENT DNN ACCELERATION

A Dissertation
Presented to
The Academic Faculty

By

Sheng-Chun Kao

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Georgia Institute of Technology
Department of Electrical and Compute Engineering

Georgia Institute of Technology

August 2022

DOMAIN-AWARE GENETIC ALGORITHMS FOR HARDWARE AND MAPPING OPTIMIZATION FOR EFFICIENT DNN ACCELERATION

Thesis committee:

Dr. Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Callie Hao
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Vivek Sarkar
School of Computer Science
Georgia Institute of Technology

Dr. Alexey Tumanov
School of Computer Science
Georgia Institute of Technology

Dr. Angshuman Parashar
Research Scientist
NVIDIA

Dr. Suvinay Subramanian
Senior Software Engineer
Google

Date approved: May 16, 2022

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Professor Tushar Krishna for the guidance and support during my PhD journey in the past four years. I learn many technical skills and also the great characteristics of being a researcher: commitment, passion, curiosity, endeavors, and so on. I am truly grateful to have him as my advisor and mentor. In my PhD journey, there are ups and downs. He is the one I can always count on, who is giving me endless support, keeping my chin up, and always cheering for me. I also learn many life advice from him. I sincerely appreciate him for being a great role model I could learn from.

I would like to pay special thanks to my mentor and collaborator throughout my PhD journey, Dr. Suvinay Subramanian. I learn many technical insights and life advice from you. My PhD journey could not be completed without your guidance.

I would like to pay special thanks to my collaborators in different exciting projects, Dr. Michael Pellauer, Dr. Angshuman Parashar, Dr. Po-An Tsai, Dr. Amir Yazdanbakhsh, Gaurav Agrawal, Dr. Shivani Agrawal, and Dr. Arun Ramamurthy. I appreciate for invaluable learning opportunities and countless hours for discussing research ideas, which led to major contributions of this thesis.

I would like to thank Dr. Hyoujhun Kwon for mentoring me in my first year of PhD and helps me understand MAESTRO thoroughly which opens up many research opportunities in my PhD. I would like to thanks Yannan (Nellie) Wu for all the help and discussion about Timeloop via emails. I thank all the members in Synergy Lab for collaborating and helping out for different tasks.

I would like to thank Dr. Tushar Krishna , Dr. Vivek Sarkar, Dr. Alexey Tumanov, Dr. Callie Hao, Dr. Angshuman Parashar, and Dr. Suvinay Subramanian for taking out time to serve on my thesis committee.

Lastly, I would like to thank my parents and family for their unconditional and endless support.

TABLE OF CONTENTS

Acknowledgments iii

List of Tables xii

List of Figures xv

Summary xxiii

Chapter 1: Introduction 1

1.1 Challenges 2

1.1.1 Linkage between HW Accelerator Flexibility and Map Space 2

1.1.2 Efficient Mapper 3

1.1.3 Scalability of MSE 3

1.1.4 Mapping and HW Co-exploration 4

1.2 Thesis Contributions 5

1.2.1 Flexion: Accelerator Flexibility and Map Space 5

1.2.2 Gamma: Efficient DNN Mapper 5

1.2.3 Demystifying MSE: Techniques for Improving the Scalability of MSE 6

1.2.4 DiGamma: Mapping and HW Co-exploration 6

1.2.5 MAGMA: Mapping across Multi-Accelerators 7

1.2.6 ConfuciuX: Automated DNN Accelerator Design 8
1.2.7 FLAT: Dataflow Technique for Mapping Attention Operation 8
1.3 Thesis Statement 9
1.4 Thesis Overview 9
Chapter 2: Background 12
2.1 DNN Workloads 12
2.2 Accelerator Hardware Configuration 13
2.3 Dataflow and Mapping 14
2.4 The Impact of Mapping 15
2.4.1 Optimal Mappings 15
2.4.2 Efficient Mappings Example 15
2.4.3 Performance-efficient v.s. Energy-efficient Mappings 18
2.4.4 Map Space Exploration 21
2.5 DNN Accelerator Cost Model 21
2.6 Optimization Methods 22
2.6.1 Heuristics-based 23
2.6.2 Feedback-based 23
2.6.3 Gradient-based 27
2.7 Summary 27
Chapter 3: Flexion: A Formalism of DNN Accelerator Flexibility and the Implication of Map Space 28
3.1 Axes of Accelerator Flexibility 29

3.1.1 Axes Taxonomy 30

3.1.2 Classes of Accelerator Flexibility 33

3.1.3 Benefits of Mapping Flexibility Per Axis 33

3.1.4 Hardware Support for Flexibility Per Axis 35

3.2 Formal Flexibility-Constrained Map Space Definition 37

3.2.1 Definitions of Map Space and Flexibility 37

3.2.2 Full and Partial Flexibility 39

3.3 Flexibility-Aware Design-Space Exploration 40

3.4 Evaluation I: Isolation Study on Flexibility Axes 41

3.4.1 Methodology 42

3.4.2 Isolation: Tile Flexibility 43

3.4.3 Isolation: Order Flexibility 47

3.4.4 Isolation: Parallelism Flexibility 47

3.4.5 Isolation: Shape Flexibility 49

3.5 Evaluation II: Accelerator Future-Proofing 52

3.6 Related Work 55

3.7 Summary 55

Chapter 4: Gamma: A GA-based Mapper 57

4.1 Challenges with Baseline Methods 58

4.2 Genetic Algorithms (GA) 58

4.3 Methodology 59

4.3.1 GAMMA Encoding scheme 59

4.3.2 Decoding Genomes into a Mapping 60

4.3.3 Algorithm Flow 63

4.3.4 Flow for Automated Mapping Search 64

4.4 Evaluation of GAMMA 65

4.4.1 Target Search Methods and Parameters. 66

4.4.2 S1: Fixed 2D Accel. 66

4.4.3 S2: Flexible 2D Accel. 69

4.4.4 S3: Scale-out Flexible 2D Accel. 70

4.4.5 Two-stage Optimization for Inter-layer 73

4.5 Related Works 75

4.5.1 Dataflow Design in DNN Accelerators 75

4.5.2 HW Mapping Space Search and Exploration 75

4.6 Summary 76

Chapter 5: Demystifying Map Space Exploration and the Improvement Techniques 77

5.1 Map Space Exploration (MSE) 78

5.1.1 Evaluation Method (Cost Model) 79

5.1.2 Representation 80

5.1.3 Exploration Method (Mapper) 80

5.1.4 Why MSE Matters 81

5.2 Quantitative MSE Analysis 82

5.2.1 Methodology 82

5.2.2 Size of Map Space 84

5.2.3 Understanding Mapper Sampling Efficiency 85

5.2.4 Understanding Mapper Search Operators 94

5.2.5 Understanding Sparse Accelerator Mappings 96

5.2.6 Lessons Learnt 98

5.3 Improving MSE 98

5.3.1 Warm-start 99

5.3.2 Sparsity-aware MSE 102

5.4 Related Works 104

5.5 Summary 105

Chapter 6: DiGamma: A HW-Mapping Co-exploration Mapper 107

6.1 Technical Approach 107

6.1.1 High-level Overview 107

6.1.2 HW-Mapping Co-optimization Framework 109

6.1.3 Encoding of Design Point 112

6.2 Optimization Algorithm 113

6.2.1 Leveraging Existing Algorithms 113

6.2.2 Background of Genetic Algorithm. 113

6.2.3 DiGamma: Domain-aware Genetic Algorithm 114

6.3 Evaluations 114

6.3.1 Setup 114

6.3.2 Comparisons with Baseline Optimization Algorithms 117

6.3.3 Comparisons with Baseline HW-Mapping Schemes 121

6.3.4 Explanation of Found Solutions 122

6.4 Related Work 122

6.5 Summary 123

Chapter 7: MAGMA: A GA-based Mapper for Multi-accelerator System 124

7.1 Optimization Framework (M3E) 125

7.1.1 Encoding 125

7.1.2 Optimization Algorithms Supported 126

7.1.3 Objective and Constraints 127

7.1.4 Building Blocks of M3E. 128

7.1.5 M3E Workflow 130

7.1.6 Search Space 131

7.2 Optimization Algorithm (MAGMA) 131

7.2.1 Why GA? 133

7.2.2 MAGMA Algorithm Details 133

7.2.3 Warm-Start of MAGMA 136

7.3 Evaluations 137

7.3.1 Methodology 137

7.3.2 Mapper Settings. 139

7.3.3 Latency-BW Characteristics of DNNs 140

7.3.4 Homogeneous Accelerators 141

7.3.5 Heterogeneous Accelerators 142

7.3.6 Flexible Accelerator 145

7.3.7 More about MAGMA Algorithm 148

7.4 Related Works 149

7.5 Summary 150

Chapter 8: ConfuciuX: A RL-based HW Resource Allocation Algorithm for DNN Accelerators 151

8.1 Methodology 152

8.1.1 DNN Model Deployment on Accelerators 153

8.1.2 RL Agent 154

8.1.3 Observation (State) 156

8.1.4 Action Space 156

8.1.5 Platform Constraint and Objective 157

8.1.6 Reward Function 159

8.1.7 Interactive Environment (Env) 160

8.1.8 Local fine-tuning using GA 160

8.2 Evaluations of ConfuciuX 162

8.2.1 Methodology 162

8.2.2 Per-layer study for LS deployment 164

8.2.3 LP Deployment 165

8.2.4 Dataflow-HW co-automation 171

8.2.5 Benefit of Two-stage Optimization 173

8.2.6 LP deployment at compile time 176

8.2.7 Policy Network Exploration 176

8.2.8 Summary 177

8.3 Related Works 177

8.4 Summary 179

Chapter 9: Conclusions and Future Works 180

9.1 Summary of Contributions 180

9.1.1 Formalism of DNN Accelerator Flexibility and the Implied Map Space 180

9.1.2 Deep Analysis and Improvement of MSE for DNN Workloads . . . 181

9.1.3 Search Algorithm/ Mapper 182

9.2 Future Work 183

9.2.1 MSE for Both Intra- and Inter- Layers 183

9.2.2 MSE for Heterogeneous System 184

9.2.3 MSE for new DNN Operation 184

9.2.4 Sparsity as new Mapping Axis 185

9.2.5 Pre-trained Mapper and Transfer Learning 185

9.2.6 Mapping and Neural Architecture Co-Exploration 186

Appendices 187

Appendix A: FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks 188

References 215

Vita 242

LIST OF TABLES

3.1 Hardware resources and baseline flexibility TOPS. 41

3.2 Area cost of accelerators with different flexibility. 41

3.3 Definition of Map Space and Flexibility. 45

4.1 Terminology in Genetic Algorithm (GA). 59

4.2 The HW resources in different platforms. 66

4.3 Three target systems (Accel's infrastructures). 66

4.4 Supported optimization algorithms in M3E. 68

4.5 End-to-end performance and energy on S3 for a suite of DNN models using fixed mappings versus GAMMA. Bold means lowest values. 72

4.6 Two stage optimization for inter-layer parallelism on ResNet-18 * and VGG16 † for a multi-accelerator (S3) pipelined deployment. In the 1st state, we optimize for latency and identify the bottleneck layer (highlighted in bold), which determines the pipeline latency. In the 2nd stage, we optimize for energy (or power) by allowing the latency of other layers to increase, while staying less than the pipeline latency. 73

5.1 The description of the relevant workloads and accelerator configurations used across evaluations. 84

5.2 MSE for workload with weight sparsity. In each columns, the blue cell shows the performance of the optimized mapping for the sparse workload; the rest of the cells shows the performance of the same mapping tested with the workload with different sparsity. We highlight the best-performing cell of each row by green text. We can observe that the blue cells overlap with green texts, indicating that different workload with different sparsity levels do require different mapping to optimize the performance. 87

5.3 The optimized EDP performance of inner and outer product style mapping on sparse-dense GEMM workloads in Bert-large model [69]. The workload density indicates the density of the sparse matrix. Bert-large KQV: the key/ query/ value projection operations. Bert-large Attn: the attention operation, Bert-large FC: the FC operations at the end of attention blocks. 88

5.4 Comparisons of sparsity-aware technique and static-density heuristic when tackling the activation sparsity. The static-density heuristic searches mapping for a fixed density level (1.0, 0.5, or 0.1), marked as blue cells. Sparsity-aware technique searches the mapping scored with its performance across 5 density level (1.0, 0.8, 0.5, 0.2, and 0.1). We highlight the best-performing one in each row with green text. It indicates that sparsity-aware technique can find mapping with comparable performance to the static-density ones across a range of sparsity (density 1.0 - 0.05). 89

7.1 Terminology used in MAGMA Algorithm. 133

7.2 Accelerators configurations/ settings of the experiments. 136

7.3 Supported optimization algorithms in M3E. 137

7.4 The performance of warm-start on (a) Mix, S4, BW=1. (b) The averaged performance across different tasks and different accelerator (S1-S6) under BW=1. All the values are normalized by the values of Trf-100-ep of each columns. Raw (highlighted in orange) is the throughput without warm-start. Trf-0-ep (highlighted in green) is warm-start and before further optimization. Trf-1-ep is warm-start with one epoch of optimization, and likewise for Trf-30-ep. Trf-100-ep (highlighted in blue) represents a full optimization process. 147

8.1 The level values of action pair. 158

8.2 Platform constraint settings. 162

8.3 Converged solution of LP deployment. 166

8.4 Converged solutions after 5000 epochs for various optimization methods across four platforms with different constraints. DNN=MobileNet-V2, Dataflow=NVDLA-style, Deployment=LP 167

8.5 Comparison of search-time and converged solutions across state-of-the-art RL techniques. 169

8.6 Dataflow and Hardware co-automation. 174

8.7 Two-stage optimization of ConfuciuX. 175

8.8 Resource assignments for LP deployment at compile time of ConfuciuX. . . 176

8.9 Different configurations of the policy network. 177

A.1 Buffer requirement for tiling granularity. M: batched Multi-head, B: Batch, H: Head, R: Row. 193

A.2 Comparisons Dataflow Configurations. 201

A.3 The HW resource configuration of cloud and edge accelerators in the evaluation sections. 202

LIST OF FIGURES

1.1 The overview of DNN Workload, Accelerator, and Mapping. 2

1.2 The proposed optimization loop for Map Space Exploration (MSE) or Design Space Exploration (DSE). 5

2.1 The loop-nest representation of CONV2D and a Matrix Multiplication (GEMM). 13

2.2 An example of (a) perfect loop-nest and (b) imperect loop-nest. 14

2.3 A DNN accelerator architecture template. 14

2.4 A weight-stationary example. This is example is referenced from [88]. . . . 16

2.5 An output-stationary example. This is example is referenced from [88]. . . 17

2.6 The HW performance of randomly sampled HW-mapping in the design space on an example layer (second layer of VGG16). (a) The Energy to Latency plot, (b) The Latency to Area plot, and (c) The HW performance statistics of the sampled HW mappings. 18

2.7 The level of buffer mapping effects. (a) option 1 mapping between MAC and L1, L2 and off-chip, (b) option 2 mapping between MAC and off-chip. . 19

2.8 The Effect of tile sizes on ameliorating BW bottleneck. (a) option 1 with larger input tile size, (b) option 2 with smaller input tile size. 20

2.9 The Effect of tile sizes on reuse. (a) option 1 with larger weight tile sizes, (b) option 2 with smaller weight tile sizes. 21

2.10 The workflow of MAESTRO [68]. 22

2.11 The categories of different exploration algorithms (Mappers) in Map Space Exploration. 23

3.1 (a) Flexibility TOPS and prior work. (b) For any workload, a unique mapping involves choosing individual points from the map-space actually supported by the target accelerator. 30

3.2 The benefit of having flexibility in (a) Tile, (b) Order, (c) Parallelism, and (d) Shape. Example GEMM operation of $Z_{MN} = A_{MK} \times B_{KN}$ on 4x4 PE array. The subscript t indicates that these are tiles of the overall computation. The inflexible accelerator is weight stationary (tensor B), using a loop ordering of K → N → M. 31

3.3 DNN accelerator with NVDLA-style mapping. 32

3.4 HW Overhead for Flexibility Support (highlighted in yellow). (a) Flexible Tile Size HW Support: Soft-partitioned Scratchpads. (b) Flexible Loop Order HW Support: Configurable Address Generators. (c) Flexible Parallelism HW Support: Spatial and Temporal Reduction. (d) Flexible Shape HW Support: NoC support for PE clustering. 35

3.5 0.85 37

3.6 Flexibility-Aware Automated Design-Space Exploration Framework. 40

3.7 Tile Axis Isolation. (a) Performance comparisons of accelerators with different level of Tile flexibility running Mnasnet model. (b) Venn diagram of Tile map space on Mnasnet. We use on-chip buffer size=4K in this experiments. Scale: Total Data points in $W_T^{\omega} = \pi(40)^2$. Runtime/Energy/EDP are normalized against values of InFlex-1000. 43

3.8 The runtime performance and workload Flexion of fully-Tile-flexible accelerators with different buffer sizes. (W-F: Workload-dependent Flexion.) 44

3.9 Order Axis Isolation. (a) Performance comparisons of accelerators with different level of Tile flexibility running Mnasnet model. (b) Venn diagram of Tile map space on Mnasnet. Runtime/Energy/EDP are normalized against values of InFlex-0100. . 46

3.10 Parallelism Axis Isolation. Performance comparisons of accelerators with different level of Parallelism flexibility running Mnasnet model with (a) 16x64 PE array and (b) 32x32 PE array. (c) Venn diagram of Parallelism map space on Mnasnet. PRuntime/Energy/EDP are normalized against values of InFlex-0010. 48

3.11 Shape Axis Isolation. (a) Performance comparisons of accelerators with different level of Tile flexibility running Mnasnet model. (b) Venn diagram of Tile map space on Mnasnet. Runtime/Energy/EDP are normalized against values of InFlex-0001. . 50

3.12 The runtime performance and workload Flexion of fully-Shape-flexible accelerators with different size of PE arrays. (W-F: Workload-dependent Flexion) 51

3.13 Runtime performance of accelerators with different flexibility. InFlex-0000-X-Opt means class-0000 (fixed) accelerator that is optimized for DNN model X (X= Alexnet, Mnasnet, etc.). The runtime in each row is normalized by the runtime of the values of InFlex-0000-Alexnet-Opt. 52

4.1 The GAMMA encoding example of (a) 1-level mapper and (b) 2-level mapper. 60

4.2 (a) GAMMA's description of a 2-level mapper and (b) its decoded description for cost model (MAESTRO) of a NVDLA-like [80] 2-level mapper. . . 61

4.3 (a)The structure and algorithm flow of GAMMA, and (b) The summary of evolution in GAMMA. 62

4.4 The performance of found solutions across a suite of optimization methods on different target systems (S1, S2, S3) and different platform constraints (Edge, Cloud) for ResNet-18. 67

4.5 The energy consumption of S2 on ResNet-18. 70

4.6 GAMMA's found mapping of early, medium, and late layer of ResNet-18 in Figure 4.4(e). 71

4.7 End-to-end latency improvement over generations with GAMMA for S3 system and edge platform constraint. 72

5.1 A loop-nest representation of a NVDLA-like [80] mapping. 78

5.2 A canonical Map Space Exploration framework. 79

5.3 Comparisons of different types of mappers. Top figures show the converge curve across number of samples. Bottom figures show the converge curve across wall clock time. 83

5.4 (a) shows the sampled points by exhaustively sampling the search space of (Resnet Conv_4, Accel-A). The 3D visualization is projected by PCA dimension reduction. (b) shows the sampled points of different types of mappers in this search space. 85

5.5 Mapping axes sensitivity analysis using the mutation operators in Gamma [5]. E.g., Tile (blue): means mutating tile only, i.e, only tile is explored, and other mapping axes are fixed, similarly for (mutate-)Order and (mutate-)Parallelism. 86

5.6 Crossover (blending two mappings) sensitivity analysis using operators in Gamma [5]. Standard-GA uses the standard mutation and crossover (without domain-specific operators along each mapping axes designed in Gamma [5]). 88

5.7 The EDP difference of the same mapping with different loop order. We sweep through all 7! order combinations assuming all the buffer level utilize the same order. The 7! different mapping leads to 16 different EDP performance, with the best and the worst EDP differs by 14.4x times (under Resnet Conv_4, Accel-B). 90

5.8 The workflow of Warm-start and Sparsity-aware techniques in MSE. 90

5.9 Performance comparisons of initialized solution by Random Init and two types of warm-start Init comparing to the final optimized performance (after search). The EDP values are normalized by final optimized EDP (green bars). 91

5.10 The performance convergence curve with random initialization and warm-start (by similarity) initialization at the (a) first layer and (b) a later layer of VGG16. 91

5.11 The benefit of warm-start (by similarity) when executing MSE for the DNN models. Warm-start MSE achieves comparable EDP performance to default MSE, but converge 3.3-7.3x faster. Different colors represent different layers of the models. 92

6.1 State-of-the-Art HW and Mapping optimization frameworks. 108

6.2 HW/Mapping Co-optimization Framework. Our three technical contributions are highlighted. 109

6.3 (a) Co-opt Framework, (b-c) the HW-Mapping encoding representation, and (d-e) the corresponding decoded accelerator configuration. (f) The formula for calculating minimum on-chip buffer requirement. (g) The definition of notations. 110

6.4 GAMMA's genetic operators† and their perturbing space‡. 115

6.5 State-of-the-Art HW and Mapping optimization frameworks. 116

6.6 Performance comparisons of different optimization algorithms. Both latency and latency-area-product are normalized by the values of CMA, the best-performing baseline (lower is better). We highlight the best performing algorithm in different tasks (DNN models) in bold. N/A means the algorithm cannot find valid solution that fits in the area constraint under the set 40K sampling budgets. 118

6.7 Latency of the found solution by different optimization scheme. Latency values are normalized by the values of best-performing baseline method (Compute-focused + Gamma). Grid-S: grid search. Buffer-focused: large buffer design. Compute-focused: large PE arrays design. Meidum-Buf-Com: medium buffer and PE arrays design. 119

6.8 The solution found by different optimization schemes and their corresponding performance on Mnasnet at edge resources. 120

7.1 The structure and flow[†] of M3E. 126

7.2 The encoding scheme of M3E. 127

7.3 (a) Mapping description from the decoder. (b) The example BW and sub-accelerators allocation results. 128

7.4 (a) Then encoding scheme of M3E. The genetic operators of MAGMA and their implying update on the mapping: (a) mutation, (b) crossover-gen, (c) crossover-rg, and (d) crossover-accel. 132

7.5 The algorithm flow of MAGMA. 132

7.6 (a) The average per-job no-stall latency and required BW for no-stall across different models on high (HW) and low (LB) bandwidth mapping style. (b) Average no-stall latency and (c) average BW required for no-stalls across all involved jobs. 140

7.7 The experiment results on multi-core homogeneous small accelerator (S1) with BW=16 across four tasks. Throughput values are normalized by the value of MAGMA. The absolute throughput values of MAGMA in (a-d) are: 249, 397, 194, and 329 GFLOPs. 141

7.8 The experiment results on multi-core heterogeneous (a)(b) small (S2, BW=16) and (c)(d) large (S4, BW=256) accelerator on Vision and Mix tasks. Throughput values are normalized by the value of MAGMA. The absolute throughput values of MAGMA in (a-d) are: 254, 271, 254, and 383 GFLOPs. 141

7.9 Performance comparisons on multi-core heterogeneous (a) small (S2) and (b) large (S4) accelerator on Mix tasks, given different BWs. Throughput values are normalized by the value of MAGMA. 142

7.10 Jobs analysis of the averaged per-job (a) no-stall latency and (b) required BW. (c) Performance evaluation of MAGMA on S3, S4, and S5 with different BW. In (a-b), we concatenate the four independent no-stall latency (averaged required BWs) into a stacked bar and show the total values. In (c), throughput values are normalized by the value of S5. 142

7.11 Jobs analysis of the averaged (a) per-job no-stall latency and (b) required BW of fixed and flexible PEs arrays. Performance evaluation of MAGMA with fixed or flexible PEs array on (c) Vision and (d) Mix. Throughput values are normalized by the value of flexible accelerator. 145

7.12 The visualization of found solution by Herald-like and MAGMA. (a)(c) shows the respective sub-accelerator allocations, and (b)(d) shows the respective BW allocations. (Mix task, S5, BW=1). 146

7.13 The reached performance of MAGMA given the same task and setting (Mix, S2, BW=16) with different group sizes. Throughput values are normalized by the value of group size=1000, ... 146

8.1 Overview of ConfuciuX. 153

8.2 DNN deployment scenarios and corresponding HW assignments. In Layer Sequential (LS), each layer of the model is mapped one by one on the entire accelerator, with all on-chip compute and memory assigned to it; in Layer Pipelined (LP), the entire model is mapped and run in a pipelined manner, with the compute and memory partitioned across all layers. 154

8.3 Different HW resource combinations with the same NVDLA-style dataflow. 157

8.4 Searching for per-layer PEs/Buffers configurations to optimize latency/energy with different techniques. Purple indicates lower (better) and red indicates higher (worse) latency/energy. **Heuristic A**: Determine the PEs/Buffers with the most compute-intensive layer (Layer-38) and apply the same configuration for all the layers. **Heuristic B**: Determine the PEs/Buffers by the configuration that optimizes end-to-end whole model latency/energy. 164

8.5 The learning curve of the critic network. 168

8.6 The fast convergence and sample efficiency of Con'X (global). 170

8.7 The solution for (a) MobileNet-V2 and (b) ResNet-50 (Obj.(min.):Latency, Cstr:IoT area). 172

8.8 Overall latency as a function of epochs across two-stage optimization in ConfuciuX (MobileNet-V2, Obj.(min.):Latency, Cstr.:IoT area). 173

8.9 Dataflow-HW co-automation with ConfuciuX for MobileNet-V2. (Obj.(min.):Latency, Cstr:IoT area). 175

A.1 The structure of attention-based models. Green matrix notation shows the size of weight tensors; black matrix notation shows the size of activation tensors; Softmax is applied on output of Logit. 189

A.2 Potential of Tensor-Tensor Fusion. The operation intensity of single and fused operators in attention layers in TrXL(-large) [70] using batch size=1. The notations are from Figure A.1. f(X, Y) means a fused X and Y operator. 190

A.3 Roofline analysis on TPU-v3 [284] for operators in BERT(-base) [69], TrXL(-large) [70], and XLM(xlm-mem-en) [71], and ResNet50 [142] using sequence length = 512. 191

A.4 (a) Baseline and (b) FLAT dataflow. FLAT performs inter-operator fusion of L, A while respecting data dependencies introduced by Softmax. FLAT-tile enables staging slices of the logits tensor in the on-chip scratchpad increasing effective memory bandwidth. This fused, interleaved execution of L, A yields higher compute utilization and improved performance. 193

A.5 For loop of fused L-Softmax-A (or shortened as L-A in the paper) and the choice of granularity. 194

A.6 Map space exploration framework. (Special Function Unit: for computing non-linear operations, e.g., softmax, activations.) 197

A.7 Compute utilization analysis of L-A operators under different dataflow granularities, running BERT with 512 sequence length with edge platform resources. Flex dataflow (Flex-X): Flex dataflow with X-granularity. FLAT dataflow (FLAT-X): Flex dataflow with X-granularity; X could be M (batch-Multi-head), B (Batch), H (head), or R (row). 202

A.8 Comparisons of compute utilization across different sequence lengths running BERT under edge platform resources with sequence length of (a) 512, (b) 4K, and (c) 64K. We sweep the available on-chip buffer from size 20KB to 2GB. We list three level of performance analysis, L-A: focusing on performance difference at the L, A operators; Block: consider all operators in the attention block; and Model: a model-wise performance. 203

A.9 Comparisons of compute utilization across different sequence lengths running XLM under cloud platform resources with sequence length of (a) 4K, (b) 16K, (c) 256K. 206

A.10 The corresponding energy consumption of each of the data-point in Figure A.8. The energy numbers are normalized by the largest energy number in each sub-plot. 207

A.11 The corresponding energy consumption of each of the data-point in Figure A.9. The energy numbers are normalized by the largest energy number in each sub-plot. 208

A.12 The design space of FLAT when running BERT with sequence length of 512 under edge platform resources. Flex dataflow (Flex-X): Flex dataflow with X-granularity. FLAT dataflow (FLAT-X): Flex dataflow with X-granularity, where X could be M (batch-Multi-head), B (Batch), H (head), or R (row). The design-point with the highest utilization, given a buffer constraint represents Flex-Opt and FLAT-Opt (Table A.2). 210

A.13 End-to-end latency breakdown running: (a) BERT on Edge accelerator, (b) XLM on Cloud accelerator. Naive_e(dge): Edge with Naive dataflow. Flex-opt_e(dge): Edge with optimal Flex dataflow. FLAT-opt_e(dge): Edge with optimal FLAT dataflow. Similarly, for Naive_c(loud) and so on. Flex-opt represents the best possible (SOTA) intra-operator dataflow obtained through exhaustive search of the intra-operator map-space. FLAT performs inter-operator L-A fusion to relieve memory-bandwidth boundedness and achieves lower run time. While FLAT realizes benefits even at modest sequence lengths, the gains are pronounced at larger sequence lengths. . . . 211

A.14 The end-to-end speedup and energy-consumption ratio of FLAT-opt_e(dge) over Flex-opt_e(dge) and FLAT-opt_c(loud) over Flex-opt_c(loud) on different models. 211

A.15 The required BW to reach compute utilization rate higher than 0.95 in the most BW-intensive L-A operator when running XLM using Flex-opt_c(loud) or FLAT-opt_c(loud). 212

SUMMARY

The proliferation of AI across a variety of domains (vision, language, speech, recommendations, games) has led to the rise of domain-specific accelerators for deep learning. At design-time, these accelerators carefully architect the on-chip dataflow to maximize data reuse (over space and time) and size the hardware resources (PEs and buffers) to maximize performance and energy-efficiency, while meeting the chip's area and power targets. At compile-time, the target Deep Neural Network (DNN) model is mapped over the accelerator. The mapping refers to tiling the computation and data (i.e., tensors) and scheduling them over the PEs and scratchpad buffers respectively, while honoring the microarchitectural constraints (number of PEs, buffer sizes, and dataflow).

The design-space of valid hardware resource assignments for a given dataflow and the valid mappings for a given hardware is extremely large (O(10^{24})) per layer for state-of-the-art DNN models today. This makes exhaustive searches infeasible. Unfortunately, there can be orders of magnitude performance and energy-efficiency differences between an optimal and sub-optimal choice, making these decisions a crucial part of the entire design process. Moreover, manual tuning by domain experts become unprecedentedly challenged due to increased irregularity (due to neural architecture search) and sparsity of DNN models. This necessitate the existence of Map Space Exploration (MSE). In this thesis, our goal is to deliver a deep analysis of the MSE for DNN accelerators, propose different techniques to improve MSE, and generalize the MSE framework to a wider landscape (from mapping to HW-mapping co-exploration, from single-accelerator to multi-accelerator scheduling). As part of it, we discuss the correlation between hardware flexibility and the formed map space, formalized the map space representation by four mapping axes: tile, order, parallelism, and shape. Next, we develop dedicated exploration operators for these axes and use genetic algorithm framework to converge the solution. Next, we develop ”sparsity-aware” technique to enable sparsity consideration in MSE and a ”warm-start” technique to solve the search

speed challenge commonly seen across learning-based search algorithms. Finally, we extend out MSE to support hardware and map space co-exploration and multi-accelerator scheduling.

CHAPTER 1
INTRODUCTION

Deep Neural Network (DNNs) have become an indispensable tool in the solution toolbox for a variety of complex problems such as object detection, machine translation, language understanding, autonomous driving, and so on. There is a growing demand for specialized DNN accelerators pursuing high performance with high energy, power, and area efficiency. Different applications, objectives, and design budgets create a massive accelerator design space.

In order to achieve high efficiency across a wide range of DNNs, state-of-the-art DNN accelerators are often designed with *flexibility* to adapt to various workloads [1, 2, 3]. These accelerators allow different strategies (i.e., *mappings*) for mapping workloads onto the accelerator to maximize performance and energy efficiency. This flexibility imposes a unique challenge for deployment: finding a high-quality mapping between a DNN workload and the flexible accelerator during runtime. From the space of all legal mappings (i.e., the *map space*) of a workload, the user of such a flexible accelerator needs to find and utilize the best mapping to unlock the full potential of the DNN accelerator.

As a result, *map space exploration* (MSE) is critical for DNN accelerator efficiency. It is a complex and challenging problem because the search space is often massive. Prior work has clearly defined the MSE problem [4, 5, 6, 7], cleanly separating it from the hardware architecture design problem, and has proposed various search algorithms (i.e., *mappers*) [4, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 7, 29, 30, 31, 32, 33] to find optimized mappings for DNN accelerators and workloads.

Despite the success achieved by these prior efforts, MSE remains a computationally challenging problem. This is because the search space for legal mappings for even a single layer of a modern DNN (e.g., ResNet-50) on a typical edge class accelerator [1] is $\sim$

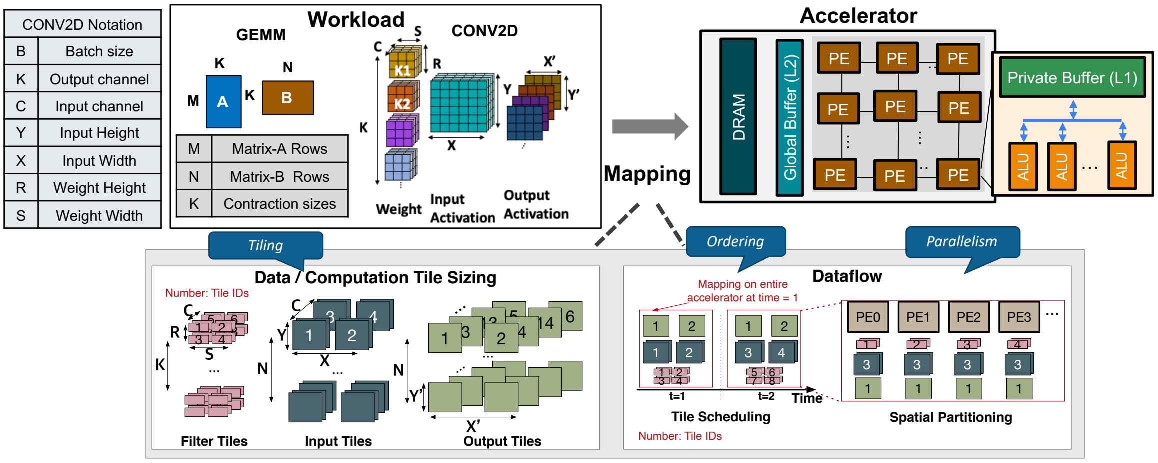


Fig. 1.1: The overview of DNN Workload, Accelerator, and Mapping.

$O(10^{24})$ [5, 6] which would require more time than the age of the earth to search exhaustively (assuming 1msec to evaluate each mapping sample). This gets exacerbated as newer and ever-larger DNN models are being created with increasing frequency, especially thanks to the success of neural architecture search techniques [34, 35, 36, 37, 38]. Furthermore, the advent of *compressed-sparse* DNNs [39, 40, 41, 42, 43, 44, 45], whose mappings are not performance-portable across sparsity levels, further increase MSE burden.

This thesis aims to develop a scalable MSE for complex DNN workloads. It tackles the following challenges.

1.1 Challenges

1.1.1 Linkage between HW Accelerator Flexibility and Map Space

Traditionally, the efficiency of domain-specific accelerator ASICs has come from *specialization*, i.e., the control path and datapath in the accelerator are tailored to the deep neural networks (DNNs) that are expected to run on the accelerator. In other words, the number of mappings that an accelerator can support (aka *map-space*) is restricted. To this end, there has been growing interest in developing *flexible* DNN accelerators. Flexibility allows the accelerator to better tailor itself to the diverse set of layer parameters within the current DNN being mapped [46, 47], instead of targeting the average case. While the notion HW flexibility described above makes intuitive sense, the field of DNN accelerators today is inconsistent about the definition. We find that "flexibility" has been used loosely to refer either to the ability to handle different loop tiling/blocking [10], and/or support for different loop orders [12], and/or the ability to spatially partition across different dimensions [48], and/or the ability to logically support different PE aspect ratios [49]. In fact, there is also inconsistency in whether flexibility is a hardware feature [46] or simply a term for compiler loop-transformations over a fixed inner tile [10, 50]. This in-formalism is a serious barrier to the adoption of flexibility features, as it hinders precise quantification of the cost/benefit tradeoff. Moreover, it is a lack of formalism on how different levels of flexibility impact the map space and the performance of the accelerator.

1.1.2 Efficient Mapper

The search algorithm in MSE is called – Mapper. Although multiple prior works [9, 51, 52, 30, 53, 54, 55, 56, 57, 18] have studied the mapping problem for DNN accelerators, the size of the map space (exceeding $O(10^{24})$ even for a single layer of a DNN) makes the problem highly challenging. To cope with this challenge, most prior works restrict the search space. For e.g., coarse-grained strided exhaustive search [23, 12, 16, 17, 22], random search [4], fixed parallelism [23, 16, 22, 56, 57, 18], or limited search for tile sizes for one or more fixed dataflows [21, 12]). Alternately, ML-based search techniques have also been leveraged for guided search to increase sampling efficiency [31, 24, 26, 27]; however, they need to restrict some aspects of the map space (e.g., fixing the parallelism levels) to adapt to the ML algorithms. Such restrictions of the mapping space can lead to local optimal mappings which are significantly sub-optimal, as recent works have highlighted [4, 8].

1.1.3 Scalability of MSE

Despite the success achieved by the design of an efficient mapper, the scalability of MSE remains challenging because of two issues: speed and dynamic sparsity in workloads. 1)

Speed. Most mappers search for mapping in a layer-by-layer fashion, the run time of MSE inevitably increases linearly with the layers of DNN models, increasing compilation times significantly. More importantly, the emergence of techniques like neural architecture search is leading to new DNN models coming out frequently with highly irregular tensor shapes. This naturally increases the demand for efficient MSE. In this regard, the search speed of MSE is a critical concern. 2) Dynamic sparsity in workloads. Mapping need to be optimized for the specific sparsity level of the workloads. While the sparsity of the weight is often fixed for a trained DNN model, the sparsity of activations is dynamic. When facing activation sparsity, we would either under-utilize the hardware because of inefficient mapping or would need to re-launch the MSE again and again for every input-activation.

1.1.4 Mapping and HW Co-exploration

The next question we want to answer is that can we extend MSE to support Mapping HW co-exploration. However, HW-Mapping co-exploration is not trivial. The search space is the cross-product of HW space (as large as $O(10^{12})$[1]) and mapping space (as large $O(10^{24})$ [5]), which can lead to an design space as large as $O(10^{36})$. Therefore basic techniques like exhaustive searches become impractical. An optimization-based algorithm (e.g., RLs, GA, simulated annealing, and so on) is needed. A naive optimization-based HW-Mapping co-optimizer can be formulated as follows. One can formulate a two-loop optimization process, where the outer-loop optimizes the HW, and the inner-loop (which takes in the HW parameters from outer-loop) optimizes the mapping or vice versa. E.g., a highly tuned mapper GAMMA [5] requires about 10 mins to converge to a mapping solution of a given HW configuration. For a two-loop optimization, the found solution at inner-loop (mapper) becomes the feedback of one single sampling point of HW optimizer at outer-loop, where outer-loop can easily require more than 10K sampling points. A naive two-loop optimization requires 1.6M sampling points and more than 1,600 hours, which is challenging for practical

[1] Assuming PEs:128x128 and maximum buffers:100MB, the number of combinations is $O(10^{12})$.

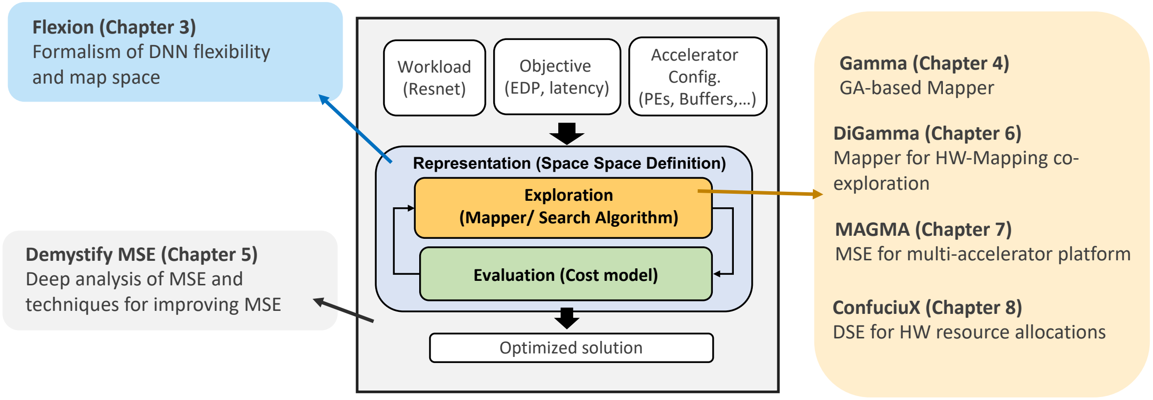

Flexion (Chapter 3)
Formalism of DNN flexibility and map space
Workload (Resnet)
Objective (EDP, latency)
Accelerator Config. (PEs, Buffers,...)
Representation (Space Space Definition)
Exploration (Mapper/ Search Algorithm)
Evaluation (Cost model)
Optimized solution
Demystify MSE (Chapter 5)
Deep analysis of MSE and techniques for improving MSE
Gamma (Chapter 4)
GA-based Mapper
DiGamma (Chapter 6)
Mapper for HW-Mapping co-exploration
MAGMA (Chapter 7)
MSE for multi-accelerator platform
ConfuciuX (Chapter 8)
DSE for HW resource allocations

Fig. 1.2: The proposed optimization loop for Map Space Exploration (MSE) or Design Space Exploration (DSE).

usage.

1.2 Thesis Contributions

1.2.1 Flexion: Accelerator Flexibility and Map Space

We present a formal definition of the flexibility of an accelerator that refers to the percentage of explorable/supported map-space out of all possible (i.e., exhaustive, as formally defined in Section §3.2) mappings for a DNN layer. As these map-spaces can be overwhelmingly complex, we identify a complementary abstraction: we taxonomize flexibility into four axes – tile size (T), order (O), parallelism (P), and array shape (S). Identifying each axis with a binary 0/1 value enables us to create 16 flexibility classes within which various accelerators can fall, enabling a systematic classification of prior work. For each axis, we identify both the hardware support needed and the software loop transformations it exposes to a mapper.

1.2.2 Gamma: Efficient DNN Mapper

We propose encoding scheme transforms the mapping problem into an optimization problem, which enables the user to directly use off-the-shelf optimization algorithms for mapping. These form our baselines. We introduce new GA operators, enabling a domain-specific flexible search space, unlike most off-the-shelf optimization algorithms. We automate Gamma as a black-box optimizer for the HW-mapping problem. This reduces the learning curve and saves manual-tuning effort for ML practitioners exploring the HW-mapping space. Gamma encapsulates an end-to-end workflow, which generates outputs compatible with an open-source cost model [58]. We have released Gamma as an open-source DNN Mapper [59].

1.2.3 Demystifying MSE: Techniques for Improving the Scalability of MSE

We compare three wide categories of mappers: random-based [4] (i.e., heuristic pruning), feedback-based [5] (i.e., black-box optimization and reinforcement learning), and gradient-based [6] (i.e., surrogate models), and analyze their trade-offs. We conduct a sensitivity analysis of different mapping axes to understand the contribution of each axis. We then perform case studies that reveal distinguishing characteristics of good and bad mappings. Based on our findings, we propose two novel techniques to enhance the state-of-the-art in MSE. (i) We propose a "warm-start" technique to initialize the MSE with previous mapping solutions from previous layers in the replay buffer based on a *similarity* metric, enabling the mapper to start at a better point and converge faster. (ii) We propose a "sparsity-aware" technique to search for a mapping that can perform well across a range of target activation sparsity. We believe these techniques can be augmented over existing MSE tools, making them more robust and scalable for future DNNs.

1.2.4 DiGamma: Mapping and HW Co-exploration

We propose a HW-Mapping co-optimization framework, which takes in any DNN model(s), design objective, budget, and constraint, and generates an accelerator design point, HW (i.e., numbers of PEs, number of memory levels, sizes of buffers at each level) and mapping (i.e., parallelism, loop order, clustering, tile sizes). We abstract the detail of performance modeling for different DNNs, and chip constraints and provide a generic interface, where many existing optimization algorithms can be plugged in. We propose an efficient design point encoding, which describes both HW and mapping with a list of parameters. Our encoding method constructs a compact representation of the cross-coupled design space that boosts the efficiency of the optimization algorithms. We propose a domain-aware genetic algorithm-based optimization method. It is specifically designed for HW-Mapping design space and comes with specialized optimization operators to step through the design space in a structured manner, and its HW exploration strategy respects the interaction between HW and mapping.

1.2.5 MAGMA: Mapping across Multi-Accelerators

We extend MSE to support mapping across multi-accelerator. We propose an optimization framework called Multi-workload Multi-accelerator Mapping Explorer (M3E). In the framework, (i) we develop an efficient encoding scheme to encode the search space of the mapping; (2) we develop several modules to enable the found mapping to effectively orchestrate the data movement across sub-accelerator cores; (3) we enable several commonly used black-box optimization algorithms and two reinforcement learning methods to be leveraged as the underlying optimization methods. In M3E, we break the multi-tenant mapping problem into two components: *sub-accelerator selection* and *job prioritization*. Sub-accelerator selection is where we assign each job a sub-accelerator to execute; job prioritization is where we order the jobs that are assigned to a sub-accelerator. Each component creates an immense design space by itself. The full design space is the combinatorial probability of both, which becomes as large as O(1e81) (§7.1.6). It also motivates us to design a sample-efficient optimization algorithm. We propose a custom genetic algorithm-based optimization method for this mapping problem, termed Multi-Accelerator Genetic Mapping Algorithm (MAGMA). We design custom genetic operators in MAGMA, which allows it to structurally explore the design space and largely increase its sample efficiency.

1.2.6 ConfuciuX: Automated DNN Accelerator Design

We also develop a framework to automate the HW resource assignment of DNN accelerator design. Different choices for HW resources can lead to drastically different latency and energy for a given DNN. Some recent studies have shown that HW resource assignment plays a more important role in determining the accelerators' performance than its dataflow [60]. However, determining the policy for assigning HW resources is still very much an open problem, with prior works on HW Design-Space Exploration almost exclusively relying on exhaustive searches [17, 61, 62, 63, 64, 65, 66, 67, 68, 18]. In this thesis, we develop an autonomous mechanism to efficiently search through the HW design-space. It takes the target model, platform constraint, deployment scenario, and optimization objective (latency/energy) as input, and determines an optimized HW assignment strategy (number of computes and buffers). We leverage reinforcement learning (RL) to perform a global coarse-grained search, followed by a genetic algorithm (GA) for fine-grained tuning.

1.2.7 FLAT: Dataflow Technique for Mapping Attention Operation

Attention mechanisms, the key building block of transformer models, have enabled state-of-the-art results across a wide range of machine learning (ML) tasks—from natural language processing (NLP) [69, 70, 71], to object detection [72, 73, 74], image classification [**vision˙longformer**, **levit**, **cvt**, **longshort˙transformer**], image generation [75, 76, 77], and music synthesis [78, 79]. we identify that the conventional dataflow/mapping methods for CONV and FC layers [80, 81, 82, 83] are inadequate for attention layers. This fundamentally tackles the challenges associated with attention layers by devising a first in its class *many-to-many inter-operator* dataflow optimization mechanism, called **F**used **L**ogit **A**ttention **T**iling. This optimization particularly fuses multiple many-to-many tensor operators, while systematically preserving their inter-operator data dependencies, leading to a significant reduction in off-chip memory bandwidth pressure. In addition, to fully realize the performance benefit of this inter-operator fusing mechanism, FLAT performs a new tiling approach across the fused operators. This tiling enables efficient staging of quadratically growing intermediate tensors of attention operations on tight-budgeted on-chip memories, leading to higher performance and energy savings and elevating the scalability of transformer models up to 64 K inputs. These benefits are unlocked with only modest hardware changes, integrating into a platform deployable on off-the-shelf DNN accelerators.

1.3 Thesis Statement

MSE bridges the gap between two active trends: (1) efficient DNN model design and the emergence of sparsity in state-of-the-art DNN models and (2) flexible hardware accelerators that support diverse mappings via configurable buffer hierarchies [84] and on-chip interconnect topologies as an answer to the first trend. MSE is crucial for extracting performance and energy efficiency from the accelerator as there can be multiple orders of of difference in performance and energy-efficiency between good and bad mappings.

1.4 Thesis Overview

This thesis is organized as follows (Figure 1.2):

- **Background and Related Works.** In chapter 2, we discuss backgrounds and related works of this thesis.
- **Accelerator Flexibility and Map Space Introduction.** In chapter 3, we formalize the definition of accelerator flexibility. Next, we discuss the implied map space of different levels of accelerator flexibility. The discussion in this chapter set up the context of the following chapters which are exploring the map space of the fully flexible or partially flexible accelerator for different DNN workloads.
- **Efficient Mapper Design.** In chapter 4, we discuss the challenge of an efficient mapper design. We develop a GA-based mapper to show the performance comparisons with other black-box methods. We show that with the domain-specific operators, Gamma,

the mapper we propose, can achieve the best sampling efficiency.

- **Scalability of MSE.** In chapter 5, we discuss the upcoming challenge of MSE for future complex DNN workloads. The DNN Workload becomes increasingly irregular because of the wide application of neural architecture search and the sparsity of DNN workloads increases with the development of new pruning techniques. These make DNN workload more complex and challenging for MSE to solve within limited time constrain. We propose two practical techniques discussed in chapter 5 to tackle these challenges.
- **HW-Mapping Co-exploration.** In chapter 6, we discuss how to extend MSE and Mapper to support a larger search space. In this chapter, we discuss how to extend the work to support HW-Mapping co-exploration. We extend the mapper in chapter 4 for this purpose and call it DiGamma.
- **Mapping Across Multi-accelerator.** In chapter 7, we discuss another dimension of extension to the DNN MSE. Previously, we focus on single accelerator mapping. In this chapter, we discuss how can we form the MSE to consider multi-accelerator mapping. We introduce what is the trending multi-accelerator system architecture, how the mapping is defined for this map space, and finally how we design a mapper for this map space.
- **Automating DNN Accelerator Design.** Previously, we focus on map space discussion, which works on a given HW accelerator. In chapter 8, we discuss the design space of an DNN accelerator. We focus on HW resource allocation problem for HW accelerator, which essentially allocates Processing Element (compute) and Buffer (memory). We use the same optimization framework we developed throughout the previous chapter for MSE and change the search space to HW resources of the accelerator, essentially becoming a design space exploration problem (DSE). We leverage RL for this DSE problem.
- **Mapping Design for New DNN Operation.** In §A.1, we show a case study on a new

DNN operation, attention. We show how to develop new mapping for attention operation. We analyze the performance bottleneck of attention and develop a fusion-based technique to tackle the problem.

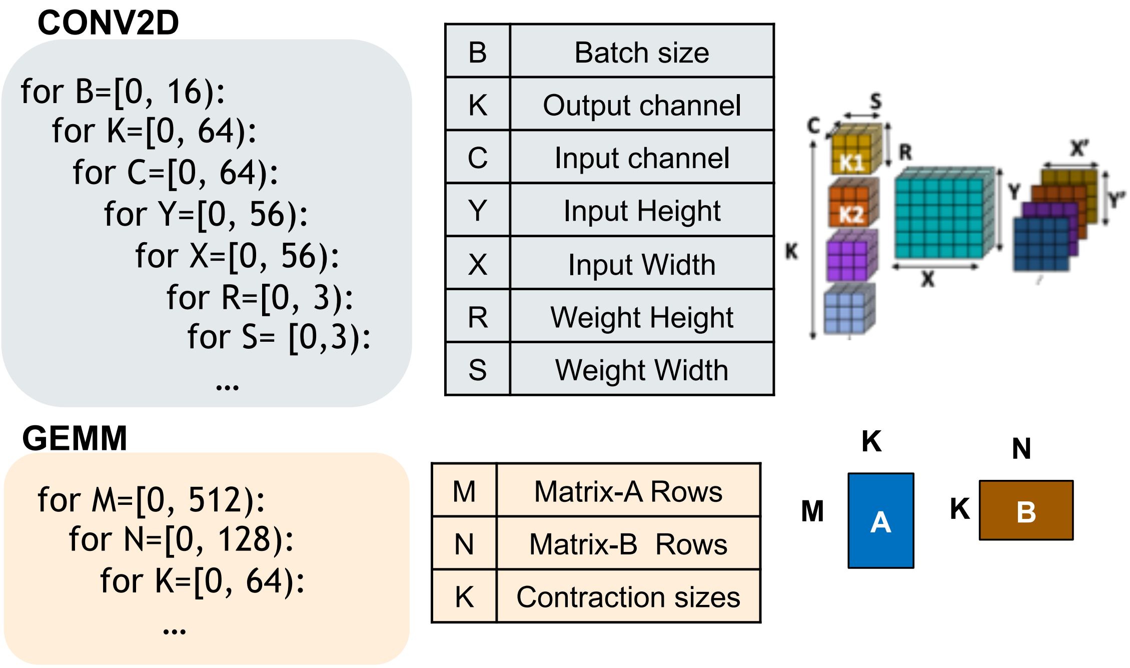
- **Conclusion and Future Works.** In chapter 9, we conclude this thesis with a summary of our contribution and future works.

CHAPTER 2
BACKGROUND

This chapter provides background for the topics covered in this thesis, including background of DNNs and DNN HW accelerators, concepts and examples of DNN mappings, and introductions of common optimization methods for design space exploration.

2.1 DNN Workloads

Different Deep Nueral Networks (DNNs) are developed for different applications. In this thesis, we focus on the performance of the DNNs (We use the term "performace" to refer to hardware-wise performance indexes such as latency, energy, power, and area. We use the term "quality" to refer to the application-wise criteria such as accuracy, precision, fitness and so on. In this thesis, we focus on DNN accelerator performance optimization.) A DNN **model** is usually made of tens to hundreds of DNN **layers** and **other operators**. There are different types computation of a DNN layer such as convolution, including 1D/2D convolution, depth-wise convolution, point-wise convolution, Fully-Connected (FC), Einsum (a general tensor computations), Attention, and so on. Also, a DNN layer could be made of different "shape". For example, different output/input channel sizes, number of nodes, sizes if hidden dimensions and so on. These DNN layers can be represented with a general loop-nest GEMM computation. For example, a CONV2D can be represented as 7 for-loops, and GEMM can be represented as 3 for-loops, as shown in Figure 2.1. Other operators include various of activations (e.g., ReLU, tanh, sigmoid), pooling, batchnorm, softmax, and so on, which could not be represented as loop-nest of GEMM computations. These operators are usually mapped on spatial function units or CPU of DNN accelerators, or on the host server, instead of the parallel processing elements (array of PEs) on the DNN accelerators. This thesis focusing on performance optimizations of "DNN layers" on the



CONV2D

```
for B=[0, 16):
  for K=[0, 64):
    for C=[0, 64):
      for Y=[0, 56):
        for X=[0, 56):
          for R=[0, 3):
            for S= [0,3):
              ...
```

B	Batch size
K	Output channel
C	Input channel
Y	Input Height
X	Input Width
R	Weight Height
S	Weight Width

GEMM

```
for M=[0, 512):
  for N=[0, 128):
    for K=[0, 64):
      ...
```

M	Matrix-A Rows
N	Matrix-B Rows
K	Contraction sizes

Fig. 2.1: The loop-nest representation of CONV2D and a Matrix Multiplication (GEMM).

DNN accelerators and leave “other operators” as future work.

In this thesis, we use individual DNN layers as our target ***workload****.*

Problem Domain. In this thesis, we consider widely used DNN layers including CONV, FC, GEMM, and Attention, whose computation are affine transformation and perfect nested loop. Affine transformation means a geometric transformation that preserves lines and parallelism [85]. For examples scaling, reflection, rotation and compositions of them in any combination and sequence are affine transformation. Most of the DNN layers are affine transformation. More general computation or imperfect loop nest [86, 87] are not considered in this thesis. The perfect loop-nest means the computation is at the inner-most loop-nest (Figure 2.1(a)), while imperfect loop-nest means the computations could be interleaved with computation (Figure 2.1(b)).

2.2 Accelerator Hardware Configuration

A canonical spatial DNN accelerator often houses an array of Processing Elements (PEs), as shown in Figure 2.3. Each PE has one to several ALU units to compute partial sums,

```
for B=[0, 16):
  for K=[0, 64):
    for C=[0, 64):
      for Y=[0, 56):
        for X=[0, 56):
          for R=[0, 3):
            for S= [0,3):
              O[..] = I[..] × W[..]
```

(a)

```
for B=[0, 16):
  for K=[0, 64):
    for C=[0, 64):
      for Y=[0, 56):
        O[..] = I[..] + A[..]
        for X=[0, 56):
          for R=[0, 3):
            for S= [0,3):
              O[..] = I[..] × W[..]
```

(b)

Fig. 2.2: An example of (a) perfect loop-nest and (b) imperect loop-nest.

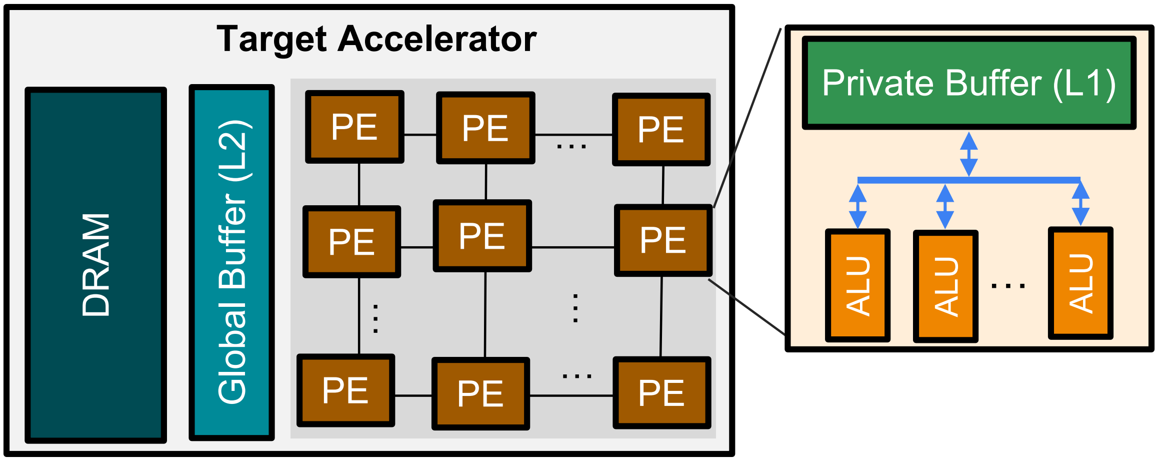


Fig. 2.3: A DNN accelerator architecture template.

and private local (aka “L1”) buffers to store weights, input activations, and partial sums. The accelerator also houses a global shared (aka “L2”) buffer to prefetch activations and weights from DRAM for the next tile of computation that will be mapped over the PEs and L1 buffers. Networks-on-Chip (NoCs) are used to distribute operands from the global L2 buffer to the L1 buffers in the PEs, collect the partial or full outputs, and write them back to the global L2 buffer.

2.3 Dataflow and Mapping

Each layer of a DNN can be expressed as a multi-dimensional loop nest over the input and weight tensors. A CONV2D layer has 6 loops (7 with batching) while a FC layer (i.e., GEMM operation) has 3 loops. At runtime, these loop bounds may be smaller than the hardware resources available (leading to under-utilization) or greater than them (leading to multiple temporal passes). Scheduling the loop involves splitting these loops into sub-partitions (tiles) and re-ordering the resulting loop nest. Different accelerators choose to execute different loop levels temporally (via serialization through the same PE) or spatially (via parallelization across distinct PEs, or clusters of PEs). The choice of parallelization may allow some input tensors to be multicasted to multiple PEs, amortizing expensive accesses. Similarly, parallelized partial-sum contributing to the same output allows for the use of efficient spatial reduction hardware. The number of tiling levels, loop order, and parallelism dimensions are collectively referred to as the accelerator's *dataflow* in prior work [83, 46]. The dataflow, with specific tile sizes at each level of the accelerator's buffer hierarchy, is called a *mapping* [47, 46], as shown in Figure 1.1.

2.4 The Impact of Mapping

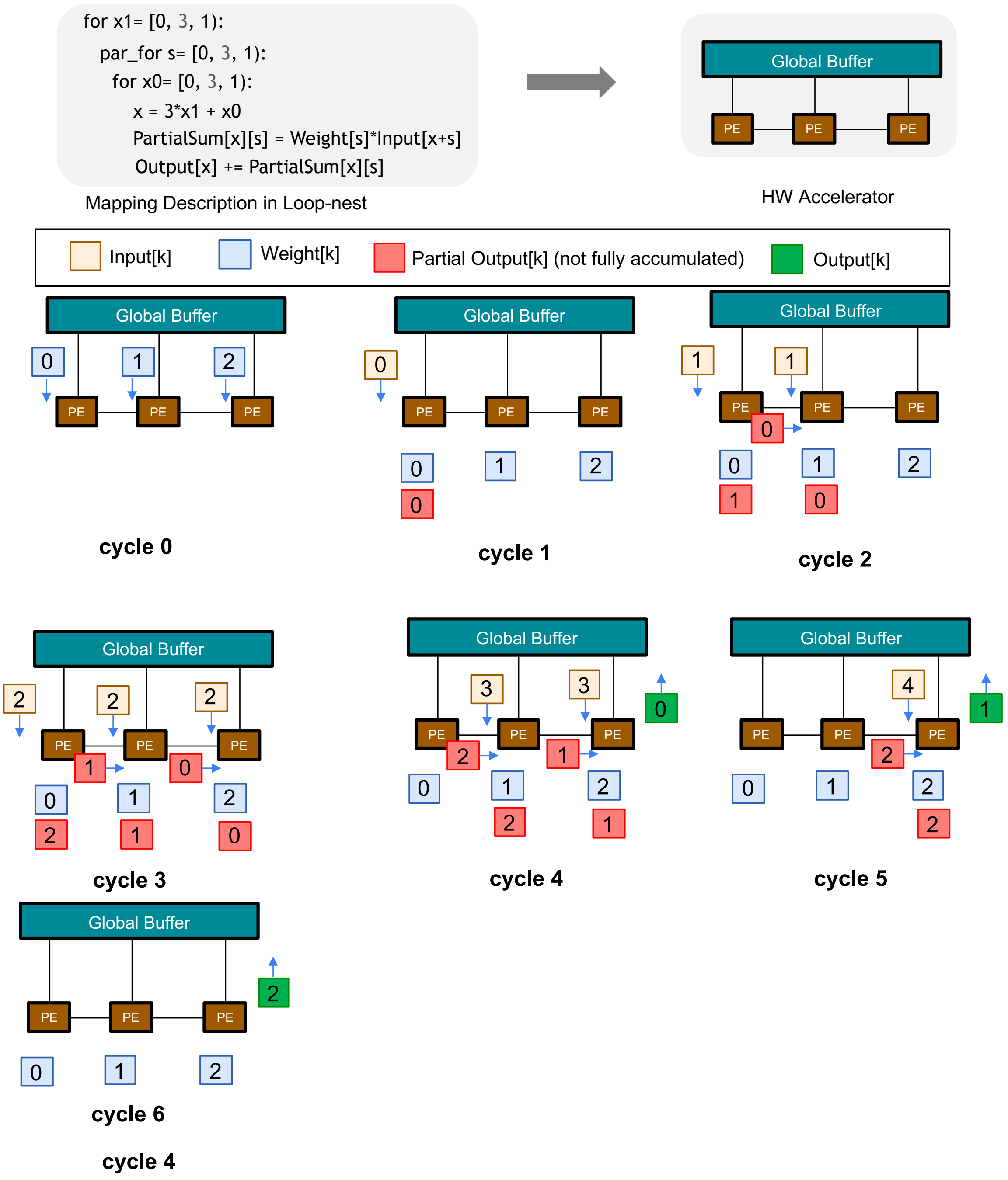
2.4.1 Optimal Mappings

The same workload can be mapped to the HW by various different ways of mappings. A good and bad mappings can have up to 2 order of magnitude difference in latency, power, and energy, as shown in Figure 2.6. This is also observed in [68, 4, 6, 10]. Therefore to search for the optimal mapping for the given workload become one critical issue.

2.4.2 Efficient Mappings Example

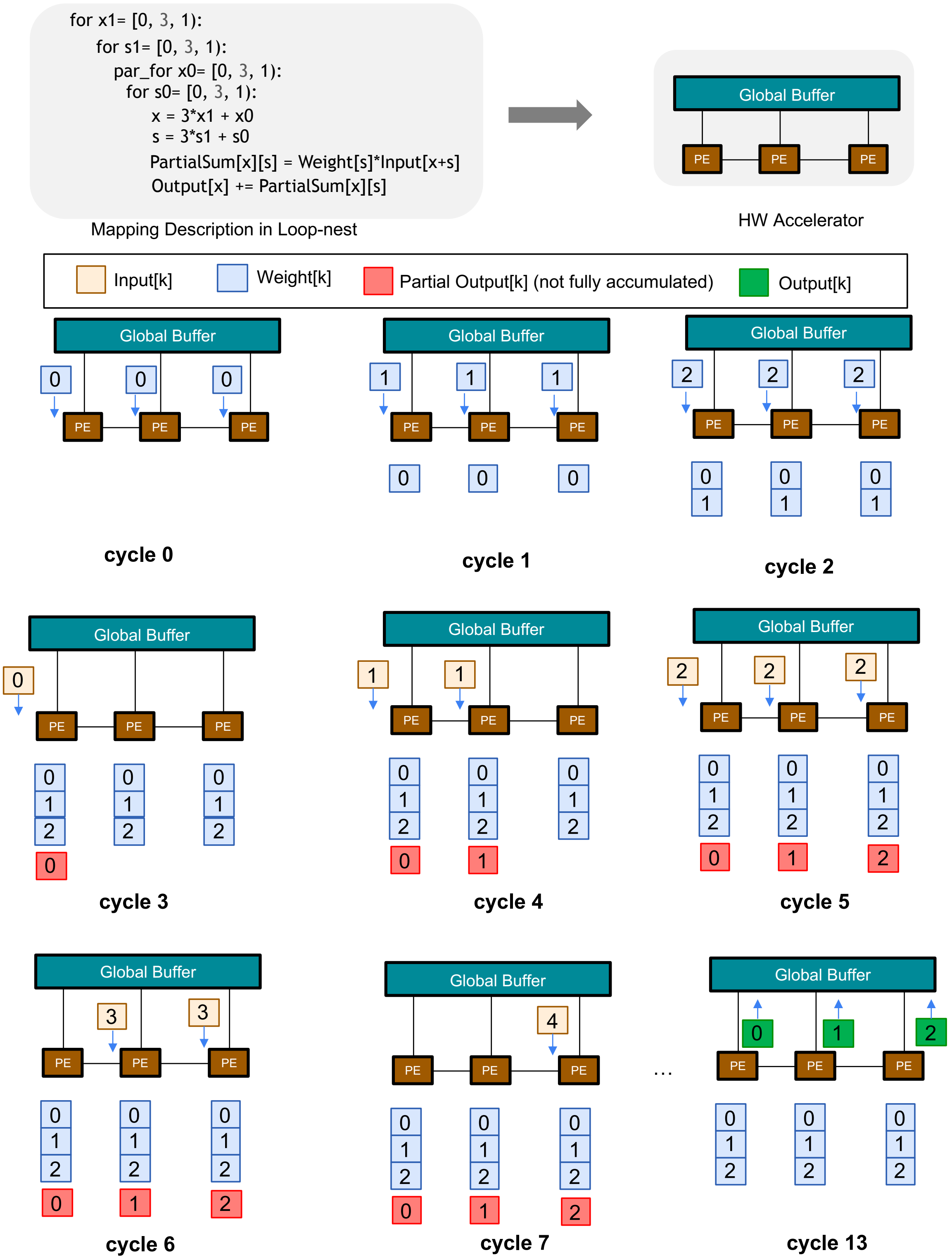
We often use "stationary" to characterize the behavior of the computation [83]. For example "weight-stationry" [80, 89, 90, 91, 92] means we organize the compute order to keep the same slices/tiles of weights on the PE array to be reused across different slices/tiles of inputs and outputs. Similarly, we have "input-stationary" reusing the inputs and "output-stationary" reusing the outputs [81, 93, 94].

We show a weight-stationary mapping in Figure 2.4, and we assume the underlying HW is a 1D systolic array. The weights are pre-fetched onto the local scratchpad of PEs and



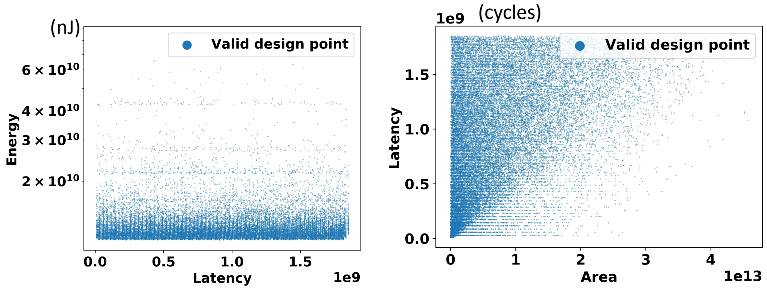
```
for x1= [0, 3, 1):
  par_for s= [0, 3, 1):
    for x0= [0, 3, 1):
      x = 3*x1 + x0
      PartialSum[x][s] = Weight[s]*Input[x+s]
      Output[x] += PartialSum[x][s]
```

Fig. 2.4: A weight-stationary example. This is example is referenced from [88].



```
for x1= [0, 3, 1):
    for s1= [0, 3, 1):
       par_for x0= [0, 3, 1):
         for s0= [0, 3, 1):
             x = 3*x1 + x0
             s = 3*s1 + s0
             PartialSum[x][s] = Weight[s]*Input[x+s]
             Output[x] += PartialSum[x][s]
```

Fig. 2.5: An output-stationary example. This is example is referenced from [88].



(a) Energy-Latency

(b) Latency-Area

	Min	Max
Latency (cycles)	8.18E+06	1.85E+09
Energy (nJ)	1.12E+10	8.34E+10
Power (mW)	1.52E+05	2.24E+09
Area (um^2)	1.77E+09	4.57E+13
SL size (Bytes)	2.62E+02	9.90E+06
SG size (Bytes)	2.82E+03	1.54E+07

(c)HW perf. statistics

Fig. 2.6: The HW performance of randomly sampled HW-mapping in the design space on an example layer (second layer of VGG16). (a) The Energy to Latency plot, (b) The Latency to Area plot, and (c) The HW performance statistics of the sampled HW mappings.

remain stationary. At each cycle, we will stream in the corresponding inputs, which also introduce the partial sum in the PEs. The partial sum will be forwarded to the neighbor PEs to accumulate the result. After all the partial sums are accumulated the output will be stored back to the global buffer.

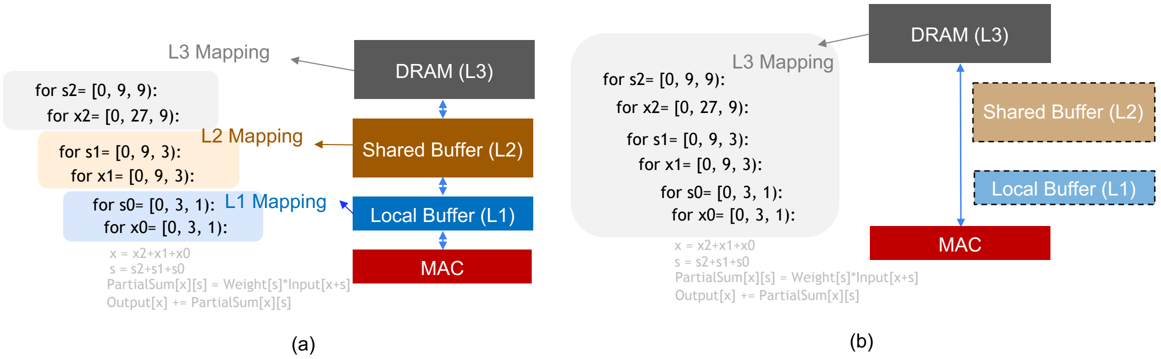
In contrast, we show a output-stationary mapping in Figure 2.5. The weights are pre-fetched onto the local scratchpad of PEs. Then we stream in the inputs. The partial sums are still collecting the multiplication result. However, the difference is that the partial sums remain stationary in each PE. They kept accumulating the results in each PE. Then after all the partial sums are collected, the outputs are stored back to the global buffer.

2.4.3 Performance-efficient v.s. Energy-efficient Mappings

In different use-case, we will target different objectives such as best performance (minimum latency), best energy-efficiency, and so on. Often, we will also use multi-objective such as performance and energy to find the mappings on the Pareto frontier. In the following, we show some illustrative examples to demonstrate what's the factors causing mappings to be performance-efficient, energy-efficient, or both performance and energy-efficient.

The Effect of Buffer Mapping

In Figure 2.7, we show a 3-level mapping. In Figure 2.7(a), each level of mapping is mapped onto one of the buffer level. For example, L3 mapping is used to described mapping from



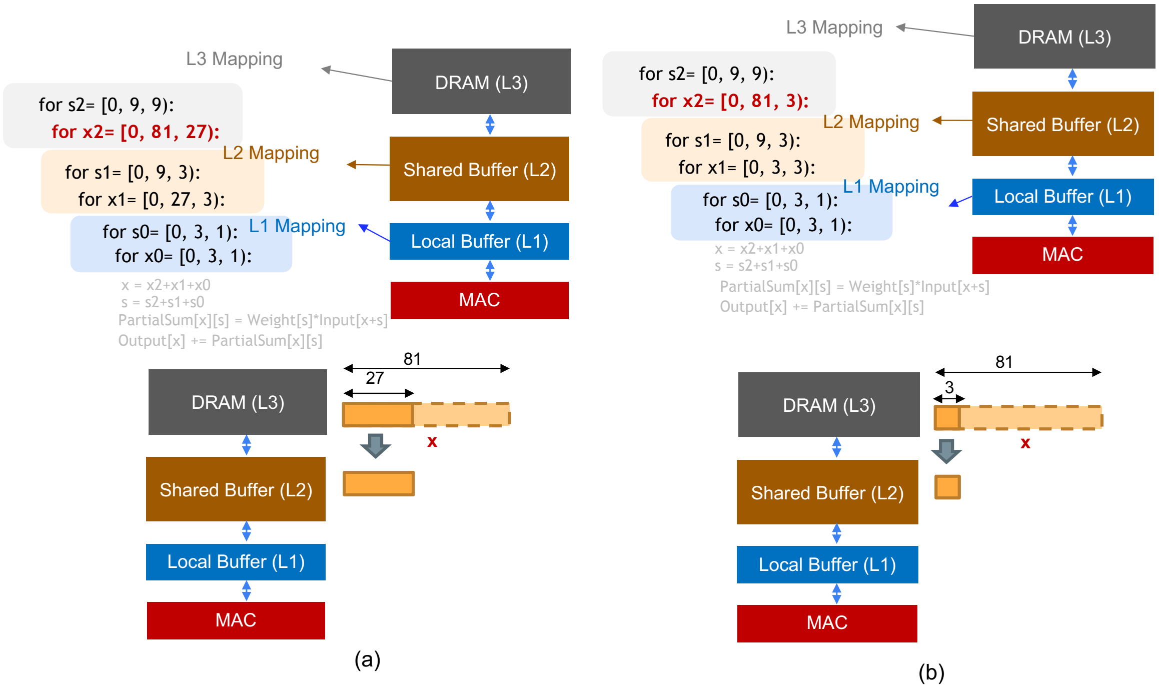
```
for s2= [0, 9, 9):
  for x2= [0, 27, 9):
    for s1= [0, 9, 3):
      for x1= [0, 9, 3):
        for s0= [0, 3, 1):
          for x0= [0, 3, 1):
            x = x2+x1+x0
            s = s2+s1+s0
            PartialSum[x][s] = Weight[s]*Input[x+s]
            Output[x] += PartialSum[x][s]
```

Fig. 2.7: The level of buffer mapping effects. (a) option 1 mapping between MAC and L1, L2 and off-chip, (b) option 2 mapping between MAC and off-chip.

off-chip to shared buffer; L2 mapping is used to described mapping from shared-buffer to local buffer. However, the same 3-level of mapping can also be mapped entirely to the off-chip buffer level, as shown in Figure 2.7(b). The on-chip shared buffer and local buffer are not utilized. The benefit of (b) is that it will hugely reduce the number of memory access of on-chip memory while the number of off-chip access is kept the same. Therefore (b) is a more energy-efficient mapping. However, off-chip memory usually has smaller BW, which might cause this mapping to be off-chip BW bounded and becoming sub-optimal with the respect to (latency) performance. However, the off-chip BW boundedness depends on the workload. If the workload has massive reuse opportunity, the limited off-chip BW could be sufficient for (b) and makes (b) an both energy and performance-efficient mapping.

The Effect of Tile Sizes on Ameliorating BW Bottleneck

In Figure 2.8, we show two mappings with different tile sizes at input (x) dimensions. In (a), we use tile size of 27 at L3 level; in (b), we use tile size of 3 at L3 level. We assume the shared L2 buffer is large enough to hold the tiles introduced by these two different kinds of tiling strategies. Fetching larger tile from off-chip can reduce the frequency to interact with off-chip, less likely to be bottlenecked by off-chip BW. For example in (b), we are frequently requesting new elements from off-chip, the speed of off-chip access need to be on-par with the MAC processing frequency to not be bottlenecked. Therefore, given a fixed sizes of buffer hierarchies, we often see mappings with larger tile sizes have better energy



```
for s2= [0, 9, 9):
  for x2= [0, 81, 27):
    for s1= [0, 9, 3):
      for x1= [0, 27, 3):
        for s0= [0, 3, 1):
          for x0= [0, 3, 1):
            x = x2+x1+x0
            s = s2+s1+s0
            PartialSum[x][s] = Weight[s]*Input[x+s]
            Output[x] += PartialSum[x][s]
```

(a)

```
for s2= [0, 9, 9):
  for x2= [0, 81, 3):
    for s1= [0, 9, 3):
      for x1= [0, 3, 3):
        for s0= [0, 3, 1):
          for x0= [0, 3, 1):
            x = x2+x1+x0
            s = s2+s1+s0
            PartialSum[x][s] = Weight[s]*Input[x+s]
            Output[x] += PartialSum[x][s]
```

(b)

Fig. 2.8: The Effect of tile sizes on ameliorating BW bottleneck. (a) option 1 with larger input tile size, (b) option 2 with smaller input tile size.

and performance efficiency.

The Effect of Tile Sizes on Reuse

Different tile sizes will also introduces different reuse opportunity. In Figure 2.9, we show two mappings with different tile sizes at weight (s) dimensions. In (a), we use tile size of 3 at L3 level; in (b), we use tile size of 9 at L3 level. Note that this is a weight-stationary mapping since we map weight (s) at the outer most loop. In (a), 3 elements of s are fetched on-chip and being reused to multiplied with x. The x will be read iteratively from off-chip to on-chip. Owing to the folding of s, the full x will be read from off-chip to on-chip iteratively by 3 times. In contrast, in (b), the full s is fetched on-chip, and hence x will only need be read from off-chip to on-chip iteratively by 1 times. Reducing the number of memory read from off-chip as in (b) can often makes the mapping more energy-efficient.

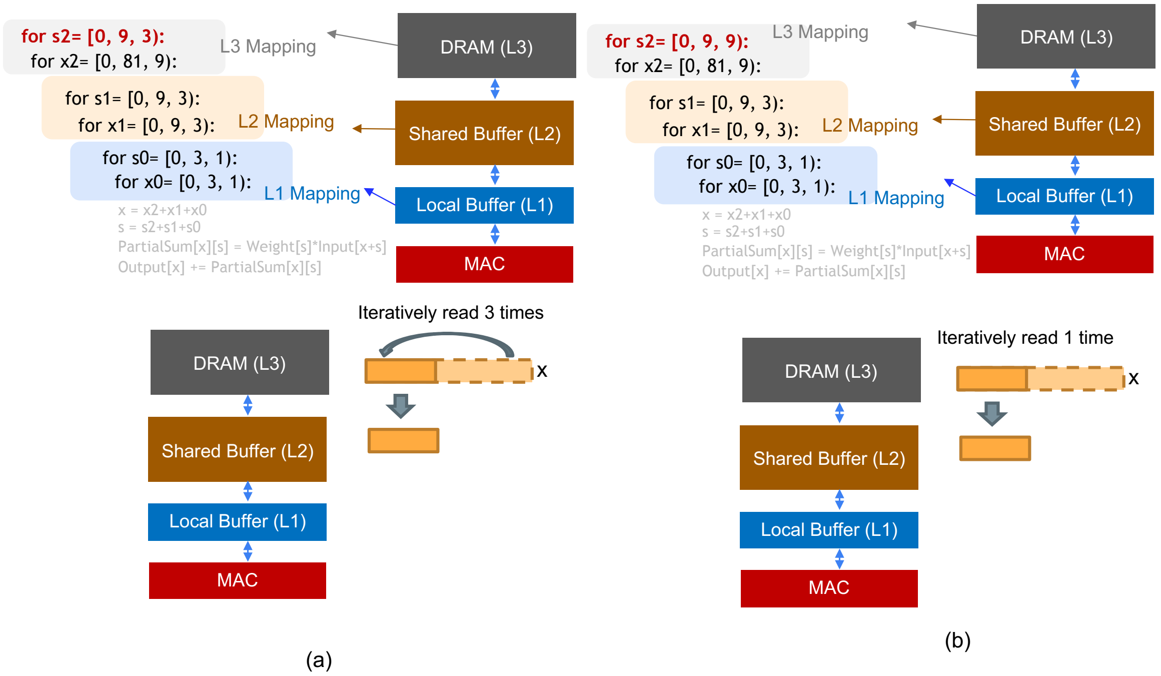


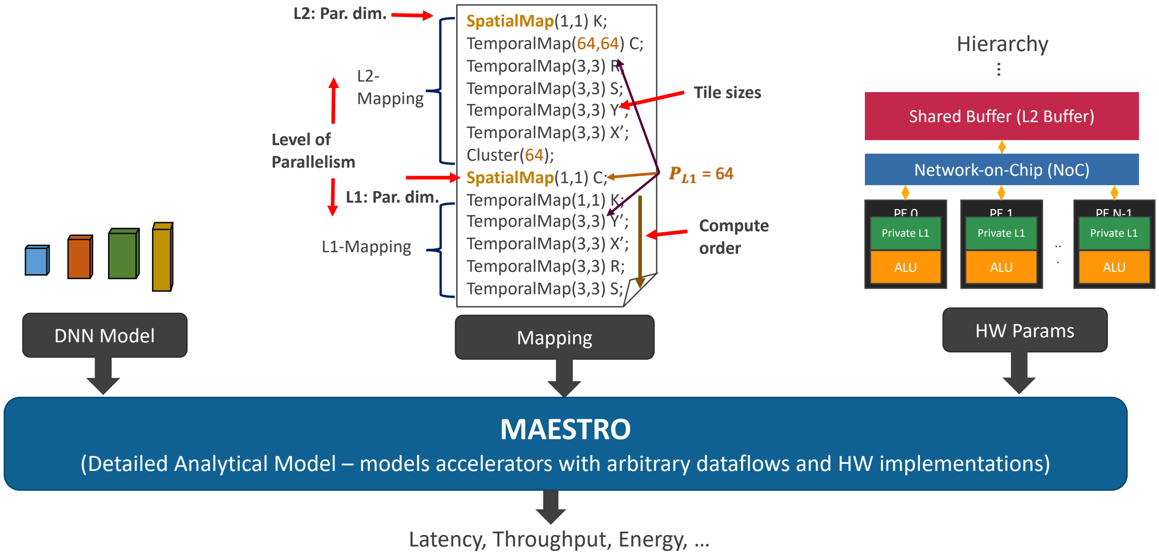
Fig. 2.9: The Effect of tile sizes on reuse. (a) option 1 with larger weight tile sizes, (b) option 2 with smaller weight tile sizes.

2.4.4 Map Space Exploration

In the above examples, we have shown the performance impact of different mapping decisions and how critical a good mapping decision is. To pursue for optimal performance, we often form a mapping optimization framework to search for the optimal mapping. This framework is called – Map Space Exploration (MSE).

2.5 DNN Accelerator Cost Model

Frameworks like MAESTRO [68] and Timeloop [4] use detailed analytical modeling to evaluate different mapping strategies of a DNN on the accelerators. We leverage MAESTRO [68] as our underlying cost model because of its ability to support the target detailed mapping space. It supports most of the common DNN layers such as CONV, depth-wise CONV, and Fully connected. As shown in Figure 2.10, it takes in a DNN model (or layer), a mapping strategy, and the HW parameters of the modeling accelerators and MAESTRO estimates the statistics such as latency, energy, runtime, power, and area. The HW parameters



```
SpatialMap(1,1) K;
TemporalMap(64,64) C;
TemporalMap(3,3) R;
TemporalMap(3,3) S;
TemporalMap(3,3) Y';
TemporalMap(3,3) X';
Cluster(64);
SpatialMap(1,1) C;
TemporalMap(1,1) K;
TemporalMap(3,3) Y';
TemporalMap(3,3) X';
TemporalMap(3,3) R;
TemporalMap(3,3) S;
```

Fig. 2.10: The workflow of MAESTRO [68].

inputs include the number of PE, local and global buffer sizes, NoC latency and bandwidth, memory read/write bandwidth, and so on. A mapping strategy describes tiling, compute order, parallelism, and level of tiling/ parallelism (Figure 2.10), which we will have a detailed discussion in chapter 3.

2.6 Optimization Methods

We introduce the optimization methods we will use in this thesis in the following. In map space exploration, we will call the the optimization methods – Mappers. We categorize them into three broad categories, heuristic-based, feedback-based, and gradient-based (Figure 2.11). The heuristic-based is often the faster to acquire one sample. The feedback-based in general will use learning methods to improve its sampling function to improve its sampling efficiency across time. The gradient-based will train a surrogate model offline to replace the external and unusually non-differentiable cost model such as MAESTRO. Then, the gradient base will use the trained surrogate model to perform gradient updates to optimize its solution. We list different methods in each category as follows.

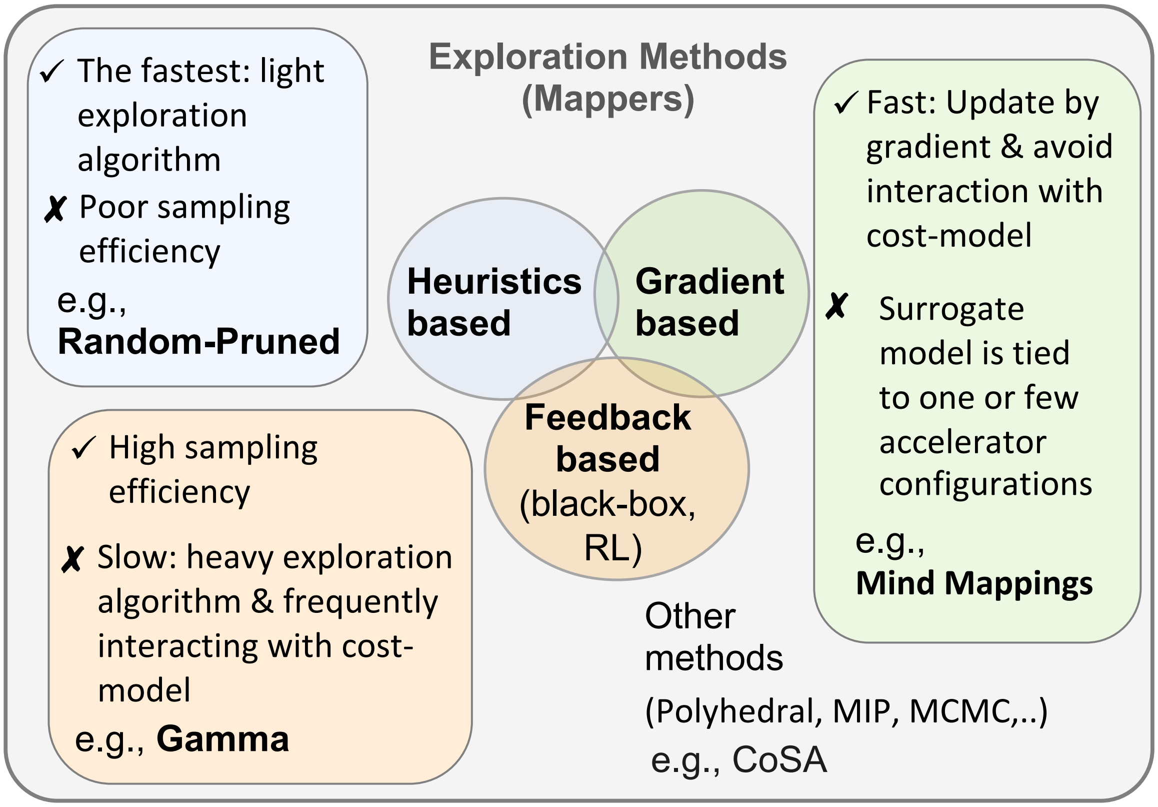

Exploration Methods (Mappers)
✓ The fastest: light exploration algorithm
✗ Poor sampling efficiency
e.g., Random-Pruned
Heuristics based
Gradient based
Feedback based (black-box, RL)
✓ Fast: Update by gradient & avoid interaction with cost-model
✗ Surrogate model is tied to one or few accelerator configurations
e.g., Mind Mappings
✓ High sampling efficiency
✗ Slow: heavy exploration algorithm & frequently interacting with cost-model
e.g., Gamma
Other methods
(Polyhedral, MIP, MCMC,..)
e.g., CoSA

Fig. 2.11: The categories of different exploration algorithms (Mappers) in Map Space Exploration.

2.6.1 Heuristics-based

Exhaustive search will lead to a global optimum but is nearly impossible to sweep for vast design spaces.

Grid search is an exhaustive search with a coarse-grain sampling step, which makes the process approachable.

Random search randomly samples design points in an unknown search space and keeps the best solution. It has been shown to be competitive for optimization problems in various fields [95, 96, 97, 98].

2.6.2 Feedback-based

Simulated annealing [99] adds an exploitation step to random search (which is always exploring). It randomly samples and accepts points that improve the objective, but also with a certain probability accepts points that may worsen the objective. The probability is controlled by a hyper-parameter *temperature*. Higher temperature will increase the probability to accept worse points, causing more randomness, and vice versa. Simulated annealing is used for compiler optimization for CPU and software [100] and also tile and loop scheduling for DNN workload [101].

Genetic Algorithm (GA) [102] is a method where we encode the dimension of each design point as a gene. With all the dimensions specified, a design point is called a genome. We initialize the algorithm with several randomly sampled design points (genomes) and these genomes form a generation. Then we evaluate the fitness of individuals of this generation. We keep well-performing individuals and use them to reproduce the next generation with mutation and crossover. Generation by generation, GA will converge to an optimized point. STOKE [103] and TensorComprehensions [31] use GA to search the space of DNN code optimization.

Bayesian Optimization [104] builds a surrogate for the objective and quantifies the uncertainty in that surrogate using a Bayesian machine learning technique. The optimization method can be constructed by the Gaussian process, Random forest, or Tree Parzen Estimator. It selects the next values to evaluate by applying criteria to the surrogate. It evolves the surrogate model and sampling criterion simultaneously. The concept is to limit the evaluation of the objective function by spending more time choosing the next values for sample efficiency. Some works use Bayesian optimization to search for DNN hyper-parameters [105, 106, 107].

Differential Evolution (DE)[108]. The algorithm flow of DE is similar to GA. The difference is that in GA we mutate the gene by random number. However, in DE we mutate by the *difference vector*, which we explain next. When mutation, we sample two parents and extract their difference on each dimension to formulate a difference vector. Next, we sample another parent and mutate it by adding the difference vector to it. DE is often found converge faster than GA [109], and is one of the most used optimization methods in many applications [110, 111, 112].

(1 + λ)-ES [113]. Evolutionary Strategy (ES) is a branch of EA, which contains only a mutation operator. The (1 + λ)-ES is a strategy, where for each parent we mutate it to reproduce λ number of mutated children. The parent and children compete with each other by their fitness values and the best one will go to the next generation.

Covariance matrix adaptation evolution strategy (CMA-ES) [114]. In CMA-ES, we maintain the mean and covariance matrix of a population, which helps the algorithm to have adaptive step-size when approaching optimum. The algorithm works as follows. We select several best-performing individuals as elites and calculate their mean. We calculate the covariance matrix of the elites with a tweak, where we calculate each variance value by the current elite and the old mean value. The made covariance matrix will reflect the degree of improvement of these elites over the last generation. Large improvement values imply the current populations are still far from optimum, leading to larger variances. Therefore when we sample the next generations by the updated mean and covariance matrix, we would take a larger step, and vice versa. The CMA-ES has been empirically outperforming other optimization methods in many applications [115].

Test-based Population-Size Adaptation (TBPSA) [116] is an augmented CMA-ES that varies the population size along evolution to tackle noisy environment. We estimate the trend of the population's fitness values. When the fitness values start to converge, the algorithm will adapt to have a smaller population size. When the environment is noisy and fitness values start to have a larger fluctuation, the algorithm will adapt to have a larger size again. TBPSA is found more robust in noisy environments [116].

Particle Swarm Optimisation (PSO) [117]. In PSO, the basic unit in the population is called particle. Different from GA, each particle only tracks the global best (gbest), which is the best performing particle (individual) in the population and has no direct interaction (crossover) with others. It updates its parameter by gbest and the self best, or so-called parent best (pbest), which is the best performing self across generations. The gbest and pbest enable particle to explore global and local optimum respectively, which update its parameter x as follow:

$$v = \omega v + C_p \times r_1 \times (pbest - x) + C_g \times r_2 \times (gbest - x) \tag{2.1}$$

$$x = x + v \tag{2.2}$$

The r_1 and r_2 are random number sampled from normal distribution $N(0,1)$. The C_p and C_g are the set hyperparameter to focus more on local or global information. ω is the hyperparameter that controls the momentum of the update. This simple optimization scheme is widely used in the optimization problems [118, 119].

Passive Portfolio (pPortfolio)[120] A well-diversified portfolio is important for optimization. The pPortfolio strategy is to maximize diversity by spreading the risk broadly to multiple instances. For example, we launch two (or K numbers of) optimization methods (CMA-ES, DE), each with a half (or 1/K numbers) of the total sampling points, and use the best performing solution provided by one of them. The pPortfolio is the opposite of an active strategy (in which we keep optimizing one algorithm) and is sometimes a better strategy for its diversity [120].

Reinforcement Learning Reinforcement Learning (RL) algorithms are often used in games [121, 122] as they are useful in sequential decisions. More formally, this is a Markov decision process (MDP). In this thesis, we show that determining the appropriate number of PEs and buffers for a series of DNN layers to minimize the overall platform latency/energy while staying within an area or power budget can be viewed as a MDP. Therefore we find RL algorithms to be a promising approach for this problem to increase the sample efficiency of the search, compared to baseline optimization methods that use no information from the current state. The goal of an RL agent is to continuously interact with an **environment**, observe the current **state**, take one or more **actions**, observe the **reward** from the environment, and update its underlying **policy network**. With time, the policy network learns to predict actions that can maximize reward. We discuss our RL-based HW resource exploration next.

2.6.3 Gradient-based

Gradient-based [6] trains a neural network-based surrogate model via offline sampling of millions of data points collected from the cost model. It uses the loss gradient to update its solution. During MSE, it utilizes gradient-descent on this surrogate model to find mappings, instead of searching.

2.7 Summary

In this chapter, we have introduced the minimum background for the following chapters, including the background of DNN accelerator, the concept of DNN mapping, the Map Space Exploration (MSE) framework, and the different optimization algorithms that are heavily used in the MSE. These background can cover the related topics discussed across different chapters. In the following, we will have a deep dive into different part of these, e.g., chapter 3 focuses on the discussion of DNN accelerator and mapping, chapter 4 focuses on optimization algorithm design, chapter 5 focuses on analysis of MSE framework, and so on.

CHAPTER 3

FLEXION: A FORMALISM OF DNN ACCELERATOR FLEXIBILITY AND THE IMPLICATION OF MAP SPACE

Traditionally, the efficiency of domain-specific accelerator ASICs has come from *specialization*, i.e., the control path and datapath in the accelerator are tailored to the deep neural networks (DNNs) that are expected to run on the accelerator. In other words, the number of mappings that an accelerator can support (aka *map-space*) is restricted.

Specialization is unfortunately a double-edged sword. While high specialization enables DNN accelerator ASICs to provide higher performance and better energy-efficiency than CPUs and GPUs, it restricts the accelerator from adapting to the growing diversity in DNN models today. This is becoming a key challenge, given how rapidly the field of ML is evolving, since it is highly probable that accelerator chips will be obsolete by the time they reach the market.

To this end, there has been growing interest in developing *flexible* DNN accelerators[1]. Flexibility provides a two-fold benefit. First, it allows the accelerator to better tailor itself to the diverse set of layer parameters within the current DNN being mapped [46, 47], instead of targeting the average case. For example, the CONV2_1 early layer in ResNet-50 has 56x56 activations with 64 channels, whereas the CONV5_3 late layer in ResNet-50 has 7x7 activations with 2048 channels. An accelerator that only supports parallelism on activations can result in severe under-utilization on the late layer, and vice versa for one that only supports parallelism on channels. Second, it helps "future-proof" the accelerator. For instance, consider an accelerator such as Eyeriss [124], developed in 2014 for accelerating AlexNet [125], the state-of-the-art network at that time for image classification, which had 3x3, 5x5 and 11x11 filters; Eyeriss is severely under-utilized for newer DNN models that

[1] This chapter builds on the idea of Flexion developed by Hyoukjun Kwon [123]

use layers such as pointwise (i.e., 1x1 filters) convolutions (e.g., MobileNetV2 [126]), as demonstrated by its successor Eyeriss_v2 [48], or attention (e.g., BERT [127]).

While the notion of flexibility described above makes intuitive sense, the field of DNN accelerators today is inconsistent about the definition. We find that “flexibility” has been used loosely to refer either to the ability to handle different loop tiling/blocking [10], and/or support for different loop orders [12], and/or the ability to spatially partition across different dimensions [48], and/or the ability to logically support different PE aspect ratios [49]. In fact, there is also inconsistency in whether flexibility is a hardware feature [46] or simply a term for compiler loop-transformations over a fixed inner tile [10, 50]. This in-formalism is a serious barrier to the adoption of flexibility features, as it hinders precise quantification of the cost/benefit tradeoff. Moreover, the addition of large map-spaces confounds the accelerator Design-Space Exploration (DSE) problem as concepts such as *partial* flexibility cannot be precisely expressed or numerically assessed by existing DSE flows.

In this chapter, we present a novel formal definition and precise quantification of accelerator flexibility. We also quantify the corresponding cost for flexibility, including area and energy. We then couple that with an abstraction that allows its incorporation into a first-of-its-kind *flexibility-aware* automated DSE, without making the problem intractable. As a debut work in this field, we cannot claim to be exhaustive. However, we advocate that our formalism can be the first step toward the accelerator community moving towards consensus on the costs and benefits of flexible accelerators.

3.1 Axes of Accelerator Flexibility

We define accelerator flexibility as *“how many different ways can we compute this workload.”* In other words, flexibility aims to describe a device’s ability to run different mappings (i.e., re-arrangements or transformations) of the same starting loop nest, as opposed to unrelated loop nests from different programs (which might be a measure of *programmability*).

We follow a bottom-up approach toward understanding flexibility. In this section, we

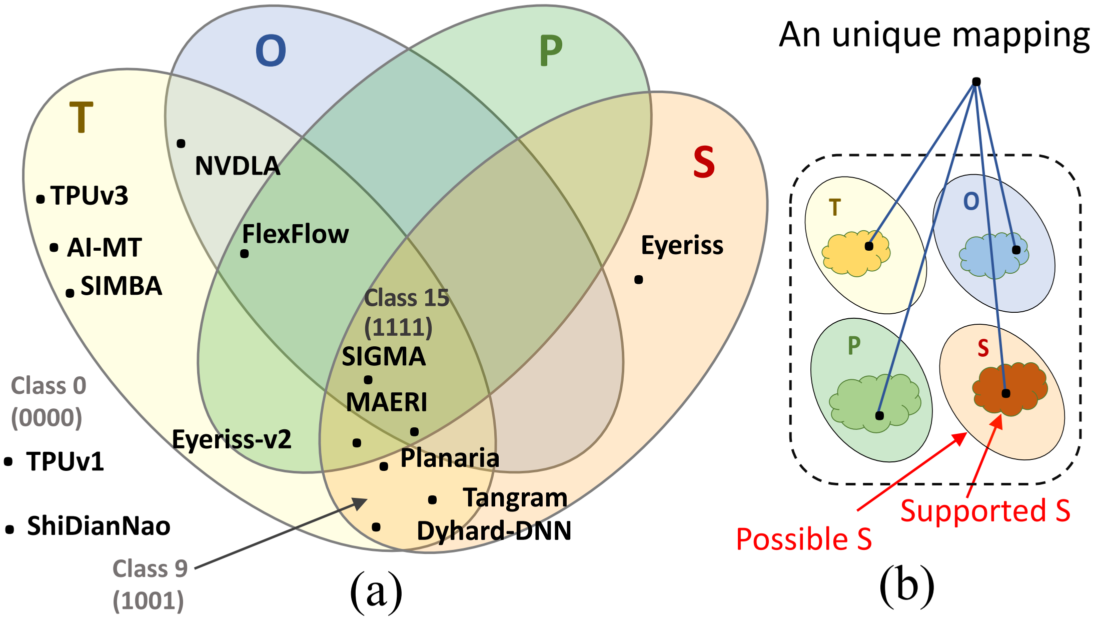


Fig. 3.1: (a) Flexibility TOPS and prior work. (b) For any workload, a unique mapping involves choosing individual points from the map-space actually supported by the target accelerator.

identify axes of flexibility and qualitatively discuss the performance benefit and HW cost for supporting each. In §3.2, we identify the map-spaces exposed by each axis. In §7.2, we discuss our mechanism to search through these flexibility-constrained map-spaces.

3.1.1 Axes Taxonomy

Informed from existing work of accelerators [128, 3, 129, 130, 131, 80, 132, 48, 81] and map space exploration tools (MAESTRO [128], Timeloop [47], TVM [24], and so on [10, 133]), where intra-layer mappings are captured as loop nests, we identified and unified the knobs that these tools vary when doing map-space exploration. We distill flexibility into four axes[2].

(1) Tile sizes (T): The ability to change bounds and aspect ratios of data tiles from one or more operand tensors per level of the buffer hierarchy [84].

(2) Loop order (O): The ability to change the loop orders iterated per tiling level.

(3) Loop parallelization (P): The ability to change which tensor dimensions are parallelized per tiling level. This represents the *spatial* partitioning of data (i.e., across PEs).

[2] We only consider intra-layer mapping (which most accelerators target) and did not consider inter-layer mapping.

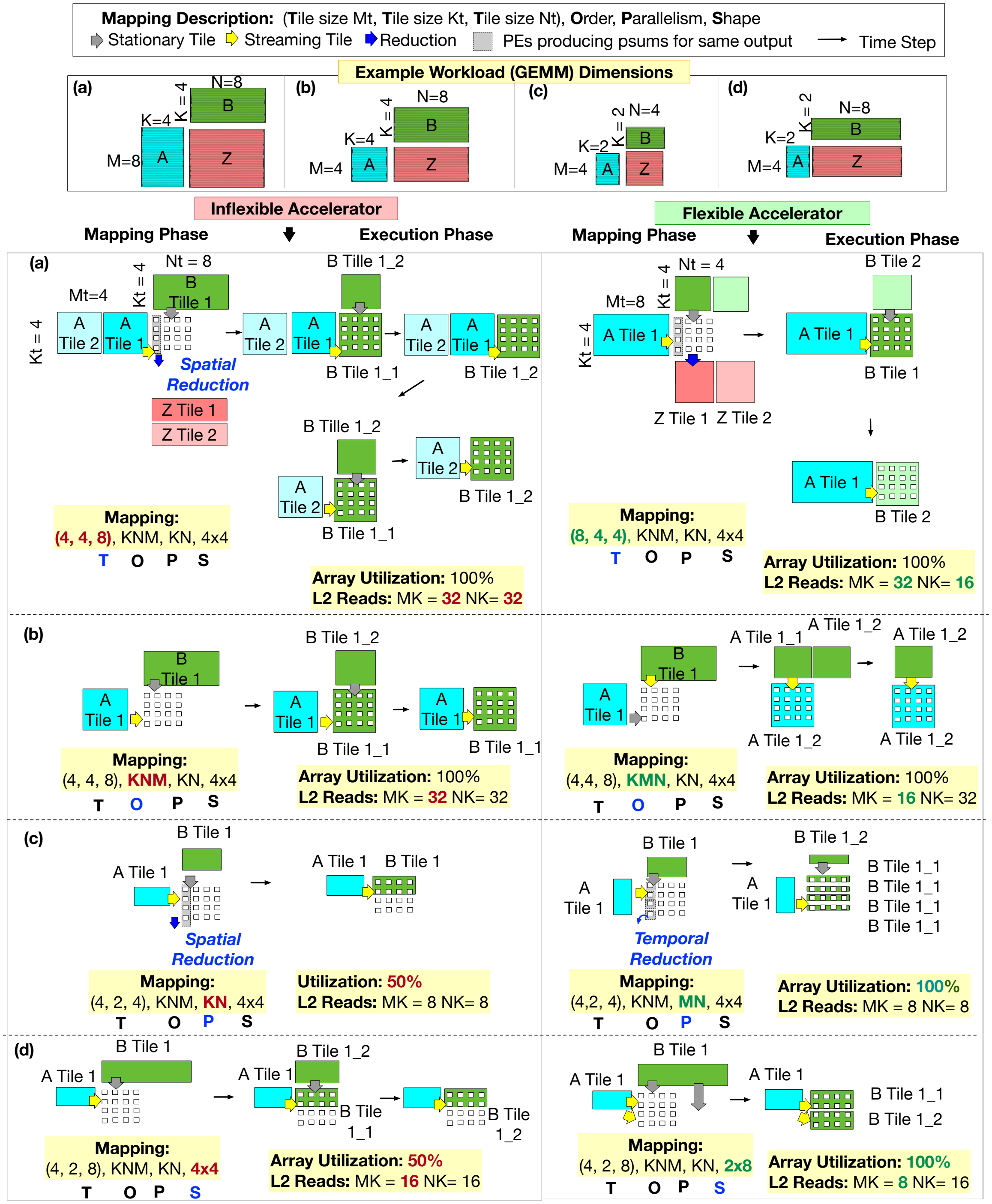
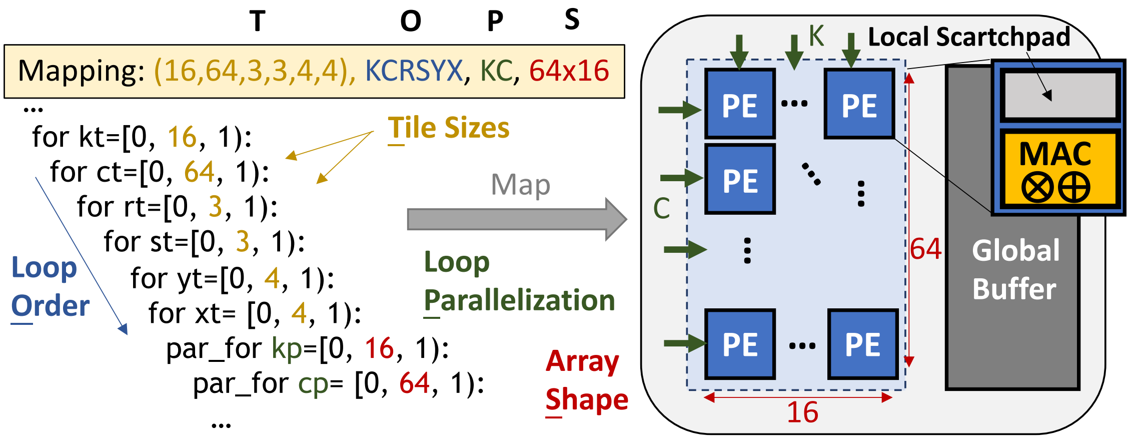


Fig. 3.2: The benefit of having flexibility in (a) Tile, (b) Order, (c) Parallelism, and (d) Shape. Example GEMM operation of $Z_{MN} = A_{MK} \times B_{KN}$ on 4x4 PE array. The subscript t indicates that these are tiles of the overall computation. The inflexible accelerator is weight stationary (tensor B), using a loop ordering of K → N → M.



Mapping: (16,64,3,3,4,4), KCRSYX, KC, 64x16

```
...
 for kt=[0, 16, 1):
   for ct=[0, 64, 1):
     for rt=[0, 3, 1):
       for st=[0, 3, 1):
         for yt=[0, 4, 1):
           for xt= [0, 4, 1):
             par_for kp=[0, 16, 1):
               par_for cp= [0, 64, 1):
                 ...
```

Fig. 3.3: DNN accelerator with NVDLA-style mapping.

(4) Array shape (S): The ability to change the logical shape and clustering of the accelerator's hardware resources. This determines the number of tiling levels and the maximum tile sizes for the tensor dimensions being mapped in parallel (i.e., spatially) over the accelerator array.

We refer to the four axes as flexibility TOPS (tiling, order, parallelism, shape), as a mnemonic with the well-known performance metric Tera-Operations Per Second. Figure 3.3 shows an example of NVDLA mapping [80], where in the 6-order for loop of convolution, tile sizes are (16, 64, 3, 3, 4, 4), loop order is (K, C, R, S, Y, X), loop parallelization is (K, C) parallelism, and array shape is (64x16).

Scope of Classification. Note that the four axes we described exhaustively capture the space of *intra-layer mappings* which has been the target of several recent mapping optimizers [6, 134, 135, 7, 47]. We assume that an accelerator runs each layer sequentially, consistent with state-of-the-art DNN accelerators [132, 124, 8, 81, 80, 3, 49], and flexible accelerators can change the mapping alone one or more of these axes for each layer. Note that inter-layer mapping strategies (e.g., loop fusion [136]) can be a outer-loop on top and not the focus of this study. Further, aspects of DNN accelerators tangentially related to the flexibility that we explicitly leave to future work include sparsity and compression, bit-serial arithmetic, and alternate algebraic refactorings such as Winograd [137] or UCNN [138], and processing-in-memory and near-memory architectures.

3.1.2 Classes of Accelerator Flexibility

We introduce a notation to concisely describe an accelerator as a binary vector $[X_T, X_O, X_P, X_S]$ where:

$$X = \begin{cases} 0 \text{ if } |Mappings_{supported}| = 1 \text{ (inflexible)} \\ 1 \text{ if } |Mappings_{supported}| > 1 \text{ (some flexibility)} \end{cases} \tag{3.1}$$

$|Mappings_{supported}|$ is the number of legal mappings for a given workload (i.e., DNN layer) over a given accelerator.

Our taxonomy creates 16 accelerator classes - with Class 0 (i.e., $[X_T, X_O, X_P, X_S] = [0,0,0,0]$) being the least flexible and Class 15 (i.e., $[1,1,1,1]$) being the most. The intent of this taxonomy is to enable a coarse classification of accelerators and allow a systematic study of cost-benefit analysis. In Figure 3.1(a), we do a best-effort classification of several current accelerators into these classes. Figure 3.1(b) then demonstrates the role of a compiler/mapper: it generates unique mappings by selecting from the subset of the map-space that is actually supported by the target accelerator [47].

3.1.3 Benefits of Mapping Flexibility Per Axis

In Figure 3.2 we show how each axis of flexibility can enhance performance and/or energy-efficiency depending on the workload dimensions. For simplicity, we use a GEMM mapped on a 4x4 array throughout the example.

In Figure 3.2(a), suppose we have a GEMM with (M=8, K=4 and N=8). It is tiled such that the size of A tiles (i.e., input activations) is 16 (M_t=4,K_t=4), and that of B tiles (i.e., weights) is 32 (N_t=8,K_t=4) - assuming this constraint comes from the sizes of the input and weight buffers around the array. The 32 weights will require two iterations or passes through the 4x4 PE array during which time 16 of the weights will remain stationary; the two sources of inefficiency are (i) the same inputs will be read from the global buffer and streamed during each iteration (increasing energy), and (ii) switching the stationary tile will lead to a stall. Now consider a flexible accelerator, where the global buffer is soft-partitioned and can be configured to store a tile of 32 inputs (M_t=8,K_t=4) and 16 weights (N_t=4,K_t=4) instead. Now, the 16 weights will remain stationary as the 32 inputs stream. Here only one iteration (although longer) will be needed, only one L2 buffer read for each input required and no stall to switch stationary tiles, potentially reducing energy and runtime.

In Figure 3.2(b), the A tile has size 16 while the B tile has size 32. Keeping the B tile stationary requires two iterations and two reads of the A tile, leading to a similar issue as Figure 3.2(a). In contrast, if we could switch the loop order to keep the input tile stationary instead as the flexible accelerator does, we can finish the computation in one iteration with fewer buffer reads, enhancing performance and energy-efficiency.

In Figure 3.2(c), suppose the GEMM is not square, but rectangular. In the default mapping discussed so far, the parallelism is across the K and N dimensions which leads to a utilization of only 50%. Here the K dimension is reduced spatially across the columns (like the TPU [132]). If the accelerator could support flexible parallel dimensions, then we could switch to MN parallelism (i.e., output-stationary dataflow), performing the reduction temporally within each PE. Now each PE operates on a separate output and we get 100% utilization.

In Figure 3.2(d), we see that the dimensions of the weight tile (2×8) do not match those of the 4x4 array, leading to two iterations, each with 50% utilization. Flexibility in the shape of the physical array would enable it to emulate a logical 2×8 aspect ratio instead, increasing utilization to 100%.

As shown in Figure 3.2, different axes of flexibility provide different opportunities to enhance utilization and/or reuse. In fact, in some cases more than one choice exists for enhancing the utilization. E.g., the benefit achieved by flexible shape for the tall-skinny GEMM in Figure 3.2(d) could also be achieved by switching to MN parallelism like

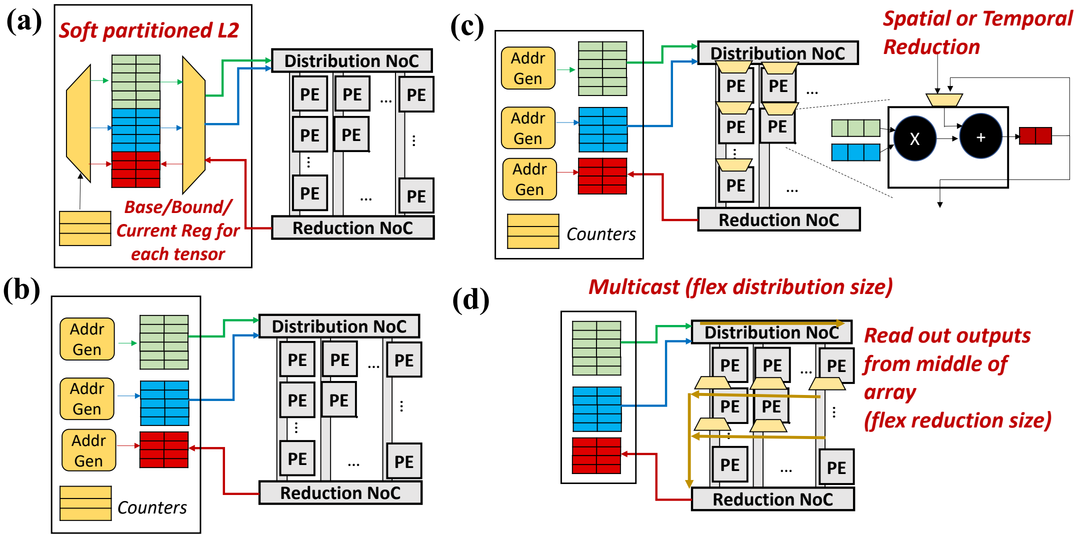


Fig. 3.4: HW Overhead for Flexibility Support (highlighted in yellow). (a) Flexible Tile Size HW Support: Soft-partitioned Scratchpads. (b) Flexible Loop Order HW Support: Configurable Address Generators. (c) Flexible Parallelism HW Support: Spatial and Temporal Reduction. (d) Flexible Shape HW Support: NoC support for PE clustering.

Figure 3.2(c). This motivates this paper's focus on dissecting the performance and energy benefits along each axis of flexibility.

3.1.4 Hardware Support for Flexibility Per Axis

However the flexibility is not free. It comes with an area cost, as we characterize later in Table 3.2. For example, a multi-casting circuit takes a larger area than a uni-cast; flexible choices of inputs introduce multiplexers whose area cost grows with number of choices. We characterize the specific hardware cost of each axis of flexibility via Figure 3.4 as follows.

Tile: To support tile flexibility, the accelerator needs to be able to access different tiles in the buffer. As shown in Figure 3.4(a), this costs additional Base (memory start point), Bound (data length), and Current (the current address pointer) Registers for input, weights, and outputs buffers, which are controlling/monitoring the data flow of accessing current tiles. In addition, to be able to support more varieties of tile shapes, the accelerator should have a software-controlled soft-partitioned buffer (instead of hard-partitions across each operand) to enable more tiling opportunities, shown in Figure 3.4(a) via a multiplexer/demultiplexer.

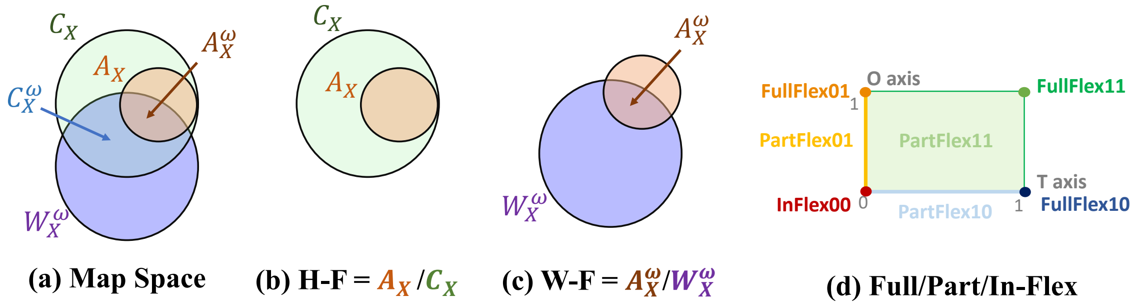
Order: To support different loop orderings, the accelerator needs to allow tiles from different tensors being stationary versus streaming. It needs to have additional address counters and address generators for inputs, weights, and outputs, as shown in Figure 3.4(b). Each PE also needs a register to store the counter value it should count up to before discarding the stationary tile.

Parallelism: To support different parallelism dimensions, we need three additional address counters and address generators. In addition, we need a dedicated multiplexer in front of each PE to select between a spatial or temporal reduction path, as shown in Figure 3.4(c).

Shape: Shape flexibility requires the ability for the accelerator to mimic (i) a larger row/column by sending the same operand across multiple rows/columns and/or (ii) a smaller row/column by breaking into multiple parts. E.g., in Figure 3.2(d), the flexible accelerator runs two spatial reductions of size two on the same column. Feature (i) needs a richer distribution NoC topology to multicast the operand across multiple rows/columns. Feature (ii) requires demuxes at the output of each PE to allow it to forward the output to the neighbor or directly to the L2 buffer via a reduction NoC that provides these connections, as Figure 3.4(d) shows. Prior work [46, 49] has discussed implementation choices for such NoCs.

We wish to highlight that beyond the area and power cost incurred by the components described above (which we find in our evaluations to not be very significant over the MAC, buffer, and NoC required by an inflexible accelerator), a key cost for adding flexibility support is that of *engineering and verification*. This is why a systematic understanding of the benefits of flexibility is valuable for DNN accelerator developers.

(a) Venn diagram of map space of a class-X target accelerator. (b) Hardware-dependent Flexion. (c) Workload-dependent Flexion. (d)A 2D example of the relationship between flexibility axes, flexibility class, and degree of flexibility. Definitions are in Table 3.3.



(a) Map Space **(b) H-F = A_X / C_X** **(c) W-F = A_X^ω / W_X^ω** **(d) Full/Part/In-Flex**

Fig. 3.5: 0.85

3.2 Formal Flexibility-Constrained Map Space Definition

Given the above insights, we now formalize how hardware flexibility and workload specialization precisely affect the feasible map space.

3.2.1 Definitions of Map Space and Flexibility

Table 3.3 summarizes the key definitions. We omit the mathematical equations in the interest of space, instead use the Venn diagrams in Figure 3.5 to guide the discussion.

Target accelerator: The accelerator whose map-space is being evaluated.

Workload: Within the scope of our evaluations, this refers to a given layer of a given DNN model, since we consider the accelerators mapping layers one by one. The venn diagrams and definitions from this section will however also hold for workloads referring to full models or fused layers.

Mapping: A design-point in the map space, which precisely describes the value for T, O, P, and S.

Class-X: We use class-X to denote the class of the target accelerator (§3.1.2). Note that accelerators with different configurations of HW resources such as number of PEs, buffers, and NoCs are all called class-X accelerators as long as their supporting flexibility axes are categorized into X class. Note that the following definitions apply to all of the 16 accelerator classes, i.e., we can use it to discuss map-space and flexibility of class-0011 with two axes enabled, class-1111 with four axes enabled, and so on.

Map space: the design space of the mapping of a class-X accelerator. E.g. the map space of class-0011 will expand along the two flexible axes P and S, while T and O are fixed values across all mappings. These fixed values are often practically hard-coded at accelerator design-time for simplifying hardware design (§3.1.4).

Hardware-dependent Map Space: We define two hardware-dependent map-spaces: C_X and A_X.

C_X is used to denote the map space of a class-X accelerator *given a constraint in total HW resources available*. Therefore, C_X is a constrained map space. Taking Class-1xxx accelerators as an example, the total buffer size within the accelerator will directly limit the number of possible tiling choices. Similarly, for Class-xxx1, the total number of PEs are the constraint. No such resource constraints exist for order and parallelism. C_X can be viewed as the map-space of a fully-flexible accelerator within that class. Note that C_X is *workload-agnostic*. For e.g., C_{1000} captures all possible tile sizes that can fit in the buffers, agnostic of tile sizes that make sense for the workload (as we discuss later in this section). Thus in Figure 3.5, it captures a map-space beyond what the specific workload needs.

Given the same HW resource constraint, the target accelerator in class-X might add *additional constraints*, creating its own map space A_X. A_X will always strictly be a subset of C_X, as shown in Figure 3.5. For e.g., given a total buffer size, an accelerator with hard partitioned buffers for inputs, weights and outputs will cover a smaller map-space (i.e., A_X) than theoretically possible in one that provides full flexibility in partitioning the buffers (i.e., C_X).

Workload Map Space: This denotes the possible mappings for a given workload ω regardless of underlying HW, noted as (W_X^{ω}). For example, for a CONV layer in ResNet50 (CONV2_1), with 64 output and 64 input channels, 3x3 weight kernel, and 56x56 activations, the entire workload map space is all the possible combinations of 6 parameters with values from 1 to its dimension size (e.g., 1-64 for output channel). However, not every tiles can fit into the limited on-chip buffer (whose map space is captured by C_X). Therefore, for C-1xxx class, C_X often cannot cover the full W_X^ω, as Figure 3.5 shows.

Workload-dependent Map Space: This is the intersection of hardware-dependent map space (C_X and A_X) and the workload map space (W_X^ω), which creates C_X^ω and A_X^ω in Figure 3.5.

Feasible Map Space: A_X^ω, the map space of the target accelerator on a given workload, is the actual map space the mapper is allowed to explore. The size of A_X^ω is the value of $|Mappings_{supported}|$ in Equation 3.1.

***Flexibility Fraction or Flexion*:** The fraction of supported map space to the possible map space.

Hardware-dependent Flexion (H-F): Accelerators in the same flexibility class can be implemented with different degree of flexibility, whose feasible map space percentage is A_X/C_X. We call this hardware-dependent Flexion (H-F).

Workload-dependent Flexion (W-F): Different workloads will have different shapes and sizes of the map space. To understand how much portion of the map space the target accelerator can support for the current workload, we define workload-dependent Flexion (W-F) as A_X^ω/W_X^ω.

3.2.2 Full and Partial Flexibility

Any accelerator in class-X can have a grey-scaled degree of flexibility support (0¡ flexibility degree ¡=1) . This is illustrated for two axes of flexibility (for simplicity) in Figure 3.5(d). For ease of referring, we use the terms FullFlex-X, PartFlex-X and InFlex-X in our evaluations in §3.4 and §3.5. For e.g., FullFlex-0001 is an accelerator that supports all possible shapes S (such as MAERI [49]), while PartFlex-0001 is an accelerator that supports a subset of shapes S (e.g., Eyeriss [124] to simplify hardware), and InFlex-0001 supports no flexibility in shape (such as the 128x128 systolic array within TPU-v3 [132])[3].

[3] InFlex-0001 is equivalent to InFlex-0000 but we make the S-bit high for ease of comparison against the partial and fully flexible versions in that class.

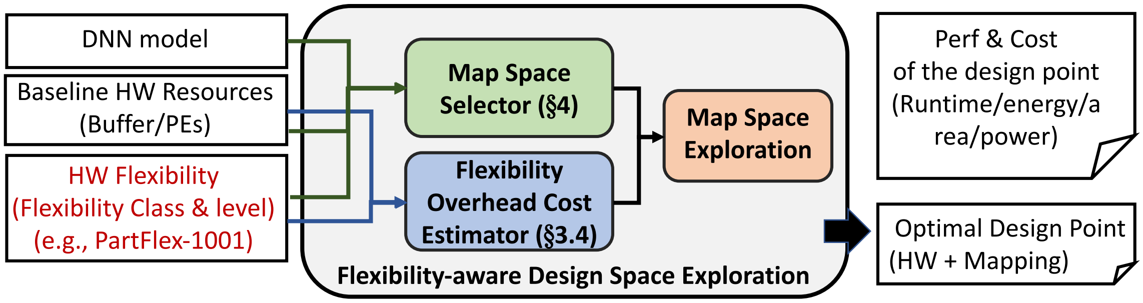

DNN model
Baseline HW Resources (Buffer/PEs)
HW Flexibility (Flexibility Class & level) (e.g., PartFlex-1001)
Map Space Selector (§4)
Flexibility Overhead Cost Estimator (§3.4)
Map Space Exploration
Flexibility-aware Design Space Exploration
Perf & Cost of the design point (Runtime/energy/area/power)
Optimal Design Point (HW + Mapping)

Fig. 3.6: Flexibility-Aware Automated Design-Space Exploration Framework.

3.3 Flexibility-Aware Design-Space Exploration

Ideally, the above formalism is (A) precise enough to capture a quantifiable constraint on the shape of the map-space, and (B) abstract enough not to make the DSE problem intractable. To demonstrate this, we extend an open-source DNN mapper, GAMMA [134, 59], which uses a genetic algorithm to search through a large search-space ($\sim 10^{24}$ [134]) in a sample-efficient manner. We use the same genetic search technique for DSE and MSE where applicable.

Modules for Flexibility-awareness. The native mapper only supports either inflexible accelerators (accelerators in class-0000 aka InFlex-0000) or fully flexible accelerator in class-1111 (FullFlex-1111). We extend it with two additional features: (i) Ability to constrain the search within 16 different accelerator classes (i.e., FullFlex-xxxx). (ii) Ability to constrain the search further in each class for PartFlex accelerators.

These features are implemented by adding different levels of constraints to the map space of the native mapper. It includes modules for understanding the flexibility specification from input, encoding flexibility into constraints and updating the mapping exploration operators [134] correspondingly[4].

Modules for Area, Power for Estimating Cost of Flexibility. To estimate hardware overhead, we implemented RTL of the various components described in Figure 3.4, synthesized them using Synopsys DC with Nangate 15nm library [139] and used Cadence Innovus

[4] We do not go into engineering details in the interest of space, but will open-source the mapper upon paper acceptance.

Table 3.1: Hardware resources and baseline flexibility TOPS.

HW Resources	Number of PEs: 1024, on-chip buffer: 100KB
Mapping Config	**T**: [K:64,C:16,Y:3,X:3,R:3,S:3] tile size, **O**: KCYXRS order, **P**: K-C parallel, **S**: 16x64 PE array

Table 3.2: Area cost of accelerators with different flexibility.

	InFlex	T-Flex	O-Flex	P-Flex	S-Flex	PartFlex	FullFlex
Area μm^2	736,843	763,874 (+0.004%)	738,458 (+0.21%)	737,720 (+0.11%)	737,055 (+0.02%)	738,209 (+0.19%)	739,576 (+0.37%)

for place-and-route. We synthesized the SRAM buffers with SAED32 education library from Synopsys, and scaled it to 15nm. The final cost model takes in the mapping constraint file to understand the required flexibility of the accelerator and outputs its area/power/energy cost. We modularized these building block and build a cost model, which is embedded in flexibility-aware DSE (Flexibility overhead cost estimator in Figure 3.6).

Toolflow. As shown in Figure 3.6, the flexibility-aware DSE takes in DNN model description, baseline HW resources, and HW flexibility specification. Three of them together defines (or so-called selects) the map space that the internal MSE tool works on. After its internal MSE tool converges or reaches a pre-defined number of sampling budget[5], it terminates the optimization and outputs the best-found design point and its HW performance (runtime, energy, area, and power).

3.4 Evaluation I: Isolation Study on Flexibility Axes

In this section, we demonstrate how architects could utilize the flexibility-aware toolflow to tractably explore different map-space constraints on candidate designs. Simultaneously, we leverage the toolflow constraints to target the DSE in order to isolate and quantify the contributions of the T/O/P/S axes.

[5] Throughout the experiments, we set the genetic algorithm hyper-parameter as 100 populations, 100 generations, and thus having 10K sampling budget. The mutation/crossover rate and execution rate of other evolving functions as 0.5. We set these by applying a hyper-parameter search [140].

3.4.1 Methodology

Workloads. We use MnasNet [141], which is a high speed and high accuracy model found by Neural Architecture Search (NAS), for our evaluation, since networks found by NAS have been recently achieving state-of-the-art performance.We also look into three types of models: Vision (Alexnet [125], Resnet50 [142], MobilenetV2 [126]), Language (BERT [127]), and Recommendation (NCF [143], DLRM [144]).

Target Accelerators. For the targeted isolation studies, we formulate three accelerators with different levels of flexibility: i) an inflexible accelerator (**InFlex**), using a fixed and unchangeable TOPS configuration, ii) a partially flexible accelerator (**PartFlex**) with some constrained capabilities that we describe in the relevant sections, and iii) a fully flexible accelerator (**FullFlex**), which can support the full map-space, as was described earlier in §3.2.2. These accelerators are built with the HW resources and baseline configuration listed in Table 3.1, and the area overhead for different flexibility is shown in Table 3.2. For each study, we characterized the full neural network across all layers and present the overall runtime, energy, and energy-delay product (EDP) for the best-found mappings. Furthermore, we then present detailed analysis of three layers with significantly different aspect ratios from MnasNet to illustrate the correlation between performance.

Optimization Objective. Across our experiments, we set the optimization objective to minimum runtime. However, other objectives such as energy, area, energy-delay-product (EDP), performance-per-watt, and so on, are all feasible. We can also use multi-objective formulation to optimize two or more objectives simultaneously.

Map-space Visualization. Finally, for each of the targeted isolation studies, we present a Venn diagram in the style of Figure 3.5 drawn to scale (provided in each figure) with the areas capturing the number of mappings. Each Venn diagram plots the average of the metrics across all layers of the model.

Axis Isolation. We investigate the impact of standalone flexibility of Tile (class-1000), Order (class-0100), Parallelism (class-0010) and Shape (class-0001) with the target accelera-

PE=16x64	Accel	Runtime	Energy	EDP	H-F	W-F	Chosen
Layer1 (32,3,224, 224,3,3)	InFlex1000	1.00	**1.00**	1.00	0.22	0.0001	K:32, C:16, Y:3, X:3, R:3, S:3
	PartFlex1000	0.02	1.04	0.02	0.22	0.37	K:32, C:3, Y:1, X:16, R:3, S:1
	FullFlex1000	**0.0035**	1.01	**0.003**	1.00	0.50	K:32, C:3, Y:2, X:16, R:3, S:3
	FullFlex1111	0.0035	1.01	0.003	-	-	K:32, C:3, Y:2, X:16, R:3, S:3
Layer16 (120,40,2 8,28,1,1)	InFlex1000	1.00	**1.00**	1.00	0.22	0.0002	K:64, C:16, Y:3, X:3, R:3, S:3
	PartFlex1000	0.43	1.01	0.43	0.22	0.54	K:64, C:10, Y:2, X:4, R:1, S:1
	FullFlex1000	**0.14**	1.01	**0.14**	1.00	0.69	K:64, C:10, Y:4, X:4, R:1, S:1
	FullFlex1111	0.05	1.00	0.05	-	-	K:64, C:8, Y:4, X:4, R:1, S:1
Layer29 (1,480,14, 14,5,5)	InFlex1000	1.00	1.00	1.00	0.22	0.0007	K:1, C:16, Y:3, X:3, R:3, S:3
	PartFlex1000	0.23	**0.55**	0.13	0.22	0.58	K:1, C:8, Y:2, X:7, R:3, S:3
	FullFlex1000	**0.08**	**0.55**	**0.04**	1.00	0.74	K:1, C:24, Y:2, X:7, R:3, S:3
	FullFlex1111	0.08	0.55	0.04	-	-	K:1, C:24, Y:2, X:7, R:3, S:3
Mnasnet-Model	InFlex1000	1.00	**1.00**	1.00	0.22	0.0006	-
	PartFlex1000	0.38	1.07	0.41	0.22	0.50	-
	FullFlex1000	**0.21**	1.05	**0.22**	1.00	0.65	-
	FullFlex1111	0.14	1.06	0.15	-	-	-

(a)

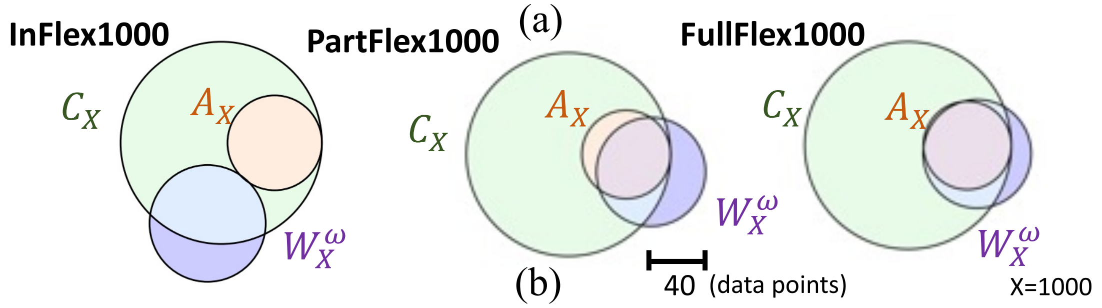


Fig. 3.7: Tile Axis Isolation. (a) Performance comparisons of accelerators with different level of Tile flexibility running Mnasnet model. (b) Venn diagram of Tile map space on Mnasnet. We use on-chip buffer size=4K in this experiments. Scale: Total Data points in $W_T^\omega = \pi(40)^2$. Runtime/Energy/EDP are normalized against values of InFlex-1000.

tors described by varying the constraints given to the tool. This aspect of the evaluation is not intended to reflect the typical usage of the tool, but rather as a limit study that characterizes the most extreme points in the overall design space.

3.4.2 Isolation: Tile Flexibility

We study the impact of tile flexibility with three kinds of accelerators: InFlex-1000 (or InFlex-0000[6]), uses the fixed baseline tile sizes; PartFlex-1000 uses hard-partitioned buffer with partition ratio of 1:1:1 for weight, input, and output buffer, and uses flexible tile sizes;

[6] InFlex-X (X=0000 to 1111) indicate the same constraint in all 16 classes.

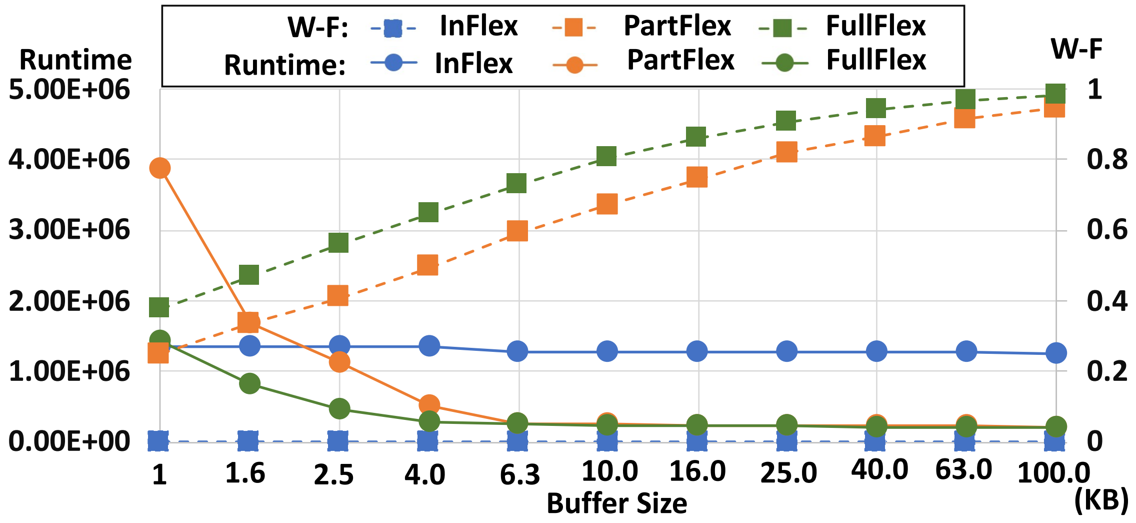


Fig. 3.8: The runtime performance and workload Flexion of fully-Tile-flexible accelerators with different buffer sizes. (W-F: Workload-dependent Flexion.)

FullFlex-1000 uses soft-partitioned buffer and flexible tile sizes. Note that the hardware trade-off between hard-partitioning and soft-partitioning significantly increases the energy per on-chip memory access, but can also result in fewer expensive off-chip accesses [47].

Map Space. Hardware-dependent Flexion (H-F) A_X/C_X is decided by the partitioning strategy which is around 0.22 for 1:1:1 partition. Value 0.22 is the area ratio of A_X over C_X in Figure 3.7(b). A_X^ω is the intersection of W_X^ω and A_X, showing how many different tiles that workload ω can explore and are supported by the accelerators. A_X^ω of InFlex-1000 is 1, since it leverages only 1 choice, while A_X^ω of PartFlex-1000 will be smaller than the one for FullFlex-1000, since hard-partitioning is a stricter constraint. These can be observed by the increase of overlapped area of W_X^ω and A_X from InFlex-1000 to FullFlex-1000. Note that the workload-dependent Flexion (W-F) A_X^ω/W_X^ω is directly related to the ratio of the supported map space to the entire map space, where larger values implies the given flexibility in the investigating accelerator can support the current workload better.

Performance. PartFlex-1000 allows the tile sizes to be flexible, which reduce runtime by 98%, as shown in Layer-1 (L1) in Figure 3.7(a). It chooses smaller K, C sizes and larger Y, X sizes to tailor for Layer-1. While enabling soft-partitioning (FullFlex-1000), with more tile choices, the runtime improves by another 6×. In Layer-16, the C tile sizes are

Table 3.3: Definition of Map Space and Flexibility.

C_X	Map space of an accelerator class (class-X)
A_X	Map space of the target accelerator in class-X
W_X^{ω}	Map space of workload ω
C_X^{ω}	$C_X \cap W_X^{\omega}$. Workload-dependent map space of class-X
A_X^{ω}	$A_X \cap W_X^{\omega}$. Feasible map space. Workload-dependent map space of the target accelerator
H-F	A_X / C_X. Hardware-dependent Flexion of the target accelerator. Supported map space percentage in the entire class-X map space.
W-F	$A_X^{\omega} / W_X^{\omega}$. Workload-dependent Flexion of the target accelerator. Supported map space percentage in the entire workload ω space.

chosen to divide the C dimension size perfectly. The tile configurations are chosen under the criterion of best runtime, and thus energy does not show significant difference. However, if we used least energy as criterion, different points would be picked, demonstrating a larger energy difference between InFlex-1000 and PartFlex-1000/FullFlex-1000. Overall, with this criterion, in class-1000, FullFlex-1000 can achieve speedup of 4.8× and 1.9× over InFlex-1000 and FullFlex-1000 in end-to-end Mnasnet.

Sensitivity Analysis: Buffer Size. The space of supported tile shapes is directly affected by the buffer size, which is the intersection area of S_X and W_S^{ω} in Figure 3.7(b). As shown in Figure 3.8, with increasing buffer sizes, the workload-dependent Flexion increases and the runtime improves. We can observe that the runtime improvement saturates at a buffer size of around 6.4KB, which is sufficient for most of the layers in MnasNet. Conversely, when the buffer size is small, the flexibility to execute on soft-partitioned buffer (FullFlex-1000) becomes more valuable to achieve a good runtime.

Takeaways. For a model like Mnasnet with large diversity in layer dimensions, we find that tile flexibility is crucial for the accelerator to capture the map-space. For the most part, the hard-partitioned PartFlex-1000 provides significant runtime benefits over an InFlex-1000 but is strictly worse than FullFlex-1000. We also observe that tile flexibility is more crucial for accelerators with smaller buffer sizes.

PE=16x64	Accel	Runtime	Energy	EDP	H-F	W-F	Chosen
Layer16 (120,40,28 ,28,1,1)	InFlex0100	1.00	1.00	1.00	0.001	0.04	YXKCRS
	PartFlex0100	0.82	1.003	0.82	0.004	0.13	KCRSYX
	FullFlex0100	**0.80**	**0.9997**	**0.80**	1.00	1.00	CXYKRS
	FullFlex1111	0.002	0.99	0.002	-	-	YCKSXR
Layer21 (40,120,28 ,28,1,1)	InFlex0100	1.00	1.00	1.00	0.001	0.04	YXKCRS
	PartFlex0100	0.77	0.998	0.77	0.004	0.13	KCRSYX
	FullFlex0100	**0.74**	**0.997**	**0.74**	1.00	1.00	CKXYRS
	FullFlex1111	0.002	1.29	0.002	-	-	KCYRXS
Layer29 (1,480,14, 14, 5,5)	InFlex0100	1.00	**1.00**	1.00	0.001	0.01	YXKCRS
	PartFlex0100	1.00	1.09	1.09	0.004	0.03	KCRSYX
	FullFlex0100	**0.997**	1.001	**0.998**	1.00	1.00	KCXYRS
	FullFlex1111	0.0003	0.48	0.0001	-	-	KSYRXC
Mnasnet-Model	InFlex0100	1.00	1.00	1.00	0.001	0.01	-
	PartFlex0100	0.90	1.02	0.91	0.004	0.03	-
	FullFlex0100	**0.89**	**0.996**	**0.89**	1.00	1.00	-
	FullFlex1111	0.004	1.00	0.004	-	-	-

(a)

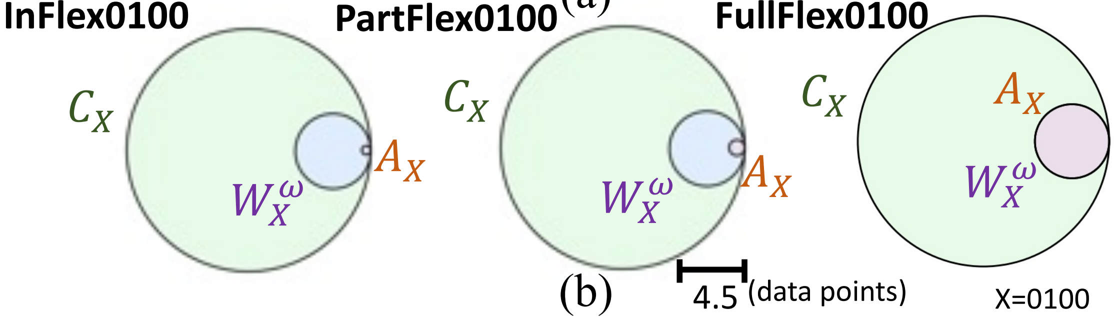


Fig. 3.9: Order Axis Isolation. (a) Performance comparisons of accelerators with different level of Tile flexibility running Mnasnet model. (b) Venn diagram of Tile map space on Mnasnet. Runtime/Energy/EDP are normalized against values of InFlex-0100.

3.4.3 Isolation: Order Flexibility

The three types of accelerators are configured as follows: InFlex-0100, using the output-stationary order YXKCRS, PartFlex-0100, with three order choices including output, input, and weight stationary order, and FullFlex-0100, with flexible order.

Map Space. H-F A_X/C_X is affected by the number of supported order choices, which depends on the available HW support described in Figure 3.4. For order, the map space does not depend on the number of compute or memory resources such as PEs and buffer sizes. Therefore, C_X can encompasses all design points in W_X^{ω}, as shown in Figure 3.9(b).

Performance. From Layer-16 in Figure 3.9(a), we observe that simply adding 3 out of 6! order choices—and consequently increasing workload Flexion by a small amount—improves runtime to a near-optimal value. This demonstrates that for accelerator design, partially supporting order flexibility may expose a better cost-performance trade-off, if the supported order choices expose distinct behaviors. In our evaluation, we also found that many different orders will end up with the similar runtime. This observation motivates the design of PartFlex–0100, a simpler and lower-cost accelerator that achieves similar performance as FullFlex–0100. Overall, in class-0100, full flexibility support can still achieve 1.12× and 1.01× speedup over the two baselines.

Takeaways. We find that the output stationary loop-order in InFlex-0100 works reasonably well across all layers. Adding support for three loop-orders (output, weight and input) in PartFlex-0100 gets nearly the same performance as supporting all 6! loop orders in FullFlex-0100.

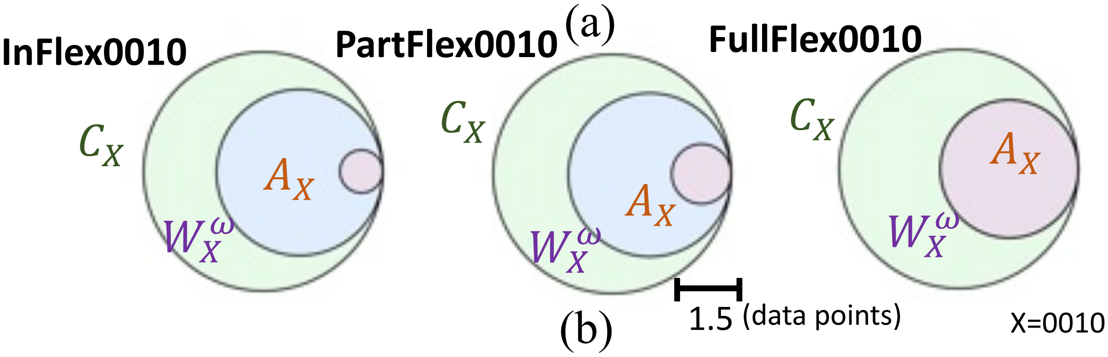
3.4.4 Isolation: Parallelism Flexibility

The three types of accelerators are as follows: InFlex-0010, using default K-C parallelism, PartFlex-0010, with a choice of K-C or Y-X parallelism, and FullFlex-0010.

Map Space. Since we investigate CONV-accel (6-dim) and consider two-way parallelism. We have C_X=6x5=30 different choices, while the A_X for InFlex-0010 and PartFlex-

PE=16x64	Accel	Runtime	Energy	EDP	H-F	W-F	Chosen
Layer10 (72,24,56, 56,1,1)	InFlex0010	1.00	**1.00**	1.00	0.03	0.08	KC
	PartFlex0010	0.48	1.59	0.76	0.07	0.17	YX
	FullFlex0010	**0.46**	1.20	**0.55**	1.00	1.00	XK
	FullFlex1111	0.00003	0.90	0.00002	-	-	XK
Layer16 (120,40,28 , 28,1,1)	InFlex0010	1.00	**1.00**	1.00	0.03	0.08	KC
	PartFlex0010	0.88	1.85	1.63	0.07	0.17	YX
	FullFlex0010	**0.48**	1.50	**0.72**	1.00	1.00	KS
	FullFlex1111	0.00006	0.92	0.00006	-	-	KC
Layer29 (1,480,14, 14,5,5)	InFlex0010	1.00	1.00	1.00	0.03	0.05	KC
	PartFlex0010	0.50	**0.50**	0.25	0.07	0.10	YX
	FullFlex0010	**0.05**	1.62	**0.09**	1.00	1.00	RS
	FullFlex1111	0.0001	0.16	0.00001	-	-	KY
Mnasnet-Model	InFlex0010	1.00	1.00	1.00	0.03	0.07	-
	PartFlex0010	0.81	**0.87**	0.70	0.07	0.14	-
	FullFlex0010	**0.59**	0.99	**0.58**	1.00	1.00	-
	FullFlex1111	0.0004	0.56	0.0002	-	-	-

(a)



(b)

Fig. 3.10: Parallelism Axis Isolation. Performance comparisons of accelerators with different level of Parallelism flexibility running Mnasnet model with (a) 16x64 PE array and (b) 32x32 PE array. (c) Venn diagram of Parallelism map space on Mnasnet. PRuntime/Energy/EDP are normalized against values of InFlex-0010.

0010 is 1 and 2, respectively.

Performance. Figure 3.10(a) shows that different workloads can have different optimum parallelism choices, and supporting only a subset of choices may not be sufficient to approach optimal performance (Layer-16, 19). Layer-29 is a depth-wise CONV, where there is no cross-input channel computation. InFlex-0010's restriction of parallelism only across K and C dimensions deprives it of other parallelism opportunities such as YX- or RS-parallelism, which could be a better choice for this layer. Overall in class-0010, FullFlex-0010 consistently achieves around 1.6× and 1.3× better runtime relative to the fixed and partially flexible accelerators, respectively.

Takeaways. We observe the flexibility in parallelism is highly-sensitive to the shape of the physical array. We also see that non-conventional parallelism choices (such as XK or KS) are found by the mapper, suggesting that supporting full flexibility in parallelism choices is valuable.

3.4.5 Isolation: Shape Flexibility

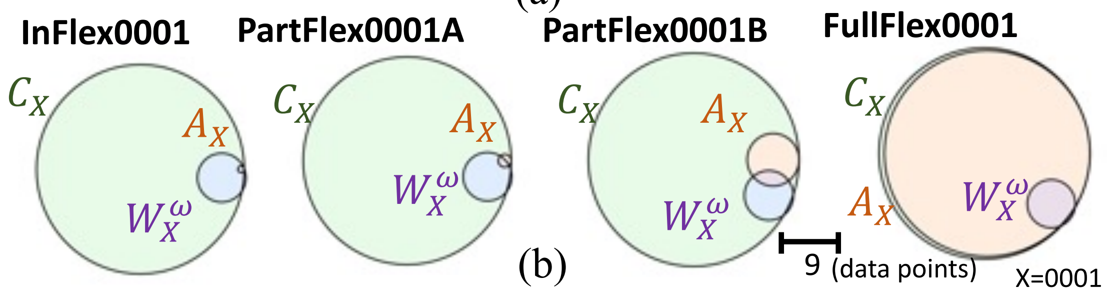
The accelerators compared are as follows: InFlex-0001, using a square PE array, PartFlex-0001-A, a flexible PE array composed of a modular 16×16 PE-array building block, PartFlex-0001-B, a flexible PE array with a 4×4 building block, and FullFlex-0001.

Map Space. C_X is decided by the size of the PE array. A_X for the PartFlex-0001 is the number of different array shapes that can be composed by the smaller building block. A smaller building block can enable more fine-grained PE array shape exploration and thus exposes a larger configuration space, which can be observed by the different size of A_X for the two partially flexible accelerators in Figure 3.11(b).

Performance. When the size of parallelism dimension is larger than the size of the building block (32×32, 16×16, 4×4, or flexible), we need to fold the computation. When they are not perfectly divisible, the last fold of computation will introduce compute under-utilization, as shown in Layer-15 and Layer-16 of PartFlex-0001-A in Figure 3.11(a).

PE=1024	Accel	Runtime	Energy	EDP	H-F	W-F	Chosen
Layer15 ParSize: [72, 40]	InFlex0001	1.00	1.00	1.00	0.001	0.01	32x32
	PartFlex0001A	0.67	1.00	0.67	0.004	0.06	48x16
	PartFlex0001B	**0.50**	**0.97**	**0.48**	0.06	0.25	24x42
	FullFlex0001	**0.50**	**0.97**	**0.48**	1.00	1.00	24x42
	FullFlex1111	0.04	0.95	0.03	-	-	72x14
Layer16 ParSize: [40, 120]	InFlex0001	1.00	1.00	1.00	0.001	0.03	32x32
	PartFlex0001A	0.75	**0.98**	0.74	0.004	0.08	16x64
	PartFlex0001B	**0.63**	0.99	**0.62**	0.06	0.25	40x24
	FullFlex0001	**0.63**	0.99	**0.62**	1.00	1.00	40x25
	FullFlex1111	0.04	0.96	0.04	-	-	8x128
Layer25 ParSize: [80, 480]	InFlex0001	1.00	1.00	1.00	0.001	0.01	32x32
	PartFlex0001A	0.89	**0.99**	0.88	0.004	0.05	16x48
	PartFlex0001B	0.89	**0.99**	0.88	0.06	0.25	16x48
	FullFlex0001	**0.87**	**0.99**	**0.86**	1.00	1.00	24x42
	FullFlex1111	0.80	1.21	0.97	-	-	24x42
Mnasnet-Model	InFlex0001	1.00	1.00	1.00	0.00	0.02	-
	PartFlex0001A	0.98	1.00	0.97	0.00	0.04	-
	PartFlex0001B	**0.96**	**0.99**	**0.95**	0.06	0.20	-
	FullFlex0001	**0.96**	**0.99**	**0.95**	1.00	0.98	-
	FullFlex1111	0.15	1.03	0.15	-	-	-

(a)



(b)

Fig. 3.11: Shape Axis Isolation. (a) Performance comparisons of accelerators with different level of Tile flexibility running Mnasnet model. (b) Venn diagram of Tile map space on Mnasnet. Runtime/Energy/EDP are normalized against values of InFlex-0001.

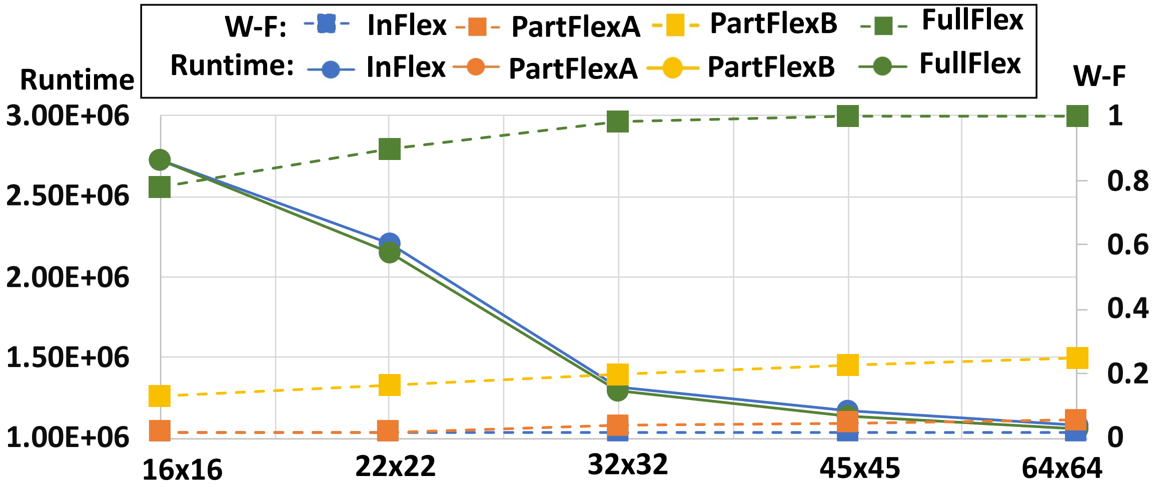


Fig. 3.12: The runtime performance and workload Flexion of fully-Shape-flexible accelerators with different size of PE arrays. (W-F: Workload-dependent Flexion)

Interestingly, PartFlex-0001-B can mostly reach optimum performance, since most of the layers have K, C dimension sizes that are multiples of 4. Note that we use the default K-C parallelism strategy in this experiment. PartFlex-0001-B reaches almost the same performance as the fully-flexible one with only 6% flexibility. Overall, we observe that full flexibility achieve speedup of 1.05×, 1.02×, and 1.001× over the three baselines, respectively.

Sensitivity Analysis: Array Size. A larger number of PEs can allow more array configuration options and hence has larger hardware Flexion, as shown in Figure 3.12. We can also observe the diminishing return around 45×45 to 64×64 points when the provided number of PEs is starting to become larger than the maximum K-C parallelism that some of the layers can leverage, leading to under utilization of the PEs.

Takeaways. We observe the flexibility in array shape is highly crucial for utilization across layers. However, it is highly sensitive to the dimensions being mapped spatially. This indicates that flexibility in P and S go hand-in-hand (more discussion on this in §3.5). One key observation is that a partially flexible accelerator using multiple fixed-size small arrays (e.g., 4x4) reaches almost the same performance as the fully-flexible one. This observation is consistent with some design choices in the existing accelerator designs, e.g., the 4×4 matrix multiplier in CUDA tensor cores [145]. The partially flexible accelerators

Flexibility Class	Alexnet	Mnasnet	Resnet50	Mbnet	BERT	DLRM	NCF
InFlex0000-Alexnet-Opt	1.00	1.00	1.00	1.00	1.00	1.00	1.00
InFlex0000-X-Opt	1.00	0.96	0.92	0.96	0.25	0.14	0.07
FullFlex1000-Alexnet-Opt	**0.14**	**0.57**	**0.07**	**0.48**	0.25	0.99	0.99
FullFlex0100-Alexnet-Opt	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FullFlex0010-Alexnet-Opt	1.00	**0.92**	1.00	**0.88**	1.00	0.25	0.57
FullFlex0001-Alexnet-Opt	1.00	**0.82**	**0.91**	**0.88**	1.00	0.58	0.63
FullFlex0011-Alexnet-Opt	1.00	**0.51**	**0.60**	**0.53**	0.90	**0.10**	0.12
FullFlex0101-Alexnet-Opt	1.00	**0.60**	**0.65**	**0.61**	0.90	0.16	0.40
FullFlex1001-Alexnet-Opt	**0.08**	**0.57**	**0.07**	**0.37**	0.22	0.32	0.18
FullFlex0110-Alexnet-Opt	1.00	**0.60**	**0.65**	**0.61**	0.90	0.16	0.40
FullFlex1010-Alexnet-Opt	**0.06**	**0.17**	**0.07**	**0.15**	**0.22**	**0.04**	**0.04**
FullFlex1100-Alexnet-Opt	**0.14**	**0.33**	**0.07**	**0.30**	**0.22**	0.66	0.60
FullFlex1110-Alexnet-Opt	**0.06**	**0.17**	**0.09**	**0.15**	**0.22**	**0.04**	**0.04**
FullFlex1011-Alexnet-Opt	**0.06**	**0.15**	**0.09**	**0.14**	**0.22**	**0.04**	**0.04**
FullFlex0111-Alexnet-Opt	1.00	**0.51**	**0.60**	**0.53**	0.90	**0.10**	0.12
FullFlex1101-Alexnet-Opt	**0.08**	**0.33**	**0.12**	**0.30**	**0.22**	0.16	0.18
FullFlex1111-Alexnet-Opt	**0.06**	**0.15**	**0.07**	**0.14**	**0.22**	**0.04**	**0.04**
PartFlex1111-Alexnet-Opt	**0.52**	**0.42**	**0.20**	**0.39**	0.26	**0.07**	0.07

Underline: Reach better runtime over InFlex0000-Alexnet-Opt.

Bold: Reach better runtime over InFlex0000-X-Opt.

Fig. 3.13: Runtime performance of accelerators with different flexibility. InFlex-0000-X-Opt means class-0000 (fixed) accelerator that is optimized for DNN model X (X= Alexnet, Mnasnet, etc.). The runtime in each row is normalized by the runtime of the values of InFlex-0000-Alexnet-Opt.

are also sufficient for many manual-design CNNs, e.g., ResNet50 [142], VGG16 [146], whose dimension sizes are mostly multiples of 64 in K, C dimensions. However, shape flexibility could become increasingly important because of two use cases, as follows. (1) More and more neural network designs are relying on NAS, which could generate more diverse layer shapes. MnasNet is one such NAS network, but it is highly modularized, while more diverse networks could be developed in future to target different applications. (2) Pruning techniques can change layer shapes in diverse ways.

3.5 Evaluation II: Accelerator Future-Proofing

In this experiment, we demonstrate how this chapter enables a first-of-its-kind experiment: assessing the advantage of adding flexibility to an accelerator at design time in terms of future-proofing, quantifying the trade-off between hardware cost and the performance gain on "future" workloads. To start, we design an inflexible accelerator tailored for Alexnet, InFlex-0000-Alexnet-Opt, by finding a fixed configuration of TOPS to optimize its runtime. This represents an accelerator such as Eyeriss [124] which was a optimized for a fixed workload in a certain year ($\sim$2014).

Fixed Alexnet-Opt Accel on Future Models. We now progress our accelerator into the future. When new neural network models (model X) are introduced. We design InFlex-0000-X-Opts, representing a new accelerator optimized just for this new workload—though for comparison purposes we always use roughly the same number of hardware resources as InFlex-0000-Alexnet-Opt. First, we use the InFlex-0000-Alexnet-Opt to run the future models and compare the performance witht the future model (X) optimized InFlex-0000-X-Opt. We observe that InFlex-0000-Alexnet-Opt can achieve similar performance to InFlex-0000-X-Opt when X is a CNN model, which follows the intuition since Alexnet is also a CNN, which might share similar characteristics. However, for fully-connected (FC) layer dominated models such as BERT, DLRM, and NCF, we see 1/0.25 = 4x to 1/0.07=14x slowdown. This shows that a fixed configuration can hurt performance when new models start to show notably different characteristics (e.g., CONV- vs FC-dominated).

Single-Axes Flexible Alexnet-Opt Accel on Future Models. Next, we go back to the "past" to explore the effect of adding single-axes flexibility when designing Alexnet-Opt accelerator, as shown in rows 3-6 of Figure 3.13.

We see that in MnasNet, FullFlex-1000-Alexnet-Opt is $1.00/0.57 = 1.75\times$ better than InFlex-0000-Alexnet-Opt. *We found tile flexibility to be crucial for the three CNNs and BERT*. However, it is not as effective in DLRM and NCF. The reason is as follows. In this experiment, InFlex-0000-Alexnet-Opt found Y-K parallelism to be optimal for Alexnet, and this stays valid for other CNNs, too. BERT is composed of matrix-matrix multiplications, the GEMM (M,N,K) dimensions got mapped to the (K_conv,C,Y) dimensions of the CONV accelerator respectively. The Y-K parallelism in the accelerator helped BERT as well.

However, DLRM and NCF use matrix-vector multiplications, where the K dimension is 1 for the vector. Mapping them on the InFlex-0000-Alexnet-Opt accelerator leads to under-utilization along the Y-parallelism dimension of the accelerator, leading to loss in performance. In order to run DLRM and NCF efficiently, the ability to switch to other parallelism strategies become crucial.

Multiple-Axes Flexible Alexnet-Opt Accel on Future Models. There are 10 other variants of combinatorial flexible accelerators, as shown in row 7-16 of Figure 3.13. We see that (P)+(S) together (0011) work well and show significant performance gain compared to the separated counterpart (0010, 0001). (T)+(P) with only two flexibility axes enabled can achieve comparable performance with FullFlex-1111-Alexnet-Opt in multiple cases. Interestingly, we see that (T)+(O) works much worse than (T)+(P). This shows that (T), which individually works well still needs to find the right other flexibility axes to have another level of performance boost.

Fully Flexible Alexnet-Opt Accel on Future Models. Finally, we investigate two kinds of flexible accelerators: FullFlex-1111-Alexnet-Opt, and PartFlex-1111-Alexnet-Opt. PartFlex-1111-Alexnet-Opt uses the configurations described in the evaluations in §3.4 (using variant A for Shape).

While more flexibility support introduces larger area cost, it is noteworthy that it brings with potential to leverage better runtime *even in the original AlexNet*. This is because AlexNet itself has a lot of variation across its layers. *A fixed accelerator that optimizes for the average-case across all layers may miss the potential to reach the best case for each individual layer.* For e.g., FullFlex-1111-Alexnet-Opt speeds up the baseline InFlex-0000-Alexnet-Opt by 1/0.06=16.7x times in Alexnet. Furthermore, FullFlex-1111-Alexnet-Opt constantly achieves better performance than InFlex-0000-X-Opts.

Takeaways. Accelerators optimized for certain layers (such as CONV2D) are more future proof, even if the layer dimensions of future models change. This is because most models show abundant parallelism across the input and output channels. Often tile flexibility seems sufficient to capture most benefits. However, for FC-dominated layer types, flexibility in parallelism and shape starts reaping more benefits since there are fewer dimensions and GEMMs can be irregular. As models with several other layer types emerge (e.g., FC/DWCONV/Attention/...), investing in flexibility is valuable.

3.6 Related Work

Flexible DNN accelerators. Flexibility support in DNN accelerators is an active topic of research, with several "flexible" accelerators proposed over the years [12, 48, 49, 3, 129]. The taxonomy from this work can tease out the specific axes of flexibility each provided and enable quantitative comparisons.

Mapping Tools. Several mapping tools have been proposed to search through map-spaces of flexible accelerators. In this work, we extend GAMMA [134] to support flexibility-aware optimization.

3.7 Summary

We demonstrate that increased formalism of the notion of flexibility leads to concrete benefits. Namely: (A) the ability to taxonomize existing accelerators along flexibility axes, (B) the ability to precisely quantify the shape of a map-space against a theoretical upper-bound, and (C) the ability to integrate flexibility into existing DSE flows. We identify that in several cases, partial flexibility along some of the axes is sufficient for capturing an optimized mapping. Most significantly, our evaluation of a 2014-style accelerator demonstrates that a design-time investment in flexibility can lead to concrete improvements in future-proofing after deployment time.

Of course, the field of deep learning is changing incredibly rapidly. Turing Complete platforms such as GPUs and CPUs will undoubtedly continue to play a key and irreplaceable role as the main platforms that allow machine learning experts to rapidly innovate. However, we believe that it is equally important that the most successful neural network architectures can be accelerated via hardware specialization, without ASIC designers risking complete obsolescence after deployment. Ultimately, the key reason flexible mapping support can be omitted from hardware designs is not area or energy, but simply design and verification effort, as well as mapper toolchain support. We hope that the research in this chapter will ultimately lead to more research that allows architects to quantify the benefits of "future-proofing" with the same level of precision as present-day budgets. Finally, while most existing DSE framework focus on finding the "optimal design point", our approach armed with the ability to systematically constructing map-spaces of fully/partially flexible accelerators, opens an interesting future avenue of research to identify "regions of optimal (i.e., high-performing) design space", leading to better explainability of the DSE algorithms and interpretability of the accelerator deign space.

CHAPTER 4
GAMMA: A GA-BASED MAPPER

In the previous section, we clearly define the map space of the DNN accelerator, which includes Tiling, Ordering, Parallelism, and Shape, four mapping axes. Next, we discuss how to create an algorithm for optimal mapping search. We call the search algorithm for mapping – mapper.

Potentially, all the optimization algorithms can be plugged into the Map Space Exploration (MSE) framework and becomes a viable mapper. However, the sampling efficiency of the mapper is the key for it to be useful. In this chapter, we will discuss why finding optimal mapping is important, how to create an efficient mapper, and what insight we learn from the development of this new mapper.

Although multiple prior works [9, 51, 52, 30, 53, 54, 55, 56, 57, 18] have studied the mapping problem for DNN accelerators, the HW-mapping search space (exceeding $O(10^{36})$ even for a single layer of a DNN, as shown later in §7.1.6) makes the problem highly challenging. To cope with this challenge, most prior works restrict the search space. For e.g., coarse-grained strided exhaustive search [23, 12, 16, 17, 22], random search [4], fixed parallelism [23, 16, 22, 56, 57, 18], or limited search for tile sizes for one or more fixed dataflows [21, 12]). Alternately, ML-based search techniques have also been leveraged for guided search to increase sampling efficiency [31, 24, 26, 27]; however, they need to restrict some aspects of the mapping space (e.g., fixing the parallelism levels) to adapt to the ML algorithms. Such restrictions of the mapping space can lead to local optimal mappings which are significantly sub-optimal, as recent works have highlighted [4, 8].

To efficiently deal with the massive search space of HW-mappings, we propose GAMMA (**G**enetic **A**lgorithm-based **M**apper for **ML** **A**ccelerators). Unlike prior works, GAMMA performs a complete search, considering all three aspects of HW-mapping (tiling strategy, computation order, and parallelization strategy). Furthermore, GAMMA can explore up to three levels of parallelism within the mapping, unlike prior works that target one to two levels. Thus GAMMA can work across both single-accelerator [82, 147] and multi-accelerator [8] systems. The key novelty in GAMMA is (i) a specialized genetic encoding of all three aspects of HW mapping, (ii) specialized mutation and crossover operators to evolve new mappings, and (iii) new genetic operators to model the behavior of adding and removing levels of parallelism. We also develop a closed-loop workflow by integrating GAMMA with a popular analytical cost-model for DNN mappings called MAESTRO [58] to fully automate the mapping search problem.

4.1 Challenges with Baseline Methods

The baseline methods in §2.6 can all be used for HW-mapping search. However many algorithms (e.g., CMA-ES, PSO, DE, and also standard GA) work in rigid search space, i.e., the number of parameters in a design point is pre-defined, which restricts the levels of parallelism (§2.3) to a pre-defined number and shrinks the potential search space by log scale. Our goal is to parameterize the level of parallelism as well. We need a framework that accepts input with flexible lengths. There are many possible ways to realize such a flexible framework, such as adding an extra auto-encoder [148] or using an sequence-to-sequence structure [149] at the input layer. However, they require another level of optimization, training, or approximation, which brings in relatively large overhead comparing to the optimization algorithm (DE, ES, standard GA) itself.

4.2 Genetic Algorithms (GA)

In this chapter, we develop a GA-based search technique. We list some common terminology for GA, namely *gene, genome, elite, population*, we will use across this chapter in Table 7.1. A genome is a mapping solution in our context. We reproduce the next generation by mutation and crossover. The goal of GA is to retain well-performing genes across the

Table 4.1: Terminology in Genetic Algorithm (GA).

Term	Description
Gene	An encoded value that represents accel. sel. or job prio. of a job.
Genome	A series of genes that represent the entire schedule about accel. sel. or job prio. of a batch of jobs.
Individual	A series of genomes that fully represent the schedule of a batch of jobs.
Generation	An entire set of individuals forms a generation. The generation evolves with time by mutation/crossover and selection of the well-performing individuals to the next generation.
Crossover	Blend two parents' genes to reproduce children's genes.
Mutation	Randomly perturb a parent's genes to reproduce children's genes.

evolution.

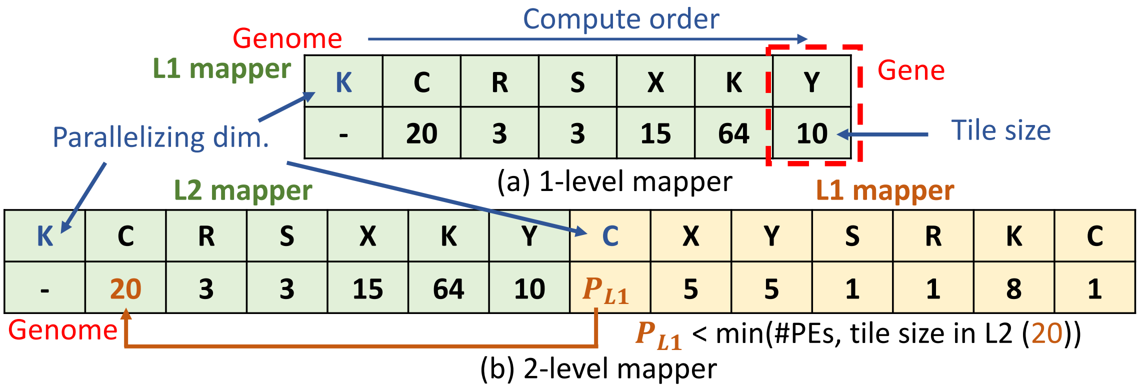
Benefits of GA. GA is one of the most popular algorithms for the scheduling problem for its lightness and simplicity [150, 151, 152, 153, 154]. Research shows GA reaches competitive performance with deep reinforcement learning [155, 156], and hyperparameter optimization problem. STOKE [103] and TensorComprehensions [31] use GA to search the space of DNN code optimization. s.

Challenges with standard GA. Standard GA still falls into the pits of the algorithm needing rigid input length. To this end, we develop a way to adopt GA to our problem by designing a novel evolution mechanism, which allows it to be flexible, without adding up immense overhead such as adding encoder or training an seq-to-seq model. We discuss these details next.

4.3 Methodology

4.3.1 GAMMA Encoding scheme

We design a specific genetic encoding scheme for the HW mapping problem. For a 1-level mapper, we encode them into a (7, 2) dimensions of the genome, which contains 7 pairs of genes, as shown in Figure 4.1(a). A pair of the gene contains a DNN layer tensor notation (e.g, K, C) and its tile size. The ordering of pairs reflects the computation order. The first pair of gene tells the parallelizing dimension. The 2-level mapper in Figure 4.1(b) is encoded in



(a) 1-level mapper

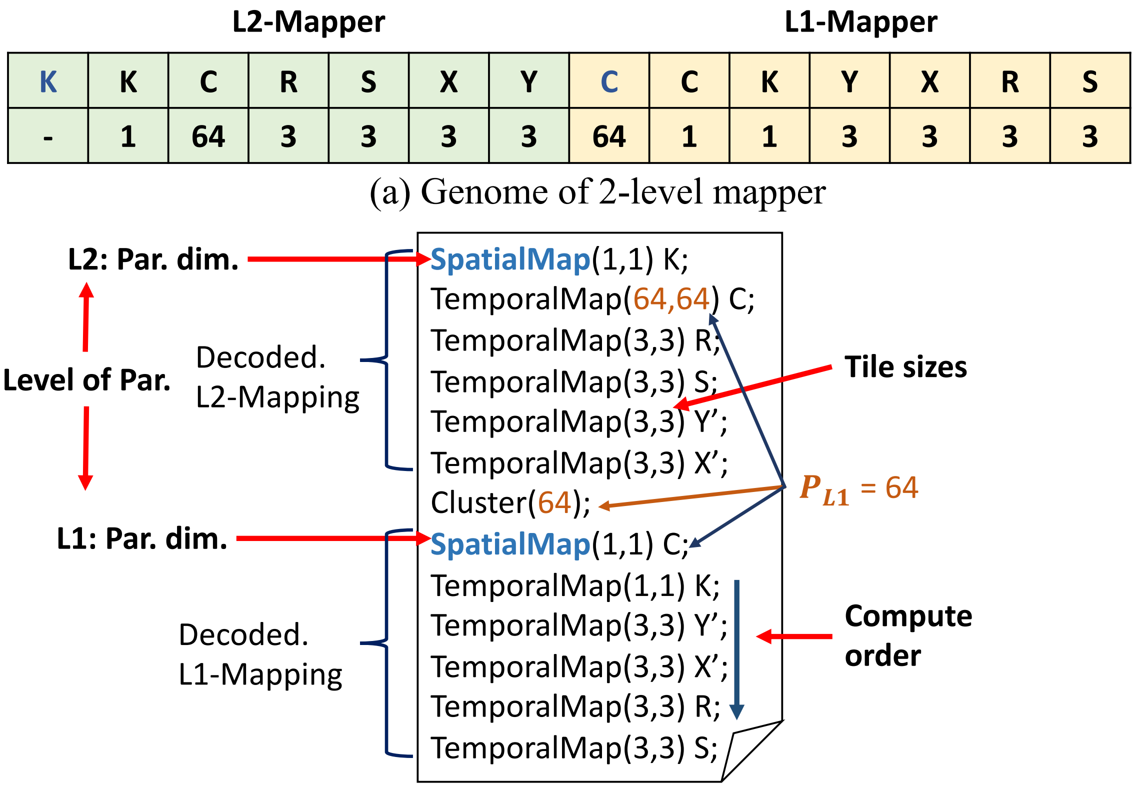
(b) 2-level mapper

Fig. 4.1: The GAMMA encoding example of (a) 1-level mapper and (b) 2-level mapper.

the same manner. P_{L1} describes number of parallel L1-mappers, which is constrained by the number of available PEs (the number of PEs defines the maximum amount of parallelism.), and the corresponding tile size of the chosen parallelizing dimension (since we need at least one element to distribute into each parallelism unit). The L1-mapper describes the inner loop. The L2-mapper describes the outer loop, while containing P_{L1} number of instances of L1-mapper.

4.3.2 Decoding Genomes into a Mapping

We outline how we describe the three aspects of mapping space in the cost model, and show how the genomes from GAMMA are decoded into the cost model's description. Figure 4.2(a) is a mapper description in GAMMA and Figure 4.2(b) is its corresponding description in the cost model (MAESTRO). The order of K, C, X, Y, R, S from left to right in (a) and top to bottom in (b) reflect the computation order. The number paired with dimension in (a) and the number inside the bracket in (b) reflects the tiling size on each dimension. In MAESTRO, we note the parallelized dimension as *SpatialMap* and remaining dimensions as *TemporalMap*. Therefore we mark the first element (indicating parallelizing dimension) in (a) as SpatialMap in (b). A level of mapper in GAMMA can be translated as a *Cluster* in MAESTRO, in which *tiling strategy*, *computation order* and *parallelism dimensions* are fully described. We formulate multiple level of parallelism by concatenating the cluster. The L1 and L2 mapper are decoded into the bottom and upper cluster.



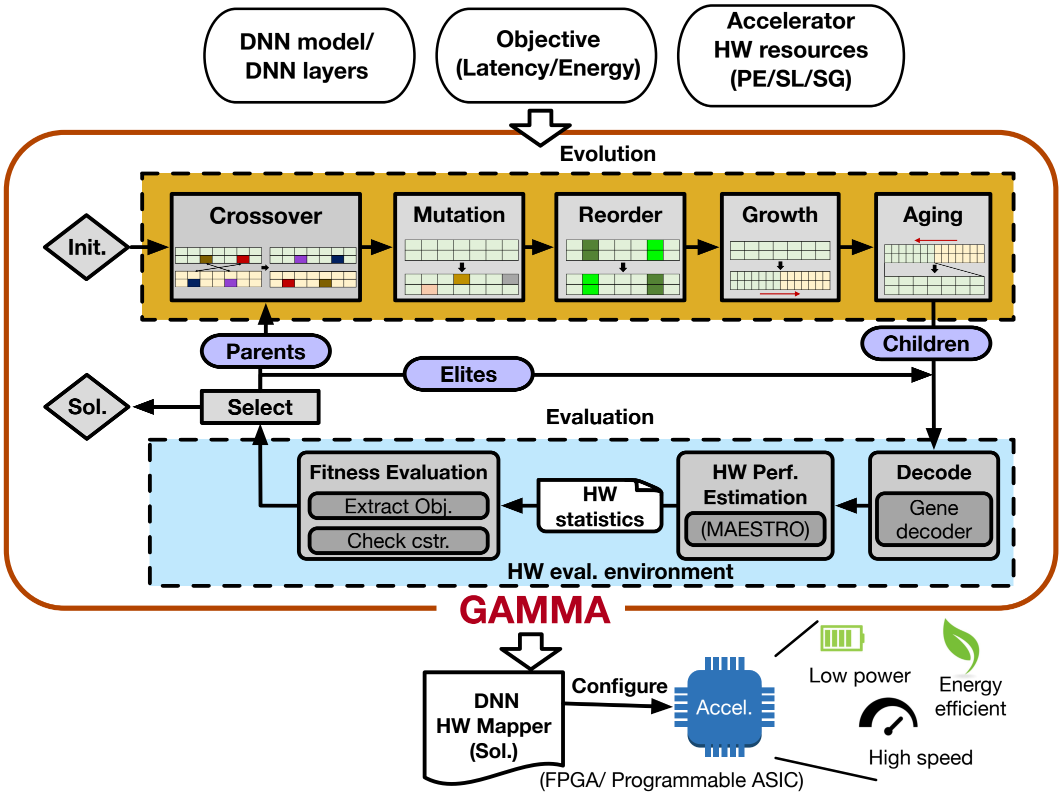
L2-Mapper							L1-Mapper						
K	K	C	R	S	X	Y	C	C	K	Y	X	R	S
-	1	64	3	3	3	3	64	1	1	3	3	3	3

(a) Genome of 2-level mapper

```
SpatialMap(1,1) K;
TemporalMap(64,64) C;
TemporalMap(3,3) R;
TemporalMap(3,3) S;
TemporalMap(3,3) Y';
TemporalMap(3,3) X';
Cluster(64);
SpatialMap(1,1) C;
TemporalMap(1,1) K;
TemporalMap(3,3) Y';
TemporalMap(3,3) X';
TemporalMap(3,3) R;
TemporalMap(3,3) S;
```

(b) Decoded 2-level genome in cost model (MAESTRO) 's description

Fig. 4.2: (a) GAMMA's description of a 2-level mapper and (b) its decoded description for cost model (MAESTRO) of a NVDLA-like [80] 2-level mapper.



(a) Structure and algorithm flow

Optimization methods		Description	Mapping Space Aspects			
			Tile Strat.	**Com. Ord.**	**Par. Dim.**	**Par. Level**
Baseline Methods		Defined a ***Par. Level***. Encode ***Tile Strat., Com. Ord., Par. Dim.*** into fixed length inputs, and evolve them with algorithms.	✓	✓	✓	✗
GAMMA	Crossover	Randomly pick two genomes from the parent subset. Interchange the value of their tile size.	✓			
	Mutation	Mutate Dim.: Sample a new Par. Dim. Mutate Tile size: Randomly pick a gene pair, sample a new tile size.	✓		✓	
	Reorder	Randomly pick two gene pairs. Swap their order.		✓		
	Growth	Concatenate a randomly initiated L1-mapper at the tail.				✓
	Aging	Remove the L1-mapper at the tail.				✓
	Summary		✓	✓	✓	✓

(b) The summary of evolution in GAMMA

Fig. 4.3: (a)The structure and algorithm flow of GAMMA, and (b) The summary of evolution in GAMMA.

4.3.3 Algorithm Flow

Figure 4.3 shows the flow of the GAMMA algorithm. We discuss the detail of each function block next. We first describe how we adopt the generic evolution operators, Crossover and Mutation, to the HW-mapping problem, and then introduce three additional evolution operators (Reorder, Growth and and Aging) in GAMMA.

Initialization. Assuming population size *P*, we randomly initialize *P* number of the 1-level mappers. The only restriction is each tile size is smaller than the corresponding layer dimension.

Evolution: Crossover. Crossover is to take advantage of the genes in some well-performing genomes, which forms a parents subset. We randomly pick two genomes from the parents subset. We blend their genes by interchanging the value of the tile size.

Evolution: Mutation - *Parallel Dim*. With a certain probability, which is set by the mutation rate of the algorithm, we mutate the parallelism dimension by randomly sampling one of the 6 dimensions of the tensor and setting it as a new parallelism dimension.

Evolution: Mutation - *Tile Size*. With a certain probability, we randomly pick paired genes and assign a new random tile size for them. If the tile-size in the mapping does not fit within the SL buffer of the PE for that operand, it is given a large penalty during its evaluation, as we discuss later in §4.3.4.

Evolution: Reorder. Reorder is another format of mutation. We pick two paired genes and swap their position in the genome, which reflect the reordering of the mapping.

Evolution: Growth. With a certain probability, we grow the genome by appending a randomly initialized 1-level genome to the current genome, as shown in Figure 4.1(b). The original L1 mapper will be promoted to L2-mapper, and the newborn genome is noted as the new L1-mapper.

Evolution: Aging. The natural phenomenon of a person's DNA keep shortening in the lifespan is known as DNA aging. With a certain probability, we will ”age” the genome by cutting out the tail of genome, an L1-mapper, which moves genome from (b) back to (a) in

Figure 4.1.

Evaluate and Selection. After evolution, we evaluate the populations by interacting with the evaluation environment (Env), which we will describe in §4.3.4. Env will feedback the fitness of each individual. We select the population that is eligible to enter the next generation by the ranking of their fitness.

4.3.4 Flow for Automated Mapping Search

Constraint

Our target is to find the HW mapping of a DNN layer that fits within limited HW resources - PEs and buffers. Different mapping lead to drastically different requirement of HW resources, especially the buffer sizes. Note that a mapping implicitly runs over multiple time iterations if the number of computations in the dimension to paralleize exceeds the number of available PEs; however, the local (SL) buffer and global (SG) buffer sizes to run each iteration (which comes from the tile sizes) is a hard constraint when searching through the map-space.

Objective

The target is to minimize the objective. The objective could be any HW performance index that the user is interested in such as latency, power, energy, area, energy-delay-product (EDP), or other combinations of them. Minimizing the objective is not a trivial task since there is no straight-forward solution even for a common tensor shape of a DNN layer. Minimizing the latency as an example, the most efficient choice of parallelizing dimension involves the shape of tensor, available PEs to parallelize, available SL/SG buffers to house the fetched data, and the tile size of each dimension. All the decisions (or genes) correlate and jointly decide the latency. Some common heuristics of parallelizing across activations dimensions at the early layers and across channel dimensions at the late layers in CNN becomes challenging when involving HW resource constraint and the multi-level parallelism flexibility of mapping strategy.

Interactive Environment (Env)

Structure: The Env is initialized with the target DNN layer, the accelerator constraint, and the optimization objective (latency/energy/power). Env contains a HW performance cost model, where we leverage MAESTRO [58] for its ability to model and evaluate arbitrary spatial accelerators and mappings. When interacting with GAMMA, Env takes in an entire generation of populations, decodes them into the input format of the cost model as describe in §4.3.2, and feeds into the cost model to gather the statistics of their HW performance. Finally, using the fitness function, which we discuss next, Env extracts fitness scores and returns them to GAMMA.

Fitness Function: We extract the corresponding *reward* value (= -*Perf. index*) from the statistics according to the set objective, and substitute it into the fitness function. We give the individual a large *penalty* - a negative infinite - when the constraint is not met. That is, the evolved mapping require more HW resources than the accelerators' constraint, which is then not suitable for the targeting accelerators. The fitness function is summarized as following.

$$Fitness = \begin{cases} reward, \text{ if constraint met} \\ -Infinite, \text{ others} \end{cases} \tag{4.1}$$

4.4 Evaluation of GAMMA

DNN Models. In our experiment, we consider five CNN models with different complexity: VGG16 [146], MobileNet-V2 [126], ResNet-50 [142], ResNet-50 [142], MnasNet [141].

HW resources of Platforms. We consider two platforms with different number of HW resource: cloud platform (which resembles the HW resources in cloud TPU[82]) and edge platform (which resembles the HW resources in Eyeriss chip[147]), as shown in Table 4.2.

Target Systems. We consider three kinds of accelerator systems as shown in Table 4.3.

Table 4.2: The HW resources in different platforms.

	# PEs	SL size (on-chip)	SG size (on-chip)
Edge Platform	168	512B	108KB
Cloud Platform	65,536	4MiB	24MiB

Table 4.3: Three target systems (Accel's infrastructures).

System	Description
S1:Fixed 2D Accel	Accelerator with 2D PE array (row x col) with fixed aspect ratio. This fixes P_{L1} to number of rows, and level of parallelism to 2. Example: TPU, NVDLA.
S2: Flexible 2D Accel	Accelerator with 2D PE array with flexible aspect ratio. The PE array can be configured to flexible 2D shape, which relax the choices of P_{L1}. We also relax the level of parallelism to be 1 or 2. Example: Eyeriss, Eyeriss_v2.
S3: Scale-out 2D Accel	Accelerator scaling out 2D structure. Accelerator comprises multiple 2D PE array instances, enabling parallelism across PE arrays. The level of parallelism is either 2 or 3. L1 and L2 mappers map within and L3 mapper maps across the PE arrays. Example: Simba, Tetris.

4.4.1 Target Search Methods and Parameters.

We compare three sets of methods, described below. We set the maximum sampling points as 10K for all methods and compare the HW performance of their searched solutions.

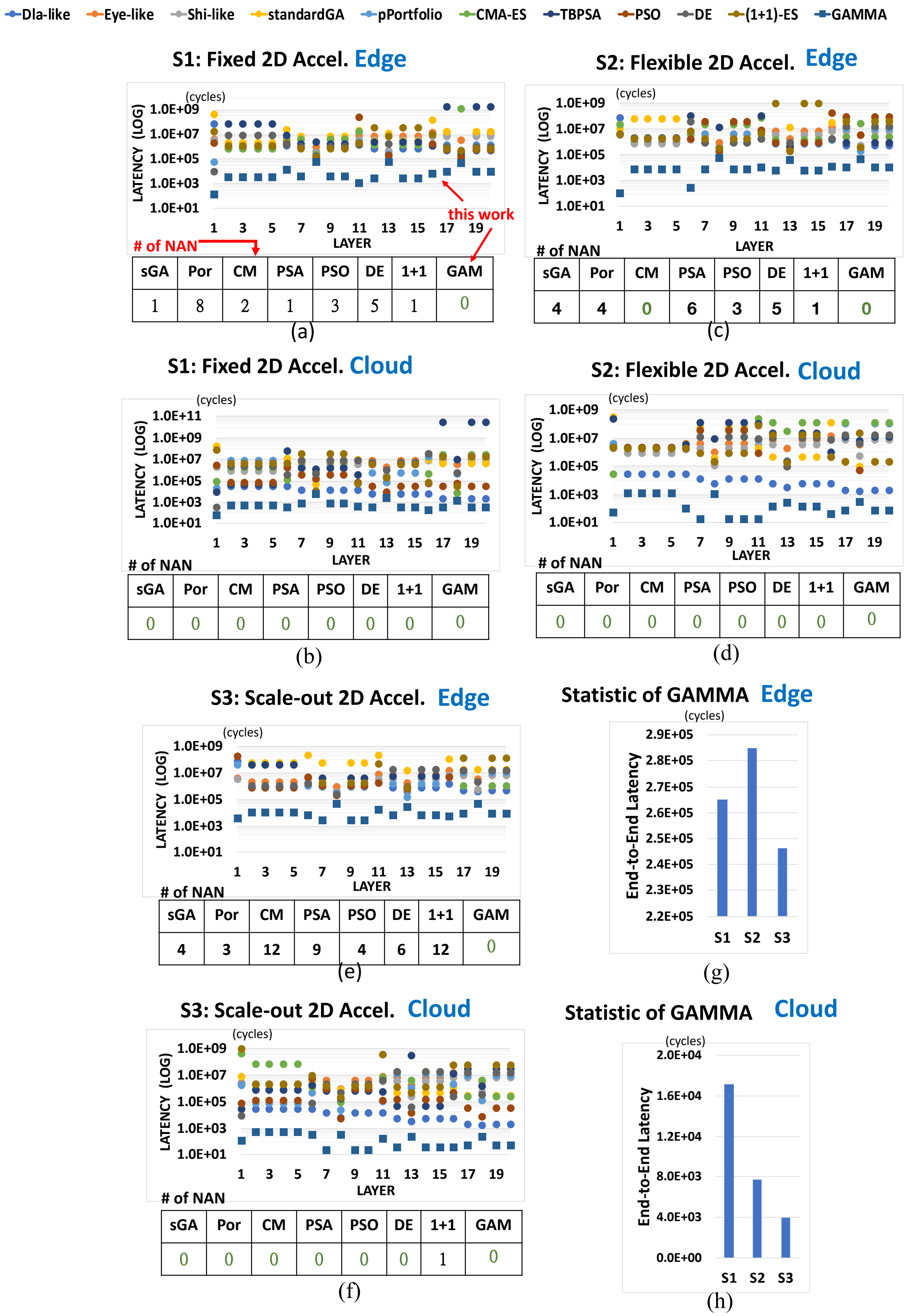
Baseline Optimization Methods. We compare with a suite of optimization methods whose implementations are adopted from Nevergrad [157]. The methods, and their experimental parameter settings are summarized in Table 7.3.

Fixed Dataflows from prior accelerators. We also compare with some widely recognized HW-mappings inspired by dataflows within prior accelerators: NVDLA-like [80] (parallelizing K and C dim.), Eyeriss-like [83] (parallelizing Y and R dim.), and ShiDianNao-like [81] (parallelizing Y and X dim). All three of them have *fixed* 2-level parallelism dimensions. We create custom mappings (i.e., dataflow + tile-size) by setting appropriate tile sizes that fit within the SL buffers for both the edge and cloud platforms .

GAMMA. We set the populations=200, generations=50, and the mutation/crossover rate and execution rate of other evolving functions as 0.5.

4.4.2 S1: Fixed 2D Accel.

In Figure 4.4(a), we run the baseline algorithms and GAMMA to search for the HW-mapping for each of the 20 layers in ResNet-18 with S1 setting (i.e., 2 levels of parallelism) and



(a) # of NAN

sGA	Por	CM	PSA	PSO	DE	1+1	GAM
1	8	2	1	3	5	1	0

(c) # of NAN

sGA	Por	CM	PSA	PSO	DE	1+1	GAM
4	4	0	6	3	5	1	0

(b) # of NAN

sGA	Por	CM	PSA	PSO	DE	1+1	GAM
0	0	0	0	0	0	0	0

(d) # of NAN

sGA	Por	CM	PSA	PSO	DE	1+1	GAM
0	0	0	0	0	0	0	0

(e) # of NAN

sGA	Por	CM	PSA	PSO	DE	1+1	GAM
4	3	12	9	4	6	12	0

(g)

(f) # of NAN

sGA	Por	CM	PSA	PSO	DE	1+1	GAM
0	0	0	0	0	0	1	0

(h)

Fig. 4.4: The performance of found solutions across a suite of optimization methods on different target systems (S1, S2, S3) and different platform constraints (Edge, Cloud) for ResNet-18.

NAN: The method cannot find a solution that fits in the platform constraint within 10K samples.

Table 4.4: Supported optimization algorithms in M3E.

Alg.	Description
AI-MT-like	A manual-tuned mapper for multi-core accelerator targeting vision and language workloads.
Herald-like	A manual-tuned mapper for multi-core heterogenous accelerator targeting vision workloads.
stdGA	Genetic Algorithm. *We use mutation rate: 0.1, crossover rate: 0.1.*
DE	Differential Evolution. *We use weighting for local DV: 0.8, weighting for global DV: 0.8, in the experiment.*
CMA-ES	Covariance Matrix Adaptation-ES. *We use 1/2 of the best performing individuals as an elite group in the experiment.*
TBPSA	Test-based Population-Size Adaptation. *We set the initial population size as 50 and let it evolve in the experiment.*
PSO	Particle Swarm Optimization. *We use weighting for global best: 0.8, weighting for parent best: 0.8, with momentum ω: 1.6.*
RL A2C	Advantage Actor-Critic. *We use policy and critic networks composed by 3 MLP layers with 128 nodes, discount factor: 0.99, learning rate: 0.0007, RMSProp optimizer.*
RL PPO2	Proximal Policy Optimization. *We use policy and critic networks composed by 3 MLP layers with 128 nodes, discount factor: 0.99, clipping range: 0.2, learning rate: 0.00025, Adam optimizer.*
MAGMA	A GA-based optimization algorithm that houses domain-specific genetic operators for multi-core heterogenous accelerator mapping problem.

edge platform constraint. We record the best solution (lowest latency) of each algorithm after they execute 10K samples (most comparing algorithms converge after 10K samples). Figure 4.4(a) shows the latency of each algorithm's solutions, where we also plot the corresponding latency when using fixed dataflow. We observe the baseline algorithms and the fixed-dataflows are with competitive performance. However, GAMMA can consistently find better solutions than both methods.

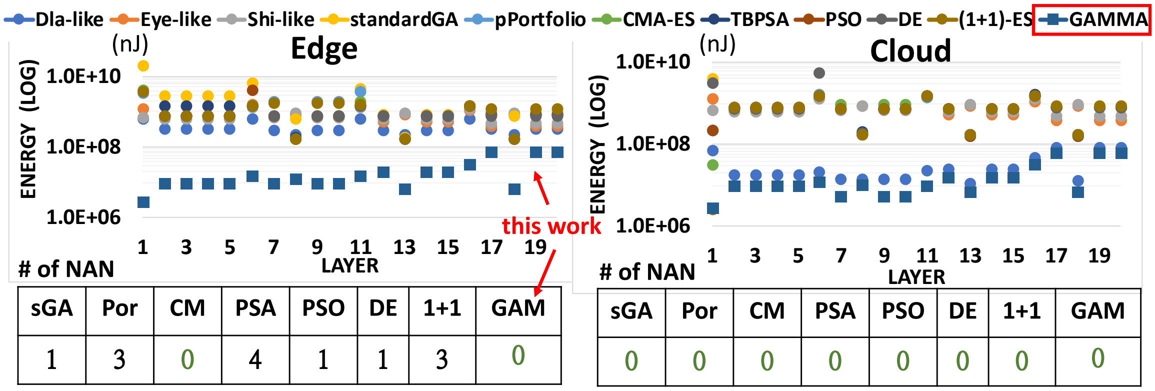
Valid solution and different platform constraints. The HW-mapping is invalid when its requirement of HW resources exceeds the platform constraint. Some methods cannot find any valid solutions that conform to the constraint after 10K sampling. Therefore some methods have no solutions (NAN) in some cases as shown in Figure 4.4(a). We did not show the result of Random Search here, since it ends up finding no solution for most of the cases, which also infers the complexity of the search space (it cannot find a valid solution in 10K samples). When we have more HW resource budget as Cloud platform in Figure 4.4(b), all baseline optimization methods can start to find valid solutions and optimize on them. This shows how the imposed constraints increase the complexity of the problem.

Despite the fact that the optimization methods fail in some cases, their solutions are competitive to manual-design ones (fixed-dataflow) when they succeed, as shown in Figure 4.4(a)-(b). It shows the potential of automating the HW-mapping design process by properly formulating it into an optimization problem, which can significantly relieve the domain expert's effort on the back-and-forth tuning process. The challenge is the occasional failing of some methods. Moreover, GAMMA can consistently find valid and better solutions than others. Comparing with others, ***GAMMA finds solutions costing 224× to 440× less latency in Edge platform and 153× to (1.3E+7)× less latency in Cloud platform.***

4.4.3 S2: Flexible 2D Accel.

Objective: Latency. Figure 4.4(c)-(d) compares latency for accelerators with flexible aspect ratios (i.e., the accelerator can support both 1-level and 2-levels of parallelism). One interesting observation is that the fixed ShiDianNao-like dataflow and NVDLA-like dataflow shows better performance than baseline optimization methods at early and late layers respectively. This is because ShiDianNao-like dataflow parallelizes along X, Y dimensions and early layers of ResNet-18 have high X-Y values; similarly NVDLA-like parallelizes along C-K and late layers have high C-K values. For the baseline methods, since the number of parallelism dimensions is fixed, we search for the best 1-level and 2-level solution for 5K points each, and pick the better one. GAMMA searches across both 1-level and 2-level via the growth and aging operators described earlier. GAMMA finds valid and better solutions than others. Compared with other techniques, GAMMA finds solutions with 209× to 1,035× less latency in Edge and 337× to (7.1E+5)× less latency in Cloud platform.

Objective: Energy. Energy consumption depends on the number of active computations,



sGA	Por	CM	PSA	PSO	DE	1+1	GAM
1	3	0	4	1	1	3	0

(a)

sGA	Por	CM	PSA	PSO	DE	1+1	GAM
0	0	0	0	0	0	0	0

(b)

Fig. 4.5: The energy consumption of S2 on ResNet-18.

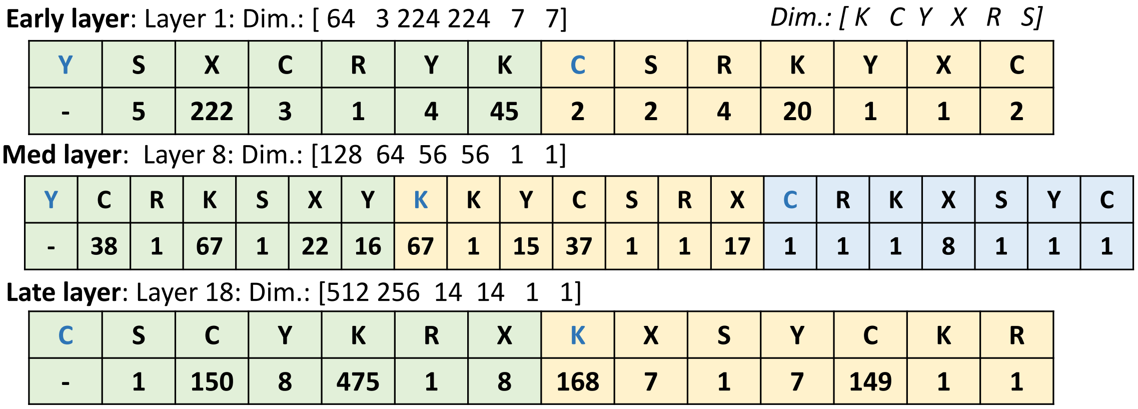
memory accesses, and SL/SG buffer usages. In the Edge platform in Figure 4.5(a), fixed dataflow show no advantage, and most optimization methods can find better solutions than the fixed dataflows. In the Cloud platform in Figure 4.5(b), optimization methods show competitive or better performance than Eyeriss-like and ShiDianNao-like dataflow. However, NVDLA-like dataflow shows high energy efficiency, since the K, C dimensions expand in ResNet-18, which gives more advantage to the dataflow that is skilled at layer with large K, C dimensions (NVDLA-like). They finish the computation with shorter time and less memory access, and hence cost less energy. However, these advantages do not show up when NVDLA-like dataflow is in the tight constraint (Edge platform), which has less SL/SG buffer and limited the parallelism opportunity. Across all fixed dataflow and optimization methods, GAMMA finds solutions costing 11× to 36× less energy in Edge platform and 2× to 42× less energy in Cloud platform.

4.4.4 S3: Scale-out Flexible 2D Accel.

In S3, we consider a more flexible scale-out infrastructure, which works in the design space of 3-level mapper.

that can not only support 2D PE arrays, but can form number of 2D PE array instances to parallelize computation in the third dimension. The number of parallelism is set to either 2-level or 3-level of parallelism. The 2-level (3-level) parallelism means we parallelize one dimension across PE arrays and one (two) dimension within PE arrays.

The growth of search space. Increasing the level of parallelism will exponentially



Early layer: Layer 1: Dim.: [64 3 224 224 7 7]

Dim.: [K C Y X R S]

Y	S	X	C	R	Y	K	C	S	R	K	Y	X	C
-	5	222	3	1	4	45	2	2	4	20	1	1	2

Med layer: Layer 8: Dim.: [128 64 56 56 1 1]

Y	C	R	K	S	X	Y	K	K	Y	C	S	R	X	C	R	K	X	S	Y	C
-	38	1	67	1	22	16	67	1	15	37	1	1	17	1	1	1	8	1	1	1

Late layer: Layer 18: Dim.: [512 256 14 14 1 1]

C	S	C	Y	K	R	X	K	X	S	Y	C	K	R
-	1	150	8	475	1	8	168	7	1	7	149	1	1

Fig. 4.6: GAMMA's found mapping of early, medium, and late layer of ResNet-18 in Figure 4.4(e).

increase the search space, which makes the performance of the optimization methods more critical to the found solutions. As shown in Figure 4.4(e)-(f), the number of cases that methods fail to find solution becomes significant. However, GAMMA can consistently find valid and better solutions, costing 241× to 644× less latency in Edge platform and 657× to (1.2E+5)× less latency in Cloud platform. Figure 4.4(g)-(h) shows the end-to-end latency of GAMMA in different accelerator systems. We can find GAMMA performs the best in S3, where the design space is several order larger than S1 and S2 but with more flexibility. It shows that GAMMA can explore the design space with sample efficiency and takes advantage of the flexibility of the mapping space.

Deep-dive into found solution. Figure 4.6 shows the HW-mapping solutions found by GAMMA on ResNet-18. At the early layer (Y, X dominant, Y=224, X=224), GAMMA found a mapper that parallelizes along Y dim at L2-mapper. At the medium layer (Y=56, K=128, C=64, X=56), GAMMA found a 3-level mapper, which maps across Y, K, and C dimensions. At the late layer (K, C dominant, K=512, C=256), GAMMA parallelize C at L2-mapper and K at L1-mapper. From the above observation, we find the automatically evolved solutions are consistent with some heuristic and insight from the manual-designed dataflows [1] [83, 81, 80].

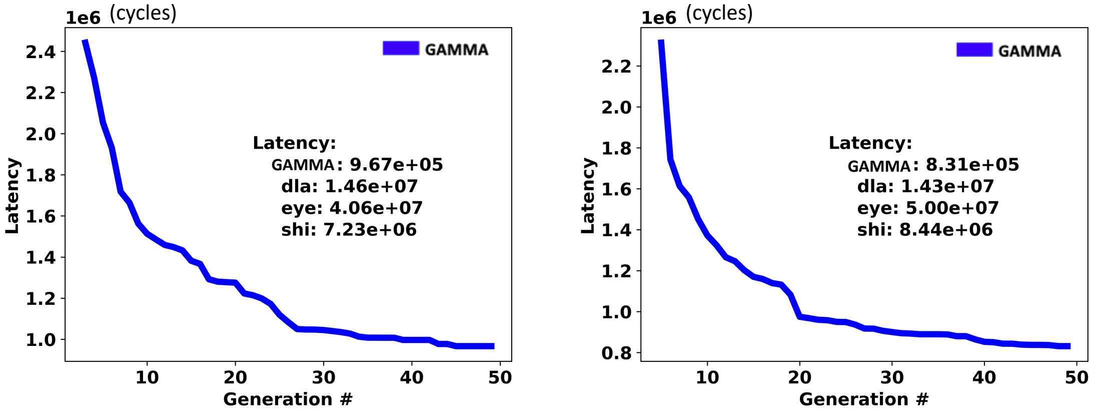
Other DNN models and end-to-end performance Table 4.5 shows the performance of

[1] The found solutions also show that by relaxing some tile size heuristics such as *deciding tile size by the integral multiple of PEs array sizes* could help reach better solutions.

Table 4.5: End-to-end performance and energy on S3 for a suite of DNN models using fixed mappings versus GAMMA. Bold means lowest values.

		MobileNet-V2				MnasNet			
Obj.	Accel.	GAMMA	NVDLA-like	Eyeriss-like	ShiDian Nao-like	GAMMA	NVDLA-like	Eyeriss-like	ShiDian Nao-like
Latency (cycles)	Edge	**9.67E+05**	1.46E+07	4.06E+07	7.23E+06	**8.31E+05**	1.43E+07	5.00E+07	8.44E+06
	Cloud	**1.85E+05**	9.33E+05	4.04E+07	6.04E+06	**1.39E+05**	4.03E+06	4.97E+07	7.46E+06
Energy (nJ)	Edge	**5.23E+08**	3.31E+09	9.62E+09	1.02E+10	**4.58E+08**	3.41E+09	9.707E+09	1.031E+10
	Cloud	**4.12E+08**	8.13E+08	9.61E+09	1.02E+10	**3.72E+08**	7.71E+08	9.706E+09	1.030E+10

		ShuffleNet				ResNet50			
Obj.	Accel.	GAMMA	NVDLA-like	Eyeriss-like	ShiDian Nao-like	GAMMA	NVDLA-like	Eyeriss-like	ShiDian Nao-like
Latency (cycles)	Edge	**5.20E+05**	8.41E+06	2.29E+07	3.89E+06	**6.49E+06**	1.04E+11	4.57E+08	1.31E+08
	Cloud	**3.41E+04**	6.28E+05	2.28E+07	3.37E+06	**3.84E+04**	2.91E+06	4.55E+08	1.13E+08
Energy (nJ)	Edge	**1.59E+08**	1.52E+09	4.647E+09	4.984E+09	**1.19E+09**	3.53E+10	1.1753E+11	1.245E+11
	Cloud	**1.17E+08**	2.57E+08	4.646E+09	4.978E+09	**1.13E+09**	2.14E+09	1.1751E+11	1.244E+11



(a) MobileNet-V2

(b) MnasNet

Fig. 4.7: End-to-end latency improvement over generations with GAMMA for S3 system and edge platform constraint.

GAMMA comparing to fixed dataflow on other widely-used DNN models. Here, for the interest of space, we only list the end-to-end performance, which is the sum of latency/energy of all the layers. Table 4.5 shows no fixed dataflow is good across all DNNs and platforms. For e.g., when considering latency, ShiDianNao-like performs the best on Edge platform, and NVDLA-like performs the best on Cloud platform. In contrast, the energy numbers follow NVDLA-like $<$ Eyeriss-like $<$ ShiDianNao-like; NVDLA-like gets advantage over the other two for energy via reuse across K and C dimensions (which dominate in most CNN-based models). Among all experiments in Table 4.5, GAMMA always provides the lowest latency and energy. ***Across models and platforms, GAMMA finds solutions costing***

Table 4.6: Two stage optimization for inter-layer parallelism on ResNet-18 * and VGG16 † for a multi-accelerator (S3) pipelined deployment. In the 1st state, we optimize for latency and identify the bottleneck layer (highlighted in bold), which determines the pipeline latency. In the 2nd stage, we optimize for energy (or power) by allowing the latency of other layers to increase, while staying less than the pipeline latency.

ResNet 18	Exp: Latency - Power				Exp: Latency - Energy			
	1st stage		2nd stage		1st stage		2nd stage	
Layer	Latency (cycles)	Power (mW)	Latency (cycles)	Power (mW)	Latency (cycles)	Energy (nJ)	Latency (cycles)	Energy (nJ)
1	4.9E+03	8.0E+02	7.1E+04	2.5E+02	1.1E+02	3.2E+06	3.8E+04	2.8E+06
2	**1.6E+05**	5.1E+02	1.6E+05	4.4E+02	1.0E+04	1.2E+08	5.2E+04	2.5E+07
6	4.2E+04	4.2E+02	1.3E+05	2.5E+02	2.9E+04	1.4E+08	6.1E+04	7.6E+07
7	4.6E+03	3.3E+02	6.5E+04	2.5E+02	8.9E+02	8.8E+06	2.4E+04	5.8E+06
8	4.9E+04	2.6E+02	1.5E+05	2.5E+02	**6.1E+04**	1.8E+07	6.1E+04	1.8E+07
11	2.9E+04	5.6E+03	1.5E+05	2.5E+02	2.6E+04	9.7E+07	3.7E+04	5.5E+07
12	1.3E+04	1.5E+04	1.4E+05	2.5E+02	2.2E+04	6.9E+07	4.2E+04	2.1E+07
13	4.1E+04	3.0E+02	1.5E+05	2.5E+02	2.5E+04	1.6E+07	5.1E+04	8.9E+06
16	2.1E+04	5.8E+02	5.9E+04	2.5E+02	2.3E+04	1.8E+08	3.7E+04	6.5E+07
18	4.9E+04	9.0E+03	1.5E+05	2.5E+02	5.5E+04	1.3E+07	5.9E+04	9.0E+06
19	2.6E+04	2.0E+04	1.4E+05	2.5E+02	1.0E+04	8.9E+07	3.8E+04	6.7E+07
Max.	**1.6E+05**	2.0E+04	**1.6E+05**	4.4E+02	**6.1E+04**	1.8E+08	**6.1E+04**	7.6E+07
Ave.	5.0E+04	**6.4E+03**	1.3E+05	**2.8E+02**	1.8E+04	**7.1E+07**	4.3E+04	**3.1E+07**
VGG16	Summary for Model VGG16							
Max.	**2.7E+06**	1.3E+05	**2.7E+06**	2.5E+02	**1.8E+06**	1.3E+09	**1.8E+06**	3.3E+08
Ave.	3.3E+05	**3.3E+04**	1.3E+06	**2.5E+02**	3.1E+05	**6.7E+08**	1.1E+06	**1.4E+08**

* We only display the layers with unique shape. Maximum and Average are calculated based on all 20 layers of ResNet-18. † We display the summary of VGG16 for the interest of space.

5× to (1.2E+5)× less latency and 2× to (1.6E+4)× less energy. Figure 4.7 tracks how GAMMA converges to its solution across generations; this shows its sample efficiency via rapid improvement over generations.

4.4.5 Two-stage Optimization for Inter-layer

So far in this chapter, we consider three systems: S1, S2, and S3, to parallelize the computation of a DNN layer, whose scenario can be termed as *intra-layer parallelism*. Next, we show how GAMMA can also be applied to the scenario of *inter-layer parallelism*. We consider a S3 system with *inter-layer parallelism* scenario, used in prior multi-accelerator systems [8, 13, 19], where each accelerator is handling one layer of a model, and the entire model is executed as layer-wise pipeline manner on the system.

Motivation The pipelined system can often bring higher throughput. However, it also owns the problem of being bottleneck by critical block. The layer-wise pipelined accelerator can be bottlenecked by some computation-heavy layer. As the bottleneck latency exists and may not be able to be further optimized, in this case, we relax other non-critical blocks by relaxing their timing constraint to achieve overall lower energy/power of the system.

Structure We apply a two-stage optimization method, where we optimize latency first and then power/energy at the second stage.

Stage I: optimize latency. We use GAMMA to find the mapping that optimizes the latency of each layer. We identify the bottleneck layer, whose latency decides the pipeline latency of the system.

Stage II: optimize power/energy. With the pipeline latency decided, we relax other layers by applying GAMMA again but optimizing power/energy at this stage with the awareness of not exceeding the pipeline latency. This is formulated by adding a heavy penalty when the searched solution exceeds the pipeline latency.

With the designed two-stage optimization, we could optimize the throughput of a layer-wise pipelined system at the first stage and further optimize its power/energy efficiency at the second stage.

Results Table 4.6 shows the HW performance of each layer in ResNet-18 in the 2-stage optimization scheme. In the *Latency-Power* experiment, we optimize latency first and their power next. After the first stage, it shows that the latency is bottleneck by the second layer, and it decides the *pipeline latency*. With the awareness of the pipeline latency, we optimize power at the second stage and find we could reduce the power by 95% comparing to the first stage when remaining at the same *pipeline latency*. The *Latency-Energy* experiment shows 58% reduction on energy consumption. Likewise, we execute the same flow on VGG16 and found it also effectively reduce the power by 99% and energy by 78% respectively.

4.5 Related Works

4.5.1 Dataflow Design in DNN Accelerators

Dataflow design has been a popular topic in the research of DNN Accelerators. Multiple hand-designed dataflows have been used across accelerators, categorized [83] as output-stationary [81, 93, 94], weight-stationary [80, 89, 90, 91, 92], row-stationary [83], input stationary, and no local reuse [158, 159]. In this work, we provide a framework to automatically determine an optimized dataflow and mapping. GAMMA can be used at compile-time to configure in the mapping if the underlying accelerator supports multiple dataflows [2, 12], or at design-time to determine the right dataflow for a custom accelerator developed for running a fixed set of DNNs.

4.5.2 HW Mapping Space Search and Exploration

Many recent works have been developed to tackle DNN HW mapping. However, since the search space is extremely large, many of them restrict the search space by considering only part of the aspects of the HW mapping search space. Some consider a limited combination of HW mappings and pick among them [12]. Some constrain the parallelizing dimension to a few choices [13, 14, 15, 16]. Some fixed the computation order to a subset of all combinations [17, 18, 19, 20, 21]. Some vastly reduce the space of tiling sizes [22] by a heuristic, or large step size, e.g., power of two [23]. Interstellar [10, 9] considers all three aspects of HW-mapping, but they constrain the search space by limiting the choice on each aspect such as the choices of loop order, parallelizing dimension. All these prior arts exclusively rely on exhaustive/random search with the help of coarse-grained striding enumeration or heuristics-based pruning. On the other hand, to search the mapping space with sample efficiency, Suda et. al[16] and TensorComprehensions [31] uses genetic algorithm, AutoTVM [101, 24] uses simulated annealing and boosted tree, Reagen et. al, [26] uses Bayesian optimization, RELEASE [27] uses RL to formulated a more guided search by ML technique. However, these ML-based algorithms need to work in a pre-defined rigid design space, where the level of parallelism is restricted, and hence the mapping space is constrained. The mappers in Timeloop [4] and Simba [8] explore the full search space; however, they rely on exhaustive/random search. In this work, we explore a full search space, but with a ML-based guided search method with sample efficiency.

4.6 Summary

GAMMA constructs and searches through a comprehensive map-space comprising of computation order, tile-sizes, parallelization strategy, and up to three parallelization levels, enabling it to target a wide variety of fixed and flexible single and multi-accelerator systems. The proposed encoding scheme transforms the HW-mapping problem to an optimization problem, which enables the user to directly use off-the-shelf optimization algorithms for mapping. These form our baselines. GAMMA introduces three new GA operators, enabling a domain-specific flexible search space unlike most off-the shelf optimization algorithms. We automate GAMMA as a black-box optimizer for the HW-mapping problem. This reduces the learning curve and saves manual-tuning effort for ML practitioners exploring the HW-mapping space. GAMMA encapsulates an end-to-end workflow, which generates outputs compatible with an open-source cost model [58]. We will open-source the GAMMA infrastructure after the paper gets published. With new DNN models and new accelerators being proposed at an unprecedented rate, GAMMA allows researchers to quickly explore the HW efficiency of emerging DNNs without time-consuming human-in-the-loop mapping and tuning processes.

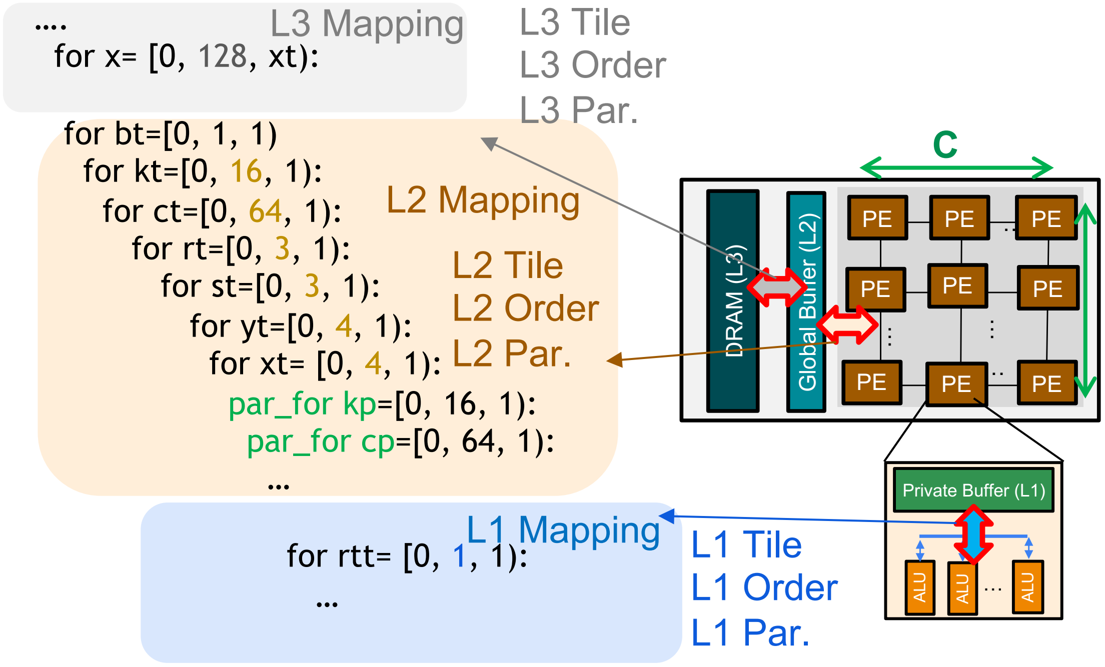
CHAPTER 5
DEMYSTIFYING MAP SPACE EXPLORATION AND THE IMPROVEMENT TECHNIQUES

In chapter 3 and chapter 4, we define the map space of DNN accelerator and propose an sampling efficient mapper to explore the map space. In this chapter, we will give a deep dive analysis on the mappers and the whole MSE framework. We want to answer why one mapper is better than the other and how the mapper actually increase the sampling efficiency. we hope these insights can propel the follow-up works on developing new and more powerful mapper.

Next, we identify the challenge of the MSE framework for the new complex DNN workloads, which is capability and the sparsity in the workloads. We propose two simple but powerful techniques to ameliorate these challenges.

Map space exploration (MSE) is critical for DNN accelerator efficiency. It is a complex and challenging problem because the search space is often massive. There are various search algorithms (i.e., *mappers*) [4, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 7, 29, 30, 31, 32, 33] to find optimized mappings for DNN accelerators and workloads.

Despite the success achieved by these prior efforts, MSE remains a computationally challenging problem. This is because the search space for legal mappings for even a single layer of a modern DNN (e.g., ResNet-50) on a typical edge class accelerator [1] is $\sim O(10^{24})$ [5, 6] which would require more time than the age of the earth to search exhaustively (assuming 1msec to evaluate each mapping sample). This gets exacerbated as newer and ever larger DNN models are being created with increasing frequency, especially thanks to the success of neural architecture search techniques [34, 35, 36, 37, 38]. Furthermore, the advent of *compressed-sparse* DNNs [39, 40, 41, 42, 43, 44, 45], whose mappings are not



```
....
  for x= [0, 128, xt):
    for bt=[0, 1, 1)
      for kt=[0, 16, 1):
        for ct=[0, 64, 1):
          for rt=[0, 3, 1):
            for st=[0, 3, 1):
              for yt=[0, 4, 1):
                for xt= [0, 4, 1):
                  par_for kp=[0, 16, 1):
                    par_for cp=[0, 64, 1):
                      ...
                  for rtt= [0, 1, 1):
                    ...
```

Fig. 5.1: A loop-nest representation of a NVDLA-like [80] mapping.

performance-portable across sparsity levels (a key finding in this chapter), further increase MSE burden.

Researching more sophisticated scalable and sparsity-aware MSE techniques is at least partially hampered by the fact that even though prior approaches have *empirically shown* that their techniques work, none of them demonstrate *why* they work and the insight behind their optimization techniques.

To this end, this chapter aims to develop scalable MSE approaches for future complex DNN workloads. However, instead of proposing yet another mapper that *just works*, we first distill the knowledge from prior mappers spanning heuristics and learning-based optimization approaches to demystify MSE as a problem. We analyze their behavior, learn from their best traits, and use these learnings to scale MSE to more complex workloads.

5.1 Map Space Exploration (MSE)

A canonical MSE framework is shown in Figure 5.2. Several prior works [4, 6, 5, 7, 160] have identified MSE as a separate problem distinct from accelerator design-space

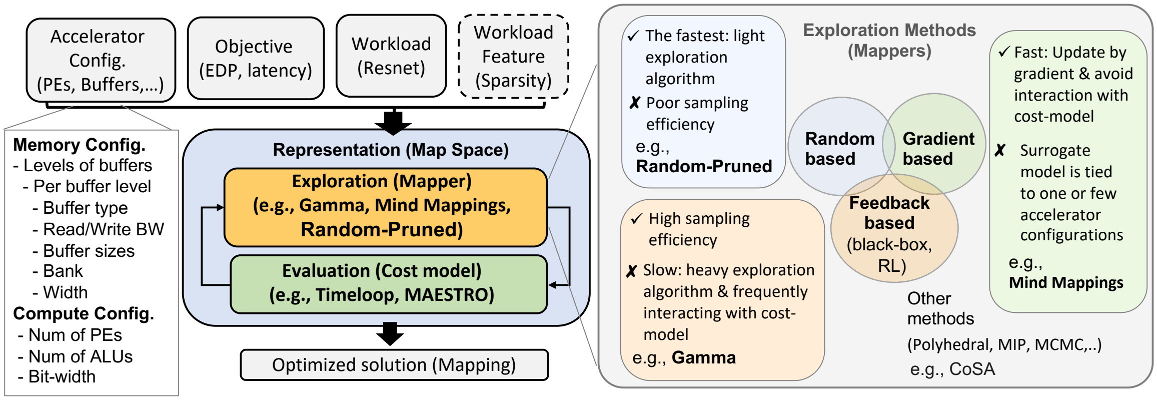

Accelerator Config. (PEs, Buffers,...)
Objective (EDP, latency)
Workload (Resnet)
Workload Feature (Sparsity)
Memory Config.
- Levels of buffers
- Per buffer level
- Buffer type
- Read/Write BW
- Buffer sizes
- Bank
- Width
Compute Config.
- Num of PEs
- Num of ALUs
- Bit-width
Representation (Map Space)
Exploration (Mapper) (e.g., Gamma, Mind Mappings, Random-Pruned)
Evaluation (Cost model) (e.g., Timeloop, MAESTRO)
Optimized solution (Mapping)
Exploration Methods (Mappers)
✓ The fastest: light exploration algorithm
✗ Poor sampling efficiency
e.g., Random-Pruned
Random based
Gradient based
Feedback based (black-box, RL)
✓ High sampling efficiency
✗ Slow: heavy exploration algorithm & frequently interacting with cost-model
e.g., Gamma
✓ Fast: Update by gradient & avoid interaction with cost-model
✗ Surrogate model is tied to one or few accelerator configurations
e.g., Mind Mappings
Other methods
(Polyhedral, MIP, MCMC,..)
e.g., CoSA

Fig. 5.2: A canonical Map Space Exploration framework.

exploration (DSE). DSE includes identifying the right compute and memory configurations for an accelerator (§2.2), as shown by the accelerator configuration in Figure 5.2, within constraints such as total FLOPS, area, and power. MSE, meanwhile, takes the accelerator configuration and DNN workloads (size, shape, and additional features such as sparsity level of weight and/or activations) as input and finds optimized mappings given an objective (e.g., latency, throughput, energy, energy-delay-product (EDP), and so on). MSE may be run at compile time within a mapping optimizer [135] after an accelerator is deployed, or at design-time in conjunction with DSE for co-optimizing the mapping and hardware configuration [25, 161].

The MSE process often includes three parts: *Representation* of search space, *Evaluation method*, and *Exploration method*. The representation will define the scope of the searching problem and the size of the search space. After the search space is defined, we form an optimization loop that includes exploration and evaluation. The optimization continues till the MSE converges, or reaches a given sampling budget or wall-clock run time budget.

5.1.1 Evaluation Method (Cost Model)

In MSE, we often rely on a DNN accelerator *cost model* to estimate the performance of a certain mapping on a given accelerator for a given workload. These cost models are typically analytical, enabling rapid evaluation of different design-points in a matter of ms. Some widely used cost models include Timeloop [4], MAESTRO [68], dMazeRunner [9], Interstellar [10], SCALE-sim [162] and others [163, 164]. These cost models can model different kinds of accelerators (systolic arrays [162], flexible spatial arrays [68, 4, 9], sparse accelerators [165], and so on) and capture each accelerator's map space in different formats. In this chapter, we use Timeloop [4] as our cost model[1] which is validated against real chips [147, 8].

5.1.2 Representation

The map space includes Tiling, Order, and Parallelism. How this map space is *represented* can often determine the success or failure of the MSE process. The efficiency of the representation ties closely to the underlying DNN accelerator cost model, described above. For example, Timeloop [4] uses loop-nest representation for mappings (thereby describing the computation to be run over space and time) while MAESTRO [68] uses a set of data-centric directives to represent which data elements are mapped over space and time; Timeloop supports detailed buffer hierarchies while MAESTRO infers the hierarchies from the mapping; Timeloop has strict constraints on tile size choices across hierarchies and parallelism dimensions while MAESTRO imposes loose constraints on them (and factors in the inefficiency by run time report), and so on. All these different details should be carefully understood to create a compact map-space representation, to avoid searching through invalid mappings. In this chapter we leverage the Timeloop representation, and ensure that all the candidate mappings generated by various mappers during MSE are all legal.

5.1.3 Exploration Method (Mapper)

The exploration algorithm in MSE (Figure 5.2) is called a mapper. Dozens of different DNN mappers have been proposed, which we categorize into *random search based* [4, 8, 9, 10, 11], *feedback-based* (including reinforcement learning and black-box optimization) [5, 24, 25,

[1] Timeloop includes *both* a cost model and mappers. Throughout this chapter, we refer to the former as Timeloop and the latter as Timeloop-mapper. Timeloop-mapper itself supports a variety of search heuristics, with the default being Random-Pruned. We also run other mappers using Timeloop as the cost model.

160, 29, 28], *gradient-based* [6], and *others* (including mathematical optimization, MCMC, polyhedral transformations, and heuristics) [7, 29, 30, 31, 32, 33] (Figure 5.2). The random search-based either apply random sampling on the search space or apply pruned random search [4, 135], which prunes off the redundant search space to increase the sampling efficiency. The feedback-based use a learning algorithm to interact with the cost model and keep improving its solution. The run time of both random search-based and feedback-based depend heavily on the run time of the cost model, potentially becoming the bottleneck of the MSE run time. Gradient-based methods uses a *differentiable* surrogate model, which eliminates this bottleneck and can update the solution directly by the gradient of the loss. We do a deeper dive within these three types in §5.2.3.

5.1.4 Why MSE Matters

MSE bridges the gap between two active trends: (1) efficient DNN model design [126, 166, 167] (which has led to a huge diversity in layer shapes/sizes and emergence of sparsity in state-of-the-art DNN models) and (2) flexible hardware accelerators that support diverse mappings (dataflows + tile sizes) via configurable buffer hierarchies [84] and on-chip interconnect topologies [2, 3] as an answer to the first trend. MSE is crucial for extracting performance and energy-efficiency from the accelerator as there can be multiple orders of of difference in performance and energy-efficiency between good and bad mappings, as prior works have demonstrated [4, 5, 6].

While several mappers are being actively developed [4, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 7, 29, 30, 31, 32, 33], there is no work, to the best of our knowledge, that has focused on understanding how different mappers navigate the map-space, how different mapping axes contribute to the performance, and trade-offs between search approaches, which is the focus of this chapter.

5.2 Quantitative MSE Analysis

In this section, we perform a quantitative analysis of the three classes of mappers described in §5.1.3 to identify *when* and *why* one works better than the other. The goal of this analysis is to educate the DNN accelerator research community on Mapper design, rather than propose yet another mapper.

5.2.1 Methodology

Workload. We consider workloads from different models: Resnet [142], VGG [146], Mnasnet [34], Mobilenet [126], and Bert-large [69]. Some frequently referenced workloads across different experiments are described in Table 5.1.

Hardware Accelerator. We model the hardware accelerator using Timeloop [4]. The hardware accelerator we evaluate houses three-level of buffer hierarchies: DRAM, a 64KB shared global buffer, and 256B private local buffer for each of the 256 PE. Each PE houses 4 ALU units (Accel-B in Table 5.1). We also model the accelerator the Mind Mappings paper [6] uses (Accel-A), whose configuration is similar but with different sizing as shown in Table 5.1.

Objective. We use multi-objective – Energy and Latency (Delay), throughout the optimization process. When optimization finishes, we select the solution with the highest Energy-Delay-Product (EDP) on the Pareto frontier. We use EDP as the performance criteria of found mapping. Note that any formulation of the objective can also be used such as power, area, performance-per-watt, performance-per-mm2, and so on.

Experiment Platform. We run experiments using a desktop with a 12-core Intel I7-6800K CPU and a Nvidia GTX1080 to train the surrogate model in Mind Mappings.

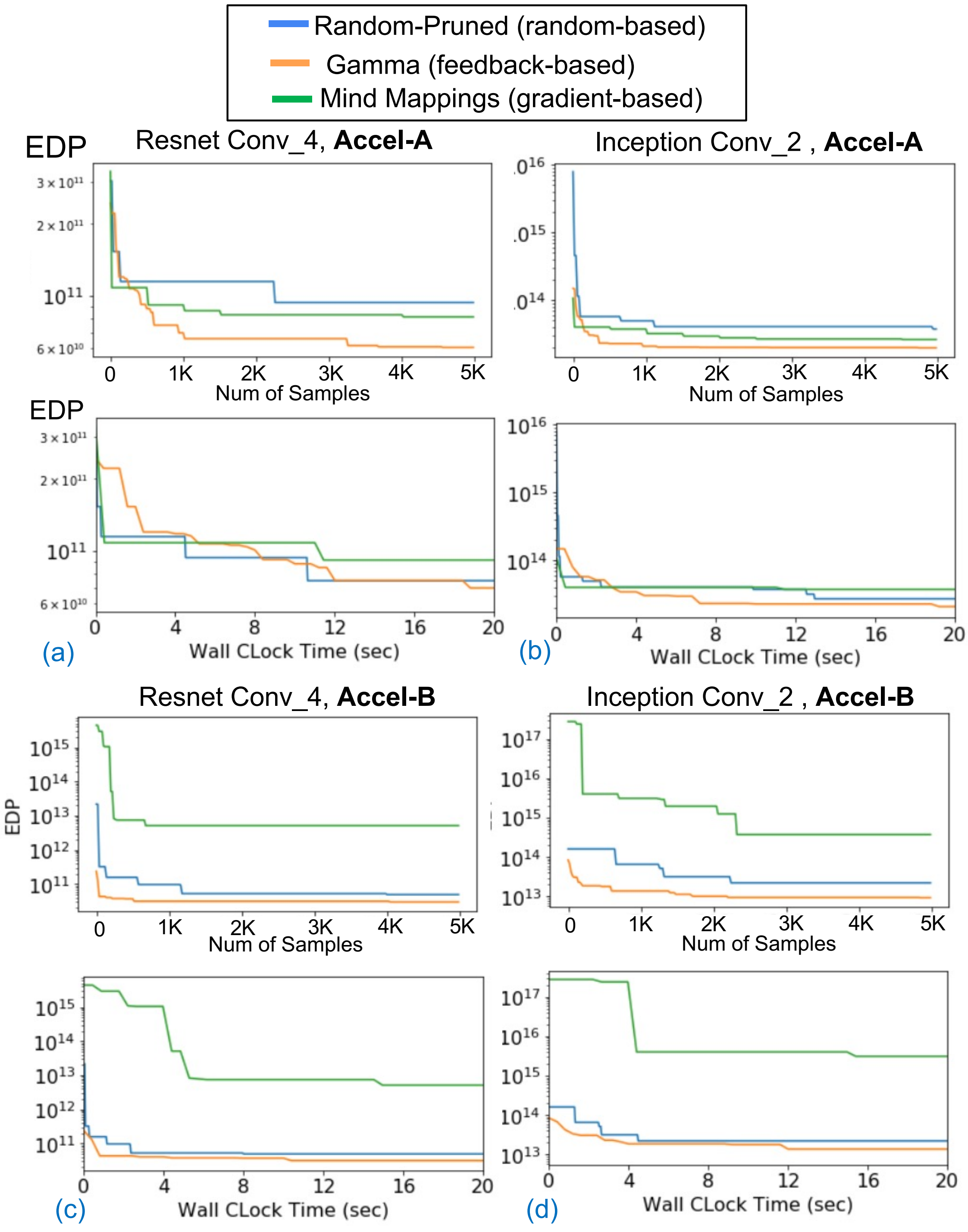


Fig. 5.3: Comparisons of different types of mappers. Top figures show the converge curve across number of samples. Bottom figures show the converge curve across wall clock time.

Table 5.1: The description of the relevant workloads and accelerator configurations used across evaluations.

Workload	(B,K,C,Y,X,R,S)
Resnet Conv_3	(16,128,128,28,28,3,3)
Resnet Conv_4	(16,256,256,14,14,3,3)
Inception Conv_2	(16,192,192,27,27,5,5)
Workload	(B,M,K,N)
Bert-Large KQV	(16,1024,1024,512)
Bert-Large Attn	(16,512,1024,512)
Bert-Large FF	(16,4096,1024,512)

	Accelerator Configuration
Accel A	512 KB shared buffer, 64 KB private buffer per PE, 256 PEs, 1 ALUs per PE
Accel B	64 KB shared buffer, 256 B private buffer per PE, 256 PEs, 4 ALUs per PE

5.2.2 Size of Map Space

The size of the map space heavily depends on representation. In this chapter, we follow the efficient representation used by Timeloop to represent the three mapping axes. We use CONV2D (7 for-loop) as workload and 3-level of buffer hierarchy (DRAM, L2, L1) as architecture configuration as an example to guide the discussion of map space.

Tile sizes. Buffers at each level of the scratchpad memory hierarchy will have a dedicated tile size for each of the dimensions, as shown by the different tile sizes within the 7 for-loops of the L2 mapping in Figure 5.1. The total possible combination depends on the tensor shape of each workload and increases exponentially with the number of buffer hierarchies.

Loop Order. Each buffer level would have a dedicated permutation of loop order. E.g., in Figure 5.1, the loop order in L2 mapping from outer to inner loop is (B,K,C,R,S,Y,X). The total combinations become $(7!)^3$ (we have 3 buffer levels in our example).

Parallelism. Parallelism happens across levels of compute units (2-level of compute units in Figure 5.1, i.e., across PEs and ALUs). At each level of the compute unit, we can choose to parallelize from 0 (no parallelism) to 7 (all parallelism) dimensions. The total combination becomes $2^{7\times2}$.

Map-Space. The Cartesian product of these sub-spaces leads to the size of the entire

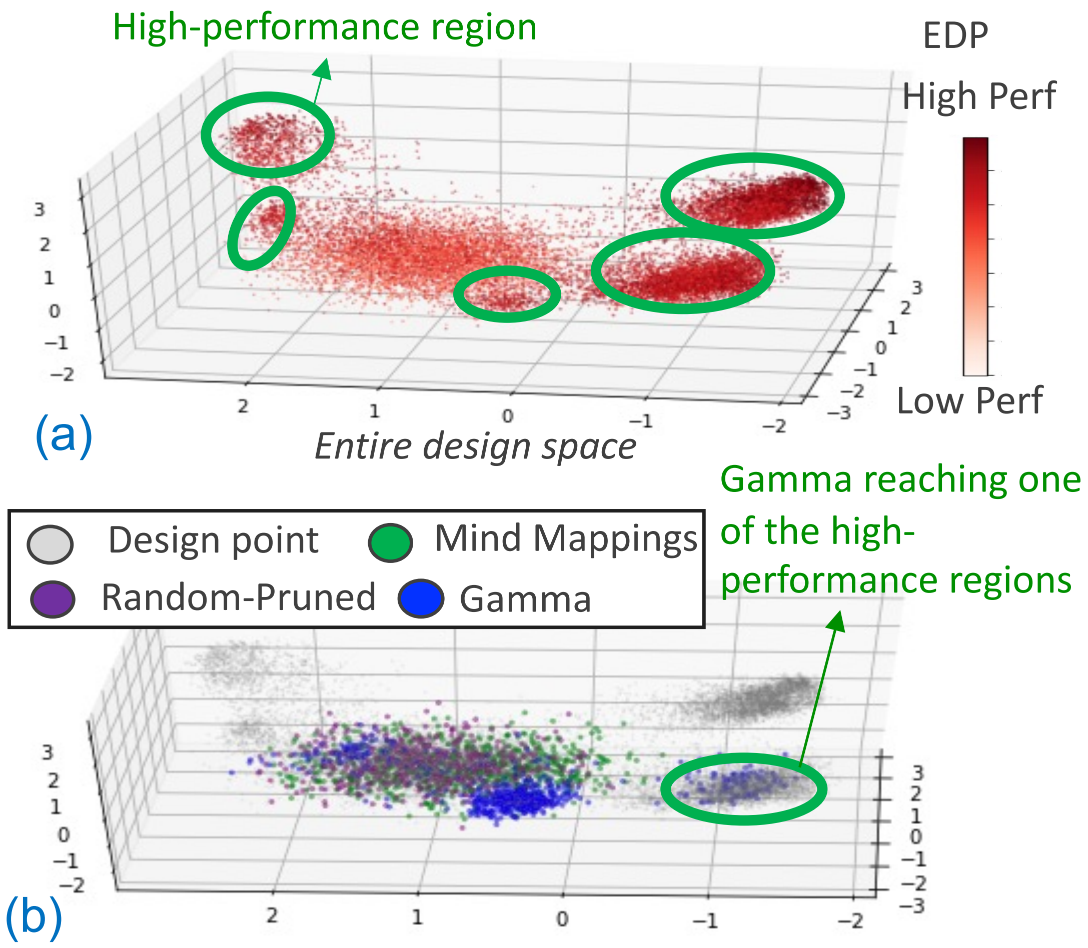


Fig. 5.4: (a) shows the sampled points by exhaustively sampling the search space of (Resnet Conv_4, Accel-A). The 3D visualization is projected by PCA dimension reduction. (b) shows the sampled points of different types of mappers in this search space.

map space, which is at the level of $O(10^{21})$ for the workloads discussed in §7.2.

5.2.3 Understanding Mapper Sampling Efficiency

As discussed in §5.1.3, we categorize state-of-the-art mappers into three major techniques (Figure 5.2). We select state-of-the-art mappers out of each category and compare their characteristics with respect to search speed and sampling efficiency[2]. We select Timeloop's Random-Pruned [4] from random-based, Gamma [5] from feedback-based, and Mind Mappings [6] from gradient-based methods[3].

[2] The performance improvement over number of sampled points.

[3] Random-Pruned and Mind Mappings both natively work with the Timeloop cost model. Gamma was originally demonstrated with MAESTRO, but we extended its open-source codebase [5] to use the Timeloop

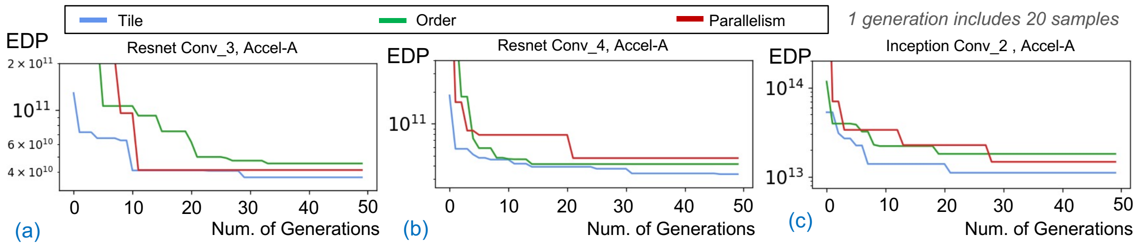


Fig. 5.5: Mapping axes sensitivity analysis using the mutation operators in Gamma [5]. E.g., Tile (blue): means mutating tile only, i.e, only tile is explored, and other mapping axes are fixed, similarly for (mutate-)Order and (mutate-)Parallelism.

- **Random-Pruned (random-based):** Random-Pruned [4] uses random sampling on a pruned search space. The pruning strategies are based on heuristics, e.g., permutations do not matter for the innermost tiling level and for tile sizes that are one [4].

- **Gamma (feedback-based):** Gamma [5], a genetic algorithm (GA) based method, keeps a population of candidate solutions, uses specifically designed mutation operators to perturb populations to explore different mapping axes (tile, order, parallelism), and uses crossover to create next generations of populations. Gamma has been shown to beat other optimization techniques, including reinforcement learning [5, 168].

- **Mind Mappings (gradient-based):** Mind Mappings [6] trains a neural network based surrogate model via offline sampling of millions of data points collected from the cost model. It uses the loss gradient to update its solution. During MSE, it utilizes gradient-descent on this surrogate model to find mappings, instead of searching.

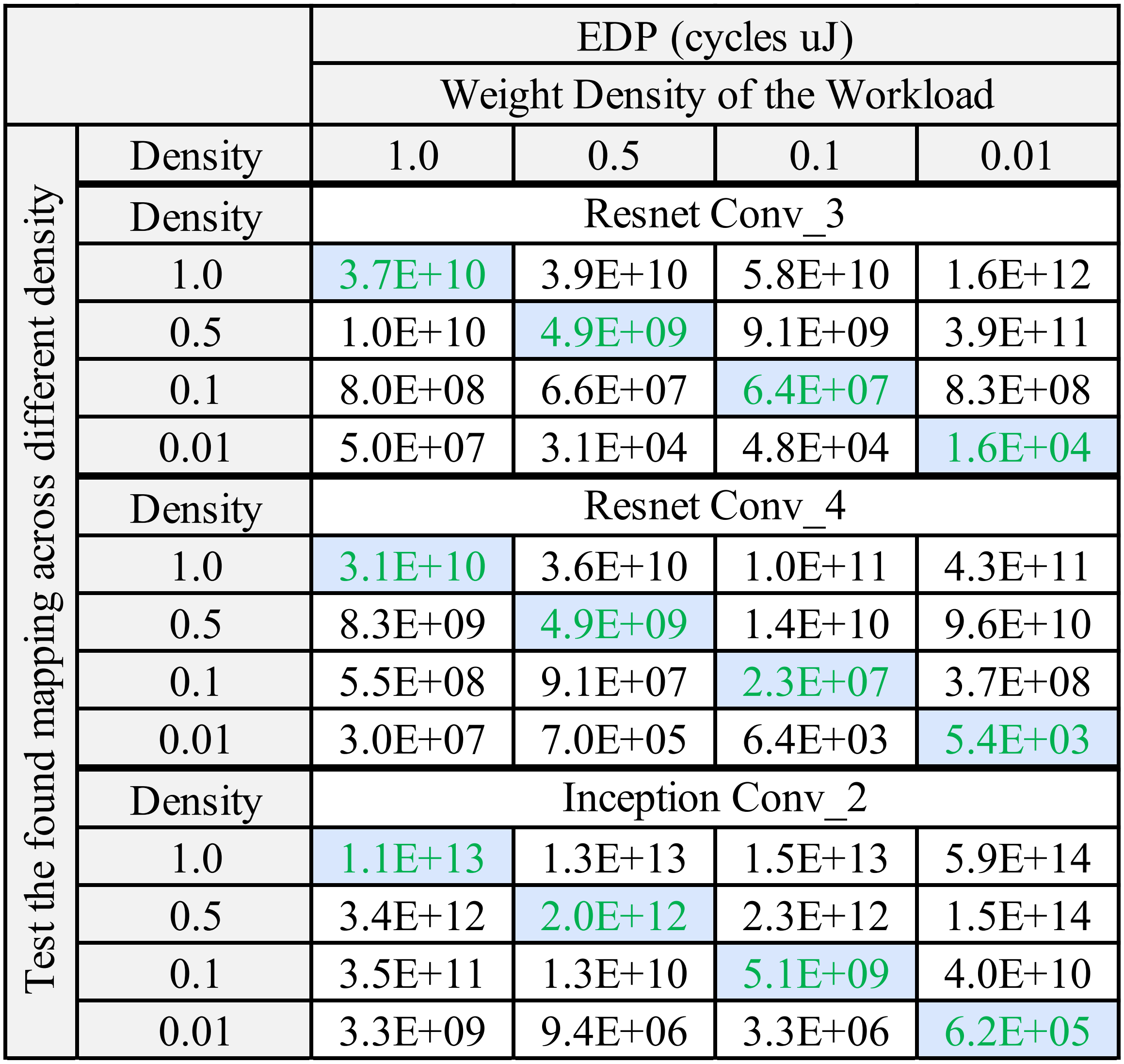
In the following evaluation case study, we show two sets of accelerator configurations: Accel-A, on which the surrogate model is trained, and Accel-B, an unseen accelerator configuration for the surrogate model.

Trained Accelerator Configuration (Accel-A)

Iso-sampling points Comparisons. We set the sampling budget to 5,000 points and compare the sampling efficiency of algorithms in the top figures of Figure 5.3(a)(b). The

cost model for this chapter. We leave the task of porting representative mappers from the *others* category (§5.1.3) on to a common cost model and analyzing them as future work.

Table 5.2: MSE for workload with weight sparsity. In each columns, the blue cell shows the performance of the optimized mapping for the sparse workload; the rest of the cells shows the performance of the same mapping tested with the workload with different sparsity. We highlight the best-performing cell of each row by green text. We can observe that the blue cells overlap with green texts, indicating that different workload with different sparsity levels do require different mapping to optimize the performance.



		EDP (cycles uJ)			
		Weight Density of the Workload			
Test the found mapping across different density	Density	1.0	0.5	0.1	0.01
	Density	Resnet Conv_3			
	1.0	3.7E+10	3.9E+10	5.8E+10	1.6E+12
	0.5	1.0E+10	4.9E+09	9.1E+09	3.9E+11
	0.1	8.0E+08	6.6E+07	6.4E+07	8.3E+08
	0.01	5.0E+07	3.1E+04	4.8E+04	1.6E+04
	Density	Resnet Conv_4			
	1.0	3.1E+10	3.6E+10	1.0E+11	4.3E+11
	0.5	8.3E+09	4.9E+09	1.4E+10	9.6E+10
	0.1	5.5E+08	9.1E+07	2.3E+07	3.7E+08
	0.01	3.0E+07	7.0E+05	6.4E+03	5.4E+03
	Density	Inception Conv_2			
	1.0	1.1E+13	1.3E+13	1.5E+13	5.9E+14
	0.5	3.4E+12	2.0E+12	2.3E+12	1.5E+14
	0.1	3.5E+11	1.3E+10	5.1E+09	4.0E+10
	0.01	3.3E+09	9.4E+06	3.3E+06	6.2E+05

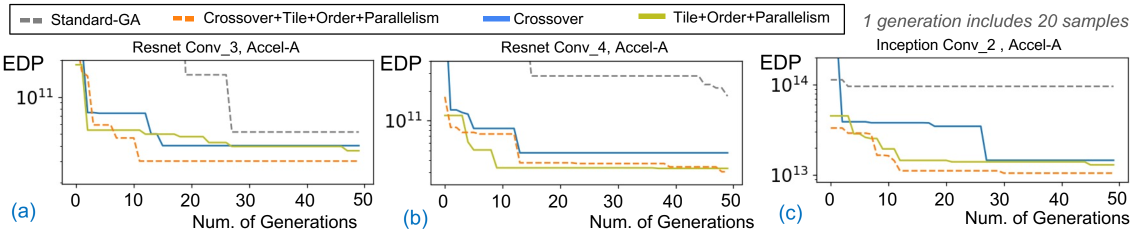


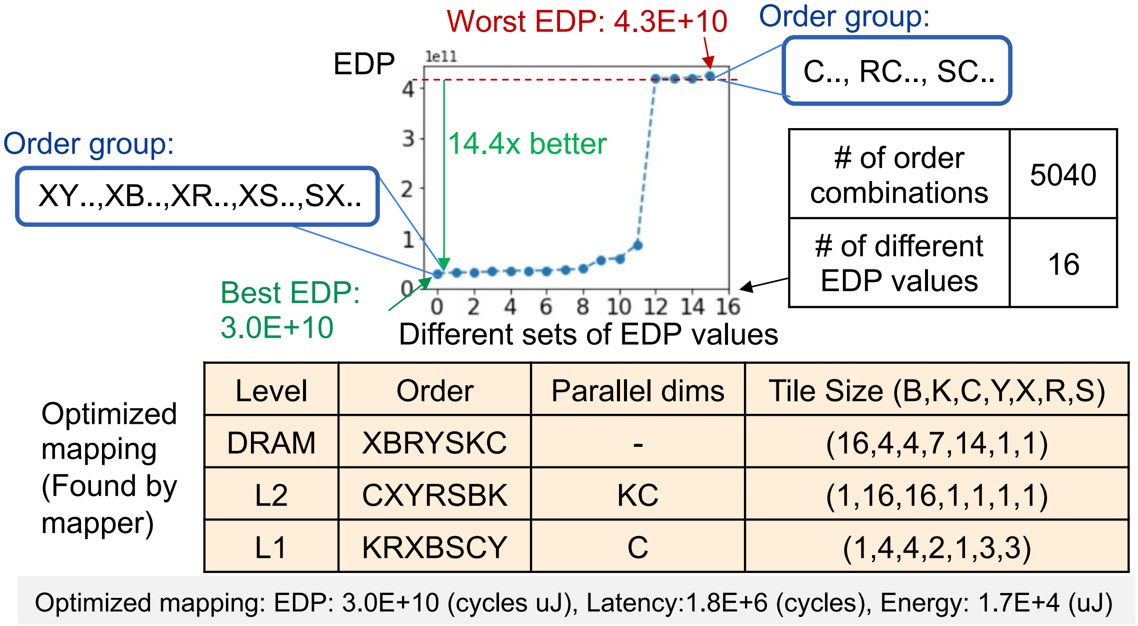
Fig. 5.6: Crossover (blending two mappings) sensitivity analysis using operators in Gamma [5]. Standard-GA uses the standard mutation and crossover (without domain-specific operators along each mapping axes designed in Gamma [5]).

Table 5.3: The optimized EDP performance of inner and outer product style mapping on sparse-dense GEMM workloads in Bert-large model [69]. The workload density indicates the density of the sparse matrix. Bert-large KQV: the key/ query/ value projection operations. Bert-large Attn: the attention operation, Bert-large FC: the FC operations at the end of attention blocks.

EDP (cycles uJ)						
	Bert-large KQV		Bert-large Attn		Bert-large FC	
Workload Density	Inner Product	Outer Product	Inner Product	Outer Product	Inner Product	Outer Product
1.0	7.6E+11	9.8E+11	1.9E+11	2.5E+11	7.8E+14	9.1E+14
0.5	1.1E+11	1.4E+11	2.8E+10	3.6E+10	1.5E+14	1.5E+14
0.1	9.0E+08	1.6E+05	3.4E+08	3.6E+08	1.4E+12	1.1E+08
0.01	1.9E+05	1.6E+05	2.0E+05	8.0E+04	1.8E+08	1.1E+08

Table 5.4: Comparisons of sparsity-aware technique and static-density heuristic when tackling the activation sparsity. The static-density heuristic searches mapping for a fixed density level (1.0, 0.5, or 0.1), marked as blue cells. Sparsity-aware technique searches the mapping scored with its performance across 5 density level (1.0, 0.8, 0.5, 0.2, and 0.1). We highlight the best-performing one in each row with green text. It indicates that sparsity-aware technique can find mapping with comparable performance to the static-density ones across a range of sparsity (density 1.0 - 0.05).

	EDP (Energy uJ)			
Workload Density	**Sparsity-aware**	**Static density 1.0**	**Static density 0.5**	**Static density 0.1**
Resnet Conv_3, Accel-B				
1.0	2.40E+13	2.39E+13	2.41E+13	2.46E+13
0.9	1.75E+13	1.94E+13	1.76E+13	1.79E+13
0.8	1.23E+13	1.54E+13	1.24E+13	1.26E+13
0.7	8.26E+12	1.18E+13	8.30E+12	8.46E+12
0.6	5.21E+12	8.69E+12	5.24E+12	5.34E+12
0.5	3.02E+12	6.06E+12	3.02E+12	3.10E+12
0.4	1.55E+12	3.90E+12	1.56E+12	1.59E+12
0.3	6.59E+11	2.21E+12	6.63E+11	6.77E+11
0.2	1.98E+11	1.00E+12	1.99E+11	2.04E+11
0.1	4.78E+10	2.65E+11	4.81E+10	4.78E+10
0.05	1.28E+10	7.34E+10	1.29E+10	2.67E+10
Inception Conv_2, Accel-B				
1.0	7.77E+15	7.77E+15	7.93E+15	7.83E+15
0.9	5.67E+15	6.33E+15	5.79E+15	5.71E+15
0.8	3.99E+15	5.00E+15	4.08E+15	4.02E+15
0.7	2.67E+15	3.84E+15	2.74E+15	2.69E+15
0.6	1.69E+15	2.82E+15	1.73E+15	1.70E+15
0.5	9.78E+14	1.97E+15	9.78E+14	9.83E+14
0.4	5.02E+14	1.26E+15	5.21E+14	5.05E+14
0.3	2.13E+14	7.16E+14	2.23E+14	2.14E+14
0.2	6.39E+13	3.22E+14	8.64E+13	6.38E+13
0.1	1.55E+13	8.37E+13	4.49E+13	1.53E+13
0.05	4.12E+12	2.25E+13	2.53E+13	3.98E+12

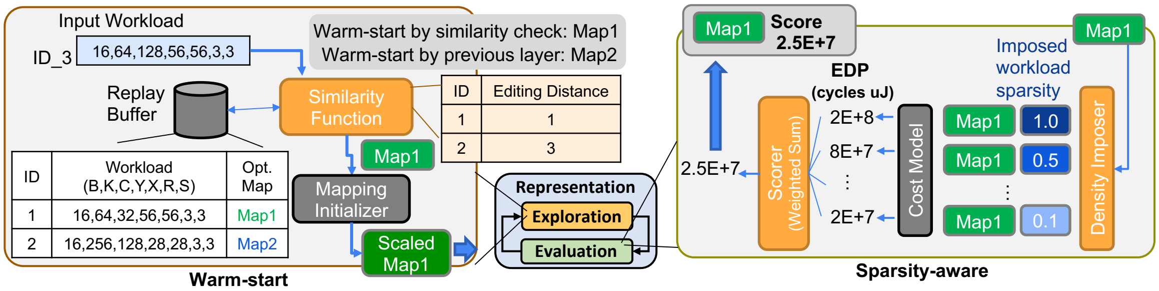


# of order combinations	5040
# of different EDP values	16

Level	Order	Parallel dims	Tile Size (B,K,C,Y,X,R,S)
DRAM	XBRYSKC	-	(16,4,4,7,14,1,1)
L2	CXYRSBK	KC	(1,16,16,1,1,1,1)
L1	KRXBSCY	C	(1,4,4,2,1,3,3)

Optimized mapping: EDP: 3.0E+10 (cycles uJ), Latency:1.8E+6 (cycles), Energy: 1.7E+4 (uJ)

Fig. 5.7: The EDP difference of the same mapping with different loop order. We sweep through all 7! order combinations assuming all the buffer level utilize the same order. The 7! different mapping leads to 16 different EDP performance, with the best and the worst EDP differs by 14.4x times (under Resnet Conv_4, Accel-B).



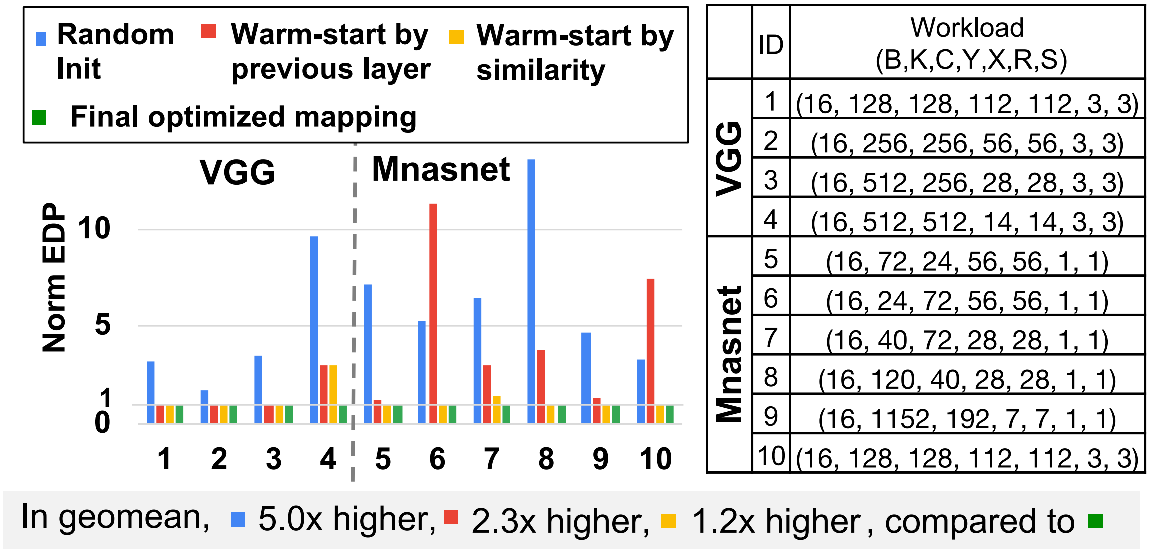
ID	Workload (B,K,C,Y,X,R,S)	Opt. Map
1	16,64,32,56,56,3,3	Map1
2	16,256,128,28,28,3,3	Map2

ID	Editing Distance
1	1
2	3

Fig. 5.8: The workflow of Warm-start and Sparsity-aware techniques in MSE.

random-based method progresses the slowest over number of samples. Among the gradient-based and feedback-based, the gradient-based method progresses faster at the start owing to its direct gradient feedback. However, with more number of samples, the feedback-based method starts to perform better. It is because the gradient-based method is more prone to fall into local optimum (discussed later) while the feedback-based methods typically work well for global optimization problems.

Iso-time Comparisons. We set a tight time budget, 20 seconds, and track the performance to wall clock time in the bottom figures of Figure 5.3(a)(b). Despite their better



	ID	Workload (B,K,C,Y,X,R,S)
VGG	1	(16, 128, 128, 112, 112, 3, 3)
	2	(16, 256, 256, 56, 56, 3, 3)
	3	(16, 512, 256, 28, 28, 3, 3)
	4	(16, 512, 512, 14, 14, 3, 3)
Mnasnet	5	(16, 72, 24, 56, 56, 1, 1)
	6	(16, 24, 72, 56, 56, 1, 1)
	7	(16, 40, 72, 28, 28, 1, 1)
	8	(16, 120, 40, 28, 28, 1, 1)
	9	(16, 1152, 192, 7, 7, 1, 1)
	10	(16, 128, 128, 112, 112, 3, 3)

In geomean, 🟦 5.0x higher, 🟥 2.3x higher, 🟨 1.2x higher , compared to 🟩

Fig. 5.9: Performance comparisons of initialized solution by Random Init and two types of warm-start Init comparing to the final optimized performance (after search). The EDP values are normalized by final optimized EDP (green bars).

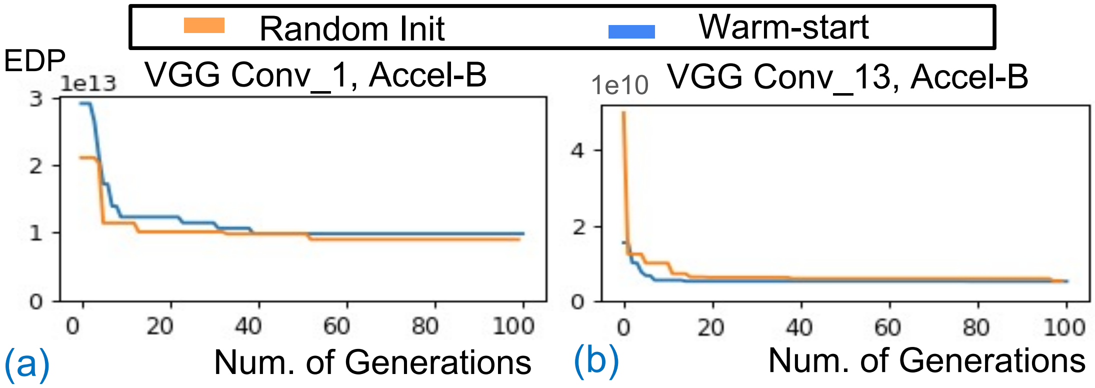


Fig. 5.10: The performance convergence curve with random initialization and warm-start (by similarity) initialization at the (a) first layer and (b) a later layer of VGG16.

sampling efficiency, the feedback-based and gradient-based methods do not show a clear edge over the random-based method within tight wall-clock run time budget. Random-based methods do not have costly built-in learning algorithms as the other two and hence can run more number of samples given the same time budget, which is essential when the run time budget is strictly tight. Specifically, the run time of the searching algorithm in Gamma and Mind Mappings is about 10x larger than Random-Pruned.

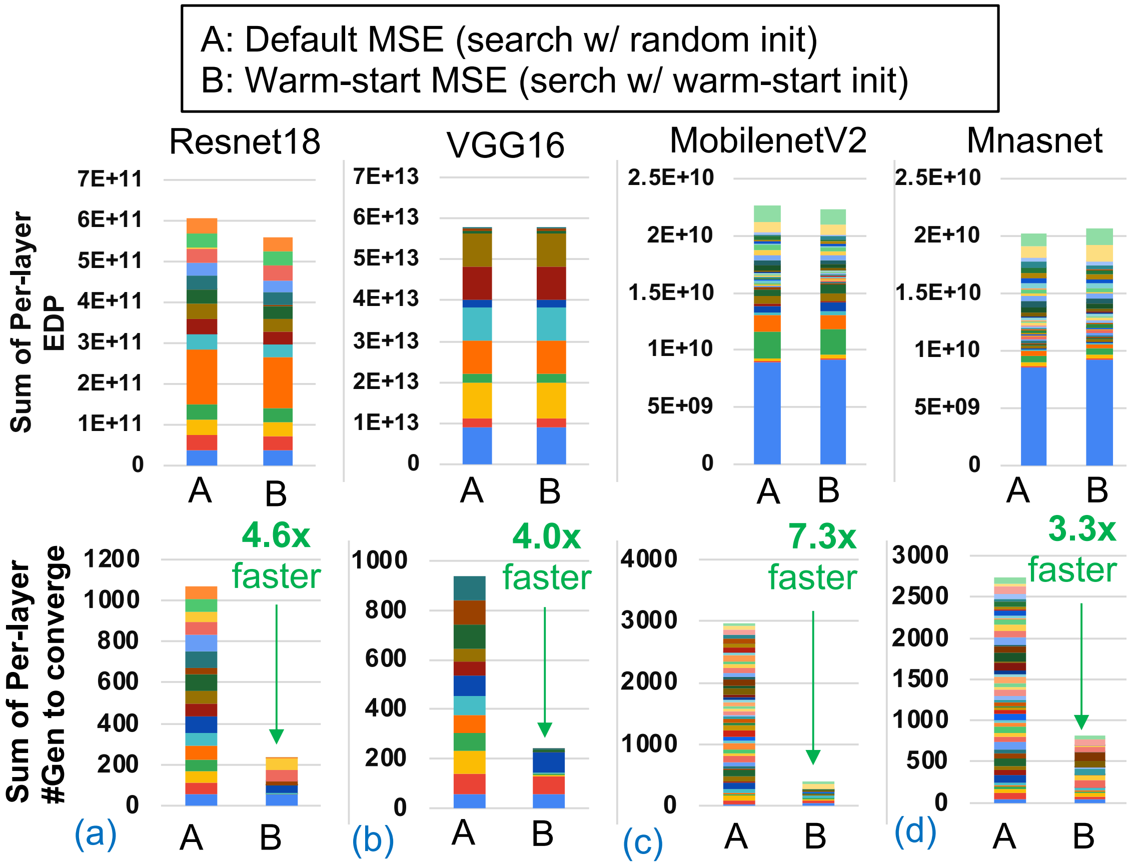


Fig. 5.11: The benefit of warm-start (by similarity) when executing MSE for the DNN models. Warm-start MSE achieves comparable EDP performance to default MSE, but converge 3.3-7.3x faster. Different colors represent different layers of the models.

Accelerator configuration not in the Training Dataset (Accel-B)

We use the same set of workloads as in Figure 5.3(a)(b), but change the accelerator configuration to Accel-B, which is not in the training dataset of the surrogate model of the gradient-based method. As shown in Figure 5.3(c)(d), the gradient-based method cannot perform as well as it did for the trained accelerator configuration, Accel-A. It demonstrates that the trained surrogate model does not generalize across accelerator configurations. Note that we can also re-train the surrogate model for the new accelerator configuration, which will recover the performance. However, it will require another full-fledged DNN training. Besides, we also need to collect 1 - 5 million of new training data to achieve quality results [6].

Variance of Accelerator Configurations. The random-based and feedback-based method take workloads and accelerator configurations as inputs and therefore are agnostic to variance in accelerator configurations. In contrast, the gradient-based method train its surrogate model based on a collected training dataset. The training dataset includes collected workloads and collected accelerator configurations. While surrogate model can generalize the workload encoding across different DNNs models [6], the generalization of accelerator configurations is more challenging since arbitrary buffer levels, buffer sizes, PE sizes, and other details (Figure 5.2) can be made. Thus the surrogate model is tied to one or few accelerator configurations.

Visualization of the Sampling Points

To better understand how different algorithms behave in the map space, we plot their sampling points in Figure 5.4 using the workload and accelerator configuration in Figure 5.3(a). Figure 5.4(a) shows the entire map space while dark red represent higher-performance points. There is a large low-performing region at the center while some small clusters of the high-performing points (green circle) scatter across the space. Figure 5.4(b) shows the points different algorithms actually sampled. Given the limited 5,000 sampling budget, the random-based method only samples around the lower-performing region because most of the design points sit here. The gradient-based method (Mind Mappings) starts with the lower-performing region and gradient-updates to the higher-performing regions at the right. However, it sits at the local optimum. The feedback-based method (Gamma) also starts with a lower-performing region but can explore a wider region faster because of its population-based method (population-based method is common in many feedback-based algorithms). Gamma reached one of the high-performance regions, as shown in Figure 5.4(b).

Takeaway:

- Learning-based methods, including gradient-based and feedback-based, can keep improving the quality of the sampling function over searching iterations, leading to better sampling efficiency.

- When the time constraint is strictly tight so that the learning-based methods cannot yet gather adequate data to improve their sampling function (i.e., still at exploration phase instead of exploitation), the random-based method is the most cost-effective choice.
- The surrogate model of the gradient-based method is trained on a collected training dataset, where the accelerator configuration is often fixed. The trained surrogate model cannot generalize across different accelerator configurations.

We pick Gamma, a feedback-based method, as our main mapper for the rest of the discussion in this chapter.

5.2.4 Understanding Mapper Search Operators

Recall that there are three mapping axes in the map space, tile, order, and parallelism. Gamma has dedicated genetic operators to explore along these axes, i.e., *mutate-tile*, *mutate-order*, and *mutate-parallelism*. It also houses a *crossover* operator to blend two high-performant mappings to create the next candidate mapping samples. Note that each genetic operator is specifically tuned to adapt to this map space as shown in the Gamma paper [5], which is the key source of sampling efficiency over other black-box optimizers, including RL and standard GA. As Figure 5.6 shows, full-fledged Gamma (dotted orange line) performs an order of magnitude better than standard GA across the three evaluated workloads.

Mapping Axis Sensitivity Analysis

In Figure 5.5, we explore each mapping axis individually (keeping the other two fixed) via the mutation operator in Gamma [5] such as mutate-tile for tile exploration, mutate-order for order exploration and so on. We find mutate-tile to have the highest impact on EDP compared to the other components.

Crossover Sensitivity Analysis

Gamma has crossover operator which blends two mapping points to create the next candidate mapping points. We execute a sensitivity analysis of crossover in Figure 5.6. We find that disabling crossover (light green) can hugely impact the potential performance compared to full-fledged Gamma (dotted orange). However, crossover-only without other operators (dark blue) is also not adequate. Crossover working with all the dedicated mutation operators for the three maxing axes (dotted orange) can maximize the sampling efficiency of the mapper (Gamma) and ends up giving the most optimized performance.

Takeaway:

- If one were to incrementally implement different exploration functions along the mapping axes, starting with the tile exploration could be the most cost-effective option.
- Blending two high-performance mappings (crossover) can effectively create another high-performance mapping.
- The ability to explore different order and parallelism dimensions choices is not as critical as tile size exploration to optimize EDP performance.
- Note that even when fixing the order or parallelism throughout the optimization process, at the initialization stage, we still randomly initialized order and parallelism for the initial populations (a groups of initial sampling points). It implies that few explorations of order and parallelism are often adequate to give competitive mapping. It is owing to the fact that many combinations of order or parallelism will lead to similar latency or energy performance, as we discuss later in §5.2.4.
- The performance difference of two mapping for the same problem can be as large as 3 orders of magnitude difference, consistent with prior works [5, 68, 4, 6].

Loop Order Sensitivity Analysis

We perform a sweep of loop order permutations to demonstrate our observation that *many order permutations lead to similar performance* as observed above. We use the found mapping in the experiment setting in Figure 5.6(a) and swap out the order permutation by enumerating through all the possibilities. The search space is as large as $(7!)^3$=1.28E+11. We add a constraint that each level of the buffer will use the same order to relax the complexity, which becomes 7!=5,040 choices. Figure 5.7 shows that there are only 16 different EDP values out of 5,040 different mappings. We can observe some patterns in each of the same performance mapping groups, as shown in Figure 5.7. For example, “XY..” means the permutation starting with XY. The loop order at the DRAM buffer level of the original mapping found by Gamma (XB..) also falls in the high-performance order group.

Takeaway. Many order permutations will lead to similar energy or latency performance. This is why various loop orders can be placed into large ”stationarity” buckets (such as weight/ input/ output/ row stationary [83, 68, 4] or inner/ outer product [165].

5.2.5 Understanding Sparse Accelerator Mappings

Need of MSE for Flexible Sparse Accelerator

There is a series of research proposing ways to prune DNN models [39, 40, 41, 42, 43, 44, 45]. However, the pruned models often cannot achieve as much performance gain in hardware as proven by the algorithmic analysis because of the increase complexity to find efficient mapping. There are several sparse accelerators [3, 169, 170, 171, 172, 173, 174, 175] for efficiently running sparse workloads, skipping zeros in the weights and/or activations. However, they often employ a fixed mapping (or a limited set of mappings). Given the nascent domain, MSE for flexible sparse accelerators is relatively unexplored, with one study looking into it [165] in contrast to several MSE studies for flexible dense accelerators [25, 160, 29, 28, 7, 9, 30, 31, 32, 33, 6, 5, 24]. This leaves MSE for sparse accelerators and workloads an area with plenty of opportunity to explore.

We use TimeloopV2, aka Sparseloop [165], as the cost model to explore the map space in a flexible sparse accelerator, and leverage Gamma as the mapper. Besides the three components mentioned before, Sparseloop also models hardware and software optimizations (e.g., power gating and compressed tensors) in sparse DNN accelerators, while preserving the same mapping representation for Gamma.

Mapping Search for Sparse Weight

For model pruning, we often focus on pruning out the weight of the models, essentially some weight becomes zero. Density 1.0 means dense weight, and density 0.5 means 50% of the weights are zero. In Table 5.2, we use workloads with different weight densities and use MSE to search for optimized mappings. The performance of found mappings are recorded in the blue cell. For example, the mapping found for Resnet CONV_3 with 0.5 density has EDP performance of 4.9E+9 (cycles uJ).

Do we need different mappings for different sparsity? We take the optimized mapping targeting a specific workload with a specific density (blue cell) and test it with the same workload with different densities. For e.g., at the top-left blue cell (Table 5.2), we have an optimized mapping for the dense workload (density 1.0). Then we use the same exact mapping and test its performance under 0.5, 0.1, 0.01 density, whose performance is recorded in the bottom cells. We perform the same experiment for the other three columns. We mark the best-performing cell across each row with green text. We can observe that the best-performing ones always located in the blue cell, meaning to optimize mapping for specific sparsity of the workload is needed to pursue the best performance.

Takeaway. A dense mapping cannot generalize across sparsity workloads. Different sparsity levels of the workload require different mappings to maximize the performance.

Sparse Inner and Outer Product

An observation that many sparse accelerators papers have made is that inner product accelerators often perform better for low sparsity workloads and outer product accelerators perform better at high amounts of sparsity [176, 172]. We study this general observation using the MSE framework. We assume the underlying sparse accelerator is flexible to support both inner and outer product style mapping. Inner and outer product styles are essentially affecting the loop order. Therefore, we fix the loop order and perform MSE for the other two axes (parallelism and tile sizes). Table 5.3 shows that the inner product style with optimized mapping consistently outperforms the outer product counterparts for workload density larger than 0.5, while the outer product style has an edge over the inner product style at densities smaller than 0.1.

Takeaway. From the viewpoint of MSE, we are able to validate the observation that inner product style mappings are better for denser workloads while outer product style works better at high sparsity.

5.2.6 Lessons Learnt

We summarize the key takeaways from our analysis:

- The feedback based mapper has the highest sampling efficiency and can directly work for any workload and accelerator configurations. However, it has the highest wall-clock time to acquire one sample.
- Tile is the most critical mapping axis to explore.
- MSE needs to consider sparsity.

5.3 Improving MSE

From our detailed analysis and takeaways from §5.2, we focus on two open-challenges for next-generation mappers.

Challenge I: Speed. As discussed in §5.2.3, while feedback-based mappers (such as Gamma [5]) offer the highest sample-efficiency, the time to run the search operators for each data point is around 20ms, 10x more costly than random-based mappers (such as Random-Pruned [4]). Moreover, since we find the optimized mapping in a layer-by-layer fashion (§5.2.3), the run time of MSE inevitably increases linearly with the layers of DNN models, increasing compilation times significantly. More importantly, emergence of techniques like neural architecture search is leading to new DNN models coming out frequently with highly irregular tensor shapes. This naturally increases the demand for efficient MSE. In this regard, the search speed of MSE is a critical concern.

Challenge II: Dynamic sparsity. Mapping need to be optimized for the specific sparsity level of the workloads (§5.2.5). While the sparsity of the weight is often fixed for a trained DNN models, the sparsity of activations is dynamic. When facing activation sparsity, we would either under-utilize the hardware because of inefficient mapping or would need to re-launch the MSE again and again for every input-activation.

We address these two challenges with two novel techniques that we believe can extend the state-of-the-art mappers.

5.3.1 Warm-start

Motivation

We introduce a **warm-start** technique to reduce the search time. It is inspired by two observations. (1) Informed by the study in §5.2.4 and §5.2.4, we know that order and parallelism are often less sensitive from workload to workload. (2) Because of the nature of the DNN operations (CONV, FC, and others), consecutive layers often have some dimensions the same or similar to each other. Therefore potentially the mapping of the later layers can be inspired by the found mapping of the previous layer.

Proposed Warm-start Search Mechanism

Figure 5.8 shows our warm-start flow. We introduce a *replay buffer* within the MSE framework which stores the optimized mapping of each workload (i.e., DNN layer) that has been run so far. We initialize the algorithm with the solution of the highest-similarity workload in the replay buffer.

MSE Flow. Warm-start works via the following flow. *Step-1:* When the new workload comes, we compare the workload *similarity* to the workloads in the replay buffer. We use *editing distance* as the similarity metric. *Step-2:* Initialize the algorithm with the mapping with the highest-similarity by (i) Inherit the order and parallelism parts of the solution, and (ii) Scale the tile sizes to match the tensor dimensions of the current workload. *Step-3:* Run the search algorithm.

Walk-Through Example. In Figure 5.8 as an example, there are two workloads that are finished with their final optimized mapping stored in the replay buffer. The next workload, workload-3, comes and will go through warm-start block before entering optimization loop. In the warm-start block, we use *editing distance* to compare the similarity between the current workload and the workloads in the replay buffer. E.g., workload-3 is only differ from workload-1 at C-dimension, leading to editing distance of 1; similarity, editing distance with workload-2 is 3 (K, Y, X). Therefore, we pick the stored optimized mapping for workload-1 (Map1), scale it to match the tensor shape of workload-3 (i.e., multiply C tile size by 2 at the outer-most tiling level (L3 mapping)), and use it as the initialized mapping for the optimization.

Similarity. Typically, for most DNNs we find that previous layer has the highest-similarity score. However, there are some exceptions: 1) the layers can come out-of-order because of other compiler decisions or 2) irregular tensor shapes of the workloads created by neural architecture search.

Evaluation

The Impact of Warm-start Initialization. Warm-start is an initialization technique. Figure 5.9 shows the effect of warm-start initialization. We measure the performance of the initialized mapping of warm-start by similarity (yellow bar), warm-start by previous layers (red bar), and the default random initialization (blue bar) in Figure 5.9. We evaluate workloads from two DNN models, VGG [146] and Mnasnet [34]. Many DNN models are made by human experts, where the shape of each layer are often designed with high regularity such as VGG [146] and Resnet [142]. In these models, warm-start by previous layers and warm-start by similarity make no difference, since the highest-similarity layers are almost always the previous layers, as shown in workload ID 1 - 4. However, the shape of the workloads in the Mnasnet, a network found by neural architecture search, are more irregular. Therefore warm-start by similarity becomes essential, providing 2x better performance than warm-start by previous layers. However, both warm-start strategies are effective and are 2.1x and 4.3x better than random initialization.

The Impact of Warm-start Search. We demonstrate how warm-start can reduce the time to converge. Figure 5.10 shows the converge curve of the first layer and a later layer to perform MSE on VGG16 [146]. For the first layers (VGG Conv_1), there are no previous solution in the replay buffer. Therefore searches with random initialization and warm-start initialization have no difference. However, for the later layers (VGG Conv_13), the searches with warm-start start with better points and converge faster.

We perform MSE for all layers in 4 DNN models with and without warm-start. Figure 5.11(a) shows that searching with warm-start does not affect the quality of the found solutions, i.e., the EDP values are as low as the default algorithm. Simultaneously, warm-start can converge **3.3x-7.3x** faster (we define time-to-converge as the time to reach 99.5% of performance improvement. In the figure we use the number of generation-to-converge, which is an equivalent index of time-to-converge.). We can observe that Mnasnet [34] yields the least speedup. It is because Mnasnet is a network found by neural architecture search whose tensor shape of each layer/ workload is more irregular. Therefore scaling from old solutions will be not as close to the optimized solutions as the ones in regular networks such as Resnet [142], VGG [146], Mobilenet [126], which are manual designed. However, even that, the warm-start for Mnasnet can still converge **3.3x** faster.

Takeaway:

- Mappings have hierarchies (§5.2.2). Each level of hierarchy is built with the micro building block formed by its down-stream hierarchy. Warm-start is essentially reusing those already found high-performance micro building blocks to construct new mappings for new workloads.
- Warm-start by similarity is beneficial for both regular (designed by human experts) and irregular (designed by ML algorithms) DNN models.
- Warm-start technique can reduce time-to-converge for MSE, which is essential for both time-critical tasks and large-scaled MSE.
- The replay buffer we use is per accelerator configuration and per DNN model, i.e., we clear up the replay buffer when evaluating new DNN model for the sake of demonstrating the learning phase of the replay buffer. However, in practice, we do not need to clear up the replay buffer. The replay buffer will become a cache of high performance mappings across models and can bootstrap a wide range of workloads.

5.3.2 Sparsity-aware MSE

Motivation

In §5.2.5 we identified the need different mappings for different sparsity of workloads. While tackling weight sparsity is straightforward because weight sparsity is often fixed at model deploy time, tackling activation sparsity is challenging. It is not practical to search for an optimal mapping for each new input-activation. We want to seek out *if we can learn a mapping that can generalize across a range of sparsity levels to tackle the dynamic sparsity in activations?*

Proposed Sparsity-aware Search Mechanism

We propose sparsity-aware mapping search, which works as follows. When executing MSE, we don't look at the actual density level of each activation (since it is dynamic). Instead, we assume and impose sparsity in the workload when executing MSE. We impose the activation to have a density from 1.0 to 0.1, which is the typical range of activation density in DNN [3, 169, 170, 171, 172, 173]. Next, when executing MSE, we score the mapping by the performance of this mapping on workload across the sweep of density levels (Figure 5.8).

Scoring a Mapping. We score a mapping by the weighted sum of the performance. We use a heuristic that "the hardware performance (e.g., latency, energy) is with positive correlation to the density of the workload" to decide the weighting. We pick the weighting by the factorial of *density*[4]. For example, assuming we have two density levels, 0.5 and 1.0, with hardware performance $Perf_{0.5}$ and $Perf_{1.0}$, then the (weighted sum) score is: $\frac{Perf_{0.5}}{0.5} + \frac{Perf_{1.0}}{1.0}$.

Evaluation

We compare the "sparsity-aware" (§5.3.2) with "static-density" in Table 5.4. Both "sparsity-aware"and "static-density" are agnostic to the actual workload density. "Static-density 1.0" always assumes the workload is dense when searching. "Static-density 0.5" searches the mapping assuming the workload has 0.5 density, and "Static-density 0.1" assumes 0.1 density. "Sparsity-aware" searches the mapping assuming the workload density range from 1.0 - 0.1. Specifically, we use 5 density levels: 1.0, 0.8, 0.5, 0.2, and 0.1 (blue cells in the first column), which are picked by heuristics. That is, when evaluating the mapping in the optimization loop, we scored the mapping by the performance of this mapping under workload density levels of 1.0, 0.8, 0.5, 0.2, and 0.1, and used the weighted sum of the performance as the final scores for the mapping. The scores are used to select which

[4] We pick the weighting linear to *density*, since we experiment on activation sparsity in our evaluation. When we experiment on weight and activation sparsity together, following the same methodology, we will use $(density)^2$.

mappings proceed to the next iteration of the optimization loop.

We test the found mappings of the four strategies (columns) in Table 5.4 by workload with density from 1.0 to 0.05. The performance of each is recorded in the corresponding rows. We make two observations: 1) The “sparsity-aware” can reach comparable performance to the “static-density” ones at the density levels, for which the “static-densities” are specifically optimized. For example, “static-density 1.0” found a mapping with EDP 2.39E+13 (cycles uJ) at density level 1.0. The mapping found by “sparsity-aware” can perform at a comparable EDP of 2.40E+13 (cycles uJ). 2) Aware of a range of sparsity (1.0 - 0.1), “sparsity-aware” can successfully find a mapping that can generalize across a range of sparsity. A fixed mapping found by “sparsity-aware” can achieve (in geomean) **99.7%** of performance to the performance of each of the mappings specifically searched for different density levels.

Takeaway:

- Our “Sparsity-aware” technique is essentially a *regularization* technique in MSE. It prevents the MSE from finding mappings overfitted for one sparsity level.
- Sparsity-aware technique enables MSE to find a fixed mapping that is generally suitable, not claiming to be optimal but is comparable, for a range of sparsity levels. It tackles the practical cases of dynamic activation sparsity in DNN networks after Relu activations or other sparsifying operations.
- We use a density range of 1.0 - 0.1 because we focus on DNNs whose activation sparsity is usually in this range [3, 169, 170, 171, 172, 173]. However, tensor operations in high-performance computing, which has different range of sparsity, can apply different density ranges.

5.4 Related Works

Map Space Exploration. Many mappers (search algorithms) with different algorithmic techniques are proposed to tackle the MSE problem. Timeloop-mapper [4], Simba [8], dmazeRunner [9], Interstellar [10], and others [11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23] use random sampling on a raw or pruned search space. Gamma [5], Autotvm [24], and others [31, 16, 168] use genetic algorithms. HASCO [25] and Reagen et. al [26] uses Bayesian optimization, RELEASE [27], ConfuciuX [160], and FlexTensor [28] uses reinforcement learning. Mind Mappings [6] uses a neural network-based surrogate model to replace the cost model and directly uses backpropagation to learn a solution that maximizes the objective. There are also other techniques such as mixed-integer programming in CoSA [7], MCMC search in FlexFlow [29], and others [30, 31, 32, 33]. There have been plenty of mappers proposed. However, a deeper analysis of how the MSE works and how different mapping axes contribute to the performance is often lacking. This is the focus of this work.

5.5 Summary

Specifically, our contributions are two-fold. (1) This is the first work, to the best of our knowledge, to *quantitatively* compare three wide categories of mappers: random-based [4] (i.e., heuristic pruning), feedback-based [5] (i.e., blackbox optimization and reinforcement learning), and gradient-based [6] (i.e., surrogate models), and analyze their trade-offs. We conduct a sensitivity analysis of different mapping axes to understand the contribution of each axis. We then perform case studies that reveal distinguishing characteristics of good and bad mappings. Our analysis reveals that: (i) random search is inefficient, (ii) gradient-based search converges fast but requires prior knowledge of the accelerator architecture, and (ii) feedback-based search is more adaptable and sample-efficient, but requires higher cost to acquire each sample. Our analysis also shows that optimality of a dense DNN mapping does not port over to a sparse DNN.

(2) Based on our findings, we propose two novel techniques to enhance the state-of-the-art in MSE. (i) We propose a "warm-start" technique to initialize the MSE with previous mapping solutions from previous layers in the replay buffer based on a *similarity* metric, enabling the mapper to start at a better point and converge faster. In our evaluations, we find

that warm-start can help the mapper converge to a similar performance point 3.3x-7.3x faster. (ii) We propose a “sparsity-aware” technique to search for a mapping that can perform well across a range of target activation sparsity. A fixed mapping found by our “sparsity-aware” approach can achieve 99.7% of the performance of each of the mappings specifically tailored for the various density levels. We believe these techniques can be augmented over existing MSE tools, making them more robust and scalable for future DNNs.

CHAPTER 6
DIGAMMA: A HW-MAPPING CO-EXPLORATION MAPPER

So far we have a thorough discussion on Map Space Exploration (MSE), where we had formally define the map space (chapter 3), proposed an efficient mapper (chapter 4), and analyze different mappers and components in MSE framework (chapter 5). The next topic we want to explore is how we can extend the MSE to support larger design space. As mention in chapter 2, the design space of DNN accelerator includes: i) HW resources and ii) mapping. So far, we focus on the discussion of optimizing mapping. In this chapter, we will discuss how to extend the mapping exploration to HW-Mapping co-exploration.

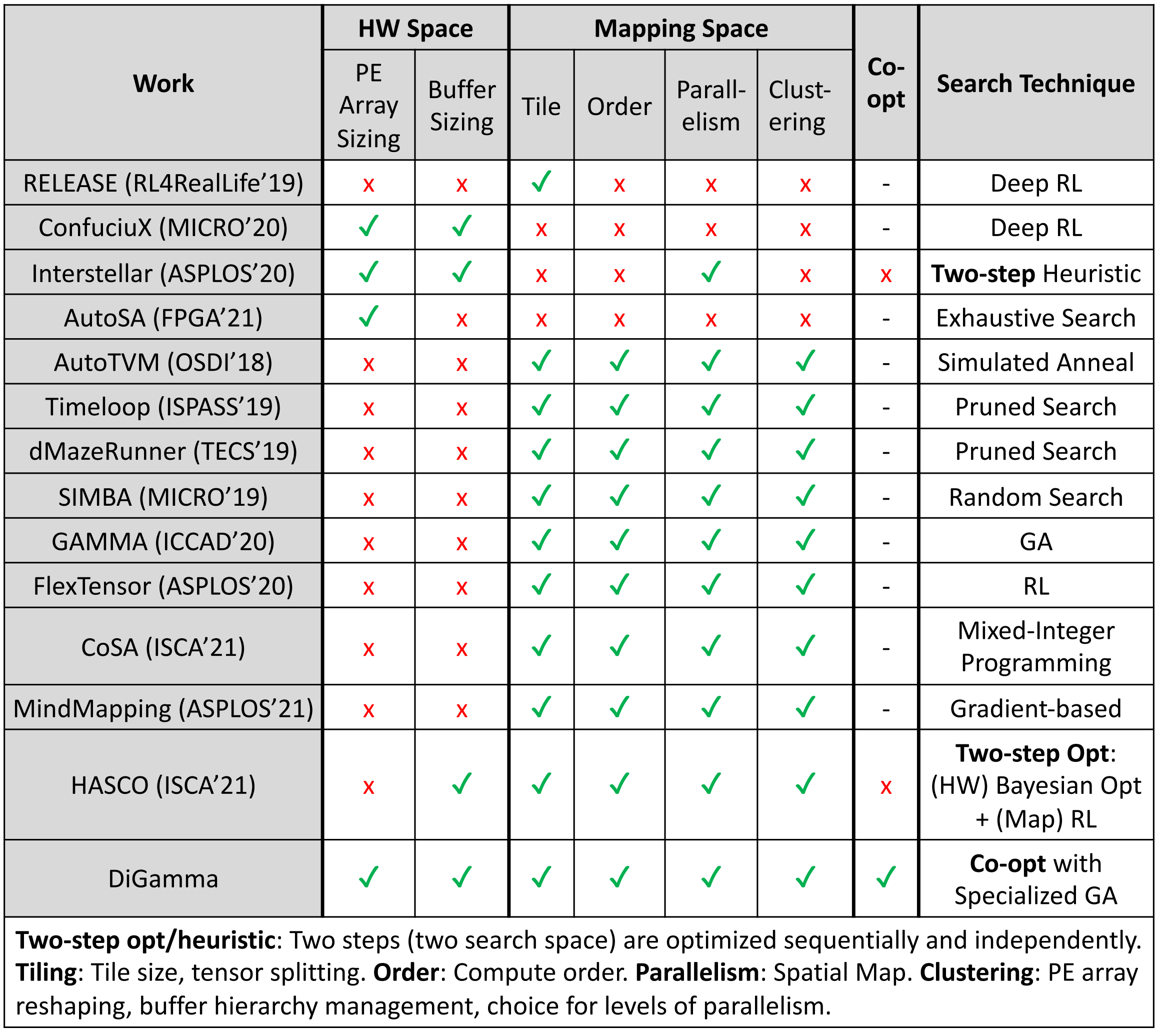
With the growth of AutoML, optimizing HW and Mapping together automatically with AI/ML offers a potential solution. However, how to effectively co-optimize the HW and mapping is still an open question. To address this, we propose a HW-Mapping co-optimization framework (Co-opt Framework) and an optimization algorithm, which search the HW resources configuration and the optimized mapping simultaneously.

6.1 Technical Approach

Problem Formulation. *Under a design budget (e.g., chip area) and given a DNN model, we aim to design an accelerator with optimized HW resource configuration for PEs, local buffer (L1), and global buffer (L2), and an optimized mapping strategy.*

6.1.1 High-level Overview

We integrate the two searching loops, HW and mapping, into one unified optimization process, and propose 1) a HW-Mapping co-optimization framework (Co-opt Framework). Note that there are two main factors deciding the effectiveness of an optimization framework: efficiency of the design-point encoding and efficiency of the optimization/ search algorithm.



Work	HW Space		Mapping Space				Co-opt	Search Technique
	PE Array Sizing	Buffer Sizing	Tile	Order	Parall-elism	Clust-ering		
RELEASE (RL4RealLife'19)	x	x	✓	x	x	x	-	Deep RL
ConfuciuX (MICRO'20)	✓	✓	x	x	x	x	-	Deep RL
Interstellar (ASPLOS'20)	✓	✓	x	x	✓	x	x	**Two-step** Heuristic
AutoSA (FPGA'21)	✓	x	x	x	x	x	-	Exhaustive Search
AutoTVM (OSDI'18)	x	x	✓	✓	✓	✓	-	Simulated Anneal
Timeloop (ISPASS'19)	x	x	✓	✓	✓	✓	-	Pruned Search
dMazeRunner (TECS'19)	x	x	✓	✓	✓	✓	-	Pruned Search
SIMBA (MICRO'19)	x	x	✓	✓	✓	✓	-	Random Search
GAMMA (ICCAD'20)	x	x	✓	✓	✓	✓	-	GA
FlexTensor (ASPLOS'20)	x	x	✓	✓	✓	✓	-	RL
CoSA (ISCA'21)	x	x	✓	✓	✓	✓	-	Mixed-Integer Programming
MindMapping (ASPLOS'21)	x	x	✓	✓	✓	✓	-	Gradient-based
HASCO (ISCA'21)	x	✓	✓	✓	✓	✓	x	**Two-step Opt**: (HW) Bayesian Opt + (Map) RL
DiGamma	✓	✓	✓	✓	✓	✓	✓	**Co-opt** with Specialized GA

Two-step opt/heuristic: Two steps (two search space) are optimized sequentially and independently. **Tiling**: Tile size, tensor splitting. **Order**: Compute order. **Parallelism**: Spatial Map. **Clustering**: PE array reshaping, buffer hierarchy management, choice for levels of parallelism.

Fig. 6.1: State-of-the-Art HW and Mapping optimization frameworks.

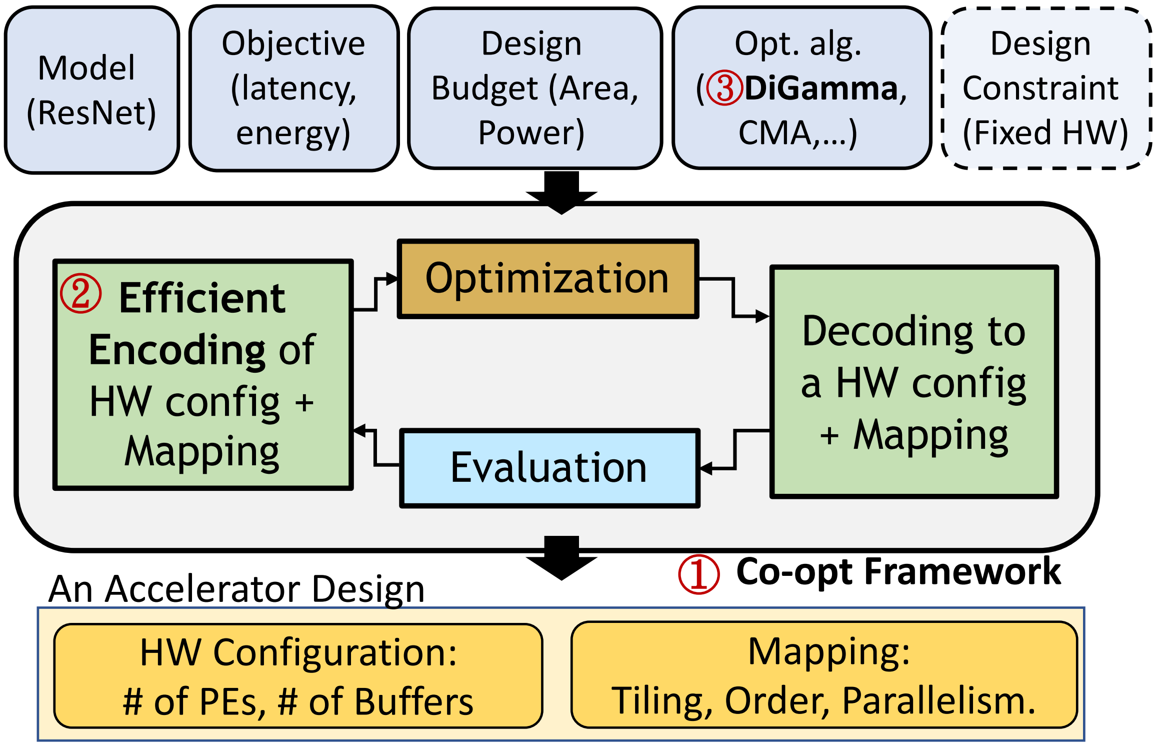
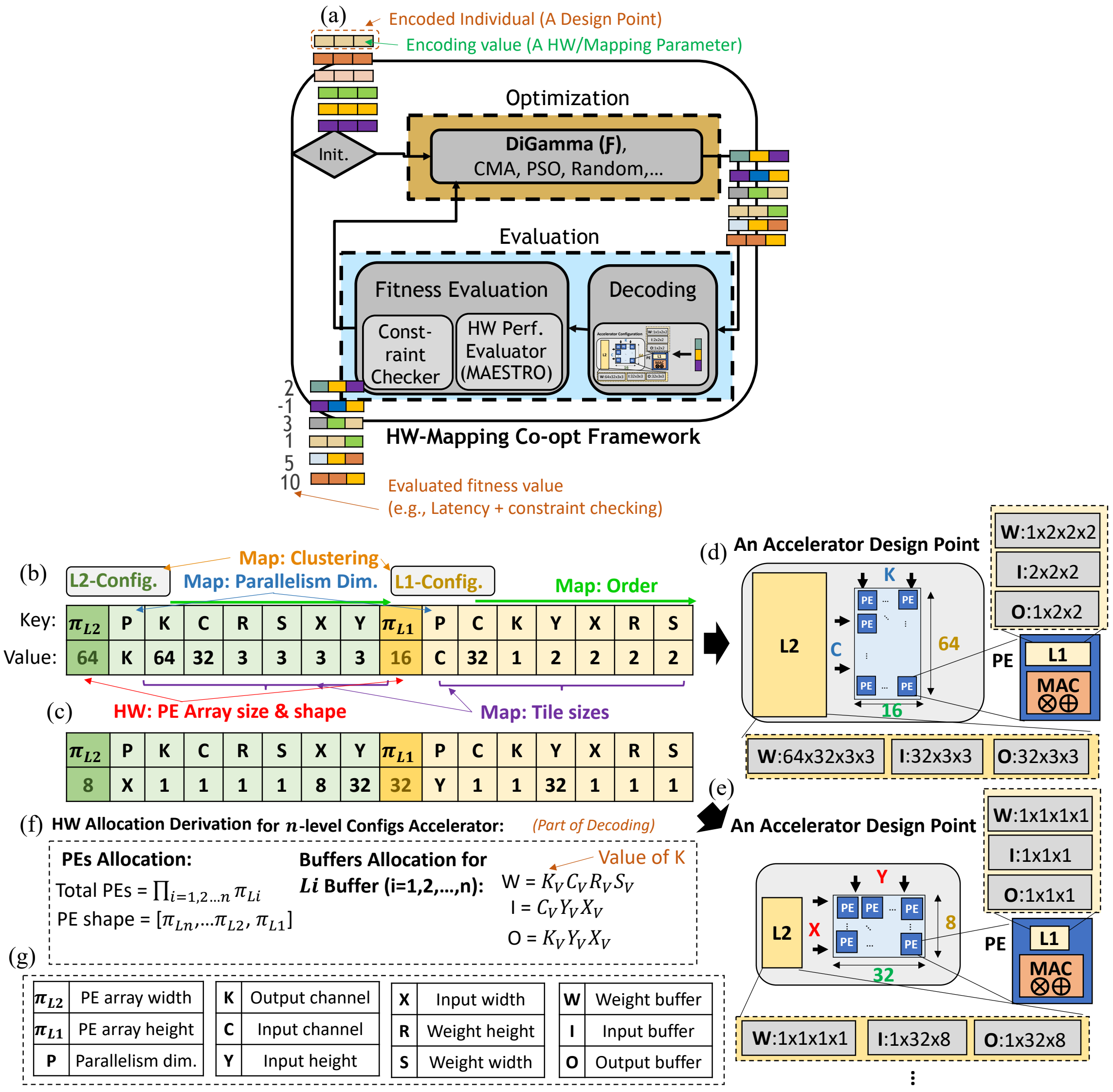

Model (ResNet)
Objective (latency, energy)
Design Budget (Area, Power)
Opt. alg. (③DiGamma, CMA,...)
Design Constraint (Fixed HW)
② Efficient Encoding of HW config + Mapping
Optimization
Decoding to a HW config + Mapping
Evaluation
① Co-opt Framework
An Accelerator Design
HW Configuration: # of PEs, # of Buffers
Mapping: Tiling, Order, Parallelism.

Fig. 6.2: HW/Mapping Co-optimization Framework. Our three technical contributions are highlighted.

In this chapter, we propose 2) an efficient encoding method for HW configuration and mapping and 3) a sample-efficient optimization algorithm (DiGamma).

6.1.2 HW-Mapping Co-optimization Framework

Co-opt Framework takes the input of target model, optimization objective, design budget, an optimization algorithm, and (optionally) a design constraint, and generates an optimized accelerator design point with HW configuration and mapping strategy. The design constraint is optional, for supporting two additional use-cases: 1) Fixed-HW: when the researcher/ engineer already has a designed accelerator and only wants to search for an optimal mapping; 2) Fixed-Mapping: when the researcher has a manual-tuned mapping (e.g., NVDLA [80]) and wants to understand the optimal HW configurations for designing a specialized accelerator (e.g., understanding the compute to memory balance). Co-opt Framework can deal with these constraints by restricting the design space accordingly. As shown in Figure 6.3(a), Co-opt Framework includes an optimization block, where different optimization algorithms



(b)

Key:	π_{L2}	P	K	C	R	S	X	Y	π_{L1}	P	C	K	Y	X	R	S
Value:	64	K	64	32	3	3	3	3	16	C	32	1	2	2	2	2

(c)

π_{L2}	P	K	C	R	S	X	Y	π_{L1}	P	C	K	Y	X	R	S
8	X	1	1	1	1	8	32	32	Y	1	1	32	1	1	1

(f) **HW Allocation Derivation for n-level Configs Accelerator:** *(Part of Decoding)*

PEs Allocation:

Total PEs = $\prod_{i=1,2\ldots n} \pi_{Li}$

PE shape = $[\pi_{Ln},\ldots\pi_{L2}, \pi_{L1}]$

Buffers Allocation for *Li* Buffer (i=1,2,...,n):

Value of K

W = $K_V C_V R_V S_V$

I = $C_V Y_V X_V$

O = $K_V Y_V X_V$

(g)

π_{L2}	PE array width	K	Output channel	X	Input width	W	Weight buffer
π_{L1}	PE array height	C	Input channel	R	Weight height	I	Input buffer
P	Parallelism dim.	Y	Input height	S	Weight width	O	Output buffer

Fig. 6.3: (a) Co-opt Framework, (b-c) the HW-Mapping encoding representation, and (d-e) the corresponding decoded accelerator configuration. (f) The formula for calculating minimum on-chip buffer requirement. (g) The definition of notations.

can be plugged and played, and an evaluation block, where the proposed design points are evaluated and scored to guide the algorithms.

Optimization Block.

One main goal of this chapter is to develop a generic framework for HW-Mapping co-optimization that is not tiled with any optimization/searching algorithms. We abstracted the underlying detail of taking different DNN models as inputs, understanding different design budgets, encoding/decoding of HW and mapping, and so on, and expose a generic interface for all the optimization algorithms. The only task left for any optimization algorithm (from any optimization library [157] or custom-designed) is to find a list of parameters (together forming a design point) that yields the highest reward (fitness). The sampling budget, which is the number of design points the algorithms are allowed to sample, is a hyper-parameter of optimization block that can be set by the users. It controls the number of optimization loops (optimization, evaluation, optimization,...) in the framework.

Evaluation Block.

The evaluation block includes a decoding module (described in §6.1.3) and a fitness evaluation module. The fitness evaluation includes a HW performance evaluator and a constraint checker (Figure 6.3(a)).

HW Performance Evaluator. We evaluate all design points using an open-source HW performance evaluator, MAESTRO [68], which has detailed micro-architectural HW models and is validated against chip prototypes. MAESTRO takes in the accelerator design (HW and mapping) and outputs a detailed HW performance report including latency, area, power, energy, and so on. In the evaluation, we use HW resource area as the constrained design budget which includes the area of PEs, L1 and L2 buffers[1]. **Constraint Checker.** In constraint checker, if the required resources (e.g., area or power) of the proposed design

[1] A complete chip area will also include NoCs, logics, routing, clock trees, pin placement, and others, which need other physical layout considerations and are not considered in MAESTRO and this chapter.

point is larger than the provided budget, we invalidate the design point by re-assigning it a negative fitness value.

6.1.3 Encoding of Design Point

Encoding: Design Space Description

One of our key contributions is a customized encoding, as shown in Figure 6.3(b-c) (notation shown in Figure 6.3(g)), to describe an accelerator design-point. Our encoding captures the compute resources, mapping, and the memory hierarchy. We show a 2-hierarchy level accelerator in Figure 6.3(b-c) as an example. Each key-value pair represents a *gene*. L1-config (yellow) shows the accelerator configuration (HW and mapping) of a 1-D PE array. π_{L1}, a HW parameter, shows the length of the 1-D PE array. The rest of the genes (P,C,K,Y,X,R, and S) describe the mapping parameters for the 1-D PE array, including tiling, order, and parallelism. The value genes describe the tile sizes of the corresponding key dimension. The order of key genes describes the compute order. P gene tells the dimension to parallelize the compute across 1-D PE array. L2-config shows the HW and mapping across several 1-D PEs arrays, effectively describing a 2-D PE array. π_{L2}, a HW parameter, shows the number of instantiated 1-D PE arrays, while the rest mapping genes decide the mapping parameters across 1-D PE arrays. Similarly, a 3-level hierarchy (i.e., several 2D arrays) can also be described.

Decoding: Design Point Derivation

When evaluating the fitness/performance of each individual's genes, we decode them back to an exact accelerator design point. Figure 6.3(d)(e) shows the decoded accelerator of the proposed design point by Figure 6.3(b)(c), respectively. The π_{L2} and π_{L1} genes decide the PE array sizes and aspect ratio. Therefore different PE arrays are configured in Figure 6.3(d)(e). The P values of L2 and L1 implies how the compute are fetched and parallelized to the PE arrays, e.g., K-C parallelism and X-Y parallelism in Figure 6.3(d)(e). The order of the genes decides the computation order for each tile in the PE arrays. Finally, tile sizes and levels of hierarchies (or called clustering) (e.g., two levels of cluster/ hierarchy in Figure 6.3(b)(c)) determine the *minimum buffer requirement* to house weight, input, and output tensor at both L2 and L1 buffers.

6.2 Optimization Algorithm

6.2.1 Leveraging Existing Algorithms

With the thriving of AutoML, many optimization algorithms have been developed for automatically searching through a given design-space. They achieve state-of-the-art performance across many domains, including neural architecture search, AI-controlled game, chip design [177], and so on. Co-opt Framework provides a generic interface enabling us to plug and play many of these existing algorithms (which we leverage from nevergrad [157]), as shown in the experiments in §6.3.2).

However, the design space of HW-Mapping co-optimization is un-smoothed and extreme large. This challenges the search efficiency of the optimization algorithms and makes some of them ineffective under limited sampling budgets (§6.3.2). This motivates our proposed algorithm, which customizes a genetic algorithm.

6.2.2 Background of Genetic Algorithm.

In genetic algorithm (GA), we often call an encoded value of a candidate a gene, each encoded candidate: an *individual*, a bag of candidates: a *population*, and one iteration of optimization loop: a *generation*. Baseline GA has two standard genetic operators: crossover (blend the genes of individuals and reproduce populations for the next generation) and mutation (perturb the genes of each individual).

Research shows GA reaches competitive performance with deep reinforcement learning [155, 156], and hyper-parameter optimization problem. Comparing to many optimizations methods, GA is light, fast, and highly parallelizable [155, 156]. However, the key challenge is its sample efficiency.

GAMMA [5] is an open-source genetic algorithm, tuned for DNN mapping optimization over given HW configurations of accelerators. Despite the effectiveness of GAMMA as a mapping search tool, using it naively to search for HW resources and mapping together will result in a two-loop optimization, which is extremely inefficien.

6.2.3 DiGamma: Domain-aware Genetic Algorithm

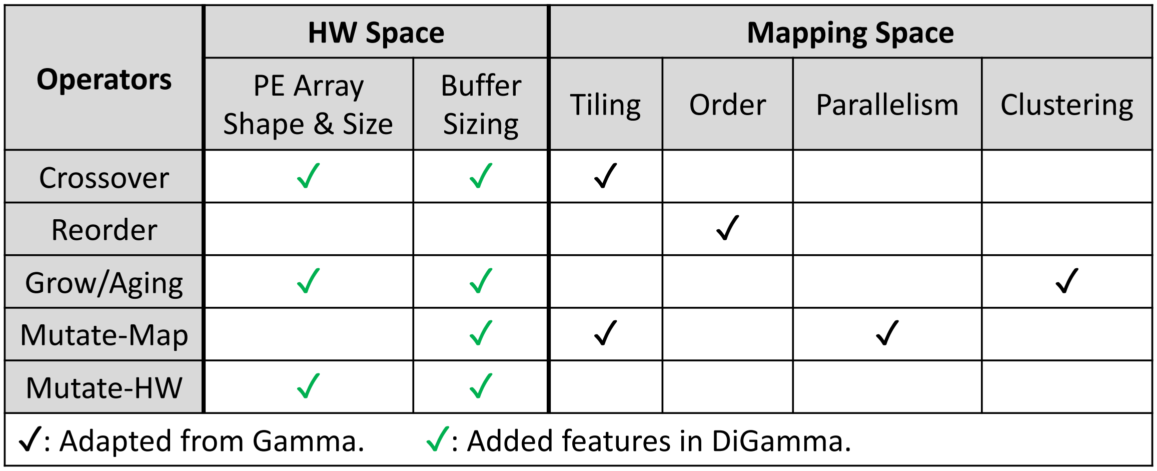
DiGamma uses the encoding presented in §7.1.1 to describe design-points. It then perturbs these to search through the co-optimization space. Rather than using conventional genetic operators (crossover, mutations) to perturb the genes arbitrarily, which is shown to have poor sample efficiency (§6.2.2), we develop *specialized genetic operators* (i.e., optimization operators) for individual HW and mapping genes to capture the structure of the design space, as described next.

The genetic operators responsible for mapping (tiling, order, parallelism, clustering) are modified from GAMMA [5]. Additionally, we implement a *HW* genetic operator to perturb the PE configuration, where the values of π_{L2} and π_{L1} decide the total number of PEs and the aspect ratio of PE array. Further, for L1 and L2 buffer sizes, we employ a buffer allocation strategy to decide the most optimized buffer allocations of a given individual (Figure 6.3(b)(c)) by the *minimum buffer requirement* derived at decoding block (§6.1.3), i.e., we allocate the exact amount of buffer needed at both L2 and L1 to maximize buffer utilization. We summarize the developed specialized genetic operators and their different perturbing ability across HW and Mapping space in Figure 6.4.

6.3 Evaluations

6.3.1 Setup

Across our experiments, we use the optimizing objective of minimum latency which becomes the performance metric when evaluating the quality of the found solution. Other objectives



Operators	HW Space		Mapping Space			
	PE Array Shape & Size	Buffer Sizing	Tiling	Order	Parallelism	Clustering
Crossover	✓	✓	✓			
Reorder				✓		
Grow/Aging	✓	✓				✓
Mutate-Map		✓	✓		✓	
Mutate-HW	✓	✓				
✓: Adapted from Gamma. ✓: Added features in DiGamma.						

Fig. 6.4: GAMMA's genetic operators† and their perturbing space‡.

†: **Mutate-HW**: Mutation operating on HW space, which tweaks the PE size/shape and also affects the allocated buffer. **Mutate-Map**: Mutation operating on mapping space and co-affecting buffer choices in HW space.

‡: Definition of each space can be found in Figure 6.5.

can also be specified to DiGamma such as power, energy, EDP.

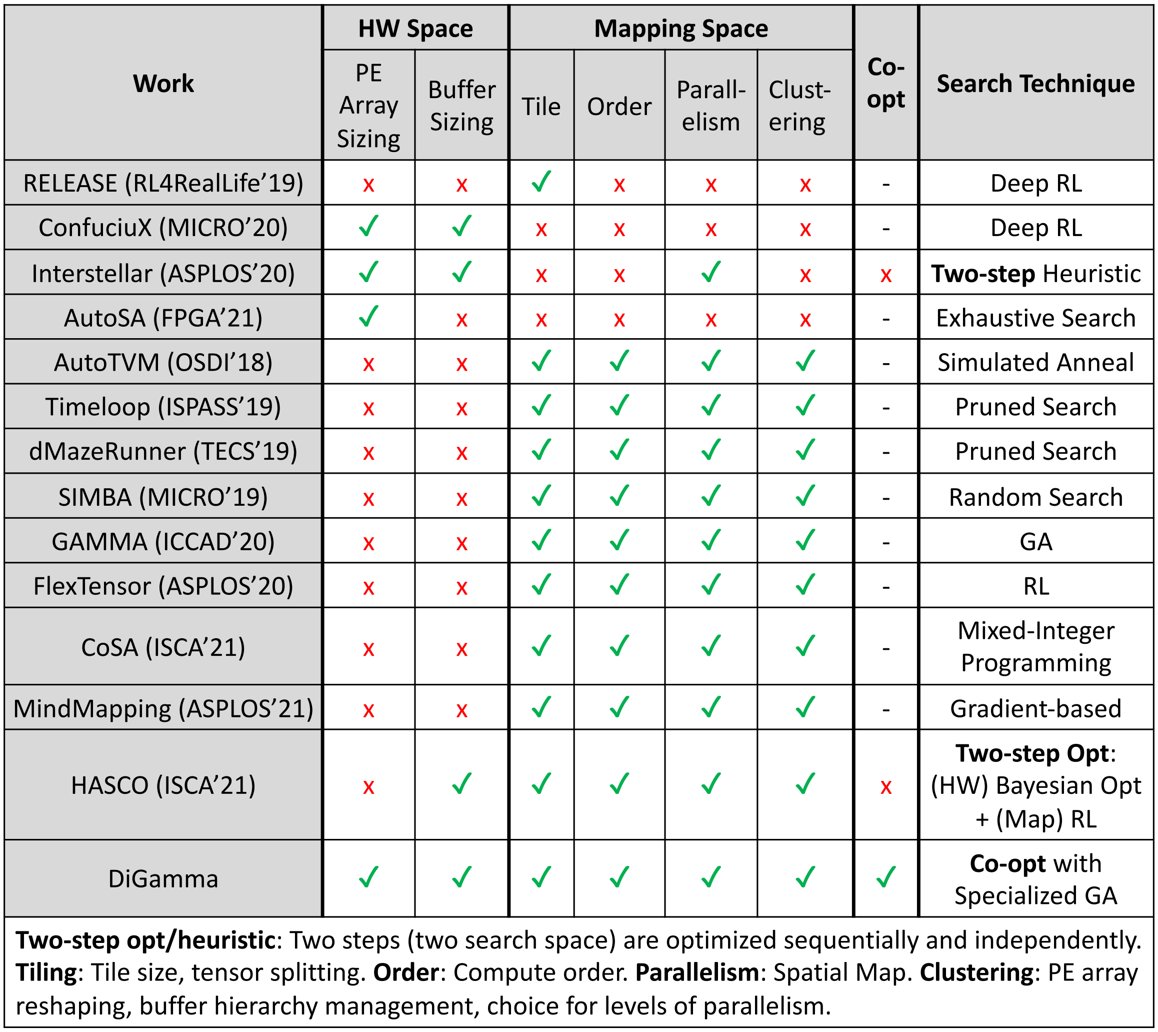
DNN Models. We experiment 7 DNN models across 3 popular DNN applications: vision (MobilenetV2, Resnet18, Resnet50, Mnasnet), language (BERT), and recommendation (DLRM, NCF).

Edge/ Cloud Platform Resources. We evaluate accelerator design under two types of platform resources: edge and cloud. We set the chip area budget for area of PEs and on-chip buffers as $0.2mm^2$ *for accelerators in edge [8, 178]* and $7.0mm^2$ *for accelerators in cloud [8].*

Area Cost Model. To estimate area cost, we implemented RTL of the various components in Figure 6.3(d-e), synthesized them using Synopsys DC with Nangate 15nm library and used Cadence Innovus for place-and-route. We synthesized the SRAM buffers with SAED32 education library from Synopsys.

Sampling Budget of Optimization Algorithms. We set the sampling budget (maximum number of sampled points throughout the search process) as 40K points for all the investigated optimization algorithms. For DiGamma, it means population size times number of generations cannot exceed 40K, which takes about 20 mins of CPU-time.

Baseline Optimization Algorithms. We take 8 other optimization algorithms, which are



Work	HW Space		Mapping Space				Co-opt	Search Technique
	PE Array Sizing	Buffer Sizing	Tile	Order	Parallelism	Clustering		
RELEASE (RL4RealLife'19)	x	x	✓	x	x	x	-	Deep RL
ConfuciuX (MICRO'20)	✓	✓	x	x	x	x	-	Deep RL
Interstellar (ASPLOS'20)	✓	✓	x	x	✓	x	x	**Two-step** Heuristic
AutoSA (FPGA'21)	✓	x	x	x	x	x	-	Exhaustive Search
AutoTVM (OSDI'18)	x	x	✓	✓	✓	✓	-	Simulated Anneal
Timeloop (ISPASS'19)	x	x	✓	✓	✓	✓	-	Pruned Search
dMazeRunner (TECS'19)	x	x	✓	✓	✓	✓	-	Pruned Search
SIMBA (MICRO'19)	x	x	✓	✓	✓	✓	-	Random Search
GAMMA (ICCAD'20)	x	x	✓	✓	✓	✓	-	GA
FlexTensor (ASPLOS'20)	x	x	✓	✓	✓	✓	-	RL
CoSA (ISCA'21)	x	x	✓	✓	✓	✓	-	Mixed-Integer Programming
MindMapping (ASPLOS'21)	x	x	✓	✓	✓	✓	-	Gradient-based
HASCO (ISCA'21)	x	✓	✓	✓	✓	✓	x	**Two-step Opt**: (HW) Bayesian Opt + (Map) RL
DiGamma	✓	✓	✓	✓	✓	✓	✓	**Co-opt** with Specialized GA

Two-step opt/heuristic: Two steps (two search space) are optimized sequentially and independently. **Tiling**: Tile size, tensor splitting. **Order**: Compute order. **Parallelism**: Spatial Map. **Clustering**: PE array reshaping, buffer hierarchy management, choice for levels of parallelism.

Fig. 6.5: State-of-the-Art HW and Mapping optimization frameworks.

widely-used and achieving state-of-the-art performance across different tasks, as baselines. The algorithms include: Random search, standard GA (stdGA), Particle Swarm Optimization (PSO), Test-based Population-Size Adaptation (TBPSA), (1 + 1)-evolution strategy ((1+1)-ES), Differential Evolution (DE), Passive Portfolio (Portfolio), and Covariance matrix adaptation evolution strategy (CMA).

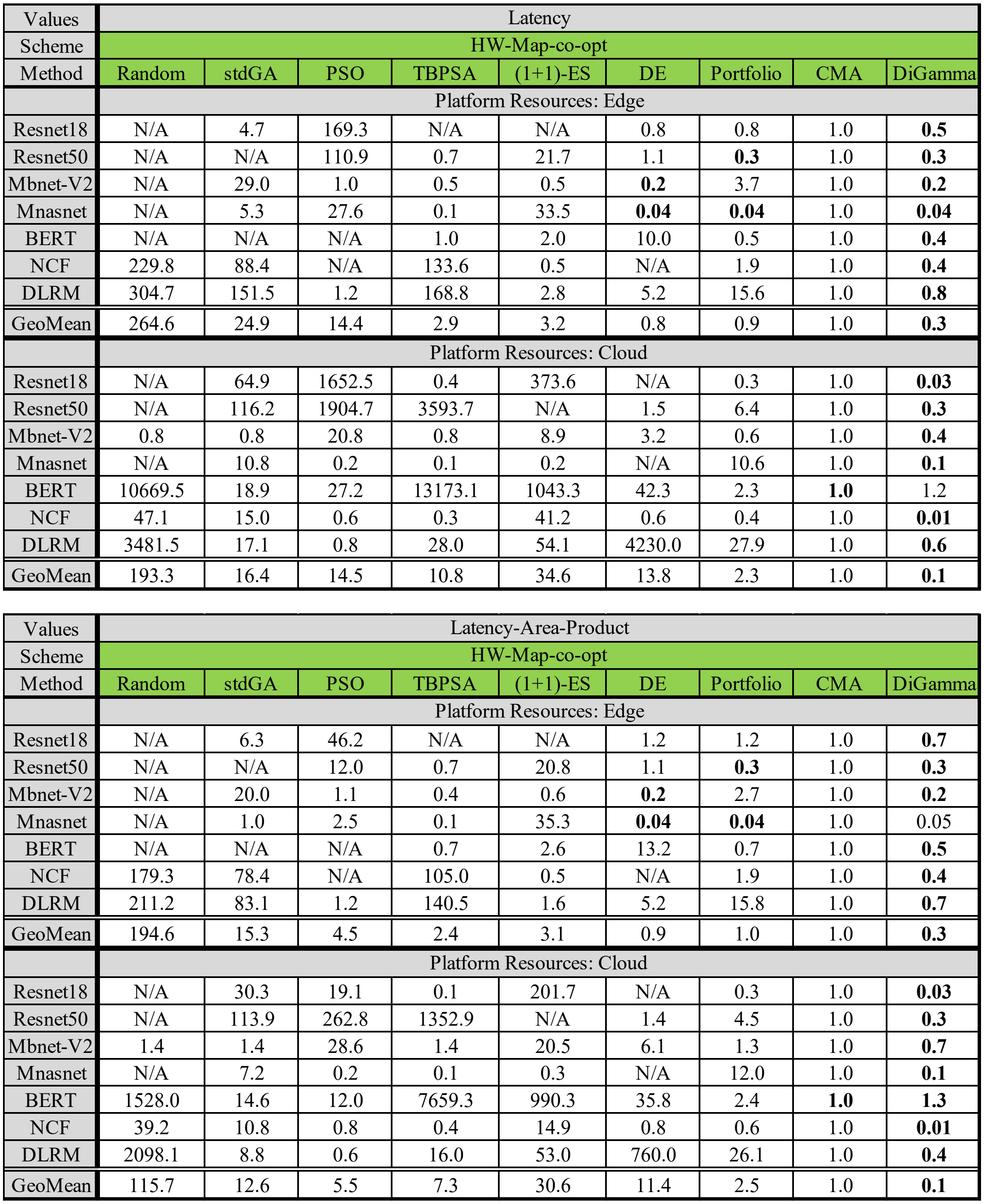
Baseline HW and Mapping Optimization Schemes. We formulate two kinds of HW and mapping optimization schemes and compare them with DiGamma, listed as follows.

- **HW-opt**: optimizing HW while mapping is fixed. The HW is optimized by grid search approach over number of PEs and buffer sizes. Note that the entire HW configuration design space is as large as $O(10^{12})$, which is hard to enumerate through, and therefore we use grid search. For mapping, we use the manual-designed NVDLA (dla)-like [80], ShiDianNao (shi)-like [81], and Eyeriss (eye)-like [178].
- **Mapping-opt**: optimizing mapping while HW is fixed. The mapping is optimized by GAMMA [5], a mapping optimizer for a given HW configuration. We cherry-picked three sets of HW configurations: Buffer-focused (small compute + large buffer), compute-focused (large compute + small buffer), and Medium-Buf-Com (medium Buffer + medium compute) configuration for both edge and cloud settings. Note that the design is area constrained, therefore compute and buffer resources are traded-off with each other.
- **HW-Map-co-opt**: using DiGamma to co-optimize both HW and mapping[2].

6.3.2 Comparisons with Baseline Optimization Algorithms

Figure 6.6 shows the achieved performance (latency) by different optimization algorithms. Note that the proposed Co-opt Framework is a supportive back-end and can work well with

[2] The hyper-parameters of DiGamma, (mutation rate, crossover rate, elite ratio, population size to number of generations ratio, and so on), are decided by a Bayesian optimization-based search process [179].



Values	Latency								
Scheme	HW-Map-co-opt								
Method	Random	stdGA	PSO	TBPSA	(1+1)-ES	DE	Portfolio	CMA	DiGamma
	Platform Resources: Edge								
Resnet18	N/A	4.7	169.3	N/A	N/A	0.8	0.8	1.0	**0.5**
Resnet50	N/A	N/A	110.9	0.7	21.7	1.1	**0.3**	1.0	**0.3**
Mbnet-V2	N/A	29.0	1.0	0.5	0.5	**0.2**	3.7	1.0	**0.2**
Mnasnet	N/A	5.3	27.6	0.1	33.5	**0.04**	**0.04**	1.0	**0.04**
BERT	N/A	N/A	N/A	1.0	2.0	10.0	0.5	1.0	**0.4**
NCF	229.8	88.4	N/A	133.6	0.5	N/A	1.9	1.0	**0.4**
DLRM	304.7	151.5	1.2	168.8	2.8	5.2	15.6	1.0	**0.8**
GeoMean	264.6	24.9	14.4	2.9	3.2	0.8	0.9	1.0	**0.3**
	Platform Resources: Cloud								
Resnet18	N/A	64.9	1652.5	0.4	373.6	N/A	0.3	1.0	**0.03**
Resnet50	N/A	116.2	1904.7	3593.7	N/A	1.5	6.4	1.0	**0.3**
Mbnet-V2	0.8	0.8	20.8	0.8	8.9	3.2	0.6	1.0	**0.4**
Mnasnet	N/A	10.8	0.2	0.1	0.2	N/A	10.6	1.0	**0.1**
BERT	10669.5	18.9	27.2	13173.1	1043.3	42.3	2.3	**1.0**	1.2
NCF	47.1	15.0	0.6	0.3	41.2	0.6	0.4	1.0	**0.01**
DLRM	3481.5	17.1	0.8	28.0	54.1	4230.0	27.9	1.0	**0.6**
GeoMean	193.3	16.4	14.5	10.8	34.6	13.8	2.3	1.0	**0.1**

Values	Latency-Area-Product								
Scheme	HW-Map-co-opt								
Method	Random	stdGA	PSO	TBPSA	(1+1)-ES	DE	Portfolio	CMA	DiGamma
	Platform Resources: Edge								
Resnet18	N/A	6.3	46.2	N/A	N/A	1.2	1.2	1.0	**0.7**
Resnet50	N/A	N/A	12.0	0.7	20.8	1.1	**0.3**	1.0	**0.3**
Mbnet-V2	N/A	20.0	1.1	0.4	0.6	**0.2**	2.7	1.0	**0.2**
Mnasnet	N/A	1.0	2.5	0.1	35.3	**0.04**	**0.04**	1.0	0.05
BERT	N/A	N/A	N/A	0.7	2.6	13.2	0.7	1.0	**0.5**
NCF	179.3	78.4	N/A	105.0	0.5	N/A	1.9	1.0	**0.4**
DLRM	211.2	83.1	1.2	140.5	1.6	5.2	15.8	1.0	**0.7**
GeoMean	194.6	15.3	4.5	2.4	3.1	0.9	1.0	1.0	**0.3**
	Platform Resources: Cloud								
Resnet18	N/A	30.3	19.1	0.1	201.7	N/A	0.3	1.0	**0.03**
Resnet50	N/A	113.9	262.8	1352.9	N/A	1.4	4.5	1.0	**0.3**
Mbnet-V2	1.4	1.4	28.6	1.4	20.5	6.1	1.3	1.0	**0.7**
Mnasnet	N/A	7.2	0.2	0.1	0.3	N/A	12.0	1.0	**0.1**
BERT	1528.0	14.6	12.0	7659.3	990.3	35.8	2.4	**1.0**	**1.3**
NCF	39.2	10.8	0.8	0.4	14.9	0.8	0.6	1.0	**0.01**
DLRM	2098.1	8.8	0.6	16.0	53.0	760.0	26.1	1.0	**0.4**
GeoMean	115.7	12.6	5.5	7.3	30.6	11.4	2.5	1.0	**0.1**

Fig. 6.6: Performance comparisons of different optimization algorithms. Both latency and latency-area-product are normalized by the values of CMA, the best-performing baseline (lower is better). We highlight the best performing algorithm in different tasks (DNN models) in bold. N/A means the algorithm cannot find valid solution that fits in the area constraint under the set 40K sampling budgets.

Value	Latency						
Scheme	HW-opt (Optimzing HW, Fixed Mapping)			Mapping-opt (Fixed HW, Optimizing Mapping)			HW-Map-co-opt
Method	Grid-S HW + dla-like	Grid-S HW + shi-like	Grid-S HW + eye-like	Buffer-focused + Gamma	Medium-Buf-Com + Gamma	Compute-focused + Gamma	DiGamma
Platform Resources: Edge							
Resnet18	7.8	94.8	29.8	3.7	2.4	1.0	**0.8**
Resnet50	3.6	88.4	27.8	3.7	1.9	1.0	**0.8**
Mbnet-V2	2.4	48.5	15.9	2.9	1.8	1.0	**0.7**
Mnasnet	2.5	61.2	20.5	3.5	2.0	1.0	**0.7**
BERT	1.3	4.1	1.1	5.6	2.8	1.0	**0.8**
NCF	1.8	908.0	482.4	5.2	2.7	1.0	**0.8**
DLRM	1.8	890.8	473.4	4.7	2.4	1.0	**1.0**
GeoMean	2.6	97.4	35.3	4.1	2.3	1.0	**0.8**
Platform Resources: Cloud							
Resnet18	8.6	877.6	275.7	1.1	1.2	1.0	**0.2**
Resnet50	3.4	786.9	141.0	1.1	0.9	1.0	**0.4**
Mbnet-V2	3.2	276.4	90.6	1.1	0.9	1.0	**0.4**
Mnasnet	3.6	322.6	107.9	1.1	0.9	1.0	**0.4**
BERT	1.0	100.3	26.6	3.0	1.5	1.0	**0.7**
NCF	1.0	12288.1	6528.1	2.3	1.2	1.0	**0.9**
DLRM	1.3	10846.3	5763.9	1.8	1.1	1.0	**0.7**
GeoMean	2.4	972.6	324.8	1.5	1.1	1.0	**0.5**

Fig. 6.7: Latency of the found solution by different optimization scheme. Latency values are normalized by the values of best-performing baseline method (Compute-focused + Gamma). Grid-S: grid search. Buffer-focused: large buffer design. Compute-focused: large PE arrays design. Meidum-Buf-Com: medium buffer and PE arrays design.

HW-opt
(Grid-S HW +
dla-like)

π_{L2}	P	K	C	Y	X	R	S	π_{L1}	P	K	C	Y	X	R	S
7	K	7	64	3	3	3	3	64	C	1	64	3	3	3	3

Mapping-opt
(Compute-focused HW +
Gamma)

π_{L2}	P	S	R	X	K	Y	C	π_{L1}	P	C	X	R	S	Y	K
56	K	3	1	9	56	1	3	9	X	1	9	1	3	1	1

HW-Map-co-opt
(DiGamma)

π_{L2}	P	Y	C	R	K	X	S	π_{L1}	P	K	S	C	X	R	Y
33	K	7	2	2	33	83	3	12	X	1	3	1	12	1	1

Scheme	Latency (cycles)	Area (mm2)	Lat-Area-Product	PE Area : Buffer Area Ratio
HW-opt (dla-like)	3.74E+06	1.99E+05	7.44E+11	67 : 33
Mapping-opt (Com-focused)	1.14E+06	1.83E+05	2.08E+11	39 : 61
HW/Map-co-opt (DiGamma)	**9.80E+05**	1.99E+05	**1.95E+11**	56 : 44
				Area Constraint: 2.00E+5 (mm2)

Fig. 6.8: The solution found by different optimization schemes and their corresponding performance on Mnasnet at edge resources.

many state-of-the-art algorithms such as DE, Portfolio, and CMA. Considering both stability (without N/A) and performance, CMA is the best-performing one among the compared baseline algorithms. The performance value of CMA represents the best performance that Co-opt Framework could bring before discussing the new algorithm, DiGamma. Therefore, we normalize the values in Figure 6.6 by the value of CMA. We walk through more detail of Figure 6.6 and discuss DiGamma's performance, next.

At edge settings, some algorithms can achieve compatible performance with DiGamma in specific tasks such as DE and Portfolio in Mnasnet, however not stable with the respect to the stability across tasks (Figure 6.6). E.g., DE did not find any solution in NCF, and Portfolio performed 15.6x worse than CMA in DLRM. Besides the latency value of different solutions, we also show their corresponding latency-area-product, since some solution/designs could trade-off areas for better latency. E.g., in BERT cases, TBPSA ranks 2nd across 8 baseline algorithms in latency performance. However, with the respect to latency-area-product, TBPSA is the best among baseline algorithms, meaning it has better area efficiency to achieve similar latency performance comparing to others. Moreover, the poor performance of standard GA contrasts the effectiveness of DiGamma's domain-aware optimization operators. **At cloud settings**, the wider design space in cloud cases increases the complexity of the optimization tasks. Multiple algorithms are not able to find any valid solutions or can only achieve relatively bad performance. In addition, some performance-leading algorithms such as DE, Portfolio, and CMA at edge settings, become unstable and have much larger swing of achieved performance across different tasks/models at cloud settings. In contrast, DiGamma can stably achieve compatible or better performance than others. Overall, comparing to best-performing baseline algorithm (CMA, Portfolio), DiGamma achieves (geomean) **3.0x** and **10.0x** better latency performance at edge and cloud settings, respectively.

6.3.3 Comparisons with Baseline HW-Mapping Schemes

In Figure 6.7 we found that, among the compared methods, using heuristic HW configuration (Compute-focused) with existing mapping searching tool (GAMMA) can yield the best performance, whose value is thus used to normalized the values in Figure 6.7. It represents the best achievable relative performance gain before proposing Co-opt Framework and DiGamma. We describe more detail of Figure 6.7, next.

Comparing with HW-opt. In this experiment, we model the scenario that the researchers designed an optimized mapping for certain DNN models such as NVDLA [80], ShiDianNao [81], and Eyeriss [178] mapping and want to explore their performance across different models. However, different tasks/ DNN models expose different characteristics (e.g., in general, CNNs are more compute-intensive, and recommendation models are more memory-intensive). To achieve better performance, the researcher could apply optimization algorithms to search for optimal HW configurations (PEs and buffers). Here, we use the grid search approach to search through different PEs and buffers configurations. In Figure 6.7, we could observe that DiGamma constantly achieve better performance across three different HW-opt methods, where DiGamma is (geomean) **3.25x** and **4.8x** better than the best-performing method (Grid-S + dla-like), in edge and cloud settings, respectively.

Comparing with Mapping-opt. In this experiment, we model the scenario that the researchers designed a mapping optimization algorithm, however relying on a pre-defined HW configuration, which become an inductive bias from the human. For example, different researchers will design different balances between compute and memory resources, where we model with three sets of configurations (§6.3.1). Note that the previously effective strategy of exhaustive grid search does not fit this scenario, since grid searching HW plus mapping optimizations will form two loops of the optimization process, whose required sampling budget and search time increase exponentially. In Figure 6.7, we can observe that DiGamma is **1.25x** and **2.0x** better than the best-performing method (Compute-focused + Gamma), in edge and cloud settings, and more importantly, without the need of human-in-the-loop to cherry-pick the HW configurations.

6.3.4 Explanation of Found Solutions

Figure 6.8 shows three solutions of different optimization schemes for Mnasnet at edge resources. In HW-opt, we could observe the output/input-channel (K-C) parallelism features of dla-like mapping. In Mapping-opt, we could observe the mapping optimizer find an unique mapping strategy, channel and activation (K-X) parallelism, which is different from dla-like (K-C), shi-like (Y-X), and eye-like (Y-R, row-stationary). DiGamma also finds a mapping with K-X parallelism for Mnasnet, however with better compute and buffer balance, therefore achieving 3.8x and 1.6x better performance than HW-opt and Map-opt.

6.4 Related Work

HW Resources Allocations Optimizations. The HW resource allocation problem has been widely-studied in the FPGA community [180, 181, 182, 17, 62, 65, 66, 63, 61, 64, 67, 183]. With the fast-growing of DNN/ML researches, the DNN ASIC accelerator design starts to find the need of autonomous optimization methods to assisted the accelerator design for speeding-up chip design time [184, 185, 186, 187, 188]. Recently, more and more ML techniques are integrated in the chip design process, such as reinforcement learning (RL) [189, 190] and GNN [191].

Mapping Optimizations. Many mapping optimization methods have been proposed. Techniques includes: formulating a more comprehensive design space for better capturing the accelerator behavior [5, 4, 8], improving the design space description for search efficiency [10, 9, 15], and incorporating different ML techniques in the optimization process [27, 26, 101, 24, 16, 31].

These prior efforts focus on the optimization of either HW or mapping independently. In this work, we co-optimize the mapping and HW together.

6.5 Summary

We propose a HW-Mapping co-optimization framework (Co-opt Framework), which takes in any DNN model(s), design objective, budget, and constraint, and generates an accelerator design point, HW (i.e., numbers of PEs, number of memory levels, sizes of buffers at each level) and mapping (i.e., parallelism, loop order, clustering, tile sizes). We abstract the detail of performance modeling for different DNNs, chip constraints and provide a generic interface, where many existing optimization algorithms can be plugged in. We propose an efficient design point encoding, which describes both HW and mapping with a list of parameters. Our encoding method constructs a compact representation of the cross-coupled design space that boosts the efficiency of the optimization algorithms. We propose a domain-aware genetic algorithm-based optimization method. It is specifically designed for HW-Mapping design space and comes with specialized optimization operators to step through the design space in a structured manner, and its HW exploration strategy respects the interaction between HW and mapping.

CHAPTER 7
MAGMA: A GA-BASED MAPPER FOR MULTI-ACCELERATOR SYSTEM

So far we discuss Map Space Exploration (MSE) or Mapping-HW co-exploration for single accelerator. However, as AI workloads continue to drive up the demand for compute, there is a trend towards building large accelerators housing several sub-accelerator/arrays. Key examples include MCM-based SIMBA [8], wafer-scale Cerebras [192] or scaled-out platforms [193, 194, 195]. Some recent studies have also explored heterogeneous multi-accelerator designs enabled via reconfiguration [129] or separate *heterogeneous* sub-accelerators [196]. In this chapter, we will discuss what is the new research opportunity with respect to mapping on these multi-accelerator architectures, what are the challenges to port existing mappers into this problem domain, and how we tackle these challenges by a new mapper design for these new DNN acceleration platforms.

With the emergence of such platforms, enabling multi-tenancy, i.e., multi-DNN mappings on the an accelerator/ platform, is a natural use-case. Data center workloads often run three categories of inference tasks: vision, language and recommendation, and in each task it involves variants of related DNN models. In this chapter, we target all three use cases and focus on batched-job tasks (jobs launched in bulk without latency constraint but with high-throughput need), e.g., Google photo auto-editing, image tagging, video processing, and voice processing.

There have been a few recent works looking into the problem of mapping multi-DNN workloads on multiple accelerator cores. PREMA [197] develops a mapper for multi-tenant language tasks, however targeting single-core accelerator. AI-MT [194] successfully designs a mapper for homogeneous multi-core accelerators and shows performance improvement over vision and language tasks. Herald [196] targets heterogeneous multi-core accelerators and systematically analyzes the benefit of heterogeneity in dataflows across the accelerator cores for AR/VR workloads (vision tasks). These works demonstrate the impact of a mapping (of DNNs) for the new multi-tenant multi-core accelerators, which is of rising interest. However, all these approaches rely on manually-designed heuristics. This limits their scalability to diverse accelerator back-ends and emerging workloads. In this chapter, we develop an automatic mapping search process which includes two specific contributions (i) an optimization framework and (ii) a novel optimization algorithm.

7.1 Optimization Framework (M3E)

We propose a mapping optimization framework for multi-tenant heterogeneous accelerator. The structure of our proposed framework called M3E is shown in Figure 7.1.

At a high-level, the M3E consists of an optimization phase and an evaluation phase. At evaluation phase, the candidate mapping is evaluated by the cost model. At optimization phase, the optimization algorithm tweak the mapping based on the feedback from the evaluation phase. The optimization-evaluation loop happen iteratively until the targeting objective converges or after a fixed set of time epochs.

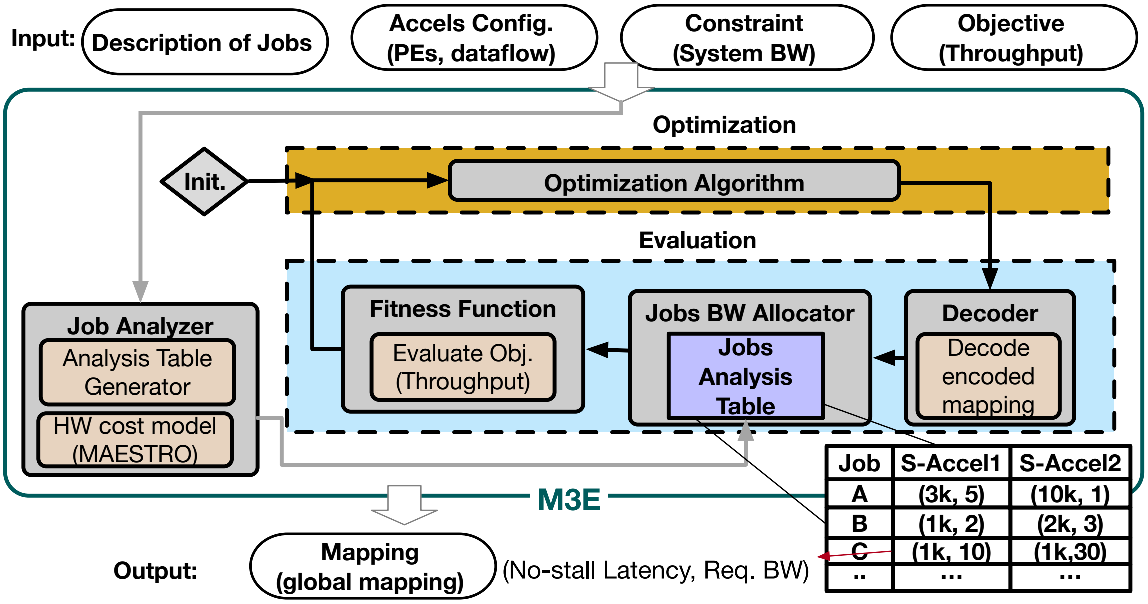
The mapping consists of two key components:

- **Sub-accelerator selection:** the assignment of each job to a specific sub-accelerator.
- **Job prioritization:** execution order of jobs on a given sub-accelerator.

To successfully frame a problem into an optimization process, there are two critical pillars: encoding (how the search space is described) and optimization algorithm (how the search space is explored). We describe them as follows.

7.1.1 Encoding

An encoded mapping should encode the joint strategy of job prioritization and sub-accelerator selection. We encode them into a series of values with two separate sections, as shown in Figure 7.2. The sub-accelerator selection section decides which job goes to which sub-accelerator. The job-prioritizing section decides the order of the jobs in each sub-accelerator.



Job	S-Accel1	S-Accel2
A	(3k, 5)	(10k, 1)
B	(1k, 2)	(2k, 3)
C	(1k, 10)	(1k,30)
..	...	...

Fig. 7.1: The structure and flow† of M3E.

The length of the section is equal to group size. A full mapping consists of two sections with total length 2x group size. We describe the encoding using the example in Figure 7.2 assuming two sub-accelerators and a group size of 5.

Sub-accelerator Selection Section. Each value describes the sub-accel ID for the corresponding job. For example, jobs J1 and J4, are assigned to sub-accel 1, and J2, J3, and J5 are assigned to sub-accel 2 as shown in the sub-accel selection part of the decoded assignment in Figure 7.2.

Job Prioritizing Section. Each value describes the priority of the corresponding job. The priority value ranges from 0 to 1, where 0 is the highest priority. We order the job assigned to a certain sub-accelerator by the order of priority value. For example, J1 runs before J4 in sub-accel 1 as shown in the job prioritizing part of the decoded assignment in Figure 7.2.

7.1.2 Optimization Algorithms Supported

With an encoded search space and the support of building blocks of M3E (described later in §7.1.4), we support several popular optimization algorithms, as shown in Table 7.3.

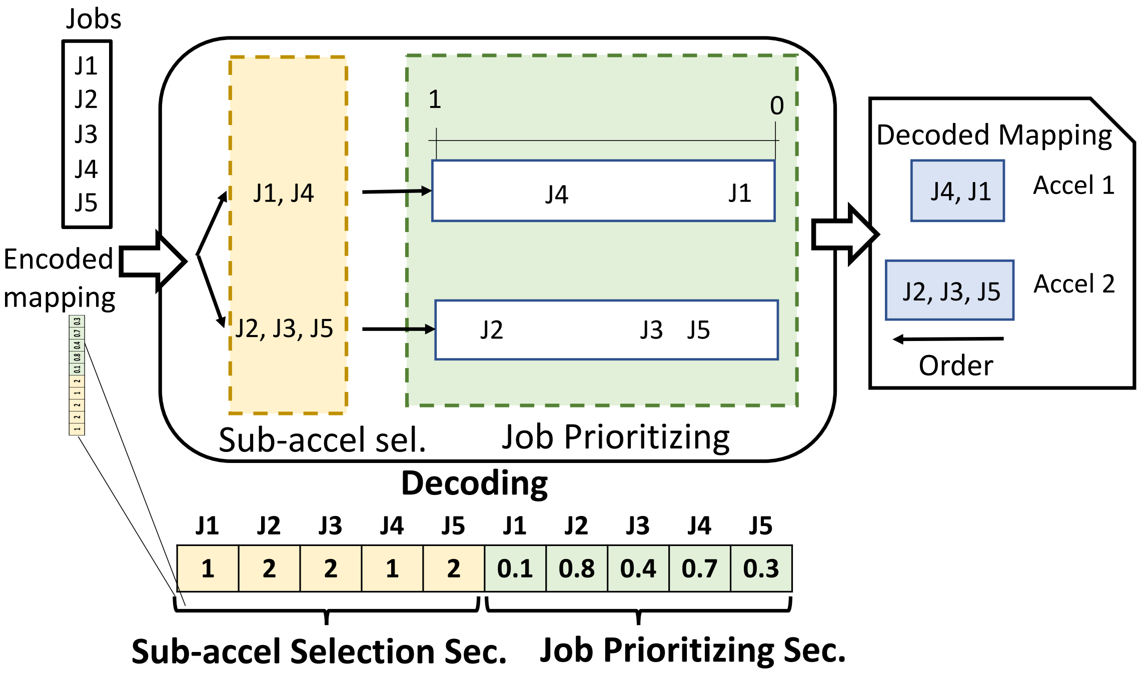


Fig. 7.2: The encoding scheme of M3E.

We include multiple black-box optimization methods such as Differential Evolution [108], Covariance Matrix Adaptation Evolution Strategy [114], and Particle Swarm Optimization [117]. In addition, we also include two widely-used reinforcement learning methods — Advantage Actor-Critic (A2C) [198] and Proximal Policy Optimization (PPO2) [199]. Finally, we also support our novel optimization algorithm called MAGMA (§7.2). With the established framework, M3E can also easily be extended to support other algorithm proposals.

7.1.3 Objective and Constraints

We target throughput as our main objective. However, other objective can also be set (e.g., latency, energy) or formulated (e.g., energy-delay-product, performance-per-watt). The objective can simply be specified as an input to the M3E, as shown in Figure 7.1. We consider BW constraint (however similarly, other constraints can also be specified). The BW constraints include accelerator-to-host BWs (e.g.,PCIe, M.2) and host memory BW (eg., DRAM/HBM BW), which are common constraints in the practical deployment

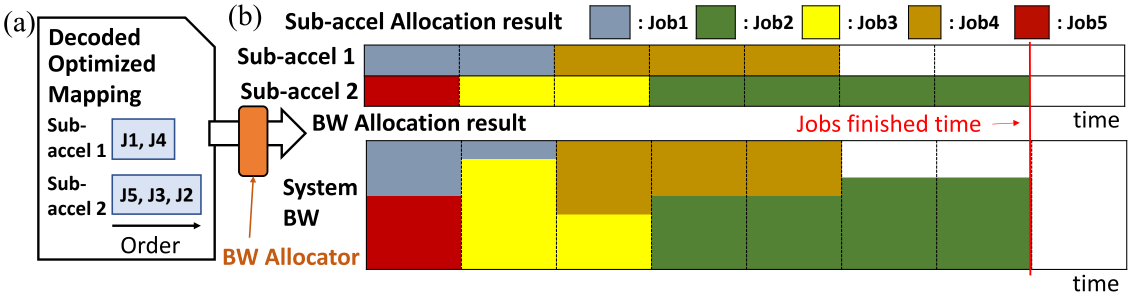


Fig. 7.3: (a) Mapping description from the decoder. (b) The example BW and sub-accelerators allocation results.

scenario [**park2018deep**, 196]. For simplicity of the optimization framework, we take the minimum of the two (the more stringent BW constraint), as the BW constraint known by the optimization framework, and we name this constraint — system BW.

7.1.4 Building Blocks of M3E.

BW Allocator

However, in a multi-core accelerator, system BW to the accelerator becomes a global shared resources between cores (sub-accelerators). To evenly allocate the same amount of BW to all the sub-accelerators is an often applied heuristics. However, it will increases the possibility of compute resource under-utilization. E.g., in a normal single-accelerator case, depth-wise CONV jobs are often more memory-intensive than regular 2D CONV jobs, which can make the accelerator under-utilized when running depth-wise CONV while it is fully-utilized when it runs 2D CONV. In the multi-core accelerator, where the system BW is a global shared resources [**park2018deep**], it gives us a chance to reallocate the BW to alleviate the under-utilization problem by proving more BW to core running memory-intensive jobs and proving only adequate BW to cores running compute-intensive jobs, which motivates the BW allocator (Algorithm Algorithm 1). The BW allocator is reallocating the BW based on the memory BW intensity of different jobs running on different sub-accelerators.

In detail, receiving the mapping, the BW allocator lookup those jobs' no-stall latency (§7.1.4) and required BW from the job analysis table (§7.1.4), and allocates the system BW to each sub-accelerator at each time frame with the ratio of their required BWs. It outputs the detailed BW allocation results, as shown in Figure 7.3(b).

Figure 7.3(b), a BW allocation results, as an example, we can tell, jobs J1 and J5 will be launched in Sub-accel-1 and Sub-accel-2, concurrently. Sub-accel-2 will be allocated more BW because it is running a more BW-intensive job. When Sub-accel-2 finishes J5 and launches J3, the BW will be re-allocated to reflect the change of live running jobs in the accelerators, where Sub-accel-1's BW is reduced and reallocated to Sub-accel-2 in Figure 7.3.

Job Analyzer

The job analyzer takes the jobs description as input and estimates the no-stall latency and its required BW for each sub-accelerator using a cost model (described below) to generate a job analysis table as Figure 7.1 shows. This table serves as a performance lookup table by the BW allocator (§7.1.4) within the optimization loop.

HW cost model for Sub-Accelerators

In M3E, we leverage MAESTRO [58] as our underlying cost model for each sub-accelerator because of its ability to support diverse accelerator dataflows and configurations[1]. It supports most of the common DNN layers such as CONV, depth-wise CONV, and fully connected. Given a DNN layer, a HW resource configuration (PE, SL size, SG size, NoC latency, and BW), and a mapping/dataflow strategy, MAESTRO estimates the statistics such as latency, energy, runtime, power, and area.

[1] In this chapter, we explore heterogeneity with the aspect of different specialized DNN accelerators configurations (PEs, buffer size, dataflows). However, M3E is general enough, so that it could also consider generic architectures such as CPUs/GPUs/TPUs by plugging in their cost models.

Job Analysis Table

Job Analyzer profiles each job in the task by the cost model [58] and stores the profiling results in the Job Analysis Table. In the optimization process, Job Analysis Table serves as a quick look-up table for fitness evaluation to avoid frequently querying the cost model. The profiling has two main information: no-stall latency and no-stall bandwidth, described next.

No-stall Latency. We define no-stall latency as the latency for running each job on each sub-accelerator, assuming it has sufficient memory bandwidth (i.e., not memory-bound).

No-stall Bandwidth. We define no-stall bandwidth as the minimum bandwidth requirement from each sub-accelerator to make it stay compute-bound, not memory-bound.

7.1.5 M3E Workflow

Set-up: At the start, the user/host sets up the optimizer by feeding in the jobs descriptions, configurations (number of PEs, dataflow) of each sub-accelerators, the system constraint (system BW), and objective (e.g., throughput).

Pre-process: **Job analyzer** Job Analyzer prepares the Job Analysis Table, as shown in Figure 7.1.

Optimization Loop: *Optimization phase:* optimization algorithm updates the encoding mapping based on the feedback from the evaluation block. *Evaluation phase:* **Decoder** decodes encoded mapping into a mapping description, as shown in Figure 7.3(a). **BW Allocator** takes in the mapping description and allocates the BW for each sub-accelerator. **Fitness function** extracts the objective and sets it as fitness value.

This finishes one loop/ epoch of optimization. The optimization loop stops when M3E reaches the set sampling budget (the number of allowed sampling data points in a search process).

Algorithm 1 BW Allocator

```
**Input:** Mapping description
**Output:** $BW_t^{alloc}$, t=1,2...T
Get $Lat_t$, an array of no-stall latency for the parallel jobs at time t, t=0
Get $BW_t^{req}$, an array of required BW for the parallel jobs at time t, t=0
$CurJobs_t = Lat_t \times BW_t^{req}$
**while** $CurJobs_t$ is not empty **do**
    **if** sum($BW_t^{req}$) $< BW_{sys}$ **then**
        $BW_t^{alloc}$ = $BW_t^{req}$
    **else**
        $BW_t^{alloc}$ =$\frac{BW_t^{req} \times BW_{sys}}{sum(BW_t^{req})}$
    **end if**
    $runtimes$ =$\frac{CurJobs_t}{BW_t^{alloc}}$
    $runtime$ = min($runtimes$)
    $CurJobs_t$ -= $runtime \times BW_t^{alloc}$
    $accel_{next}$ = argmin($CurJobs_t$)
    $t$ += $runtime$
    Fetch the next $Lat$ and $BW^{req}$ of $sub - accel_{next}$, compute $CurJob_t$ and insert into $BW_t^{req}$, $Lat_t$ and
    $CurJobs_t$.
**end while**
```

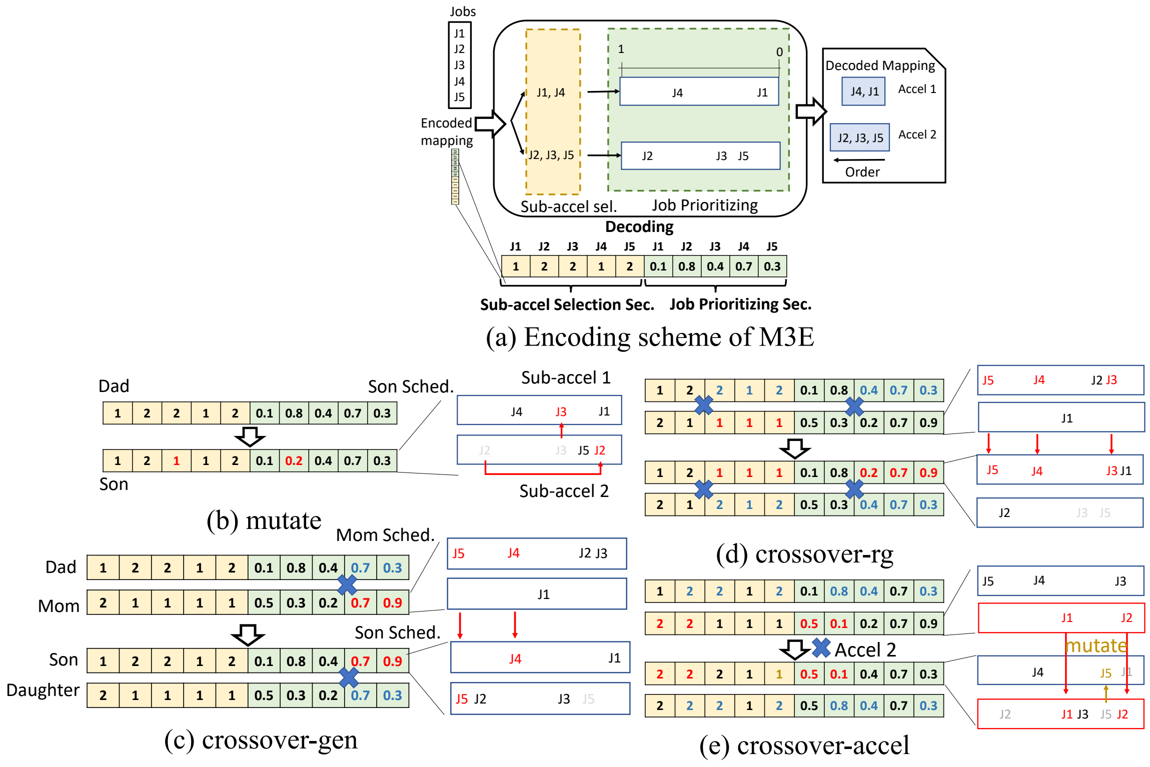
7.1.6 Search Space

The full search space of the proposed framework is the combinatorial combination of the choices for sub-accelerator selection and job prioritizing. Assuming the accelerator has 4 sub-accelerator and we use the group size of 60. The size of the design space is $(60!)/(15!)^4 \times (15!)^4 = 60! = O(1e81)$ which is extremely massive. Therefore the *sample efficiency*[2] of the optimization methods, which decides the convergent rate, becomes a key factor. We describe our proposed sample-efficient optimization method, next.

7.2 Optimization Algorithm (MAGMA)

MAGMA is a GA-based search technique. Its key difference from standard GA is that it customizes the optimization algorithm's exploration momentum and mechanism (i.e., genetic operators in GA context) for the target search space.

[2] Performance improvement over the number of sampling budget.



(a) Encoding scheme of M3E

(b) mutate

(c) crossover-gen

(d) crossover-rg

(e) crossover-accel

Fig. 7.4: (a) Then encoding scheme of M3E. The genetic operators of MAGMA and their implying update on the mapping: (a) mutation, (b) crossover-gen, (c) crossover-rg, and (d) crossover-accel.

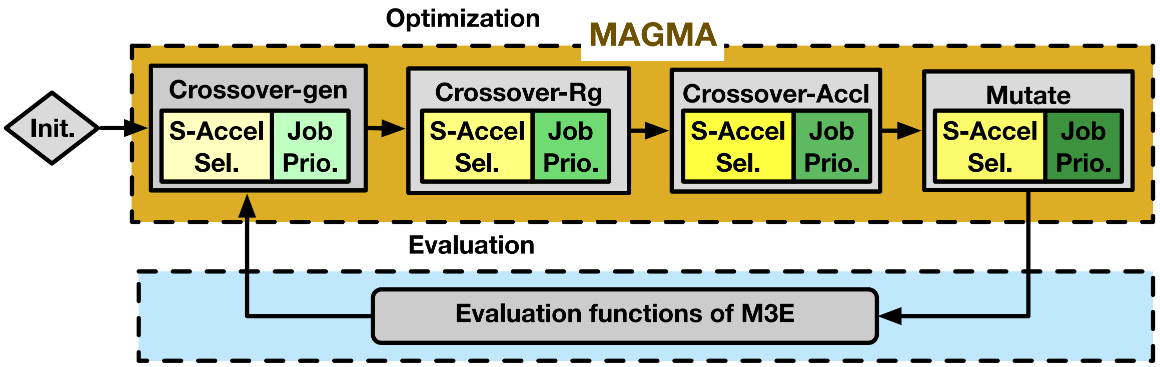


Fig. 7.5: The algorithm flow of MAGMA.

Table 7.1: Terminology used in MAGMA Algorithm.

Term	Description
Gene	An encoded value that represents accel. sel. or job prio. of a job.
Genome	A series of genes that represent the entire schedule about accel. sel. or job prio. of a batch of jobs.
Individual	A series of genomes that fully represent the schedule of a batch of jobs.
Generation	An entire set of individuals forms a generation. The generation evolves with time by mutation/crossover and selection of the well-performing individuals to the next generation.
Crossover	Blend two parents' genes to reproduce children's genes.
Mutation	Randomly perturb a parent's genes to reproduce children's genes.

7.2.1 Why GA?

Research shows GA reaches competitive performance with deep reinforcement learning [155, 156], and hyper-parameter optimization problem. STOKE [103] and Tensor Comprehensions [31] use GA to search the space of DNN code optimization. From a search time perspective, GA is light and fast [155, 156] comparing to many optimizations methods since the optimization mechanism in GA uses simple operations (e.g., crossover and mutations). A key challenge with standard GA however is that it is not sample-efficient. We address this issue using our customized operators (§7.2.2).

7.2.2 MAGMA Algorithm Details

Terminology and Basics of GA

We list the common terminology of GA we use throughout the paper in Table 7.1, namely *gene, genome, individual, generation*. The basic mechanism in GAs is to create a population of individuals in each generation. All individuals are evaluated and sorted based on their fitness. The best performing individuals are used as parents to create a new population of individuals using genetic operators (§7.2.2). The goal of GA is to perturb genes (i.e., components of the schedule) and retain well-performing ones across generations.

Genetic operators

Standard GA Operators. The standard genetic operators in GA consist of mutation and crossover. The standard mutation operator randomly mutates some genes. The standard crossover operator samples a pivot point and exchanges the genes of parents according to that pivot point. The sampling efficiency of the GA relies on the efficiency of the genetic operators to sample high-quality next generation.

MAGMA Operators. In MAGMA, we inherit the standard mutation mechanism and design *three specialized crossover genetic operators*. Different crossover operators are designed to preserve different dependency of genes while exploration. They allow us to explore the scheduling problem in a more strategical manner. We describe the genetic operators next.

Mutation. During mutation, we randomly select multiple genes (according to the mutation rate) and mutate them to random values. Figure 7.4(a) shows an example when mutating at the third and second genes of two genomes respectively. On the right side of the figure, it shows how the son's genes/schedule are generated by the dad's mutation. J3 is moved to sub-accel 1 because of the first mutation. J2 is moved to a higher priority in sub-accel 2 because of the second mutation. In our experiments, we use a mutation rate of 0.05.

Crossover-gen. This is a genome-wise crossover. First, we randomly sample a type of genome to crossover. Next, we randomly sample a pivot point and exchange the genes of the genomes. There are two benefits of genome-wise crossover. First, we keep the perturbation to the level of the genome, which potentially keeps the good characteristics of the other un-touched genomes, and therefore is more stable throughout the evolution. Second, we eliminate the order dependency of the genomes. The genomes are independently representing their features, where the order of them provides no information (i.e., representing Sub-accel Sel. genome first and Job Prio. Genome later does not make the J5 of Sub-accel Sel. and J1 of Job Prio. strongly correlated despite their being next to each other.). Therefore, a genome-wise crossover, which operates genomes independently, enables us to perturb the gene without unnecessary assumptions of the genome order. Crossover-gen becomes the major crossover function, which we set the crossover rate as 0.9.

Figure 7.4(b) shows an example that we pick the second genome (Job Prio.) as the crossover region and the third location of the region as the pivot point. With the respect of schedule change after crossovering, in the example, the orders of J4 and J5 in mom's schedule are passed to son's schedule.

Crossover-rg. This is a range crossover mechanism structured to preserve the the dependency of genes across genomes. For example, in Figure 7.2, the first and the sixth genes are dependent, since they are both representing some features for J1. We randomly pick a range of genome (e.g., the 3rd to the 5th locations of each genome) and simultaneously crossover all the genes falling into the picked region from both genomes, and thus the cross-genome dependency is preserved. With the respect of scheduling change after crossovering, the order and accel selection of J3, J4, and J5 are exchanged between two individuals, as shown in Figure 7.4(c). Crossover-rg has crossover rate of 0.05.

Crossover-accel. This is a crossover method to preserve the dependency of job ordering within an sub-accelerator. We randomly select a sub-accelerator and pass the job ordering information of this sub-accelerator to the children. For example, in Figure 7.4(d), we select sub-accel 2. Next, we check the Sub-accel Sel. genome of Mom, copy the genes related to sub-accel 2 (the first and second genes of both genomes in (e)), and paste them to son's genomes.

To increase load balancing, the original jobs assigned to sub-accel 2 in Son will be randomly mutated. Crossover-accel has crossover rate of 0.05.

Hyper-parameter Tuning

The above mentioned mutation, crossover rates, populations sizes, and elite ratios are hyper-parameters in MAGMA. We applied a hyper-parameter search via a Bayesian optimization

Table 7.2: Accelerators configurations/ settings of the experiments.

Setting	Description	# of sub-accels	(height of PE array, dataflow style, buffer)
S1	Small Homog	4	4x(32, HB, 146KB)
S2	Small Hetero	4	3x(32, HB, 146KB), 1x(32, LB, 110KB)
S3	Large Homog	8	8x(128, HB, 580KB)
S4	Large Hetero	8	7x(128, HB, 580KB), 1x(128, LB, 434KB)

S5	Large Hetero BigLittle	8	3x(128, HB, 580KB), 1x(128, LB, 434KB)
			3x(64, HB, 291KB), 1x(64, LB, 218KB)

S6	Large Scale-up	16	7x(128, HB, 580KB), 1x(128, LB, 434KB)
			7x(64, HB, 291KB), 1x(64, LB, 218KB)

2D PE array: h (height) × w (width) (PEs), w=64, in the experiments			

framework [140] to select a set of hyper-parameters that makes MAGMA achieve the highest performance across multiple workloads.

7.2.3 Warm-Start of MAGMA

In this section, we present the techniques we implement to enable the warm-start of the algorithm. Warm-start is a well-known technique in black-box optimizations to enable faster convergence or reaching better objective value. Warm-start works as follows. There are series of tasks to be solved by the optimization algorithms. If the current task is the same or similar to the previous solved tasks, we can take the previous solution to initialize the algorithms. We also implement an warm-start engine, which recognize if the current task fall within the same types of tasks (Vision, Recommendation, or Language), i.e., whether it is similar to the previous solved task. If tasks are within the same type of task, the warm-start engine will take over the initialization job from Init engine (initializing algorithm randomly). In our experiment (§7.3.7), we found warm-start is a useful add-on technique in MAGMA.

Table 7.3: Supported optimization algorithms in M3E.

Alg.	Description
AI-MT-like	A manual-tuned mapper for multi-core accelerator targeting vision and language workloads.
Herald-like	A manual-tuned mapper for multi-core heterogenous accelerator targeting vision workloads.
stdGA	Genetic Algorithm. *We use mutation rate: 0.1, crossover rate: 0.1.*
DE	Differential Evolution. *We use weighting for local DV: 0.8, weighting for global DV: 0.8, in the experiment.*
CMA-ES	Covariance Matrix Adaptation-ES. *We use 1/2 of the best performing individuals as an elite group in the experiment.*
TBPSA	Test-based Population-Size Adaptation. *We set the initial population size as 50 and let it evolve in the experiment.*
PSO	Particle Swarm Optimization. *We use weighting for global best: 0.8, weighting for parent best: 0.8, with momentum ω: 1.6.*
RL A2C	Advantage Actor-Critic. *We use policy and critic networks composed by 3 MLP layers with 128 nodes, discount factor: 0.99, learning rate: 0.0007, RMSProp optimizer.*
RL PPO2	Proximal Policy Optimization. *We use policy and critic networks composed by 3 MLP layers with 128 nodes, discount factor: 0.99, clipping range: 0.2, learning rate: 0.00025, Adam optimizer.*
MAGMA	A GA-based optimization algorithm that houses domain-specific genetic operators for multi-core heterogenous accelerator mapping problem.

7.3 Evaluations

7.3.1 Methodology

Target DNN Models

We consider three different types of tasks/ applications with their corresponding models collected from PyTorch [200]: Vision ([201, 202, 203, 34, 126, 204, 205, 206, 207, 146]), Language ([208, 209, 127, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225]), and recommendations ([226, 227, 228, 144, 229, 230]). The models are the ones used for the batched high-throughput applications (e.g., photo auto-editing, image tagging, and video / voice processing) that we are targeting.

Task and Benchmark

We categorize the jobs into Vision, Language (Lang), Recommendation (Recom), and Mix (a complex tasks with vision, language, and recommendation model involved simultaneously), four different types of tasks. We build a benchmark including different tasks motivated by the Facebook's inference accelerator jobs [**park2018deep**, **anderson2021first**], edge data centers for AI applications [**richins2020missing**], Herald [196]. AI-MT [194], and others [**shen2019nexus**]. In the benchmark, we collect models from the four different types of tasks and create several workloads. Each workload contains hundreds to thousands of jobs (one job include a batch of activations and weight parameters of a layer). We chopped them into several "dependency-free" group similar to prior works [194]. The objective of the optimization algorithm is to execute these group with the highest possible throughput. We set the default group size to be 100 but also study the effect of group size in §7.3.7.

Accelerators

We consider two classes of accelerator: Small and Large. For each class, we consider multi-core homogeneous and heterogeneous accelerator settings with different PEs, dataflow, and on-chip buffer. We construct six different multi-core accelerators, motivated by [196, 194, 82, 8], as our test-bed in Table 7.2. S1 and S3 represent homogeneous accelerators. S2, S4-6 represent heterogeneous accelerators. The accelerators are modeled with MAESTRO [58]. We uniformly set one dimension of the 2D PEs array to 64[3] and scale the PEs array size by increasing the other dimension. We consider three kinds of PEs configuration: 32 × 64 for Small accelerator [231, 232, 233, 234, 235], 64 × 64 and 128 × 64 for Large accelerator. The dataflow style (discussed next) and target tile sizes determine the buffer sizes for both SL and SG [68].

Sub-Accelerator Dataflow Styles. For our evaluations, we pick two distinct dataflow

[3] Based on our observation, most of the popular models that we collected, especially language and recommendation ones, are manually designed to have the tensor shape formed by the multiples of 64. Setting one dimension to 64, which aligns with the tensor shape, ensures higher utilization rate.

styles for the heterogeneous sub-accelerators: High Bandwidth usage dataflow style (HB) (inspired by NVDLA) [80]) and relatively Low Bandwidth usage dataflow style (LB) (inspired by Eyeriss [178]). The HB-style parallelizes across channel dimensions, and shows high-efficiency on late layers for CNN-based (vision) models, while the LB-style parallelize across activations dimensions and excels on the early layers of CNN-based models [68]. For Language and Recommendation, we found the HW-style is more compute efficient but BW intensive, while LB-style is less compute efficient but also less BW demanding (Figure 7.6). Therefore we house both these sub-accelerators in a BW constrained accelerator platform to act as a good test for our optimizer to learn and exploit their difference. M3E is general enough to run with any heterogeneous combination of two or more accelerator styles.

System BW. The accelerators are executing under frequency 200MHz and bit-width of 1 Byte. For the system BW, at the Small accelerator, we consider the BW to be range from 1GB/s to 16GB/s, which is the range of DDR1-DDR4 BW [236] and PCIe1.0 - PCIe3.0 [237] BW; at the Large accelerator, we consider the BW to be range from 1GB/s to 256GB/s, which is the range of DDR4-DDR5 [238] and HBM BW [239] and PCIe3.0 - PCIe5.0 and upcoming PCIe6.0 BW [237].

Evaluation Metric In all experiments, we use throughput as objective.

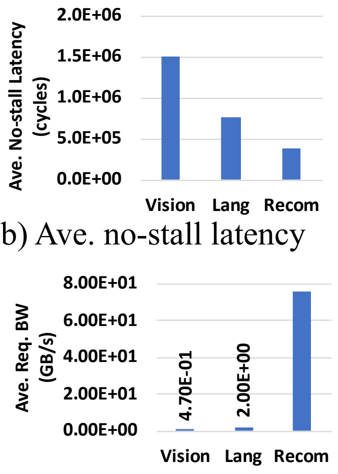
7.3.2 Mapper Settings.

Baseline Manual-tuned Mapper. We use the mapper from Herald [196] and AI-MT [194] as the baseline methods (Herald-like and AI-MT-like). Note that the mapper in Herald [196] is manual-designed targeting multi-core heterogeneous system with Vision tasks, and the mapper in AI-MT is manual-designed targeting multi-core homogeneous system with Vision and Language tasks. In our evaluation, we also tested their performance in Recommendation and Mix tasks and on both homogeneous and heterogeneous accelerators.

Optimization methods. We enable many commonly-used optimization methods in M3E. The specific hyper-parameters settings are listed in Table 7.3. For fair comparisons,

		Ave. no-stall latency		Ave. Req. BW (GB/s)	
		(dla,64)	(eye,64)	(dla,64)	(eye,64)
Vision	MobileNetv2	3.1E+05	1.1E+06	6.8E-01	8.8E-02
	Resnet50	8.5E+04	7.0E+06	5.2E-01	3.6E-04
	Shufflenet	9.6E+04	4.9E+05	1.5E+00	2.3E-02
	Ave.	1.7E+05	2.8E+06	9.0E-01	3.7E-02
Laang	GPT2	1.7E+04	3.5E+06	3.4E-01	1.6E-05
	MobileBert	5.5E+02	1.1E+05	1.1E+01	5.0E-04
	TransformerXL	4.5E+03	9.2E+05	1.3E+00	6.4E-05
	Ave.	7.4E+03	1.5E+06	4.1E+00	1.9E-04
Recom	DLRM	2.4E+02	9.7E+05	5.6E+01	2.5E-04
	WideDeep	2.8E+02	1.1E+06	3.0E+01	5.2E-06
	NCF	4.6E+01	1.8E+05	3.7E+02	6.4E-05
	Ave.	1.9E+02	7.6E+05	1.5E+02	1.1E-04

(a) Jobs analysis



(b) Ave. no-stall latency

(c) Ave. BW requirement

Fig. 7.6: (a) The average per-job no-stall latency and required BW for no-stall across different models on high (HW) and low (LB) bandwidth mapping style. (b) Average no-stall latency and (c) average BW required for no-stalls across all involved jobs.

all optimization methods are given the same sampling budget, 10K data points. Note that batched-job tasks are not latency sensitive, where the jobs and optimization process are executed off-line. Therefore the search time of different methods is not our main concern; instead, the objective of these tasks are to utilize the underlying hardware as efficient as possible, i.e, maximizing the throughput of the underlying hardware.

MAGMA. MAGMA is also one of the optimization methods. We set the number of individuals in a generation, to be as large as group size. We also constraint MAGMA to have the same 10K sampling budget, and use population size of 100, and thus have 100 epochs for optimizing. As for search time, we run the experiments on a desktop with Intel i9-9820 CPU. MAGMA takes about 0.25 seconds per epoch, and 25 seconds for a full optimization process.

7.3.3 Latency-BW Characteristics of DNNs

We start by showing the latency characteristics and bandwidth requirements of the DNN models from the three types of tasks when running by itself on two separate dataflow styles (HB and LB). We show three of the models from each type of tasks and the average across

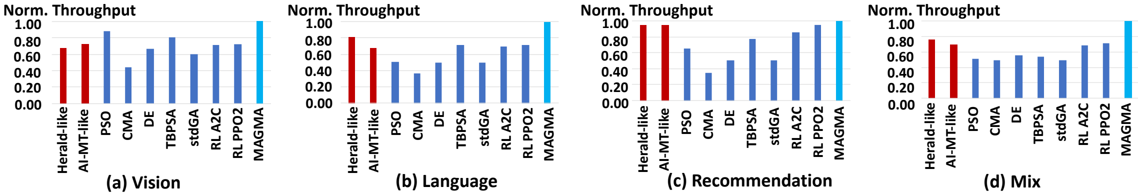


Fig. 7.7: The experiment results on multi-core homogeneous small accelerator (S1) with BW=16 across four tasks. Throughput values are normalized by the value of MAGMA. The absolute throughput values of MAGMA in (a-d) are: 249, 397, 194, and 329 GFLOPs.

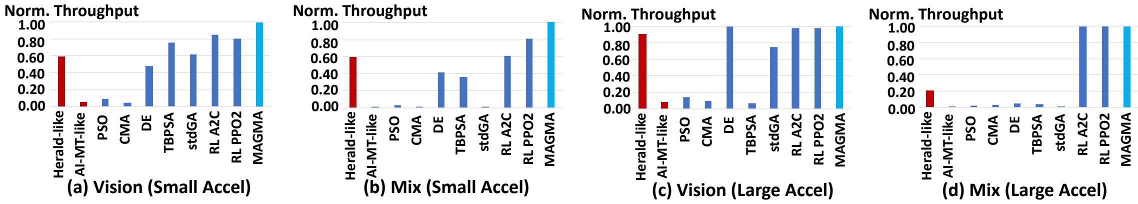


Fig. 7.8: The experiment results on multi-core heterogeneous (a)(b) small (S2, BW=16) and (c)(d) large (S4, BW=256) accelerator on Vision and Mix tasks. Throughput values are normalized by the value of MAGMA. The absolute throughput values of MAGMA in (a-d) are: 254, 271, 254, and 383 GFLOPs.

all the models in that type of tasks in Figure 7.6(a). The average values across all model across both accelerators are plotted in Figure 7.6(b-c). In general, we can see that the per-job latency of the Vision models is higher because more compute is needed in the CONV dominant models. However, CONV is generally less memory-bound than FC. The data also shows that usually Vision has the lowest BW requirement, and Recommendation has the largest.

7.3.4 Homogeneous Accelerators

We examined the Small homogeneous accelerator (S1) with system BW=16 GB/s across different tasks. As shown in Figure 7.7, Herald-like and AI-MT-like mapper works rather well across four different tasks. The result form AT-MT-like shows that even though it is designed considering vision and language tasks, it also work adequately well in recommendation and even mix task. Likewise, Herald-like is designed for vision task and work well when applying to others. For optimization methods, they can reach similar performance as Herald-like and AI-MT-like. Note that, these optimization methods are not originally

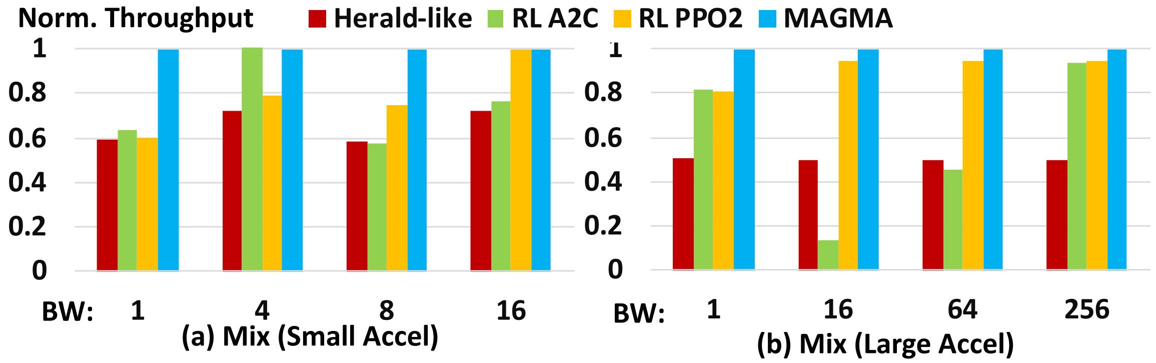
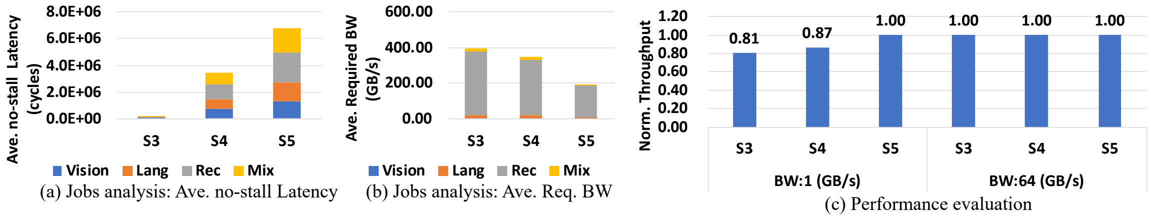


Fig. 7.9: Performance comparisons on multi-core heterogeneous (a) small (S2) and (b) large (S4) accelerator on Mix tasks, given different BWs. Throughput values are normalized by the value of MAGMA.



(a) Jobs analysis: Ave. no-stall Latency

(b) Jobs analysis: Ave. Req. BW

(c) Performance evaluation

Fig. 7.10: Jobs analysis of the averaged per-job (a) no-stall latency and (b) required BW. (c) Performance evaluation of MAGMA on S3, S4, and S5 with different BW. In (a-b), we concatenate the four independent no-stall latency (averaged required BWs) into a stacked bar and show the total values. In (c), throughput values are normalized by the value of S5.

designed for this specific mapping task. However, they work adequately well working in the M3E framework. Overall, MAGMA outperform others. MAGMA reach performance (geomean) 1.4x and 1.41x better than Herald and AI-MT, and (geomean) 1.6x better than other optimization methods.

7.3.5 Heterogeneous Accelerators

We examined the Small (S2) and Large (S4) heterogeneous accelerators across different tasks in Figure 7.8. In the following results, we will focus on presenting the result of Mix task, since it is a more complex task and is a fair realistic use-case in nowadays' inference data centers [**park2018deep**, **anderson2021first**]. In Figure 7.8, we also present Vision task result as baselines, since both Herald-like and AI-MT-like has Vision task as target.

Small Heterogeneous Accelerators (S2). As shown in Figure 7.8(a), Herald-like performs well, while AI-MT-like has comparatively lower performance. It shows the different characteristic of two algorithms. Herald-like is designed for heterogeneous accelerator while AI-MT-like is targeting homogeneous accelerator, which explains the performance difference. For optimization methods, many of them reach comparable performance to Herald-like, while PSO and CMA have lower performance (comparable to AI-MT-like). For a more complex Mix task (Figure 7.8(b)), AI-MT-like has comparatively worse performance. Many optimization methods undergo lower performance, too. However, the two RLs methods stands out. Overall, MAGMA outperforms others in both tasks. by geomean, MAGMA is 2.3x better than Herald, 39.5x better than AI-MT, 13.4X better than optimization methods excluding RLs. RLs and MAGMA have compatible performance and MAGMA achieve slightly better result 1.01x better.

Large Heterogeneous Accelerators (S4). In large accelerator, the mapping task becomes more complex since the design space of the mapping grows. As shown in Figure 7.8(c)(d), Herald-like perform rather well in Vision task. However, at a more complex Mix task and a more complex large accelerator case, Herald-like starts to undergo lower performance. Many more basic optimization process cannot tackle the large and complex design space as well (Figure 7.8(d)). However, RLs starts to shine and reach good performance. Overall, MAGMA outperform others in both tasks. by geomean, MAGMA is 1.7x better than Herald, 52x better than AI-MT, 10x better than optimization methods excluding RLs, and 1.3x better than RLs. Note that the contribution of this chapter is both the framework M3E (which enables the other optimization methods) and algorithm MAGMA. Before this chapter, the best performing mapper in Large heterogeneous accelerator setting is Herald-like, which harvests only 20% of maximum throughput enabled by M3E in Mix task, as shown in Figure 7.8(d).

BW-limited Environment. We examine the performance of Small accelerator at BW=16GB/s and Large accelerator at BW=256GB/s. However, in a heavy-loaded inference data center, the BW is a precious resource, where a big portion of it could also be occupied by other applications and leads to a more BW-limited environment. At a more BW-limited environment, mappers become crucial for smartly ordering the jobs to exploit the limited BW. We examine the effect of BW by a BW sweep in Figure 7.9. For both Small and Large accelerators, with the decrease of BWs, MAGMA stands out more obviously by reaching better relative performance. For example, in Figure 7.9(a), MAGMA is (geomean) 1.2x better than others when BW=16GB/s, but MAGMA is 1.6x better than other when BW=1GB/s. Similar observation can be made in Figure 7.9(b).

Sub-accelerator Combinations

In this experiment, we examine the performance change in different settings, S3 (Large Homogeneous), S4 (Large Heterogeneous), S5 (Large Heterogeneous, BigLittle) of the Large accelerator.

Homogeneous versus Heterogeneous. In the following experiment, we discuss the performance implication of a homogeneous versus heterogeneous accelerator using MAGMA algorithm. The LB-style sub-accelerators usually take larger runtime but lower BW requirements than HB-style in language and recommendation tasks, as shown in Figure 7.6(a). The jobs analysis in Figure 7.10(a-b) reflect the fact that S4, in general, induces more no-stall latency but requires less BW than S3. Therefore, when BW is limited (BW=1), the heterogeneous setting enables accelerator to leverage the difference of BW requirement among sub-accelerators to relax the BW contention. Thus S4 reaches better performance than S3 at BW=1 in Figure 7.10(c). However, when the BW is mostly sufficient (BW=256GB/S), the performance will reflect more of the behavior of the no-stall latency. Thus S3 reaches better performance.

Bigs versus BigLittle. We consider an accelerator with a smaller setting, BigLittle (S5), comparing to Bigs (S3, S4). It is obvious when the BW budget is sufficient (BW=256GB/S), BigLittle will perform worse than both of the Bigs (S3, S4) as shown in Figure 7.10(c), and can be verified by the jobs analysis in Figure 7.10(a). However, BigLittle has smaller

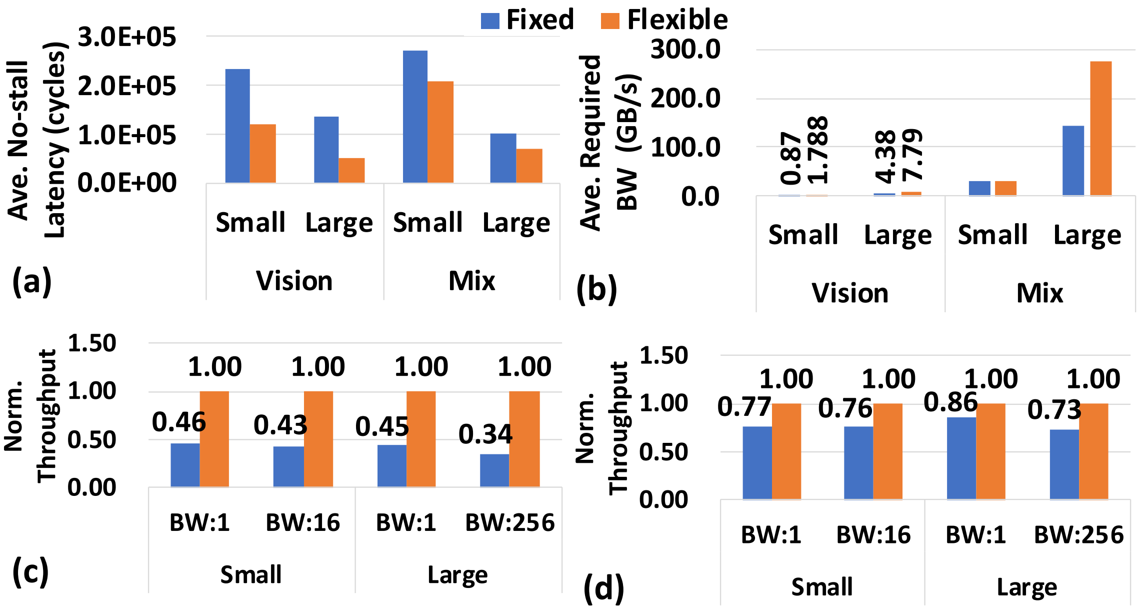


Fig. 7.11: Jobs analysis of the averaged (a) per-job no-stall latency and (b) required BW of fixed and flexible PEs arrays. Performance evaluation of MAGMA with fixed or flexible PEs array on (c) Vision and (d) Mix. Throughput values are normalized by the value of flexible accelerator.

BW requirement because of its smaller sub-accelerator size, as shown in Figure 7.10(b). Therefore, as shown in Figure 7.10(c), when the BW is limited (BW=1), BigLittle (S5) with the least amount of resources reaches the best performance. This observation shows that in a multi-core heterogeneous system, in addition to making the compute cores more powerful (adding more compute resources to them), striking the balance between each cores is another key consideration for boosting the performance.

7.3.6 Flexible Accelerator

In this experiment, we consider accelerators where the PE array dimensions are configurable, such as FPGAs [240], CGRA [2], or programmable accelerators [241, 242, 243], and demonstrate their performance by applying mapping found by MAGMA.

Accelerator Configuration. We extend the setting of S1 (Small, fixed) and S3 (Large, fixed) to have flexible accelerators. The number of PEs in the sub-accelerator are fixed (the same as in Table 7.2). However, the *shape* of 2D PE arrays is flexible, that is we

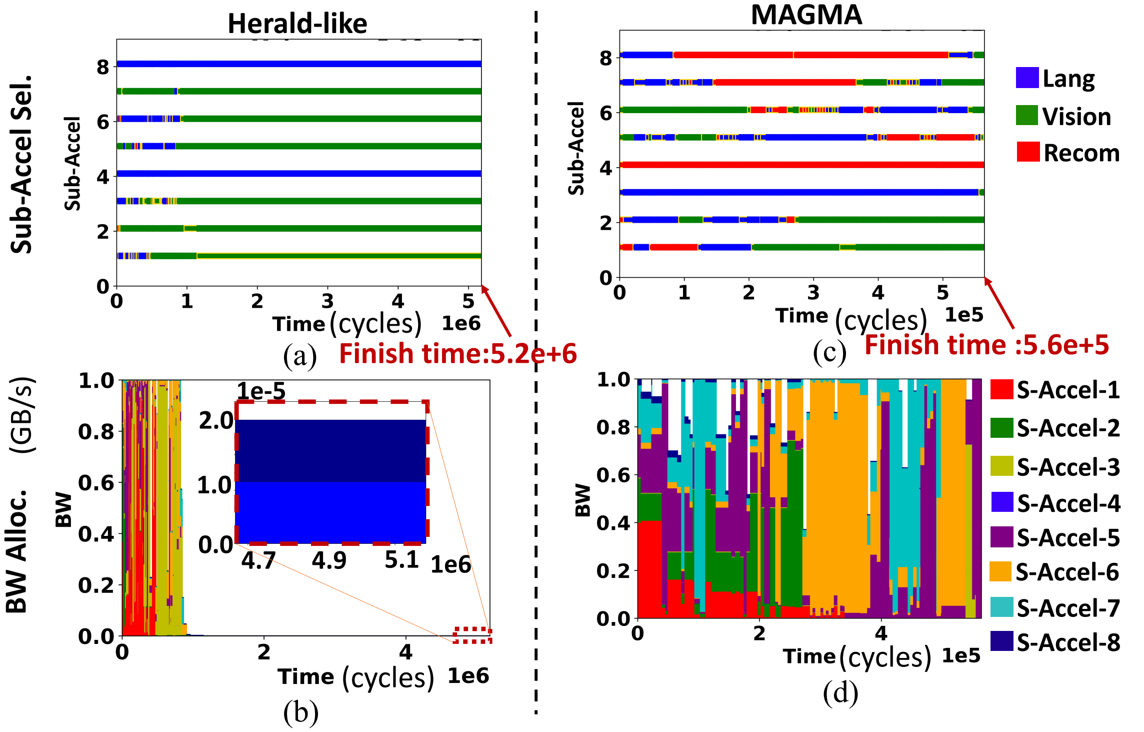


Fig. 7.12: The visualization of found solution by Herald-like and MAGMA. (a)(c) shows the respective sub-accelerator allocations, and (b)(d) shows the respective BW allocations. (Mix task, S5, BW=1).

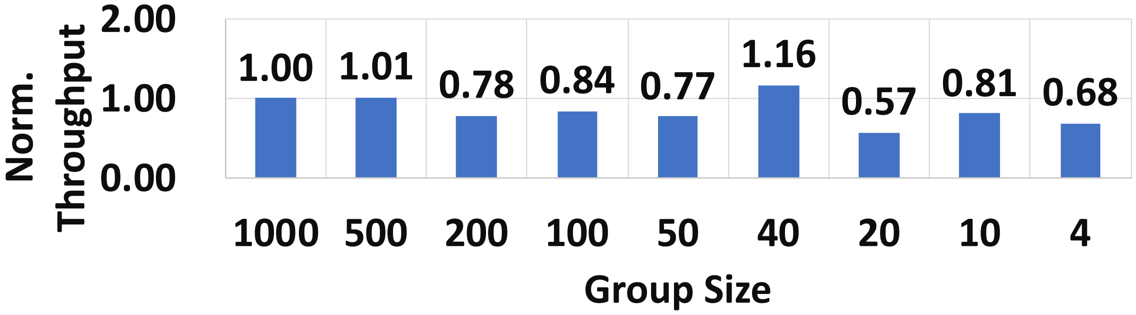


Fig. 7.13: The reached performance of MAGMA given the same task and setting (Mix, S2, BW=16) with different group sizes. Throughput values are normalized by the value of group size=1000,

Table 7.4: The performance of warm-start on (a) Mix, S4, BW=1. (b) The averaged performance across different tasks and different accelerator (S1-S6) under BW=1. All the values are normalized by the values of Trf-100-ep of each columns. Raw (highlighted in orange) is the throughput without warm-start. Trf-0-ep (highlighted in green) is warm-start and before further optimization. Trf-1-ep is warm-start with one epoch of optimization, and likewise for Trf-30-ep. Trf-100-ep (highlighted in blue) represents a full optimization process.

Mix-S4 BW=1	**Insts0 (Optim -ized)**	**Insts1 (Warm -start)**	**Insts2 (Warm -start)**	**Insts3 (Warm -start)**	**Insts4 (Warm -start)**	**Ave. (Warm -start)**
Raw	0.02	0.04	0.02	0.09	0.05	0.03
Trf-0-ep	1.00	0.32	0.60	0.78	0.58	0.51
Trf-1-ep	1.00	0.43	0.73	0.96	0.88	0.68
Trf-30-ep	1.00	0.99	0.98	0.99	0.99	0.99
Trf-100-ep	1.00	1.00	1.00	1.00	1.00	1.00

(a) Perf. on Mix, S4, BW=1

Averaged across S1-S6				
BW=1	**Mix**	**Vision**	**Lang**	**Rec**
Raw	0.02	0.04	0.014	0.004
Trf-0-ep	0.48	0.28	0.52	0.88
Trf-1-ep	0.67	0.40	0.79	0.95
Trf-30-ep	0.97	0.93	0.97	0.99
Trf-100-ep	1.00	1.00	1.00	1.00

(b) Ave. perf, across S1-S6

can configure the routing among the PEs. This enables the sub-accelerator to run various dataflows or mappings [2]. The maximum size of SLs are fixed as 1KB in each PE, and SGs are fixed as 2MB in each sub-accelerator.

Dataflow Strategy. We pick the dataflow strategy of the sub-accelerator to maximize the utilization of the PEs array. In order to maximize the utilization, we will align the PEs array dimension to be the factor of the the parallelizing dimension of the tile as much as possible. For example if the parallelizing dimension of the tile is (2, 15), which is going to map over the y and x dimension of the PEs array with 16 PEs. The potential PE array shape could be 2×8 while aligning to the factor of y dimension, or 3×5, 5×3, and 1×15 while aligning to the factor of x dimension. We examine these combinations, evaluate their expected latency by the HW cost model, and pick the lowest latency one as our PE array configurations.

Evaluations. From the performance analysis in Figure 7.11(a-b), we can observe that for both Vision and Mix tasks, *flexible* outperforms *fixed* in ave. per-job no-stall latency, owing to its ability to maximizing the utilization rate of the PEs array. However, it would also incur higher BW requirement. It is because the flexible mapping we found is to maximize the PE utilization rate, which also increases the number of data to fetch per tile to keep PEs busy.

From all scenario in Figure 7.11(c-d), *flexible* outperforms *fixed*. The results conclude that with flexible accelerators (ASIC of FPGA), we could further increase the accelerator performance without providing additional compute HW resources (PEs) if the accelerators (or sub-accelerators) have configurable PEs array shape.

7.3.7 More about MAGMA Algorithm

Analysis of found solutions. To understand the effect of different mapping, we show the detailed sub-accelerator selection and the corresponding BW allocation results of mapping found by two of the mappers, Herald-like and MAGMA. We showcase what is actually happening on the accelerator in an execution duration of one group of jobs in Figure 7.12. We found that MAGMA can distribute the BW-intensive jobs (Recommendation, Language) across the runtime to balance the BW requirement (Figure 7.12(c-d)). In contrast, Herald-like (Figure 7.12(a-b)) tries to use BW intensively at the beginning, which causes BW competition. Finally, it causes longer finish time of a group of jobs comparing to MAGMA.

Warm-start of MAGMA. In the following, we show the usefulness of warm-start technique in MAGMA. In the experiments, we use MAGMA to optimize on a group of jobs, Insts0. Then, we test, and optimize on the other four different group of jobs. Table 7.4(a) shows that by directly applying previous knowledge (Trf-0-ep), we could achieve **16x better performance** than the usual starting points, randomly initialization (Raw). By warm-start followed by one epoch/ step of optimization (Trf-1-ep), we could already receive 93% of the expected performance gain of a full optimization (Trf-100-ep). We execute the same experiment for different types of tasks and for different setting (S1-S6) (Table 7.4(b)). We can observe for BW-intensive tasks, Language and Recommendation, the previous knowledge become more important, and therefore the performance gain from the warm-start become significant. Overall, by warm-start and before further optimization is run (Trf-0-ep), MAGMA can achieve **7.4x** to **152x** better performance than the the usual starting points

(Raws).

Ablation Study of Group Size. Throughout the evaluation, we use the benchmark with a set group size of 100, It establishes a fair comparisons of the performance of different mappers. Also, in practice, group size is often a pre-defined system parameter as the formulated benchmark. However, a larger (or smaller) group size is also valid. We execute a group size sweep in Figure 7.13 using MAGMA algorithm. It tells that increasing or decreasing the group size does not affect the overall performance drastically. However, a too small group size (e.g., 4) will lead to lower performance.

7.4 Related Works

Mapping DNN Jobs on Single Accelerator. Several mappers have been proposed for the problem of mapping a single DNN layer efficiently on an accelerator. These include manual-designed mapping search [17, 12], heuristic-based mapping search [4, 23, 21, 18] and optimization/ML methods [189, 5, 6, 7, 16]. These works fall within the local mapping phase within individual accelerator cores.

Multi-tenant Mapping for DNN Accelerators. Prophet [244] builds a runtime prediction model for multi-tenancy on GPU. AI-MT [194] develops a heuristic for DNN job mapping for multi-PE arrays. Prema [197] explores preemptive multi-tenancy on a NPU. Herald [196] and Planaria [129] use manual-designed mapping for assigning jobs to sub-accelerators or reconfigurable PEs array. SCARL [245] utilizes RL for the mapping problem. In this work, we target multi-DNNs mapping and compared MAGMA against prior arts [194, 196], black-box optimizations [116, 102, 108, 114, 117], and RLs [198, 199].

Multi-tenant Scheduling for CPUs and GPUs. Multi-tenancy has been investigated for decades for multi-tasking on a single CPU and job ordering in CPU clusters [246, 247] or in GPUs [248, 249]. GAs [150, 151, 152, 153, 154], PSO [250], CMA-ES [251], and other optimizations have also been used. Some works leverage RL for jobs ordering over clusters such as DeepRM [252], Decima [253] and Thamsen *et al.* [254]. However, they presume a unified abstraction of the underlying cluster, where heterogeneity of the system is not considered.

7.5 Summary

This chapter presents a mapping optimizer for multi-tenant DNN accelerators. The key takeaways are as follows. (i) Heuristic and optimization methods have been used successfully for the design space of local-mapping (i.e., dataflow design). However, global-mapping forms a new drastically different search space. A new mapper for global mapping is needed for upcoming platforms. (ii) The search space for this (global-) mapping is extremely enormous. The search sample-efficiency of baseline optimization methods is not sufficient to find optimized solutions. (iii) We develop an optimization algorithm called MAGMA that customizes its exploration momentum and mechanism (genetic operators in this chapter) for the target search space and outperform the existing related works and other well-known optimization methods.

CHAPTER 8
CONFUCIUX: A RL-BASED HW RESOURCE ALLOCATION ALGORITHM FOR DNN ACCELERATORS

So far we base our discussion from the stand points of map space exploration. In chapter 3, we discuss how the accelerator's HW flexibility support affects the size of map size and its performance on different DNN workloads. Then in chapter 4-7, we discuss given a fully or partially flexible accelerator how we efficiently search for the best mapping for it. In this chapter, we will discuss the DNN accelerator design from the stand point of HW resource Design space Exploration (DSE). That is, in this chapter, we pre-defined the flexibility of the accelerator either it is completely inflexible or with limited flexibility. Then, we discuss how different HW resource allocation (with the pre-defined flexibility) impact the accelerator performance. In chapter 3-7, we are using the perspective of DNN compilers to infer the accelerator performance across DNN workloads, which helps DNN architect to make different design decision. In this chapter, we will use the perspective of DNN architect to show how the architect can leverage the learning from chapter 3-7, develop a DSE framework for HW resource allocation task, and improve the efficiency of its searching algorithm.

Given a set of supported mappings/ dataflows (because of the pre-defined HW flexibility), the assignment of HW resources is the next crucial part of the DNN accelerator design process. In fact, for the same dataflow, different choices for HW resources can lead to drastically different latency and energy for a given DNN. Some recent studies have shown that HW resource assignment plays a more important role in determining the accelerators' performance than its dataflow [60]. However, determining the policy for assigning HW resources is still very much an open problem, with prior works on HW Design-Space Exploration almost exclusively relying on exhaustive searches [17, 61, 62, 63, 64, 65, 66, 67, 68, 18].

The focus of this chapter is on the aforementioned HW resource assignment problem. The HW resource assignment depends on how they will be used by the DNN during runtime. We consider two deployment scenarios in this chapter. Layer Sequential (LS) involves mapping and running the DNN layer by layer on the accelerator, while Layer Pipelined (LP) maps and runs the entire DNN model over the accelerator. The LS approach is typically leveraged in cloud settings for larger models [142] that do not fit on-chip, while the LP approach is popular when running smaller optimized models [126, 141, 166] on IoT devices.

The HW resource assignment problem is an optimization problem where the design goal is to achieve an objective such as minimum end-to-end latency or energy, while meeting some platform (IoT/Cloud) constraints such as maximum power or chip area. The design-space of valid solutions is non-trivial. Consider an LP deployment; suppose we have a total of P PEs and B buffers that fit within the area/power budget and need to be divided among N layers of the DNN. Assuming each layer gets at least one PE and one buffer, the number of combinations for PEs and buffers is $\binom{P-1}{N}$ and $\binom{B-1}{N}$ respectively [255]. This makes the total possible design choices $\binom{P-1}{N} \times \binom{B-1}{N}$, which is $O(10^{72})$ for an accelerator with 128 PEs, 128 buffers running the 52-layer MobileNet-V2. This design-space is nearly impossible to enumerate to search exhaustively for an optimum solution, as we discuss in.

8.1 Methodology

In this chapter, we cast the DNN accelerator resource assignment DSE challenge as a reinforcement learning (RL) problem using REINFORCE [256] for a global search, followed by a GA for local fine-tuning. Figure 8.1 demonstrates the workflow of ConfuciuX. We provide a high-level overview next.

The inputs to ConfuciuX are the target DNN model, the deployment scenario, the optimizing objective, and the platform constraints, and it outputs an optimized HW resource assignment strategy, as shown in Figure 8.1. The first stage of ConfuciuX trains a RL

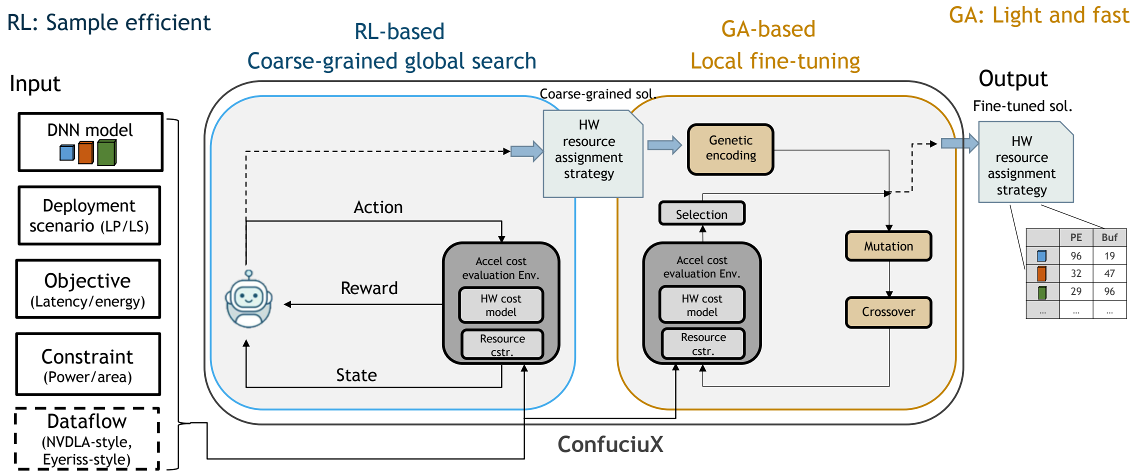


Fig. 8.1: Overview of ConfuciuX.

agent to recommend an optimized assignment of PEs and buffers for a target DNN, platform constraint, and deployment scenario. The agent is trained by having it continuously generate resource assignments as "actions" which are evaluated by a detailed but fast analytical model for DNN accelerators called MAESTRO [58] that acts as the environment (Env). The "rewards" output by the environment are used to train the underlying policy network, with the aim of maximizing the reward.

We incorporate platform constraints (area/power) as inputs to the environment to punish actions that violate the constraints. To speed up the search process, the RL agent searches through the HW assignments in a coarse-grained manner. Once it converges to an optimized strategy, we fine-tune the assignment further using GA. We discuss the various components of ConfuciuX next.

8.1.1 DNN Model Deployment on Accelerators

We focus on two DNN model deployment scenarios illustrated in Figure 8.2.

Layer Sequential (LS). We use the same underlying architecture to run the DNN model layer-by-layer. The specific (PE, buffer) design-point is chosen at design-time by some heuristic such as one that performs the best for most layers of the target DNNs. Naturally, over-provisioning and under-utilization may happen for some of the layers during

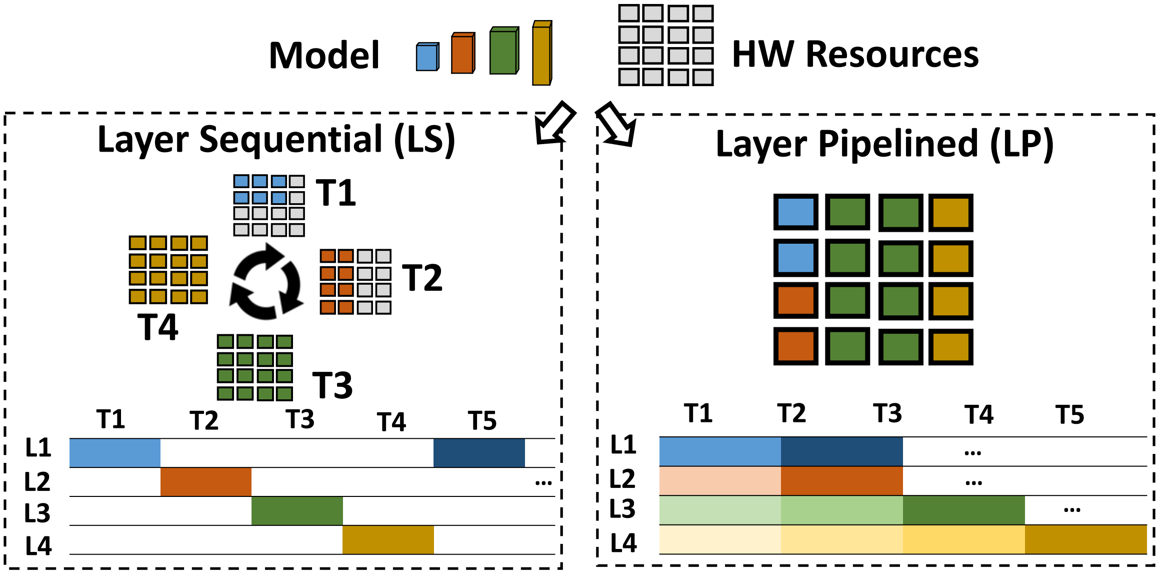


Fig. 8.2: DNN deployment scenarios and corresponding HW assignments. In Layer Sequential (LS), each layer of the model is mapped one by one on the entire accelerator, with all on-chip compute and memory assigned to it; in Layer Pipelined (LP), the entire model is mapped and run in a pipelined manner, with the compute and memory partitioned across all layers.

deployment since they favor different HW resource configuration [89, 257, 258, 259]. We quantify this further in §8.2.

Layer Pipelined (LP). With the advancement of technology, more computation logic can sit in a single chip. Many efficient models are being designed to fit completely onto the chip for embedded platforms [166, 126]. LP maps and runs the entire DNN model over the accelerator. For this model, we assume an underlying accelerator can heterogeneously partition the (PE, buffer) resources at either design-time (e.g., ASIC [241, 242, 243]) or compile-time (e.g., CGRA [2]/FPGA [18]). The challenge becomes finding the optimum (PE, buffer) distribution for each layer, which is crucial for maximizing performance [17, 61, 62, 63, 64, 65, 66, 67].

8.1.2 RL Agent

In our system, the RL agent processes the target DNN model in a layer-wise manner. We term the whole process (episode in RL parlance) as an epoch. We treat each layer as a different time-step. At each time-step, the agent makes two actions per-layer: the number of PEs and Buffers. It interacts with the environment to collect the rewards. We feed the rewards, along with the previous layer's actions to the policy function, to help the agent optimize sequential decisions. The policy network gets updated at the end of each epoch. An epoch terminates when the agent fails the constraint or successfully made $2N$ actions for a N-layer model.

Choice of RL Algorithm: REINFORCE. Modern RL algorithms [199, 260, 261, 262, 198, 263] typically use two underlying neural networks - an "actor" and a "critic". The actor formulates the policy for taking actions while the critic approximates the value function that predicts expected reward to help the training of policy. We experimented with a suite of RL algorithms for ConfuciuX and found that REINFORCE [256] works best. We show these results in §8.2.3. REINFORCE only has an actor network (no critic), and updates its underlying policy network directly using rewards from the Env. Since the design space of HW resource assignments is extremely discrete and irregular, we observed that RL algorithms with critic networks fail to approximate the value function accurately and in turn disturb the policy learning process. We show this later in §8.2.3.

Policy Network Architecture. The policy network in REINFORCE is a neural network tasked to learn the policy to maximize the probability of receiving better reward. We use an RNN as the policy network with one LSTM hidden layer of size 128. The reasoning behind an RNN-based network is as follows. We impose a hard constraint on the overall area (or power) consumption. Each action (PE, buffer) adds to area/power. Thus, any future action should depend on the previous action. The recurrent connections in the RNN capture this relationship and learn the constraint. We implemented and evaluated both RNN-based and MLP-based policy networks and provide a quantitative analysis in §8.2.7. We found RNN is essential for constraining the full trajectory of actions to conform to a global constraint.

8.1.3 Observation (State)

We construct a 10 dimensional observation space. At t^{th} time step, the observation (O_t) is expressed as follows

$$O_t = (K_t, C_t, Y_t, X_t, R_t, S_t, T_t, A_t^{PE}, A_t^{Buffer}, t) \tag{8.1}$$

The layer shape (assuming convolutions) results in the first 7 dimensions[1]. K_t and C_t are number of output and input channels. Y_t and X_t are the size of Y and X axis of the input activations. R_t and S_t are the size of Y and X axis of the weight kernel. T_t is the indicator of layer type such as CONV or DWCONV (Depth-wise CONV). A_t^{PE} and A_t^{Buffer} are the actions of the previous layer which we feed into the RNN controller. The last dimension t indicates t^{th} layer. Finally, we normalize all the dimensions of observation to the range of [-1, 1] to stabilize the training.

8.1.4 Action Space

At each time step, the agent makes an action pair (PE, Buffer), which formulates the action space. To efficiently step through the huge design space, the RL agent uses coarse-grained steps to navigate through it. In particular, we use $L = 12$ different values for the PEs and Buffers, as shown in Table 8.1. We demonstrate the effect of L later in §8.2.7. The specific values for PE at each level are chosen by the marginal observed return of HW performance to the number of PEs. For example, increasing PE from 1 to 2 could potentially double the HW performance, while increasing PE from 64 to 65 would provide slight or mostly no improvement. We choose the Buffer size value at each level according to the input of dataflow-style at design time. In NVDLA-style, with 3×3 weight as an example, we dispatch the computation to each PE along K dimension (Figure 8.3). Each PE would receive k number of 3×3 weights, 3×3 corresponding inputs, and generate k number of

[1] for other layers like MLP/GEMM, we use three dimensions (M,N,K) to describe the (M,K), (K,N) and (M,N) matrices.

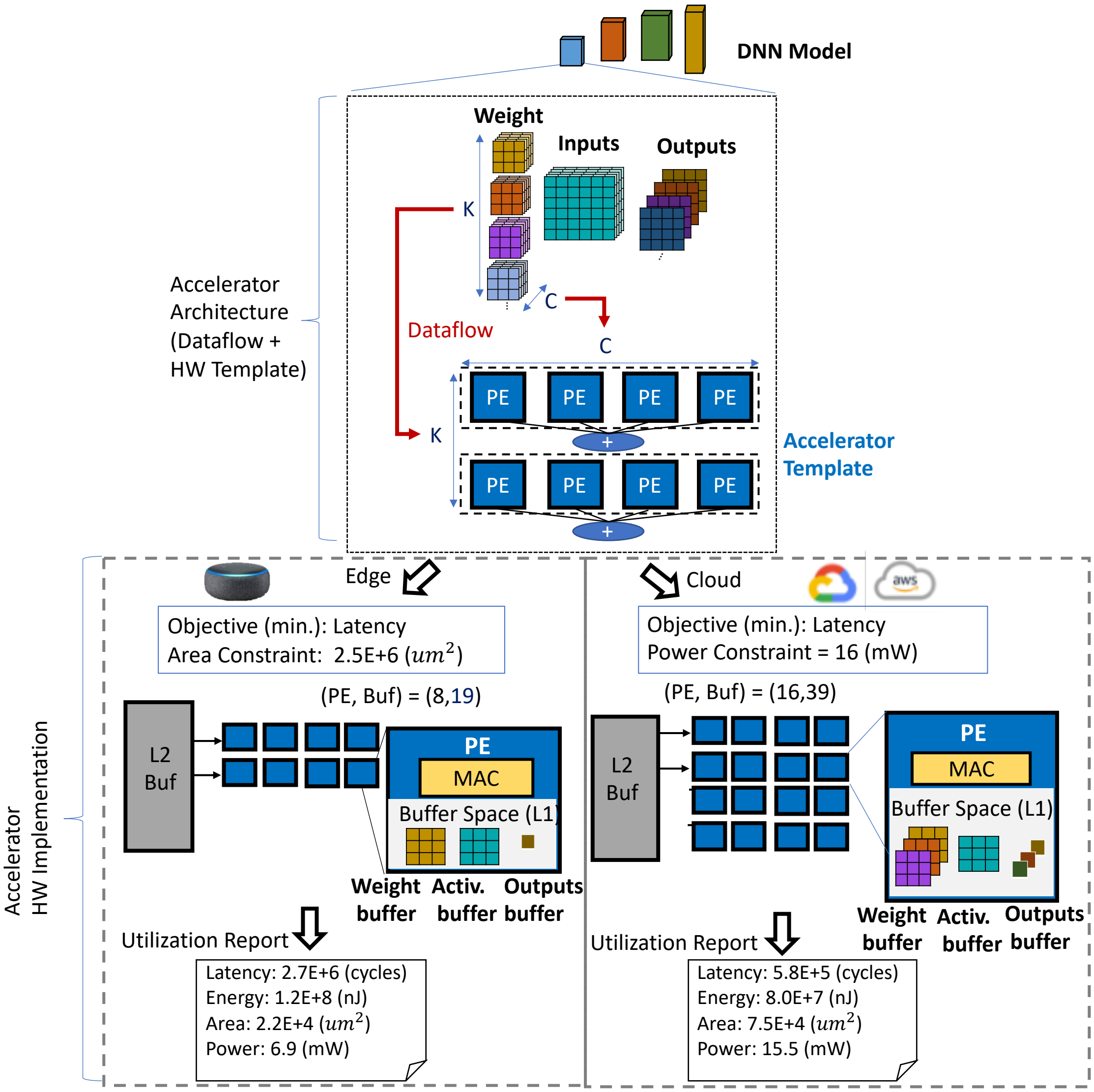


Fig. 8.3: Different HW resource combinations with the same NVDLA-style dataflow.

outputs, which makes the buffer size $9 \times k + 9 \times 1 + 1 \times k$, where k=1, 2,..., 12, as shown in Table 8.1. Note that once the RL agent converges, ConfuciuX uses fine-grained steps using GA to get to an optimized configuration, as described later in §8.1.8.

8.1.5 Platform Constraint and Objective

Each action pair (PEs, Buffers) defines the per-layer power/area constraint consumption and the per-layer energy/latency cost, which are our optimization targets. The goal of the accelerator design process is to optimize the cost for running the entire model, while meeting

Table 8.1: The level values of action pair.

	Action level values
PEs	1, 2, 4, 8, 12, 16, 24, 32, 48, 64, 96, 128
Buffers (e.g., NVDLA-style)	19, 29, 39, 49, 59, 69, 79, 89, 99, 109, 119, 129

the platform constraint.

Constraint. The accelerator is constrained by the budget of the targeted platform. We consider two categories of constraints: power and chip area. We have full flexibility to design architecture such as assigning a different number of PEs and Buffers or changing dataflow-style, as long as the design meets the constraint. In this chapter, we evaluate power and area constraints across cloud and IoT platforms, as described in §7.2.

Objective. We evaluate two design objectives in this chapter: minimum overall latency and minimum overall energy cost when the entire model is run on the accelerator, either via LS or LP. Other objectives can also be considered (say EDP or Power/Area for instance). While approaching the objective, the design should always fit the platform constraint. Minimizing the latency and energy is a non-trivial task since their dependence on the number of PEs and Buffers is not straight-forward. For e.g., increasing PEs would increase the level of parallelism; however, it would also increase the number of fetched data, which could potentially increase the latency. As for energy, increasing PEs and Buffers would increase the power; however, it could potentially decrease the energy because of the shorter execution time.

Co-optimization of layers. The RL agent is trained to be aware of the platform constraints. The agent should learn to optimize the resource assignment for each layer and the allocation of the constraint budget at hand to each layer simultaneously. We use RNN as the backbone of RL agent to enable it to memorize its entire epoch of decisions so that it could be aware of the consumption of the total budget. The Env checks the budget that is still left (L^{budget}) at every time step, and penalizes the RL agent once it is violated.

8.1.6 Reward Function

Reward. Since we are executing in a sparse reward domain, where the performance is only given at the end of the episode, we train the agent with a temporal layer-wise performance feedback for reward shaping. The sum of the layer-wise performance does not directly indicate the final entire model performance, which is our objective. However, it guides the RL agents.

We construct the reward function R as follows.

$$R = \begin{cases} P_t - P^{min}, \text{ if } L^{budget} \geq 0 \\ Penalty, \text{ otherwise} \end{cases} \tag{8.2}$$

P_t is the HW performance[2] of the current layer. P^{min} is the current lowest layer-wise performance across all time-steps and all epochs. This is tracked during the training process.

We find that the P^{min} term stabilizes the training. The insight behind it is as follows. First, as shown in Figure 8.3, the reward value for HW performance, such as number of cycles, can be extremely large, which can make the relative improvement seem insignificant across epochs. Thus, keeping a P^{min} across all epochs emphasizes the relative difference. Second, the term P^{min} makes the reward always positive while the platform constraint is not violated, which makes the RL agent easier to learn from positive reward and negative penalty.

Penalty. We penalize the RL agent when the resource constraint is violated. To teach the RL agent to forbid the failing point with reasonable penalty, we accumulate all the rewards experiences in this episode, and use negative of the accumulated value as a penalty. The reason is that the range of reward for different HW performance (latency, energy) can have an order of magnitude difference. Therefore, a threshold-based constant penalty [264, 265, 266], which is usually applied, is not feasible. Also, we need the penalty that is at the correct

[2] We use the term performance for generality. It could be latency, or energy, or any other objective we are minimizing.

scale so that it is large enough to penalize the agent and small enough to not deviate the learned policy too much once bad decision is made.

At the end of the episode, we normalize rewards in each time step to standard distribution and use the standardized reward to train the agent. We also apply a discount factor (d). We empirically found $d = 0.9$ is a generic good default value for this problem.

8.1.7 Interactive Environment (Env)

Structure. The Env is initialized with the target model(s), dataflow, platform constraint, and the optimizing objective (latency/energy). Env tracks the consumed constraints of each time step and the P^{min} across all episodes.

HW performance estimator (eval). We use MAESTRO [58], an open-source DNN accelerator microarchitectural model, to determine the performance of each accelerator design-point during the training process. MAESTRO takes the DNN model, dataflow-style, and HW configuration as an input. Internally, it estimates all possible reuse opportunities for the given dataflow and HW resources, and estimates statistics such as latency, energy, runtime, power, and area. MAESTRO's HW model assumes a spatial DNN accelerator with PEs, L1 buffers, a shared L2 buffer, and an NoC between the buffers. It can support any dataflow (specified via a data-centric DSL [68]). The number of PEs is an input parameter, while the L1 and L2 buffer sizes are estimated based on the tile-sizes for the dataflow. It supports both layer-wise and model-wise evaluation.

8.1.8 Local fine-tuning using GA

We use a two-stage optimization to search for a fine-grained solution, as shown in Figure 8.1. The first and major part is the RL based coarse-grained global search. The second is the Genetic Algorithm (GA) based fine-grain local search. We use two-stage optimization for efficiency, since increasing the level of actions, L by 1 would increase the design space by $(\frac{L+1}{L})^{2N}$.

RL shows higher sample efficiency and converges to better optimum point comparing to other optimization methods, as shown later in §8.2. GA is simple and fast, but converging to less optimum value comparing to RL or sometimes cannot converge. According to the observation of the behavior of GA, it sometimes fails to learn the constraint and optimize the objective simultaneously, leading to a great portion of populations actually violating the constraint, which pollute the genomes of the future generation. However, if we start GA with a good initialization and mutate/crossover genes carefully, which decreases the complexity of the problem, GA could reach good result. Therefore, GA becomes a good candidate as a second stage fine-tuning if we initialize it with the first-stage solution. Even though a continuous RL algorithm [260, 261, 262] could be another candidate for the second stage, we find that the problem complexity of the second stage is simple that GA is adequate to tackle it. The details of the GA algorithm are described next.

Initialization. Assuming a DNN model with N layers, a design-point would include N actions for PEs and N actions for Buffers. We encode this design-point into a genome with $2N$ genes, where a gene represents an action for PE or Buffers. We initialize the first population with the genome formulated by the solution from the first (RL) stage.

Local mutation. We mutate the gene locally. We only mutate the gene by a step difference of the current value. For e.g., for a gene representing PE=64, we could mutate it to value in the range of [60, 68] when the step is 4. This conservative mutation can reduce the number of invalid genomes, which does not conform to the constraint, and assure we have good portion of valid parents to reproduce.

Local crossover. The crossover of two genomes is unlikely to conform to the constraint, since it can break the learnt relationship between HW resource assignment of each layer. For e.g., suppose we have parents A and B, both with good fitness and lie within the constraint. However, A tends to assign more HW resources on early layers and B tends to assign more HW resources on late layer. When we blend their genes for the next generation, the platform constraints might get violated by some children: a child with early genes from A and late

Table 8.2: Platform constraint settings.

Platform Constraint	Descriptions
Unlimited	No constraint. Since we set each action to 12-level, we measured the maximum constraint (power/area) consumption, $C_{power/area}^{max}$, by evaluating entire model with uniform action pair (p_{12th}, b_{12th}).
Cloud	Loose constraint. We set the constraint at *50%* $C_{power/area}^{max}$.
IoT	Tight constraint. We set the constraint at *10%* $C_{power/area}^{max}$.
Extreme small IoT (IoTx)	Extremely tight constraint. We set the constraint at *5%* $C_{power/area}^{max}$.

genes from B may over-request HW resources for every layer, violating the constraints. Alternately, a child with early genes from B and late genes from A may under-request HW resources for each layer, leading to less performance. Thus, we crossover the genome locally within a parent by exchanging genes for (PE, Buffers) between two layers of a model. In other words, we pick two pairs of genes representing the (PE, Buffers) of two layers of a models and swap them. This conservative self-crossover preserves most of the learnt relationship between layer and resources and adds an exploration effect.

8.2 Evaluations of ConfuciuX

8.2.1 Methodology

DNN Models

In our evaluations, we consider three CNN models with different complexity: MobileNet-V2 [126], MnasNet [141], and ResNet-50 [142]. We also evaluate three GEMM-based ML models: GNMT [267] for machine translation, transformer [268] for language understanding and NCF [269] for collaborative filtering.

Accelerator Platforms

We consider three different classes of platforms: Cloud server, IoT device and extremely small IoT, and, for comparison, an unconstrained platform as shown in Table 8.2. We consider three dataflows: NVDLA-style [80] (-dla) (parallelizing K and C dim.), Eyeriss-style [83] (-eye) (parallelizing Y and R dim.), ShiDianNao-style [81] (-shi)(parallelizing Y and X dim). We use L=12 levels of action values for PE and Buffers, where $(p_{n_{th}}, b_{k_{th}})$ represents assigning n_{th}-level of PEs and k_{th}-level of Buffers.

Baseline Optimization/Search Methods

We evaluate the following optimization methods as baselines.

Grid search. We enumerate through the design space with the stride of s in the L=12 level, (e.g., $(p_{1_{th}}, b_{1_{th}})$, $(p_{1_{th}}, b_{(1+s)_{th}})$...). We set maximum epochs Eps. We emulate through the design space until the number of sampling points reached Eps.

Random search. We randomly sample Eps design points and keep the best solutions as a result.

GA [102]. The baseline is a general GA algorithm, not the specially designed local fine-tuning one as described in §8.1.8. The GA is set with 100 population, and $\left\lceil \frac{Eps}{100} \right\rceil$ generations. The mutation rate and crossover rate is set as 0.05.

Simulated Annealing [99]. The simulated annealing is implemented with temperature of 10 with step size of 1 and adopted to discrete integer space.

Bayesian optimization [104]. We set the algorithm to run for Eps iterations, where Eps points are sampled by the algorithm. We adopt it to discrete integer space. We set the number of the optimizer to 5 for the Gaussian process, since we empirically find this setting has better performance.

State-of-the-art RL algorithms. We consider state-of-the-art RL algorithms that are successful in many control problems. We consider both continuous and discrete methods. We compare with **A2C** [198], **ACTKR** (Actor Critic using Kronecker-Factored Trust Region) [263], and **PPO2** [199]. Both continuous and discrete versions of the three algorithms are experimented. Across all the experiments, we found the discrete version converge to better value. Hence, we will only show the result of the discrete version in the comparisons

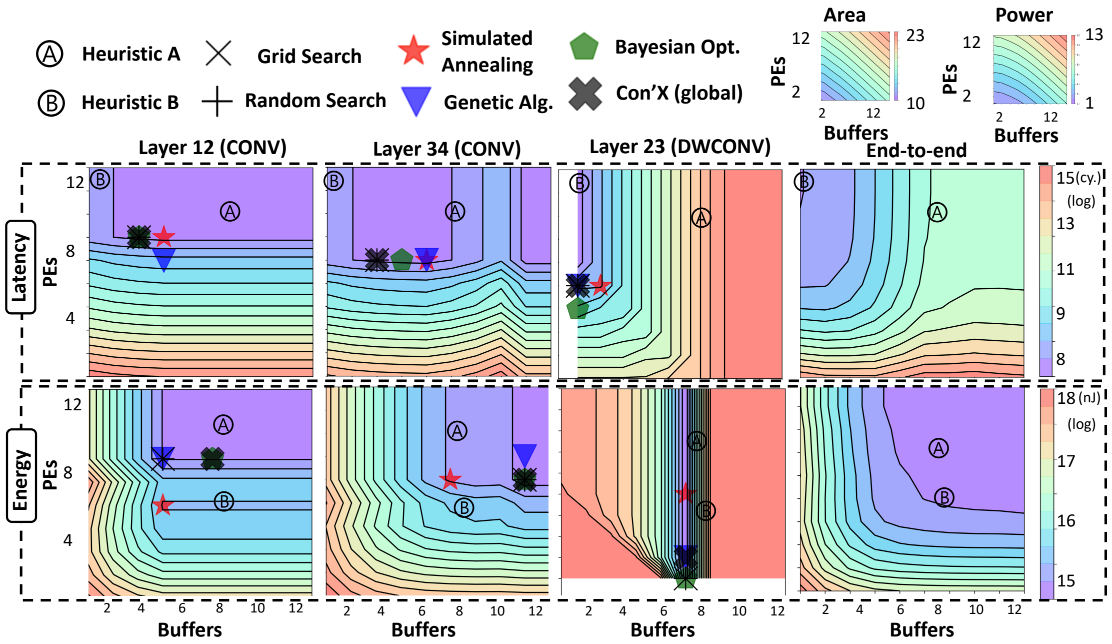


Fig. 8.4: Searching for per-layer PEs/Buffers configurations to optimize latency/energy with different techniques. Purple indicates lower (better) and red indicates higher (worse) latency/energy. **Heuristic A**: Determine the PEs/Buffers with the most compute-intensive layer (Layer-38) and apply the same configuration for all the layers. **Heuristic B**: Determine the PEs/Buffers by the configuration that optimizes end-to-end whole model latency/energy.

table. We also consider **DDPG** [260], **SAC** [262], and **TD3** [261] in continuous space. All comparisons run for Eps epochs.

ConfuciuX (Global)

We only consider the first-stage global search, Con'X (global), throughout the comparisons against baseline methods, for fairness. The second-stage fine-tuning can be added on top of the first-stage results. The benefit of the second-stage is explicitly discussed in §8.2.5.

8.2.2 Per-layer study for LS deployment

We start by showing the HW performance of different action pairs $(p_{n_{th}}, b_{k_{th}})$ with 12 level of values each, which is Y-axis and X-axis in Figure 8.4. We sweep through $(p_{n_{th}}, b_{k_{th}})$ with exhaustive search and color it with their corresponding latency/energy value. Red indicates large latency/energy values while purple indicates small ones. For each layer, the contour is drastically different. Each layer require distinct action pairs $(p_{n_{th}}, b_{k_{th}})$ to reach optimal values (purple). The contour becomes a flat region when the PEs or Buffers are over-provisioned. Latency of layer-12 as an example, when PE is larger than the 9_{th} level and Buffer is larger than the 3_{rd} level, the latency remains the same because of over-provisioning. The two separate purple region in latency of Layer-34 indicates that there are two region of tiling size, which we map to the buffer, can optimize the latency. For Layer-23 (DWCONV), increasing the tile size of the mapping dimension (K) does not help because of the irrelevance of each output channel (K) in DWCONV. As for energy, larger number of PEs and Buffers can potentially decrease the energy because of shorter execution time as in layer-12 and layer-34. We can observe there are sweet spot for buffer size in Layer-23, where all the channel is mapped to one PE. At this end, increasing PE would not increase the energy, since extra PE will be idle. Also decreasing Buffers cause more times of fetching, which increase the energy consumption.

Con'X consistently finds the optimal action pair for each layer, and its solution is as-good or better (fewer PEs and buffers for same latency or energy) than the baseline methods and two common heuristics. Figure 8.4 also shows that there is no action pair that suits all the layers. Thus, for a LS scenario, a designer can use Con'X to find optimal configurations for each layer, and then pick the one that provides optimum values across most layers.

8.2.3 LP Deployment

Next, we consider LP deployment (i.e., all layers of the model mapped on the accelerator) with platform constraints. For all the comparisons, we compare the algorithm performance by comparing their best solutions after $Eps = 5,000$ epochs.

Converged solutions across DNNs, Dataflows, and Platforms

We ran baseline optimization methods and RL algorithms for a suite of DNNs (CNN- and GEMM-based) with varying dataflow styles and platform constraints. The objective is set

Table 8.3: Converged solution of LP deployment.

Model	Obj. (min.): Latency Cstr.: Area	Optimization Results (cycles)		
		GA	PPO2	**Con'X (global)**
MbnetV2-dla	IoT	NAN	3.6E+07	**3.2E+07**
MbnetV2-eye	IoTx	NAN	4.1E+07	**3.7E+07**
MbnetV2-shi	IoTx	NAN	4.9E+07	**4.0E+07**
Mnasnet-dla	Cloud	**2.1E+07**	2.3E+07	**2.1E+07**
Mnasnet-eye	IoTx	NAN	4.1E+07	**3.9E+07**
Mnasnet-shi	IoTx	NAN	4.4E+07	**4.3E+07**
Resnet50-dla	Cloud	**2.2E+08**	2.4E+08	**2.2E+08**
Resnet50-eye	Cloud	2.3E+08	2.3E+08	**2.2E+08**
Resnet50-shi	Cloud	3.2E+08	3.4E+08	**3.1E+08**
GNMT-dla	IoTx	NAN	2.4E+08	**5.4E+07**
GNMT-eye	IoT	9.8E+07	1.4E+08	**9.8E+07**
GNMT-Shi	IoT	2.5E+10	**2.4E+10**	**2.4E+10**
Transformer-dla	IoTx	NAN	7.3E+08	**1.6E+06**
Transformer-eye	IoT	3.5E+06	6.6E+05	**1.9E+05**
Transformer-shi	IoT	7.3E+08	**7.2E+08**	**7.2E+08**
NCF-dla	IoTx	NAN	7.3E+07	**7.2E+07**
NCF-eye	Cloud	1.20E+06	1.4E+06	**1.1E+06**
NCF-shi	IoT	**6.6E+08**	**6.6E+08**	**6.6E+08**
Bold indicates the best results among GA, PPO2 and this work.				
NAN indicates that constraint (area) not met in Eps (5000) epochs.				

Table 8.4: Converged solutions after 5000 epochs for various optimization methods across four platforms with different constraints. DNN=MobileNet-V2, Dataflow=NVDLA-style, Deployment=LP

Objective (min.)	Constraint	Optimization Results (cycles)					
		Grid	Random	SA	GA	Bayes. Opt.	**Con'X (global)**
Latency	Area: Unlimited	5.3E+08	3.6E+07	6.2E+07	**2.1E+07**	3.7E+07	**2.1E+07**
Latency	Area: Cloud	5.3E+08	3.4E+07	9.6E+07	**2.1E+07**	3.7E+07	**2.1E+07**
Latency	Area: IoT	5.3E+08	7.1E+07	NAN	NAN	7.2E+07	**3.2E+07**
Latency	Area: IoTx	5.3E+08	NAN	NAN	NAN	NAN	**5.6E+07**
Latency	Power: Cloud	5.3E+08	3.6E+07	9.9E+07	**2.2E+07**	3.7E+07	**2.2E+07**
Latency	Power: IoT	5.3E+08	NAN	1.2E+08	NAN	1.6E+08	**3.7E+07**
Latency	Power: IoTx	5.3E+08	NAN	NAN	NAN	NAN	**5.6E+07**
Energy	Area: Unlimited	8.5E+09	1.8E+09	1.3E+09	**1.2E+09**	1.5E+09	**1.2E+09**
Energy	Area: Cloud	8.5E+09	1.8E+09	1.3E+09	**1.2E+09**	1.5E+09	**1.2E+09**
Energy	Area: IoT	8.8E+09	NAN	NAN	NAN	2.4E+09	**1.4E+09**
Energy	Area: IoTx	8.6E+09	NAN	NAN	NAN	NAN	**1.7E+09**
Energy	Power: Cloud	8.5E+09	1.8E+09	1.3E+09	**1.2E+09**	1.5E+09	**1.2E+09**
Energy	Power: IoT	8.6E+09	NAN	NAN	NAN	4.1E+09	**1.4E+09**
Energy	Power: IoTx	8.6E+09	NAN	NAN	NAN	NAN	**1.7E+09**
Results: Latency: (cycles), Energy: (nJ). **Bold** indicates the best results among algorithms.							
NAN indicates that constraint (area/power) not met in Eps (5000) epochs.							

to minimize the latency of the entire model. Therefore the lower the reached value, the better the solution is. In the interest of space, we show the results with the best performing baselines, GA and PPO2, in Table 8.3. GA can reach good optimized value when the constraint is loose (cloud), but it fails in some tight constraint cases (IoT, IoTx). Both PPO2 and Con’X(global) can find solutions in any type of constraint. Across all the experiments, Con’X(global) finds the solution with the same or better performance than PPO2 and GA.

Deep-dive with optimization methods

Table 8.4 compares the solutions attained by various optimization methods and Con’X(global) for MobileNet-V2 under four platform constraints for a NVDLA-style accelerator. The objective is set to minimize the latency or energy of the entire model. Random, SA, and GA fail to come up with a feasible solution when faced with tight constraint (IoT). Also,

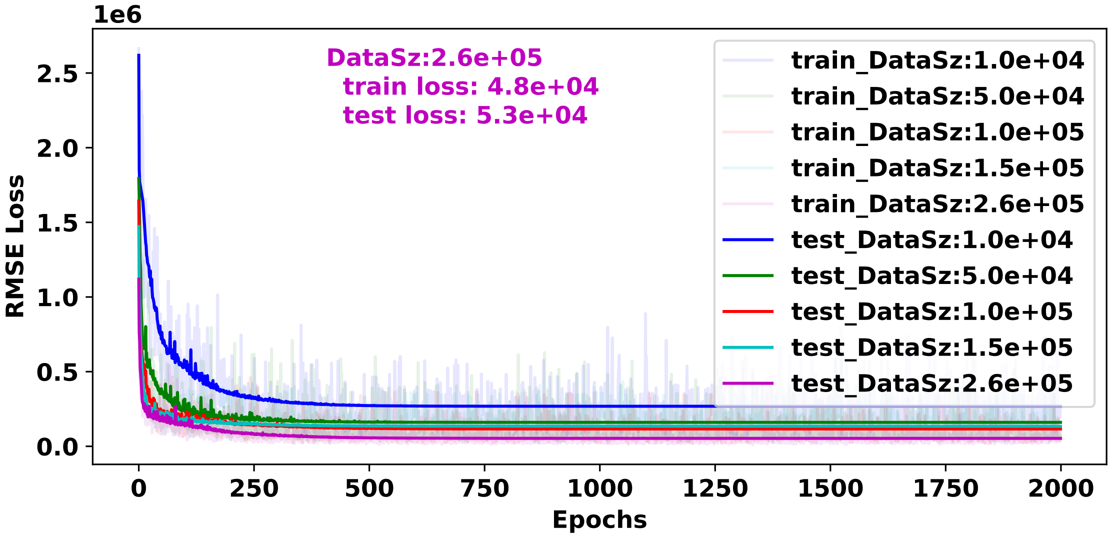


Fig. 8.5: The learning curve of the critic network.

Bayesian optimization fails in extreme tight constraint (IoTx). Con'X(global) successfully learns the constraint behavior and optimizes the objective together. Con'X(global) generates the most optimized design points with 86% lower latency and 70% lower energy, on average across baselines.

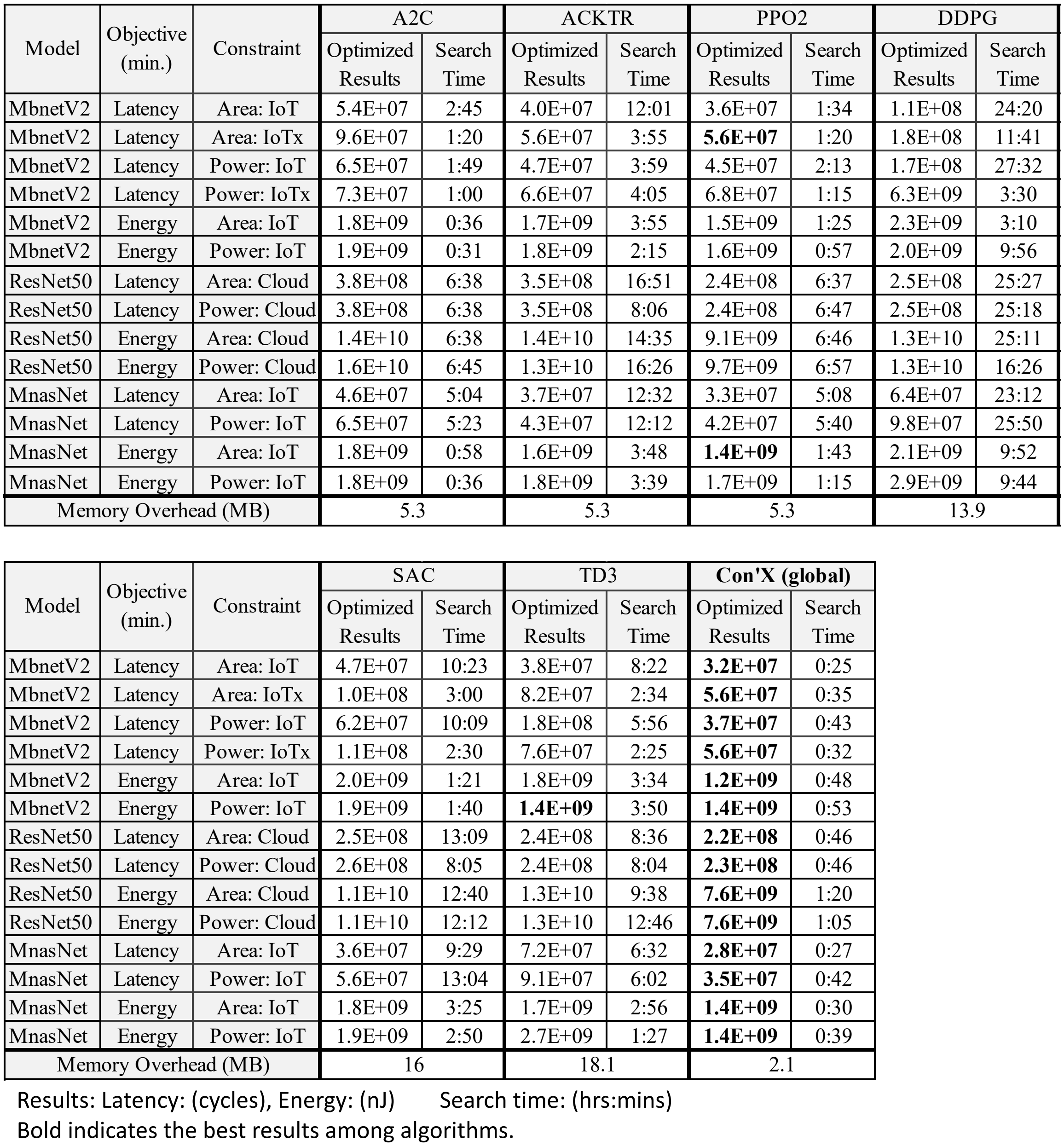
Deep-dive with RL algorithms

We compare Con'X(global) with other state-of-the-art RL algorithms in the same setting as the previous experiment, as shown in Table 8.5. All the RL agents are able to find feasible solutions in all situations. Considering the complexity of the algorithm, DDPG [260], SAC [262], and TD3 [261] generally consume more search time and memory overhead. Across all comparisons, we find Con'X(global) and PPO2 [199] reach better objective value. Con'X(global) converges to the optimized value 4.7 to 24 times faster than alternate RL algorithms.

Analysis of critic networks.

In many advanced RL algorithms such as A2C [198], ACTR [263], PPO2 [199], DDPG [260], SAC [262] and TD3 [261], critic networks are used to approximate the underlying value functions, which in turn train the policy network. The REINFORCE-based

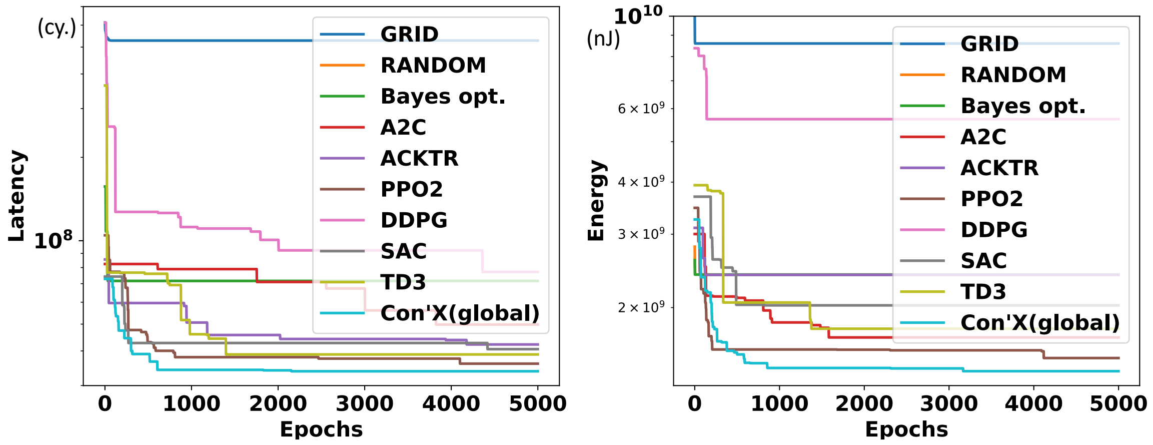
Table 8.5: Comparison of search-time and converged solutions across state-of-the-art RL techniques.



Model	Objective (min.)	Constraint	A2C		ACKTR		PPO2		DDPG	
			Optimized Results	Search Time	Optimized Results	Search Time	Optimized Results	Search Time	Optimized Results	Search Time
MbnetV2	Latency	Area: IoT	5.4E+07	2:45	4.0E+07	12:01	3.6E+07	1:34	1.1E+08	24:20
MbnetV2	Latency	Area: IoTx	9.6E+07	1:20	5.6E+07	3:55	**5.6E+07**	1:20	1.8E+08	11:41
MbnetV2	Latency	Power: IoT	6.5E+07	1:49	4.7E+07	3:59	4.5E+07	2:13	1.7E+08	27:32
MbnetV2	Latency	Power: IoTx	7.3E+07	1:00	6.6E+07	4:05	6.8E+07	1:15	6.3E+09	3:30
MbnetV2	Energy	Area: IoT	1.8E+09	0:36	1.7E+09	3:55	1.5E+09	1:25	2.3E+09	3:10
MbnetV2	Energy	Power: IoT	1.9E+09	0:31	1.8E+09	2:15	1.6E+09	0:57	2.0E+09	9:56
ResNet50	Latency	Area: Cloud	3.8E+08	6:38	3.5E+08	16:51	2.4E+08	6:37	2.5E+08	25:27
ResNet50	Latency	Power: Cloud	3.8E+08	6:38	3.5E+08	8:06	2.4E+08	6:47	2.5E+08	25:18
ResNet50	Energy	Area: Cloud	1.4E+10	6:38	1.4E+10	14:35	9.1E+09	6:46	1.3E+10	25:11
ResNet50	Energy	Power: Cloud	1.6E+10	6:45	1.3E+10	16:26	9.7E+09	6:57	1.3E+10	16:26
MnasNet	Latency	Area: IoT	4.6E+07	5:04	3.7E+07	12:32	3.3E+07	5:08	6.4E+07	23:12
MnasNet	Latency	Power: IoT	6.5E+07	5:23	4.3E+07	12:12	4.2E+07	5:40	9.8E+07	25:50
MnasNet	Energy	Area: IoT	1.8E+09	0:58	1.6E+09	3:48	**1.4E+09**	1:43	2.1E+09	9:52
MnasNet	Energy	Power: IoT	1.8E+09	0:36	1.8E+09	3:39	1.7E+09	1:15	2.9E+09	9:44
Memory Overhead (MB)			5.3		5.3		5.3		13.9	

Model	Objective (min.)	Constraint	SAC		TD3		**Con'X (global)**	
			Optimized Results	Search Time	Optimized Results	Search Time	Optimized Results	Search Time
MbnetV2	Latency	Area: IoT	4.7E+07	10:23	3.8E+07	8:22	**3.2E+07**	0:25
MbnetV2	Latency	Area: IoTx	1.0E+08	3:00	8.2E+07	2:34	**5.6E+07**	0:35
MbnetV2	Latency	Power: IoT	6.2E+07	10:09	1.8E+08	5:56	**3.7E+07**	0:43
MbnetV2	Latency	Power: IoTx	1.1E+08	2:30	7.6E+07	2:25	**5.6E+07**	0:32
MbnetV2	Energy	Area: IoT	2.0E+09	1:21	1.8E+09	3:34	**1.2E+09**	0:48
MbnetV2	Energy	Power: IoT	1.9E+09	1:40	**1.4E+09**	3:50	**1.4E+09**	0:53
ResNet50	Latency	Area: Cloud	2.5E+08	13:09	2.4E+08	8:36	**2.2E+08**	0:46
ResNet50	Latency	Power: Cloud	2.6E+08	8:05	2.4E+08	8:04	**2.3E+08**	0:46
ResNet50	Energy	Area: Cloud	1.1E+10	12:40	1.3E+10	9:38	**7.6E+09**	1:20
ResNet50	Energy	Power: Cloud	1.1E+10	12:12	1.3E+10	12:46	**7.6E+09**	1:05
MnasNet	Latency	Area: IoT	3.6E+07	9:29	7.2E+07	6:32	**2.8E+07**	0:27
MnasNet	Latency	Power: IoT	5.6E+07	13:04	9.1E+07	6:02	**3.5E+07**	0:42
MnasNet	Energy	Area: IoT	1.8E+09	3:25	1.7E+09	2:56	**1.4E+09**	0:30
MnasNet	Energy	Power: IoT	1.9E+09	2:50	2.7E+09	1:27	**1.4E+09**	0:39
Memory Overhead (MB)			16		18.1		2.1	

Results: Latency: (cycles), Energy: (nJ) Search time: (hrs:mins)
Bold indicates the best results among algorithms.



(a) Obj.(min.): Latency, Cstr.: IoT area

(b) Obj.(min.): Energy, Cstr.: IoT area

Fig. 8.6: The fast convergence and sample efficiency of Con'X (global).

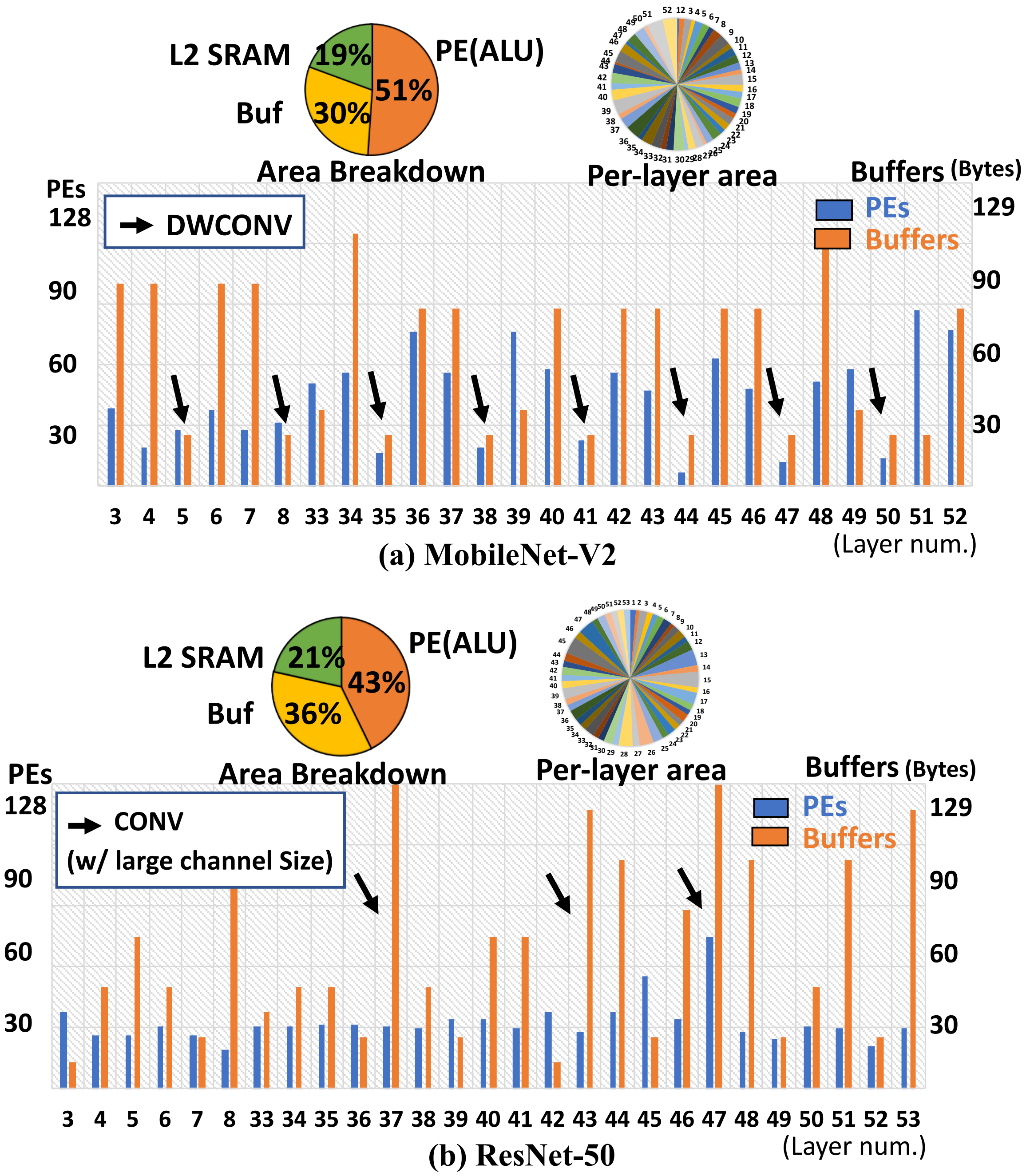
used in ConfuciuX, on the other hand, only has an actor network that learns directly from the reward. As Table 8.5 shows, we found that REINFORCE [256] in Con'X(global) converges to better solutions than all the actor-critic RL algorithms. Our intuition is that this is because the function of the HW performance of the accelerator are too discrete and irregular for a critic neural network to learn well, and this in turn adversely affects the learning of the policy networks. To verify this intuition, we extract the critic network from the implemented alternate RL algorithms [199, 260, 261, 262, 198, 263] and conduct a standalone experiment to test its ability to approximate the underlying value function. The task is to take the "state values" as input and predict the corresponding reward of that state. We use per-layer latency of MobileNet-V2 as reward. We use mean square error and gradient decent to train the network. We show the root mean square error (RMSE) when training with different size of data, as shown in Figure 8.5. 260,000 is the maximum possible data points critic network can experience under the RL tasks of $Eps = 5,000$ with MobileNet-V2. We can observe that the training and testing loss is hard to converge to a feasible value (the best RMSE is 5.3e+4, which means the predicted latency (reward) by critic network is in average 5.3e+4 cycles difference to the ground-truth ones) which means the critic network did not learn reward value well. This could potentially misguide the policy network.

Sample efficiency and convergence

In the experiments against baseline optimization methods and other RL algorithms, we found Con'X(global) has the fastest convergence rate. We show two convergence traces as examples in Figure 8.6 for MobileNet-V2. With rapid convergence, our method heads toward the objective with more sample efficiency. On the contrary, the exhaustive search needs to enumerate and search through L^{2N}, 12^{104}=$O(10^{112})$, data points for the search space of 52-layer MobileNet-V2 with two actions per layer and 12 level of values per action, which is near impossible to finish.

8.2.4 Dataflow-HW co-automation

We extend ConfuciuX to co-automate the per-layer dataflow style decision. Rather than manually picking one of the dataflow style, we let the agent make this decision. To take one step further, we let the agent do fine-grained per-layer dataflow style decision, which we termed as MIX-strategy. The agent now makes three decisions per-layer: PEs, Buffers, and dataflow style. We found Con'X-MIX can not only pick the best dataflow-style for a model but also take advantage of MIX-strategy to pick different dataflow-style in different layers, as shown in Figure 8.9 for MobileNet-V2.In general, if there are no HW resource constraints, system will favor eye/shi at early layers (larger activations), which parallelize along activations dimensions, and favor dla at late layers (larger K/C), which parallelizes along channel dimensions (K/C) in CNN-based networks. However, when considering HW constraint, it becomes a compound decision trading-off among PE, Buffers, dataflow-style, and area. From the experiment listed in Table 8.6, we can observe that in a more relaxed constraint, dla performs better than the other two since most layers in CNN-based networks have large K/C dim. However, in a tighter constraint, the parallelization ability of dla will be restricted; Eye/shi, which parallelize activations dim (whose values shrink layer-by-layer quickly in most CNNs) become more efficient choices. This observation can also explain the fact that system chooses eye/dla for some of the later layers in Figure 8.9. From the



(a) MobileNet-V2

(b) ResNet-50

Fig. 8.7: The solution for (a) MobileNet-V2 and (b) ResNet-50 (Obj.(min.):Latency, Cstr:IoT area).

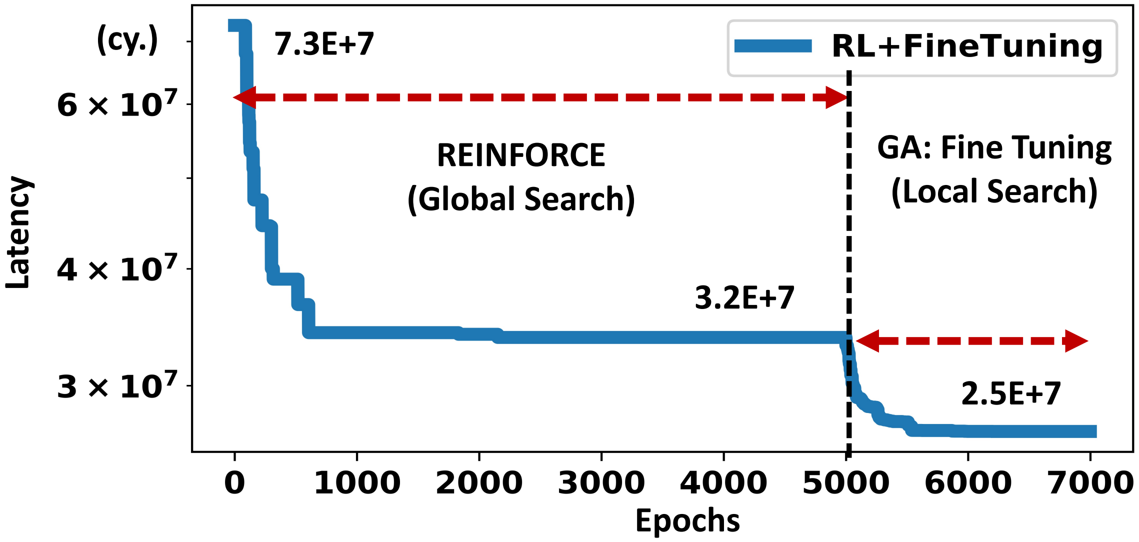


Fig. 8.8: Overall latency as a function of epochs across two-stage optimization in ConfuciuX (MobileNet-V2, Obj.(min.):Latency, Cstr.:IoT area).

experiments listed in Table 8.6, Con'X-MIX further improves the optimization results by 4% to 69% comparing to the best-performing Con'X-dla/shi/eye.

8.2.5 Benefit of Two-stage Optimization

In the above comparing with baseline experiments, we did not use local fine-tuning for fairness. We now show its effectiveness. We use local GA of 20 populations and run for 2,000 generations. We use local crossover rate of 0.2, local mutation rate of 0.05, and local mutation step of 4 .

The effect of fine-tuning

We show the two stage optimization results in Table 8.7. We show one of the trace of reached-value along epochs in Figure 8.8, which is the first row in Table 8.7 and the third row in Table 8.4. In this case, pure GA cannot find valid solution because of the tight constraint (IoT). The first-stage global search of Con'X learns to generate a valid solution first, whose value is recorded as initial valid value in Table 8.7. Then, Con'X starts to optimize the value while conforming to constraint and reached an optimized point. The first

Table 8.6: Dataflow and Hardware co-automation.

Model	Obj. (min.): Latency Cstr: Area	Optimization Results (cycles)			
		Con'X-dla (global)	Con'X-shi (global)	Con'X-eye (global)	**Con'X-MIX (global)**
MbnetV2	IoT	3.2E+07	3.1E+07	2.9E+07	**2.0E+07**
MbnetV2	IoTx	5.6E+07	4.0E+07	3.7E+07	**3.5E+07**
MnasNet	Cloud	2.1E+07	3.0E+07	2.9E+07	**6.6E+06**
MnasNet	IoT	2.8E+07	3.5E+07	3.4E+07	**1.8E+07**
ResNet50	Cloud	2.2E+08	3.1E+08	2.2E+08	**7.7E+07**
ResNet50	IoT	4.4E+08	4.3E+08	3.1E+08	**3.0E+08**
ResNet50	IoTx	6.3E+08	6.2E+08	4.9E+08	**4.4E+08**
GNMT	Cloud	2.4E+10	1.1E+07	9.7E+07	**6.0E+06**
NCF	Cloud	6.6E+08	2.6E+06	1.1E+06	**6.8E+05**
NCF	IoT	2.6E+06	6.6E+08	1.2E+06	**7.0E+05**

stage improves the values from 56% to 99% compared to the initial valid values. Then, local fine-tuning using GA to further optimizes the solutions, and they improves by another 7% to 93% than the output of the first stage, which are 66% to 99% improvement over the initial value.

Analysis of Design-Points found by ConfuciuX

In Figure 8.7, at the top, we show how ConfuciuX allocates area to different components (total PE, total buffers, and per-layer) for MobileNet-V2 and ResNet-50 in an experiment with total area constrained. The per-layer assignment is highly heterogeneous, which can be seen by the per-layer PE and Buffer assignment shown at the bottom of Figure 8.7. In particular, in MobileNet-V2, we observe that the DWCONV layers are assigned less resources on both PEs and Buffers. This could be because they require less computation and we are limited by the platform area constraint, which makes the agent reduce the assigned resources. In ResNet-50, we find that the agent assigns more Buffers to the layers that have the larger number of input/output channel size (e.g., layers 37,43, and 47).

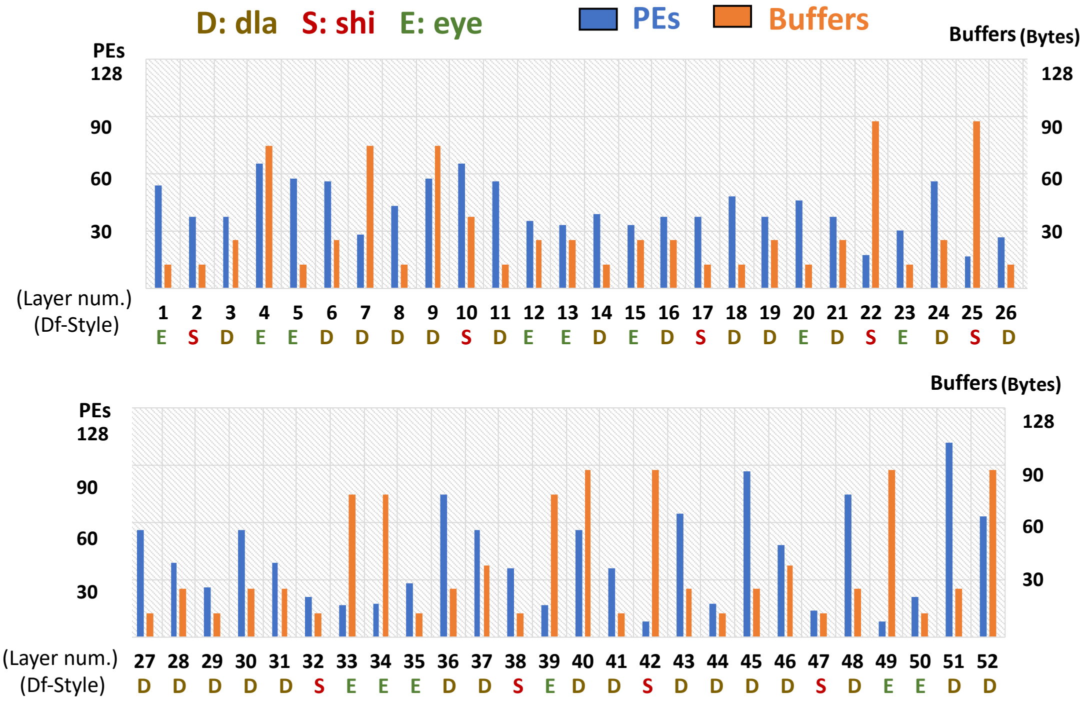


Fig. 8.9: Dataflow-HW co-automation with ConfuciuX for MobileNet-V2. (Obj.(min.):Latency, Cstr:IoT area).

Table 8.7: Two-stage optimization of ConfuciuX.

Model	Obj. (min.): Latency Cstr: Area	Initial valid value (cy.)	Con'X (global search)		**Con'X (fine-tuning)**	
			Optimized Results (cy.)	Impr. (%)	Optimized Results (cy.)	Impr. (%)
MbnetV2-dla	IoT	7.3E+07	3.2E+07	56.1%	**2.5E+07**	22.7%
MnasNet-dla	IoT	1.0E+08	2.8E+07	71.7%	**2.3E+07**	17.9%
ResNet50-dla	Cloud	1.4E+09	2.2E+08	84.1%	**2.0E+08**	7.0%
ResNet50-dla	IoT	7.1E+08	4.4E+08	37.9%	**3.3E+08**	24.0%
GNMT-dla	IoT	4.6E+09	9.5E+06	99.8%	**6.2E+06**	34.6%
NCF-dla	IoT	6.6E+07	2.6E+06	96.1%	**1.8E+05**	93.1%

Table 8.8: Resource assignments for LP deployment at compile time of ConfuciuX.

Objective: Latency		Baseline-dla			ConfuciuX-dla					
					1st: global search			**2nd: local-finetuning**		
		Used Cstr.		Optimized	Used Cstr.		Optimized	Used Cstr.		Optimized
Platform Constraint	Model	PEs	Bufs	Results (cy.)	PEs	Bufs	Results(cy.)	PEs	Bufs	Results(cy.)
Cloud FPGA Cstr: PE: 4096, Buf: 8KB	ResNet50	4081	7786	2.352E+08	4080	6338	**2.346E+08**	4096	7958	**2.2E+08**
	MbnetV2	4056	7716	2.5E+07	4032	3710	**2.3E+07**	4088	7912	**2.0E+07**
Edge FPGA Cstr: PE: 256, Buf: 4KB	ResNet50	212	3206	1.8E+09	256	2290	**1.5E+09**	256	3962	**1.2E+09**
	MbnetV2	208	3356	1.13E+08	256	2468	**1.08E+08**	256	3838	**6.9E+07**

Objective: Latency		ConfuciuX-MIX					
		1st: global search			**2nd: local-finetuning**		
		Used Cstr.		Optimized	Used Cstr.		Optimized
Platform Constraint	Model	PEs	Bufs	Results	PEs	Bufs	Results(cy.)
Cloud FPGA Cstr: PE: 4096, Buf: 8KB	ResNet50	4000	7758	**8.1E+07**	4092	7942	**6.6E+07**
	MbnetV2	4064	3944	**9.4E+06**	4096	7898	**6.0E+06**
Edge FPGA Cstr: PE: 256, Buf: 4KB	ResNet50	256	3942	**1.1E+09**	256	3922	**8.0E+08**
	MbnetV2	248	3442	**8.7E+07**	256	3920	**5.7E+07**

8.2.6 LP deployment at compile time

ConfuciuX can also be used for LP deployment at compile time. One common use-case is for FPGA-based accelerator design. As is common for FPGAs, we impose the maximum number of PEs and Buffers as constraint (which would depend on the specific FPGA board). We consider both cloud and edge FPGAs as constraints. The baseline is configured with uniform number of PE and Buffers for each layer with NVDLA-style dataflow. In Table 8.8, we show that Con'X(global)-dla performs better than baseline-dla. Then we show that local-finetuning in Con'X(global)-dla can further improves the value by 7% to 36%. Finally, we show the two stage results of ConfuciuX-MIX, where the final reached value is 50% to 72% better than baseline-dla.

8.2.7 Policy Network Exploration

We show our design decision process of the policy network. First is the action levels, L, where we pick L=12, in the experiments. By decreasing L, we decrease the complexity of the problem but worsen the granularity, and vice versa. As shown in Table 8.9, L=12 is the sweet spot we found. We also experimented with different type of policy networks: MLP-based

Table 8.9: Different configurations of the policy network.

Net Type	Obj. (min.): Latency Cstr: Area	Action Level: 10		**Action Level: 12**		Action Level: 14	
		Optimized Results(cy.)	Used Cstr.	Optimized Results(cy.)	Used Cstr.	Optimized Results(cy.)	Used Cstr.
MLP	Cloud	2.3E+07	63.9%	**2.2E+07**	96.1%	3.0E+07	26.3%
RNN	Cloud	**2.1E+07**	86.7%	**2.1E+07**	95.0%	2.3E+07	68.8%
MLP	IoT	3.4E+07	97.5%	**3.3E+07**	97.6%	4.2E+07	95.2%
RNN	IoT	3.3E+07	99.2%	**3.2E+07**	96.6%	4.2E+07	89.1%
MLP	IoTx	8.3E+07	99.0%	9.4E+07	88.8%	**4.6E+07**	97.9%
RNN	IoTx	7.1E+07	97.4%	**5.6E+07**	90.0%	5.4E+07	99.8%

and RNN-based, as Table 8.9 shows. We found RNN-based networks converging to better results, which may be owing to the fact that RNN is taking advantage of remembering the consumed constraint of previous layers.

8.2.8 Summary

We summarize some key results here. For global search, we observe that RLs can explore an extremely large design space more effectively and efficiently compared to the baseline optimization methods. Next, we find that REINFORCE, which does not rely on value network, can converge faster and reach similar or better results than alternate RL methods in the discrete and irregular HW performance exploration problem. Next, we demonstrate that our formulation of REINFORCE-based (Con'X(global)) can not only explore the HW configuration but also effectively explore the dataflow-style decision simultaneously and further optimize the results by 4% to 69%. Finally, after a coarse-grained solution is found, we show that using a specialized GA for fine-tuning the result locally can optimize the result by another 7% to 93%.

8.3 Related Works

Accelerator HW Design-Space Exploration. Fine-grained HW resource assignment has been studied extensively for LS deployment on FPGAs [180, 181, 182]. Whole-model LP deployment has been shown to be more efficient than LS deployments with uniform resource assignments for every layer [17, 61, 62, 63, 64, 65, 66, 67]. Many works have focused on allocating resources for convolution layers in a LP deployment within one FPGA [17, 65], across multiple FPGAs [63, 61, 64, 67, 183] or in cloud FPGA platforms [66]. Some works have focused on HW DSE for ASIC accelerators[184, 185, 186] or templated systolic array structures [187, 188]. Some general frameworks execute the design space exploration at the architecture level, supporting both ASIC and FPGA [60, 68]. Yang et. al, [60] further shows that the HW resource assignment dominates the performance of accelerator comparing to dataflow exploration. For design space exploration, most of these prior works employ grid/exhaustive search, while techniques for pruning the exploration spaces are manually developed. However, with myriads of DNN models being designed on a daily basis, it becomes harder to manually design and tune the policy for the newly constructed search space. In this work, we develop a ML-based method to automate the search process with high sample efficiency for both LP and LS scenario.

ML-based methods for DNN compilation and mapping. ML methods have found value in mapping/compiling DNNs over hardware. TensorComprehensions [31] uses genetic algorithm, AutoTVM [101, 24] uses simulated annealing and boosted tree, Reagen et. al, [26] uses Bayesian optimization, RELEASE [27] uses RL, ATLAS [270] uses black box optimizations, some compiler design [271, 272] use profile-guided optimization to perform target-independent front-end compiler optimizations on DNNs or linear algebra computations. Some recent works use RL on HW/SW co-exploration to explore both DNN and its mapping over hardware [273, 274, 275, 276]. The problem of mapping the DNN computation graph over multiple devices (CPU/GPU/TPU [82]) has also been explored through manual heuristics [277, 278, 279] and RL [280, 281, 282]. In contrast to these works, this work looks at fine-grained design-time assignment of compute and memory within an accelerator.

Dataflow style optimization. Architecture design of ML accelerators include resource assignment and dataflow style design. Dataflow style is a scheduling and compiler optimization problem, which has been studied for decades for the generic platform such as CPU or GPU [9, 51, 52, 30, 53, 54, 55, 56], or for FPGA [57, 18], while they apply grid search for reaching their objectives. For ML accelerators, some mainstream dataflow style are manually designed and proven to be efficient, becoming prominent or commercialized [83, 82, 81, 80]. In this work, we focus on the resource assignment part of the accelerator design flow, and utilize some prominent dataflow styles [83, 82, 81, 80].

8.4 Summary

While efficient DNN models and dataflow style are widely studied for ML accelerators, HW resource assignment is relatively unexplored. In this chapter, we propose ConfuciuX, an autonomous strategy to find out the optimized HW resource assignment for a given DNNs, a dataflow style and platform constraints. ConfuciuX leverages RL for the global search, augmented with GA for fine-tuning. We quantitatively experiment on different models, platform constraints and dataflow styles. ConfuciuX demonstrates the highest sample-efficiency compared to other optimization and RL methods. this chapters shows the promise of leveraging ML within the DNN accelerator design workflow, with opportunities for future work across new ML algorithms for learning dataflow/hardware behavior, and DNN-dataflow-hardware co-design.

CHAPTER 9
CONCLUSIONS AND FUTURE WORKS

9.1 Summary of Contributions

9.1.1 Formalism of DNN Accelerator Flexibility and the Implied Map Space

We demonstrate that increased formalism of the notion of flexibility leads to concrete benefits. Namely: (A) the ability to taxonomize existing accelerators along flexibility axes, (B) the ability to precisely quantify the shape of a map-space against a theoretical upper-bound, and (C) the ability to integrate flexibility into existing MSE flows. We identify that in several cases, partial flexibility along some of the axes is sufficient for capturing an optimized mapping. Most significantly, our evaluation of a 2014-style accelerator demonstrates that a design-time investment in flexibility can lead to concrete improvements in future-proofing after deployment time.

Of course, the field of deep learning is changing incredibly rapidly. Turing Complete platforms such as GPUs and CPUs will undoubtedly continue to play a key and irreplaceable role as the main platforms that allow machine learning experts to rapidly innovate. However, we believe that it is equally important that the most successful neural network architectures can be accelerated via hardware specialization, without ASIC designers risking complete obsolescence after deployment. Ultimately, the key reason flexible mapping support can be omitted from hardware designs is not area or energy, but simple design and verification effort, as well as mapper toolchain support. We hope that this thesis will ultimately lead to more research that allows architects to quantify the benefits of “future-proofing” with the same level of precision as present-day budgets. Finally, while most existing DSE frameworks focus on finding the “optimal design point”, this thesis, armed with the ability to systematically construct map-spaces of fully/partially flexible accelerators, opens an interesting future avenue of research to identify “regions of optimal (i.e., high-performing) design space”, leading to better explainability of the DSE algorithms and interpretability of the accelerator design space.

We believe this thesis will help both compiler developers and architects systematically develop flexibility-aware accelerator designs. Given that flexibility is often not added in HW to simplify engineering efforts, we envision partial flexibility being ripe for future work. In the future, we hope the analysis can be expanded to cover even more complex aspects of data orchestration and generalized beyond DNNs.

9.1.2 Deep Analysis and Improvement of MSE for DNN Workloads

Map Space Exploration is critical for DNN accelerator efficiency. It is a complex and challenging problem because the search space is often massive. This is because the search space for legal mappings for even a single layer of a modern DNN (e.g., ResNet-50) on a typical edge class accelerator is $\sim O(10^{24})$ which would require more time than the age of the earth to search exhaustively (assuming 1msec to evaluate each mapping sample). This gets exacerbated as newer and ever-larger DNN models are being created with increasing frequency, especially thanks to the success of neural architecture search techniques. Furthermore, the advent of *compressed-sparse* DNNs, whose mappings are not performance-portable across sparsity levels (a key finding in this thesis), further increases MSE burden.

Researching more sophisticated scalable and sparsity-aware MSE techniques is at least partially hampered by the fact that even though prior approaches have *empirically shown* that their techniques work, none of them demonstrate *why* they work and the insight behind their optimization techniques.

In this thesis, we develop scalable MSE approaches for future complex DNN workloads. However, instead of proposing yet another mapper that *just works*, we first distill the knowledge from prior mappers spanning heuristics and learning-based optimization approaches to demystify MSE as a problem. We analyze their behavior, learn from their best traits, and use these learnings to scale MSE to more complex workloads.

We propose a "warm-start" technique to initialize the MSE with previous mapping solutions from previous layers in the replay buffer based on a *similarity* metric, enabling the mapper to start at a better point and converge faster. We propose a "sparsity-aware" technique to search for a mapping that can perform well across a range of target activation sparsity. A fixed mapping found by our "sparsity-aware" approach can achieve 99.7% of the performance of each of the mappings specifically tailored for the various density levels. We believe these techniques can be augmented over existing MSE tools, making them more robust and scalable for future DNNs.

9.1.3 Search Algorithm/ Mapper

The design of the Mapper is critical for the success or failure of the MSE. For different applications we demonstrated across different chapters, we follow a workflow described next. 1) Study the design/map space of the problem. 2) Analyze the performance impact of different mapping axis. 3) Transform the problem into an optimization problem that can be plugged into the MSE optimization loop. 4) Develop or adapt open-sourced cost model (evaluation method) for the problem. 5) Develop Mapper (searching algorithms) for the specific problem. The mapper we developed includes:

- Gamma: GA-based mapper with domain-specific genetic operator design
- DiGamma: The next version of Gamma, which supports HW and Mapping co-exploration by extending the genetic encoding and operators in Gamma.
- Magma: A GA-based mapper for searching optimal mapping across multiple accelerators.
- ConfuciuX: An RL-based mapper for HW resource allocation in DNN accelerator.

9.2 Future Work

9.2.1 MSE for Both Intra- and Inter- Layers

Throughout the thesis, we follow the strategy of layer-by-layer mapping of the DNN models. Each of the mappings is optimized for a layer inside a DNN model, i.e., intra-operator mapping. The benefit of mapping is a more efficient data orchestration across buffers and PEs and exploiting the reuse opportunity in the workloads. However, across layers, there are also activation reuse opportunities that we did not leverage. In many compiler frameworks, there are naive heuristics to leverage the immediate activation reuse called – fusion. However, they often only support basic functionality such as fusing CONV with Relu, and fusing CONV with Add. The inter-layer (inter-) fusion such as fusing CONV with CONV is known to be challenging. It is hard to efficiently implement it. However, potentially with the help of MSE framework, we could leverage the mapper to explore this huge intra- and inter-layer map space and yield further performance breakthroughs.

Moreover in chapter 8, we explore the potential of using RL and RNN to form a inter-layer mapper. RL and RNN helps relate one layer of the model to another, learn the global constraint and local expense, and optimize for accumulated "model-wise" reward. Indeed, ConfuciuX focused on HW resource allocations, whose problem domain is different. However, it demonstrates the promising outlook of inter-layer optimization, whose idea might be leveraged for mapping search problem as well.

An interesting future work is to extend the intra-layer MSE to intra- and inter- layer co-exploration MSE by leveraging the learnings from ConfuciuX. However, the challenge for this future direction is obvious – the size of map space. But, we believe this is an interesting and profitable direction for future work.

9.2.2 MSE for Heterogeneous System

As AI workloads continue to drive up the demand for computing, there is a trend towards building large accelerators housing several sub-accelerator/arrays. Key examples include MCM-based SIMBA [8], wafer-scale Cerebras [192] or scaled-out platforms [193, 194, 195]. Some recent studies have also explored heterogeneous multi-accelerator designs enabled via reconfiguration [129] or separate *heterogeneous* sub-accelerators [196]. Moreover, the system could house multiple of these DNN accelerators, clusters of GPUs/ TPUs, and even CPUs. How to efficiently map the DNN workloads across the hydrogenous system is still an open problem.

In chapter 7, we show preliminary work on extending the MSE idea to optimize the throughput of the MCM-based accelerators. However, because of the inevitable search time of the MSE, the mappings are searched off-line. It is useful for multiple applications we identify in chapter 7, however also challenging to be deployed on many real-time system. A potential approach could be the combination of heuristic and MSE where we analyze and identify the sweet spot of performance and search-time. Another potential approach could be we run MSE and extract the learned knowledge of them and encode them into a fixed rule-based mapping algorithm. This is a thriving new research area. Multiple options could be explored and we leave them as future work.

9.2.3 MSE for new DNN Operation

DNN models with new DNN operations are constantly being proposed. For example, DNN models such as Mobilenet featuring depth-wise separable convolution [126], Efficientnet featuring compound model scaling [166], Once-for-all featuring progressive shrinking of network [38], SparseTransformer featuring sparsified attention layer [167], and many others. New DNN operations could create another mapping axis that the canonical MSE didn't explore. Therefore including those mapping axes into the MSE to enrich the map space and release the potential performance gain of these new DNN operations become an interesting future work. For example, in §A.1, we analyze and show that attention operation cannot be efficiently explored with the typical MSE framework because of its quadratic memory bottleneck. We develop a specific mapping/ dataflow for the attention operation leveraging the inter-layer fusion technique. We expect the inter-layer fusion can be added to the MSE framework to make MSE support attention better. Also, we only demonstrate a specific example of attention operation. There could be different unexplored mapping axis for different DNN operations, which could also be studied and added to the MSE.

9.2.4 Sparsity as new Mapping Axis

We have seen in Figure 5.2 that sparsity does matter for the performance of MSE. In Figure 5.2, we didn't go into the detail of the compression format, the sparsity pattern, sparsity-aware training and so on. Since sparsity is an critical factor of MSE performance, we believe some of these sparsity parameters can potentially become new mapping axes that we consider in MSE. It would enable MSE to automatically figure out the best sparsity strategy, which could inform the DNN model designer to design new pruning strategies and also inform the HW architect to cleverly decide the flexibility support for the diverse sparsity pattern. Also, an extended study on how to formalize flexibility support for sparsity is an interesting future work.

9.2.5 Pre-trained Mapper and Transfer Learning

As shown in chapter 3 and Figure 5.2, there are some general mapping strategies that are transferable between similar workloads. For example, in Figure 5.2, we show that Parallelism and Order mapping axes are transferable between CONV workloads. However, these transferable patterns are found by extensive experiments and distilled by human. DNNs have proven their ability to learn generalized knowledge that can achieve or surpass human ability. We believe, it is with high potential we could use DNN models to learn the generalized patterns of mapping, which might "re-discover" some heuristics made by human and more excitingly find new patterns that is never explored. We have a preliminary work on using Transformer pre-training techniques to learn general mapper [283]. We believe there could be substantial of future work on this promising direction.

9.2.6 Mapping and Neural Architecture Co-Exploration

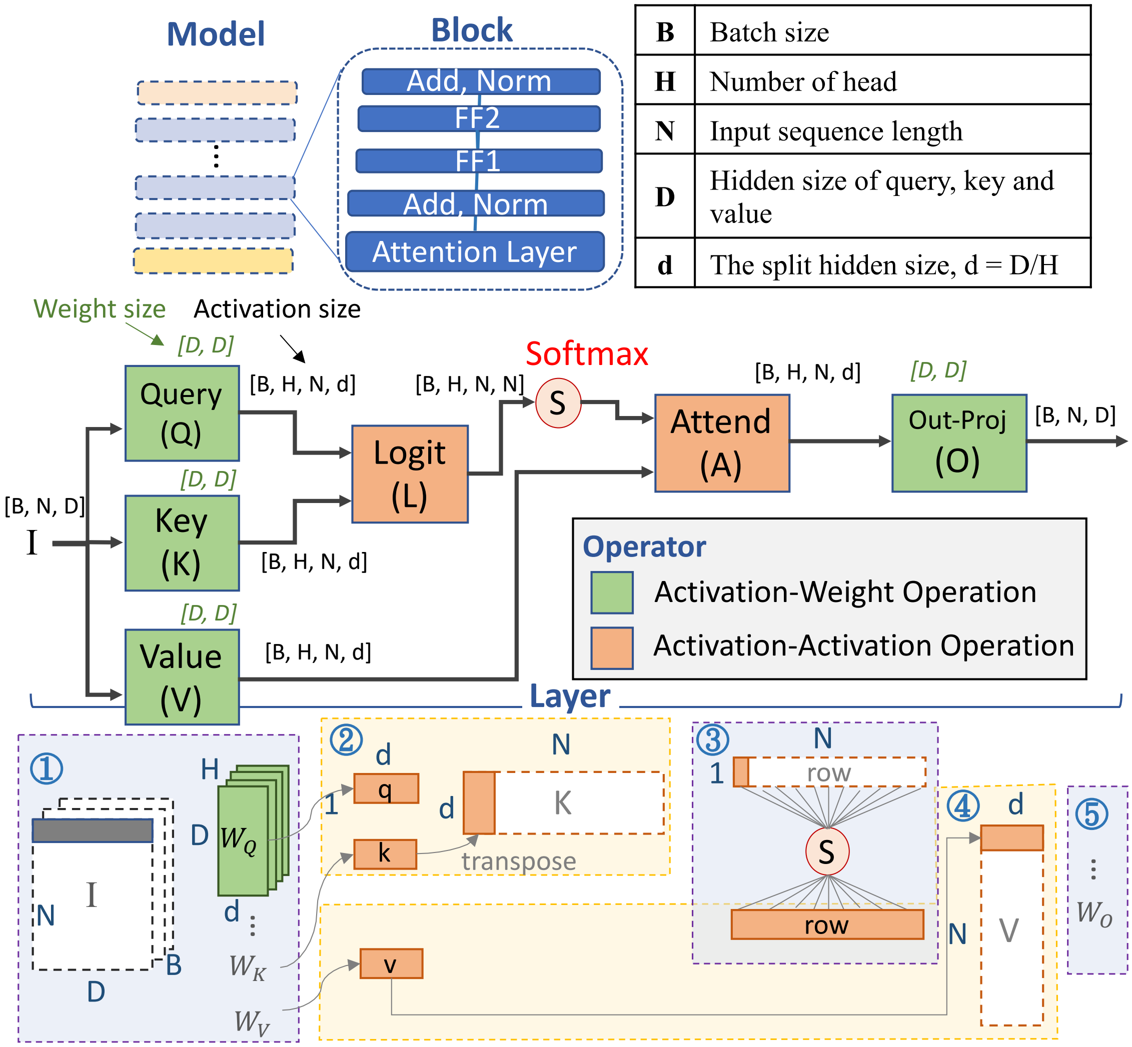
HW-aware Neural Architecture Search (NAS) has been a popular topic in recent years. However, most HW-aware NAS utilizes estimations such as FLOPs or the number of parameters as HW performance, which are the imprecise estimation of the HW performance. Recent works start to incorporate a more detailed HW cost model to provide HW performance feedback, which improves the imprecise issue. However, they often need to employ a fixed mapping to simplify the problem. From the study of this thesis, we understand the performance difference between good and bad mapping can be orders of magnitude difference. Therefore, a fixed mapping could actually bias the NAS to tailor for a bad mapping but not optimize for the best possible model quality and performance. We envision, with the ripe of MSE framework, MSE could be potentially plugged into the NAS procedure and push the Pareto Frontier of HW-aware NAS.

Appendices

APPENDIX A
FLAT: AN OPTIMIZED DATAFLOW FOR MITIGATING ATTENTION BOTTLENECKS

In this thesis, we mainly focus on some extensively used DNN operations such as CONV2D, Depth-wise CONV, Point-wise, CONV, Fully Connected, GEMM. However, new DNN operations are also being proposed with the rapid evolution of machine learning. Some new DNN operations are proven to be useful and can break through the state-of-the-art performance. One well-known example is the success of Transformer models, whose key structure is its attention operation. To support the new attention operation efficiently becomes a new and challenging problem. In this chapter, we discuss what is the key challenge when running attention operation on a canonical DNN accelerator and propose a new mapping/ dataflow method to ameliorate the challenge.

This chapter fundamentally tackles the challenges associated with attention layers by devising a first in its class *many-to-many inter-operator* dataflow optimization mechanism, called **F**used **L**ogit **A**ttention **T**iling. This optimization particularly fuses multiple many-to-many tensor operator, while systematically preserving their inter-operator data dependencies, leading to a significant reduction on off-chip memory bandwidth pressure. In addition, to fully realizing the performance benefit of this inter-operator fusing mechanism, FLAT performs a new tiling approach across the fused operators. This tiling enables efficient staging of quadratically growing intermediate tensors of attention operations on tight-budgeted on-chip memories, leading to higher performance and energy savings and elevates the scalability of transformer models up to 64 K inputs. These benefits are unlocked with only modest hardware changes, integrating into a platform deployable on off-the-shelf DNN accelerators.



B	Batch size
H	Number of head
N	Input sequence length
D	Hidden size of query, key and value
d	The split hidden size, d = D/H

Fig. A.1: The structure of attention-based models. Green matrix notation shows the size of weight tensors; black matrix notation shows the size of activation tensors; Softmax is applied on output of Logit.

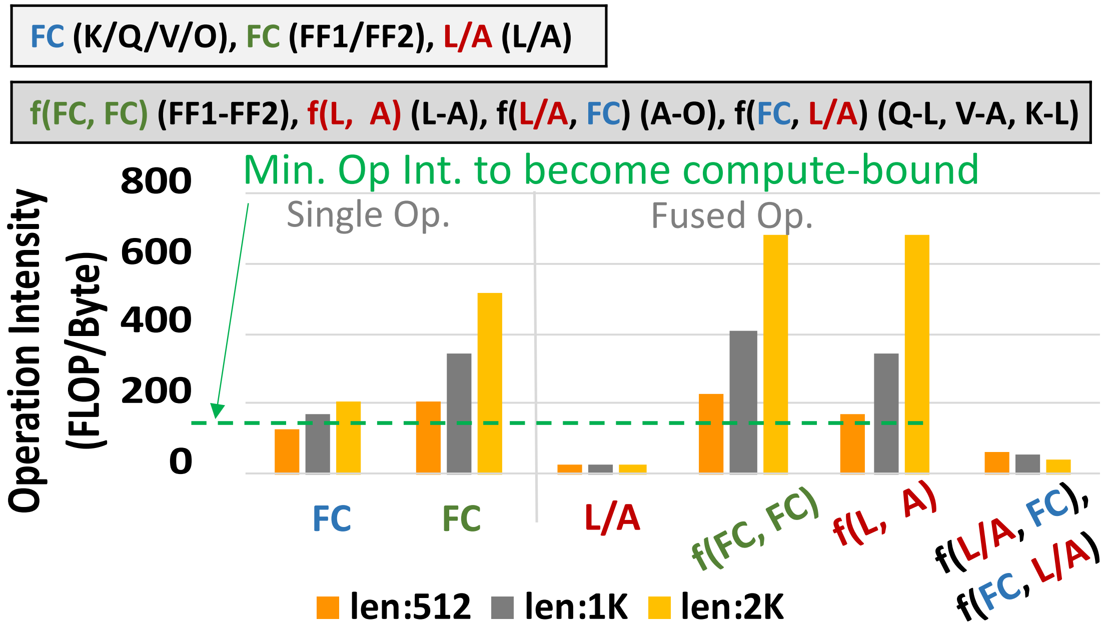


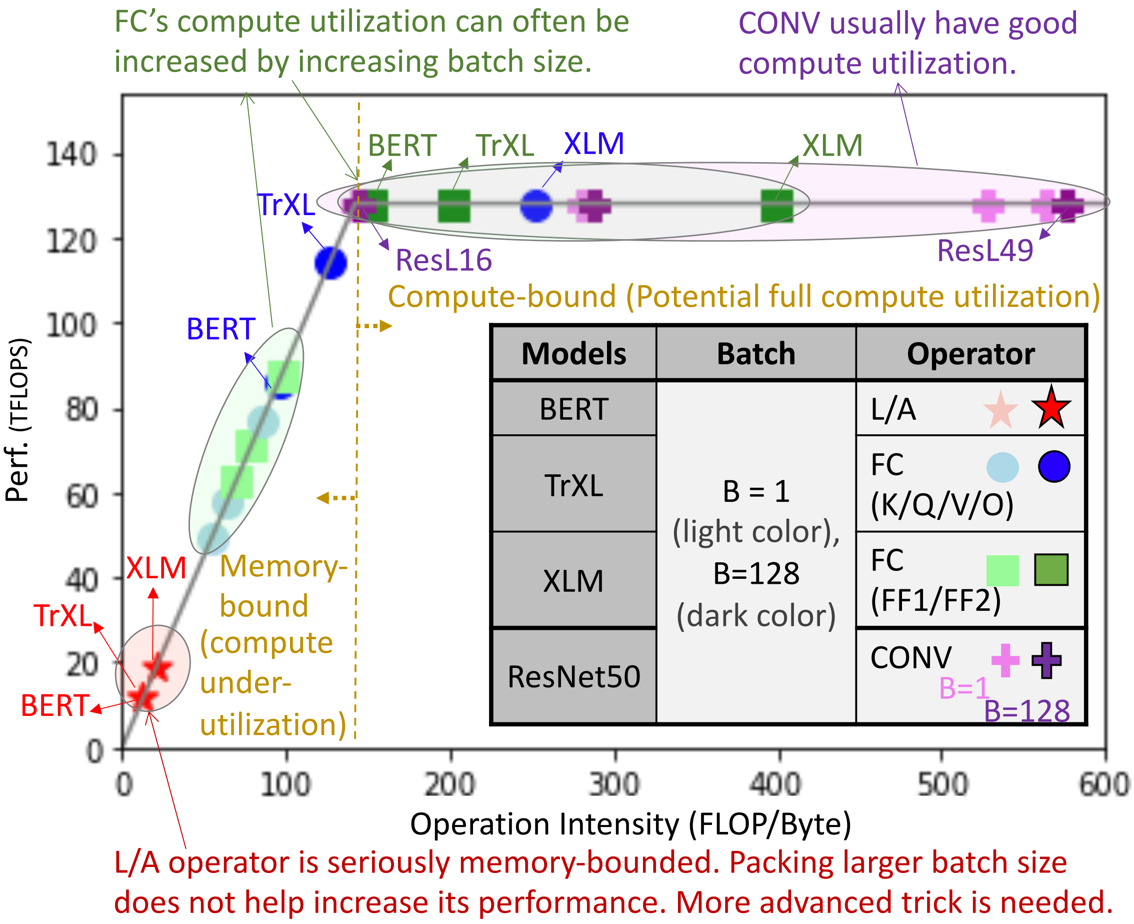
Fig. A.2: Potential of Tensor-Tensor Fusion. The operation intensity of single and fused operators in attention layers in TrXL(-large) [70] using batch size=1. The notations are from Figure A.1. f(X, Y) means a fused X and Y operator.

A.1 FLAT Dataflow Concept

We design a specialized dataflow strategy, Fused Logit Attention Tiling (FLAT), targeting the two memory BW-bound operators in the attention layer, L and A. FLAT includes both intra-operator dataflow and a specialized inter-operator dataflow, executing L and A in concert.

A.1.1 Identifying Tensor Fusion Opportunity

Figure A.2 plots the operation intensity of single and fused operators in attention layers of an attention-based model [70]. The green dotted line marks the operation intensity threshold (ridge point) from memory to compute boundedness in TPU-v3 [284]. We observe that for FC-based operators (K/Q/V/O), the operational intensity is sufficient to be compute-bound, while for L/A it is low (as we had also observed via Figure A.3). However, after fusing L and A (f(L, A)), the effective operational intensity (of the fused operator) is higher. This motivates us to explore L and A fusion.



FC's compute utilization can often be increased by increasing batch size.

CONV usually have good compute utilization.

Compute-bound (Potential full compute utilization)

Memory-bound (compute under-utilization)

Models	Batch	Operator
BERT	B = 1 (light color), B=128 (dark color)	L/A ☆ ★
TrXL		FC (K/Q/V/O) ● ●
XLM		FC (FF1/FF2) ■ ■
ResNet50		CONV ✚ ✚ B=1 B=128

L/A operator is seriously memory-bounded. Packing larger batch size does not help increase its performance. More advanced trick is needed.

Fig. A.3: Roofline analysis on TPU-v3 [284] for operators in BERT(-base) [69], TrXL(-large) [70], and XLM(xlm-mem-en) [71], and ResNet50 [142] using sequence length = 512.

Why not fuse other operators? We did not fuse other operator pairs such as f(Q, L), f(A, O), or f(V, A), for three reasons. (1) The operational intensity is often sufficient and can be increased by leveraging batch size to reach compute-bound (Figure A.3). (2) Fusing two FCs (f(FC, FC)) can achieve higher operational intensity; however since the operator is already compute-bound, there is not much value in leveraging fusion (and the additional complexity). (3) We often need finer-granularity dataflow schemes to fit fused operator tensors on-chip; however fusing two activation-weight computation (f(FC, FC)) can trade-off (weight) reuse opportunity and may reduce actual achievable performance (§A.2.3).

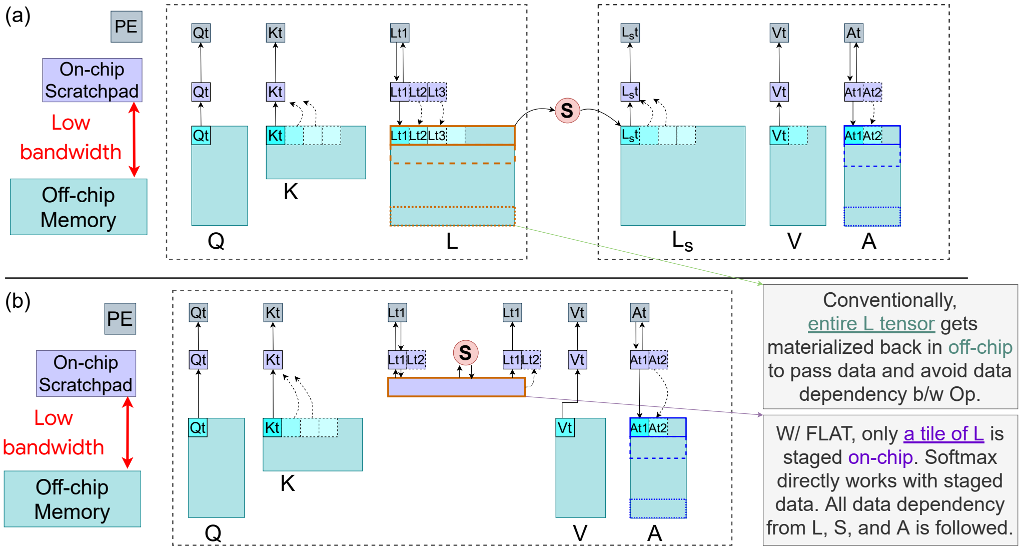
Why not fuse multiple operators? We did not fuse multiple operators such as f(L, A, O) or f(K, L, A) for two reasons. (1) Fusing L/A with FC such as f(A, O) or f(K, L) can drop the potential performance of FCs compared to their single operator performance (Figure A.2). (2) The more operators we fuse, the more data we need to stage partially on-chip. Since the on-chip memory is often extremely limited, we need to execute the fused operators at a much finer granularity, which may lead to a degradation in achievable performance (§A.2.3). With these analyses, we decide to fuse only L and A.

A.1.2 Challenges with Tensor Fusion Implementation

Fusing L and A operators introduces two key challenges that we discuss here. §A.2 presents implementation details.

Challenge 1: Respecting data dependencies across operators. Fusing L and A causes its unique challenge of data dependency owing to the many-to-many Softmax operation between them. Softmax requires a reduction along a specific dimension of the tensor before scaling individual elements. Arbitrary inter-loop tiling as employed by prior CONV/FC fusion techniques [285, 67] violates this data dependency constraint.

Challenge 2: Effectively handling large intermediate tensors that do not fit in on-chip memory. Recall that the intermediate tensor between L and A has size $O(BHN^2)$.



Conventionally, entire L tensor gets materialized back in off-chip to pass data and avoid data dependency b/w Op.

W/ FLAT, only a tile of L is staged on-chip. Softmax directly works with staged data. All data dependency from L, S, and A is followed.

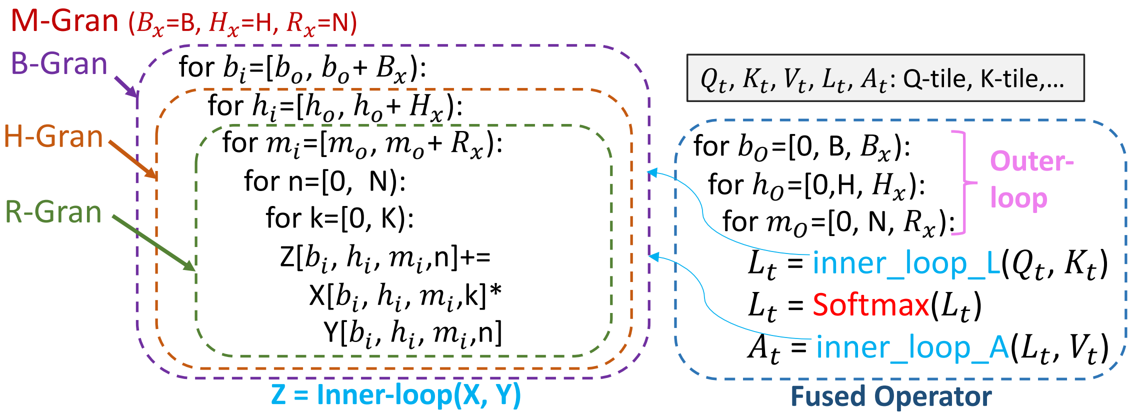
Fig. A.4: (a) Baseline and (b) FLAT dataflow. FLAT performs inter-operator fusion of L, A while respecting data dependencies introduced by Softmax. FLAT-tile enables staging slices of the logits tensor in the on-chip scratchpad increasing effective memory bandwidth. This fused, interleaved execution of L, A yields higher compute utilization and improved performance.

Table A.1: Buffer requirement for tiling granularity. M: batched Multi-head, B: Batch, H: Head, R: Row.

Granularity	**M-Gran**	**B-Gran**	**H-Gran**	**R-Gran**
Buffer Requirement	O(8BDN+ BHN^2)	O(8DN+HN^2)	O(8Nd+N^2)	O(4Rd+ 4Nd+RN)

This can easily exceed the on-chip memory capacity of DNN accelerators. Further, the specific size of the on-chip memory may be highly variable across different accelerators.

Owing to the above challenges, conventionally, we often do not apply tensor fusion to attention layer and stick to operator-by-operator operation scheme, as shown in Figure A.4(a). In this work, we use FLAT to enable L-A fusion operation scheme, as in Figure A.4(b). We describe details next.


M-Gran (B_x=B, H_x=H, R_x=N)
B-Gran
H-Gran
R-Gran
Q_t, K_t, V_t, L_t, A_t: Q-tile, K-tile,...
Outer-loop
Z = Inner-loop(X, Y)
Fused Operator

```
for $b_i$=[$b_o$, $b_o$+ $B_x$):
  for $h_i$=[$h_o$, $h_o$+ $H_x$):
    for $m_i$=[$m_o$, $m_o$+ $R_x$):
      for n=[0, N):
        for k=[0, K):
          Z[$b_i$, $h_i$, $m_i$,n]+=
            X[$b_i$, $h_i$, $m_i$,k]*
              Y[$b_i$, $h_i$, $m_i$,n]
```

```
for $b_O$=[0, B, $B_x$):
  for $h_O$=[0,H, $H_x$):
    for $m_O$=[0, N, $R_x$):
      $L_t$ = inner_loop_L($Q_t$, $K_t$)
      $L_t$ = Softmax($L_t$)
      $A_t$ = inner_loop_A($L_t$, $V_t$)
```

Fig. A.5: For loop of fused L-Softmax-A (or shortened as L-A in the paper) and the choice of granularity.

A.2 FLAT Dataflow Implementation

To fuse two tensor operators X and Y, we divide the loop nests into two groups: outer-loop and inner-loop. We use L and A for illustration as shown in Figure A.5, but the principles are applicable to any set of consecutive tensor operators. The outer-loops are shared across L and A. The inner-loops are unique for each operator. After fusion, the fused operator has two inner-loops, which we run one after another (interleaved), and iterate through the shared outer-loop. Considerations for tile sizes to address the data dependence and on-chip memory constraints (§A.1.2) are discussed in this section.

A.2.1 FLAT-tile and Execution Granularity

FLAT employs two levels of tiling: intra-operator tiling and inter-operator tiling. We name each tile in inter-operator tiling, a FLAT-tile. FLAT computes FLAT-tile activations from L and feeds it through Softmax and to A. FLAT-tiles, the inner-loop in Figure A.5, essentially specifies how many slices of the partial intermediate tensor are calculated in one pass of the fused-operator in Figure A.4(b).

The minimum granularity of the FLAT-tile is determined by the data dependence constraint of Softmax and called *row-granularity* (discussed in §A.2.2), for effectively collecting a group/tile of (input) data that fulfills the Many-to-Many dependency pattern of Softmax.

We progressively build larger (coarser-grain) tiles, namely, tiling multiple number of rows at a time (R_x), multiple number of heads (H_x), and finally, multiple number of (micro-)batches (B_x) in the tile. We refer to these as Row (R-Gran), Head (H-Gran), and Batch (B-Gran) granularity respectively (discussed in §A.2.3). Further, the most intuitive baseline of moving the entire intermediate tensor (namely the entire output of L) on-chip is referred to as Batch-Multi-Head granularity (M-Gran), as shown in Figure A.5.

A.2.2 Managing Constraints from Data Dependency

Basic execution unit: Row-granularity. The Softmax reduction is along the key dimension: this effectively captures the relative weight of each token in the query sequence against other tokens in the key sequence. The minimum Softmax execution requires an [1, N] input array, which in turn requires a query of [1, D] and a key of [D, N], as illustrated in Figure A.1 Step-2 and Step-3[1]. This forms our basic tiling unit (finest granularity)—*row-granularity*, which respects the data dependency introduced by the Softmax while keeping minimum number of elements to pass between L and A. FLAT restricts the tile sizes to operate in multiples of this row-granularity.

A.2.3 Managing Constraints from On-Chip Memory

M-Gran, B-Gran, H-Gran: Leveraging Limited Reuse of f(L, A)

Coarser granularities require staging larger tiles in the on-chip memory. As sequence lengths increase this can increase rapidly (recall the O(N^2) growth). To fit into the limited on-chip memory, one may target finer granularities, e.g., moving from M-Gran to B-Gran (i.e., effectively tiling micro-batches). In general, while this helps reduce the size of the tile, when we are tiling two operators at finer granularity at the outer-loop, we may trade-off the reuse opportunity at the inner-loops. For example, for f(FC, FC) and f(CONV, CONV), when decreasing the batch size (i.e., micro-batching), we directly reduce the number of

[1] Note that here we are describing fused L-A operators, where the K, Q, and V tensors are already calculated and prepared in Step-1.

times a weight can be reused. The weights need to be re-fetched again and again for each micro-batch. This effect is exacerbated when considering finer granularities such as H-Gran for the weight-activation K/Q/V/O operators. The reduced reuse opportunity by inter-operator tiling reduces the achievable performance, even though the fused operator has large operational intensity (Figure A.2). In contrast, L and A are activation-activation operations. Each new activation of L needs to compute with a new activation of A, i.e., there are no reuse opportunities at the algorithmic level. Decreasing the tiling granularity (M-Gran to B-Gran to H-Gran), does not preclude any reuse opportunity, since there are no reuse opportunities at the algorithmic level. Thus, the finer M-Gran, B-Gran, H-Gran in FLAT are well-suited for f(L, A).

R-Gran: Extreme Large Sequence Range

To enable very long sequence lengths [286, 287, 288], but with limited on-chip memory resources [284, 289], we need to tile at even finer granularity, namely R-Gran. However, finer granularities come with an associated trade-off: when we reduce the number of rows (R_x), we will also reduce the reuse opportunity in the matrix multiplication itself. For example, even for L/A fusion, using fewer rows means the same key vectors need to be fetched multiple times across the interleaved cross-operator outer loops. Further, reducing number of rows at the outer-loop could also decrease the achievable performance at the inner-loop, e.g., not enough dimension size to fully utilize PE array. Thus, FLAT co-explores inter-operator (optimizing the outer-loop) and intra-operator dataflow (optimizing the inner-loop) to mitigate these potential sources of inefficiencies.

On-chip Buffer Requirement

Table A.1 lists the required on-chip buffer size using FLAT. We derive the R-Gran value here (others follow similar reasoning). L operator consumes (Rd+Nd)x2 size of the on-chip buffer (2 to account for double buffering), and A consumes (Nd+Rd)x2. RN for buffering the

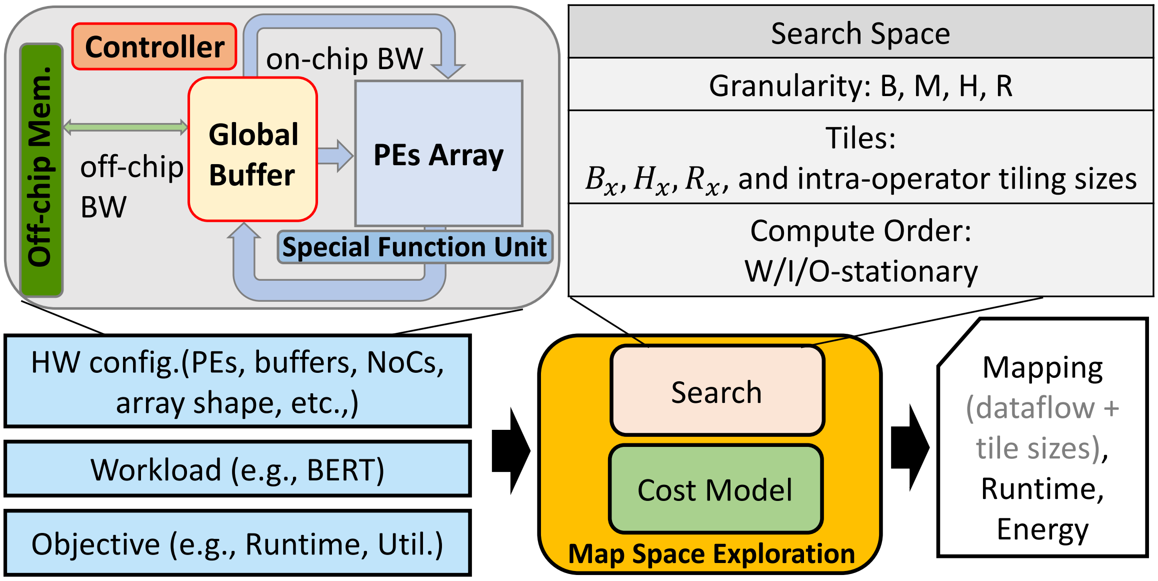


Fig. A.6: Map space exploration framework. (Special Function Unit: for computing non-linear operations, e.g., softmax, activations.)

intermediate tensor (FLAT-tile) (no double buffering since it does not interact with off-chip memory), whose on-chip buffer requirement is shown in Table A.1.

A.2.4 HW support to implement FLAT

FLAT requires minimal HW support: (1) controller to recognize our fine-grained dataflow, and (2) on-chip buffer to be *software-addressable* to support tiling. These features are already supported by most accelerators [284, 8, 194, 289].

A.3 Evaluation Methodology

A.3.1 Modeling Methodology

Accelerator simulation. We developed a detailed analytical cost model to estimate the performance and energy consumption of FLAT across a range of hardware accelerators configurations, following the same methodology as prior work [4, 58]. We meticulously model the major microarchitectural blocks commonly shared by most DNN accelerators as outlined in Figure Figure A.6. Based on this model, we collect the relevant architectural details, which later are used to compute the accelerator performance metrics. Using this simulation methodology enables us to study and grasp the key differences across a range of dataflow optimizations, while supporting different classes of hardware accelerators, including TPU-v3 [284], Eyeriss [147], Simba [8]. One of the key features of our simulation methodology is the detailed modeling of the accelerator memory hierarchy to systematically assess the memory-boundedness of attention operators and their pressure on off-chip memory bandwidth.

Energy model. Collecting the detailed activity counts from the analytical model, we use Accelergy [290] framework to estimate the energy consumption for the major microarchitectural blocks. That includes compute, on-chip memory, and off-chip memory communications. Note that FLAT neither alter the total number of computations nor the total number of accesses to the global buffer. Instead, it optimizes the number of off-chip memory accesses, which is the major contributor to the overall accelerator energy consumption [291, 83].

Comparison to prior work. There are several popular open-sourced DNN accelerator modeling frameworks, such as Timeloop [4], MAESTRO [58], and others [10, 9, 162, 163]. However, they have limited support for cross-operator fusion or cross-layer performance modeling. For example, Timeloop [4] and MAESTRO [58] only model the performance of an accelerator in a single-layer manner, which narrows their application in modeling fusion opportunities. In contrast, our framework evaluates the performance of DNN models in both single-layer and cross-layer manner, enabling various cross-operator fusion studies. To ensure the integrity and correctness of our framework, we compared the simulation results from our framework under single-layer modeling to MAESTRO [58]. The performance metrics are within 1% difference to MAESTRO's results.

Workloads. We study a range of recent attention-based models, including BERT-Base [69] (BERT), TransformerXL [70] (TrXL), FlauBERT [292], T5 [293] (T5), and XLM-MLM-En [71] (XLM). We evaluate these models under different sequences lengths ranging from $N = 512$ to $N = 64K$ to imitate attention-based models with long sequence length [294, 286]. We also study a future-proofing sequence length of size $N = 256K$. We use a batch size of 64 for all the models. Note that the batch size choice is immaterial to our dataflow optimization.

Accelerator configurations. We evaluate the benefits of FLAT on two different accelerator regimes, namely cloud and edge accelerators. As outlined in Table A.3, we set the accelerator configurations in our model following the designs proposed for cloud [284, 289] and edge [1, 295, 296] accelerators. In all the evaluations, we allot sufficient FLOPs to the Special Function Unit (Figure A.6) in order to eradicate the expected compute bottlenecks, uniformly across all the dataflow variants.

Evaluation metrics. For all the evaluations, we use performance and energy savings as efficiency metrics. For comparisons between different models, we normalize the runtime of each dataflow by the ideal runtime of the target workload as follows:

$$Util = \frac{Runtime_{ideal}}{Runtime_{dataflow}}$$

; where $Runtime_{ideal}$ is the arithmetic optimal runtime of the current workload. That is, the total computes in a model divided by the peak FLOPs of the target accelerator. $Runtime_{dataflow}$ represents the achieved runtime by a dataflow optimization. This *normalized runtime* metric explains how far the current dataflow is from its arithmetic optimum. This metric is an indication of the distance to the dataflow compute-boundray in the roofline model as well as compute resource utilization ($Util$).

Map-space exploration workflow. We also integrate a map-space exploration (MSE) workflow (Figure A.6) into our simulation framework. The main purpose of this exploration workflow is to carry out a search algorithm in a predefined map space governed by the cost model. In this work, we use exhaustive search to find the optimal design point uniformly across all the dataflow optimizations. MSE includes both intra- and inter-operator dataflow optimization space (enabling optimal dataflow comparisons with and without FLAT technique later in Table A.2 and §A.4). The relevant architectural parameters for this optimization space are outlined in Figure A.6.

A.3.2 Details of Major Microarchitectural Units

Compute model. We model the compute array as a collection of processing elements with configurable bandwidth from/to the global on-chip buffers. The compute array model supports common intra-operator dataflow, including weight, input, and output stationary. In addition, we model various data distribution and reduction NoCs, including systolic, tree, or crossbar structures to study the the trade-offs between compute bandwidth and distribution-collection time [68, 2]. Following this methodology, we model TPU [82] (systolic-array) as well as other spatial array acccelerаors, such as Eyeriss v2 [1] and MAERI [2]). We also carefully model the overhead of switching tiles for filling and draining data to reflect the cold start and tailing effect. Finally, we account for softmax operation runtime in all the evaluations.

Buffering model. Studying dataflow optimization techniques demand for a detailed modeling of buffers. To achieve this objective, we model PE arrays with local scratchpad for input, weight, intermediate results, and output storage. We add the on-chip global buffer to store the intra- and inter-operator tiles. The performance model also includes the data spilling overhead. That is when the live memory footprint (buffer requirement for staging data on-chip) is larger than the on-chip global buffer capacity.

Memory bandwidth. Since there are multiple microarchitectural units that access the on-chip and off-chip memories, we model them as limited bandwidth shared-hardware resources. That is, if the access rate to a shared memory resource exceeds a pre-defined bandwidth, the data accesses are throttled. This overhead manifests as longer runtime.

A.3.3 Overview of the Evaluations

We organize the evaluation results as follows:

(1) Efficacy of the FLAT dataflow (§A.4.1, §A.4.2, §A.4.3): We first fix the “headline”

Table A.2: Comparisons Dataflow Configurations.

Dataflow	Design Point	Description
Naïve (intra-operator dataflow)	**Naive**	Intra-operator weight-stationary dataflow with fixed tile size, seen in many DNN accelerators [1][2][27]
Flex (intra-operator dataflow)	**Flex-opt**	We exhaustively search for optimal intra-operator dataflow, reflecting the optimal solution can be found in existing intra-operator MSEs/mappers [31][37][45]
FLAT (intra- and inter-operator dataflow)	**FLAT-opt**	We exhaustively search for optimal intra-operator as well as inter-operator dataflow, enabled by FLAT.

"HW resources" (i.e., FLOPs and off-chip memory bandwidth) as outlined in Table A.3 and sweep-and-explore other microarchitectural parameters relevant to the dataflow efficiency, including on-chip memory size and dataflow variations (Naive, Flex, FLAT). The on-chip memory size assesses the dataflow optimizations associated to the large intermediate tensor size in the attention layers. We demonstrate the benefits of FLAT across a range of hardware and dataflow configurations, without biasing to any specific design point.

(2) Concrete comparison of accelerator design points (§A.5): Next, we pick two specific hardware design points, namely a cloud and an edge accelerator, with headline HW resources that closely resemble a TPU-v3 [284] and edge-TPU [296], respectively. For baseline design, we use the best possible dataflow optimization in the intra-operator map space following prior techniques [6, 5, 68, 4, 7, 147, 284, 80, 81]. We call this performant baseline accelerator Flex-opt, against which we compare FLAT-opt, our proposed inter-operator and tiling dataflow optimization.

Table A.3: The HW resource configuration of cloud and edge accelerators in the evaluation sections.

Platform	Number of PEs	On-chip BW	Off-chip BW
Edge	32x32	1TB/sec	50GB/sec
Cloud	256x256	8TB/sec	400GB/sec

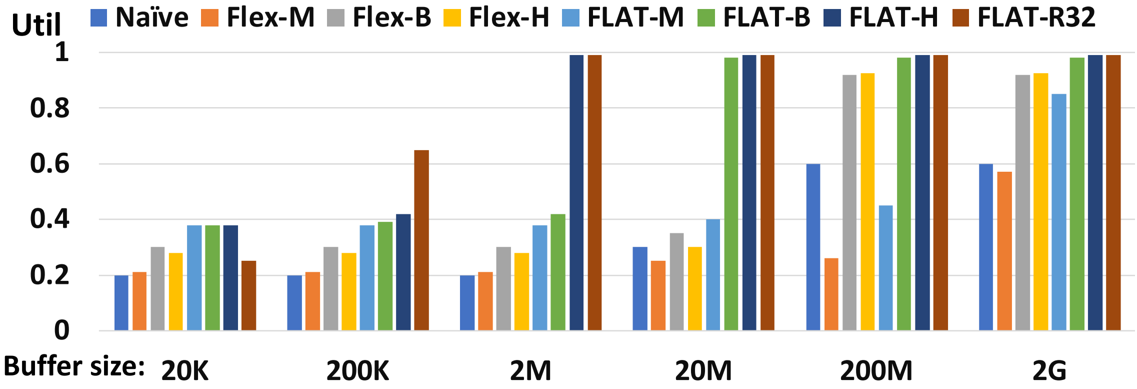


Fig. A.7: Compute utilization analysis of L-A operators under different dataflow granularities, running BERT with 512 sequence length with edge platform resources. Flex dataflow (Flex-X): Flex dataflow with X-granularity. FLAT dataflow (FLAT-X): Flex dataflow with X-granularity; X could be M (batch-Multi-head), B (Batch), H (head), or R (row).

A.4 Evaluation I: FLAT Dataflow Efficacy

A.4.1 Utilization

Edge accelerator. As described in §A.3.1, lower *Util* implies higher run time and lower throughput. Figure A.7 shows the utilization of different dataflow configurations (as outlined in Table A.2). The Naive dataflow barely achieves to 0.2 utilization for a buffer size ≤ 10 MB. Increasing the on-chip buffer to an excessive 100 MB yields a peak utilization of merely 0.6 for this dataflow. This low utilization is mainly attributed to the bandwidth-boundedness of attention operations when no dataflow optimization is employed.

Leveraging efficient staging of tensor in on-chip memory, Flex-M can potentially increase the utilization. However, when the on-chip buffer size is not sufficiently sized to house the intermediate tensors, the accelerator requires to fetch the intermediate tensors partially from the on-chip memory and the rest from off-chip memory breaking into the lim-

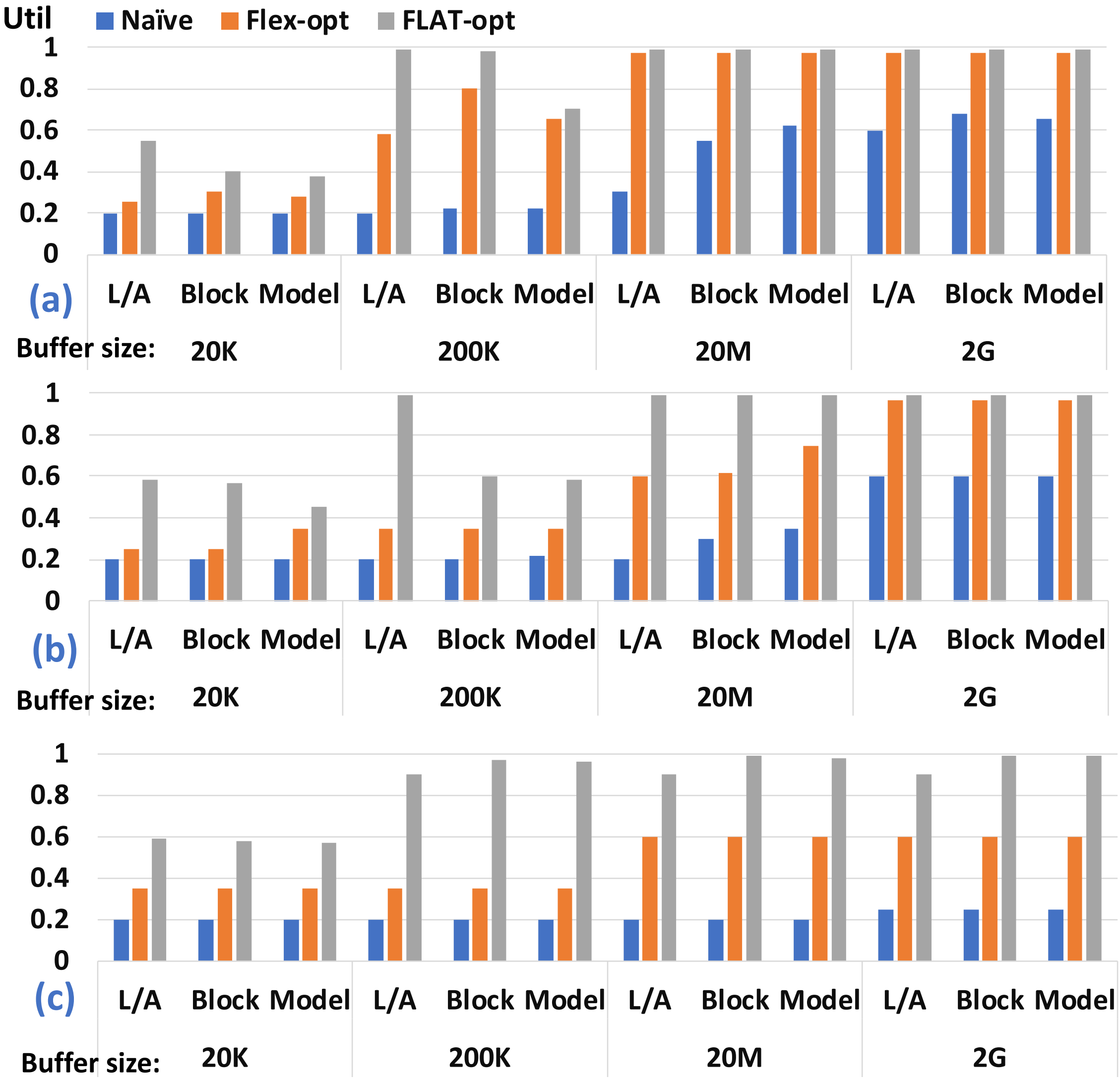


Fig. A.8: Comparisons of compute utilization across different sequence lengths running BERT under edge platform resources with sequence length of (a) 512, (b) 4K, and (c) 64K. We sweep the available on-chip buffer from size 20KB to 2GB. We list three level of performance analysis, L-A: focusing on performance difference at the L, A operators; Block: consider all operators in the attention block; and Model: a model-wise performance.

ited off-chip bandwidth wall. Compared to Naive dataflow, this extra pass between on-chip and off-chip memories diminishes the potential benefits of Flex-M under tightly-budgeted on-chip buffers scenarios. As we can see in Figure A.7, even increasing the on-chip buffer capacity to 2 GB is not adequate to fulfill the on-chip buffer size requirement for Flex-M. Flex-B and Flex-H are two variants with finer granularity of storing the intermediate results. They have smaller memory footprint for on-chip buffer that translates to a reduction in the buffer requirement to around 160 MB, while reaching to 0.92 utilization, comparing to 0.6 utilization with 100 MB on-chip storage with Naive dataflow.

Comparing to Flex which only exploits single operation execution (intra-operator), FLAT further enables cross-operator execution (inter-operator), which gains cross-operator data reuse opportunity and reduces the off-chip memory access. The fine-grained execution schemes of FLAT significantly reduces the on-chip buffer requirement, from 2 GB (Naive) and 160 MB (Flex-H) to just 0.6 MB (FLAT-Rx). These results show the effectiveness of cross-operator data reuse in mitigating the bandwidth-boundedness and improving the utilization from 0.6 (Naive) and 0.92 (Flex-H) to a nearly optimal value of 0.99 (FLAT-Rx).

Sensitivity to sequence length. Note that, Flex-opt and FLAT-opt represents the optimal design points in the Flex and FLAT dataflow design space, respectively. Figure A.8 and Figure A.9 compare the compute utilization of these dataflows by sweeping the sequence length for an edge and cloud accelerators, respectively. Flex design space exhaustively has both dataflows of state-of-the-art DNN accelerators [6, 5, 68, 4, 7, 147, 284, 80, 81] as well as their other relevant configurations. On the other hand, Flex-opt represents the most optimal design point across all the evaluations.

As Figure A.8 shows, FLAT-opt consistently outperforms Naive and Flex-opt. Analyzing the results indicate that though tensor-tensor fusion seems to be complicated and deemed as non-profitable, FLAT can efficiently execute tensor-tensor fusion in attention layers and harvest the highest performance gains. In Figure A.8, as the sequence length increases, the on-chip buffer requirement increases quickly (Table A.1). Under this scenario, most of the accelerator design points in Flex design space starts to hit the memory boundedness. However, applying the FLAT technique, we can effectively reduce the memory requirement and thus providing a better scalability to sequence length. At the optimal design point (FLAT-opt) reaches nearly 1.0 compute utilization with 10×-100× less on-chip buffer requirement, a scarce and critical hardware resource for accelerators, compared to Flex-opt.

Sensitivity to different operator fusion. So far, we analyze the operator fusion for only L and A operators, while keeping all the other attention operators non-fused (See the justification for this decision in §A.1.1). As shown in the first row of Figure A.8 from left to right, we observe that the effect of L/A operators are diluted as more operator fusions are unlocked. In attention-based models, FC/GEMM and attention operators, namely L and A, are generally the most dominant computation. For FC/GEMM, the typical single (intra-)operator dataflow is often sufficient to reach a high compute utilization, and hence FLAT-opt and Flex-opt performs equally well for these operators. As we can see, for the sequence length below 512, both Block-level and Model-level performance is dominated by FC/GEMM operators. Therefore, the gains from Flex-opt and FLAT-opt are immaterial. The significant gains from our approach emerge when the sequence length increases beyond 512 to 4K, 16K, and to 64K. Under these long-sequence lengths, the runtime contribution of L and A operators grows from 12% to 49%, 79%, and 94%, respectively. This increase causes our proposed FLAT-opt to outperform Flex-opt significantly even in Block and Model level scenarios.

Cloud accelerator. Figure A.9 shows the evaluation of different dataflow optimizations for a cloud platform when the sequence length ranges from 4K to 256K. For L/A, Flex-opt reaches to nearly 0.8 utilization, however with the contingency of a 2GB on-chip buffer requirement. Moreover, when the sequence length surpasses 16K, the peak utilization reduces to around 0.6. On the other hand, FLAT-opt reaches to nearly 1.0 utilization, while significantly reduces the on-chip buffer requirement by 100x-1000x comparing to Flex-opt.

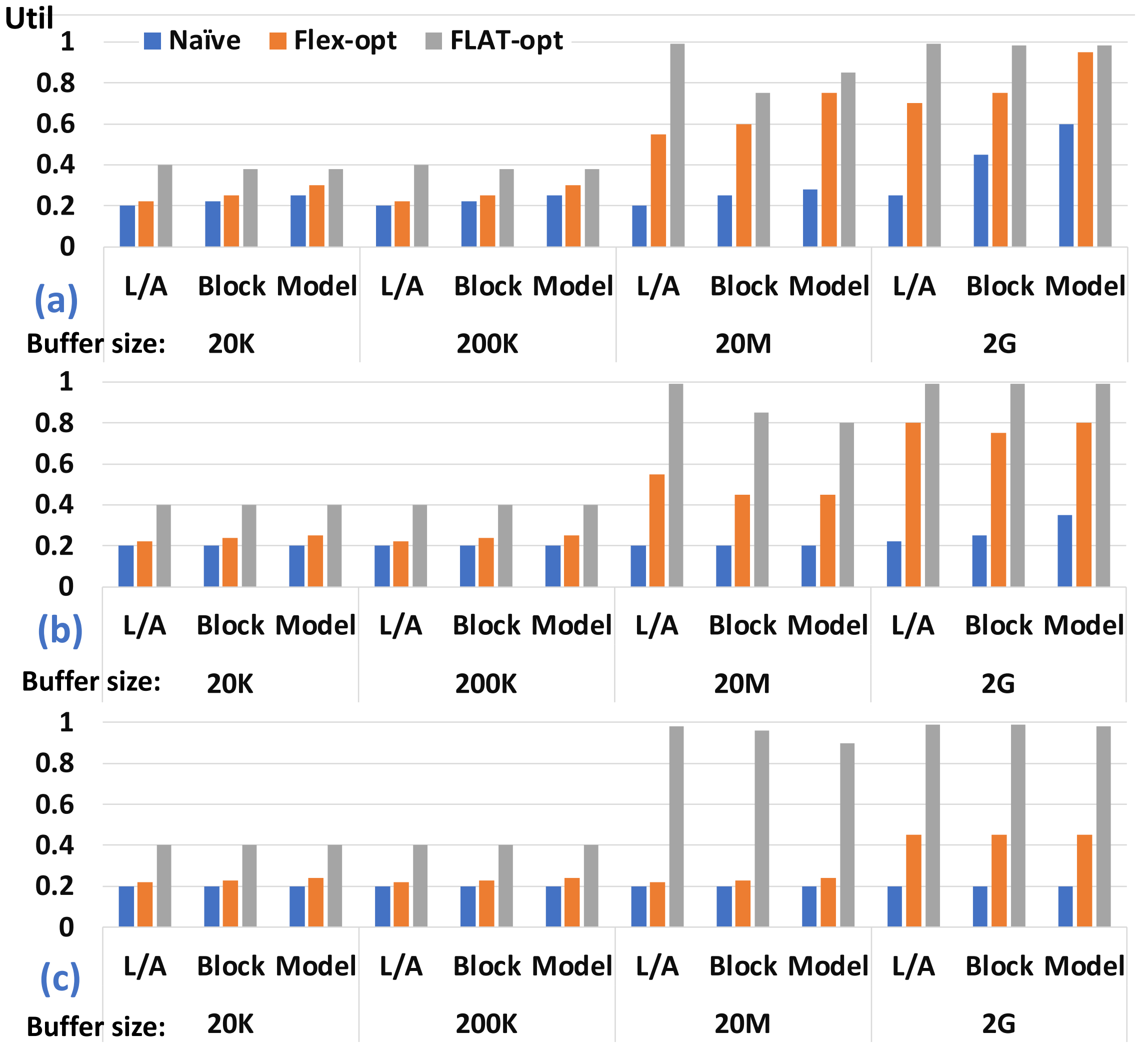


Fig. A.9: Comparisons of compute utilization across different sequence lengths running XLM under cloud platform resources with sequence length of (a) 4K, (b) 16K, (c) 256K.

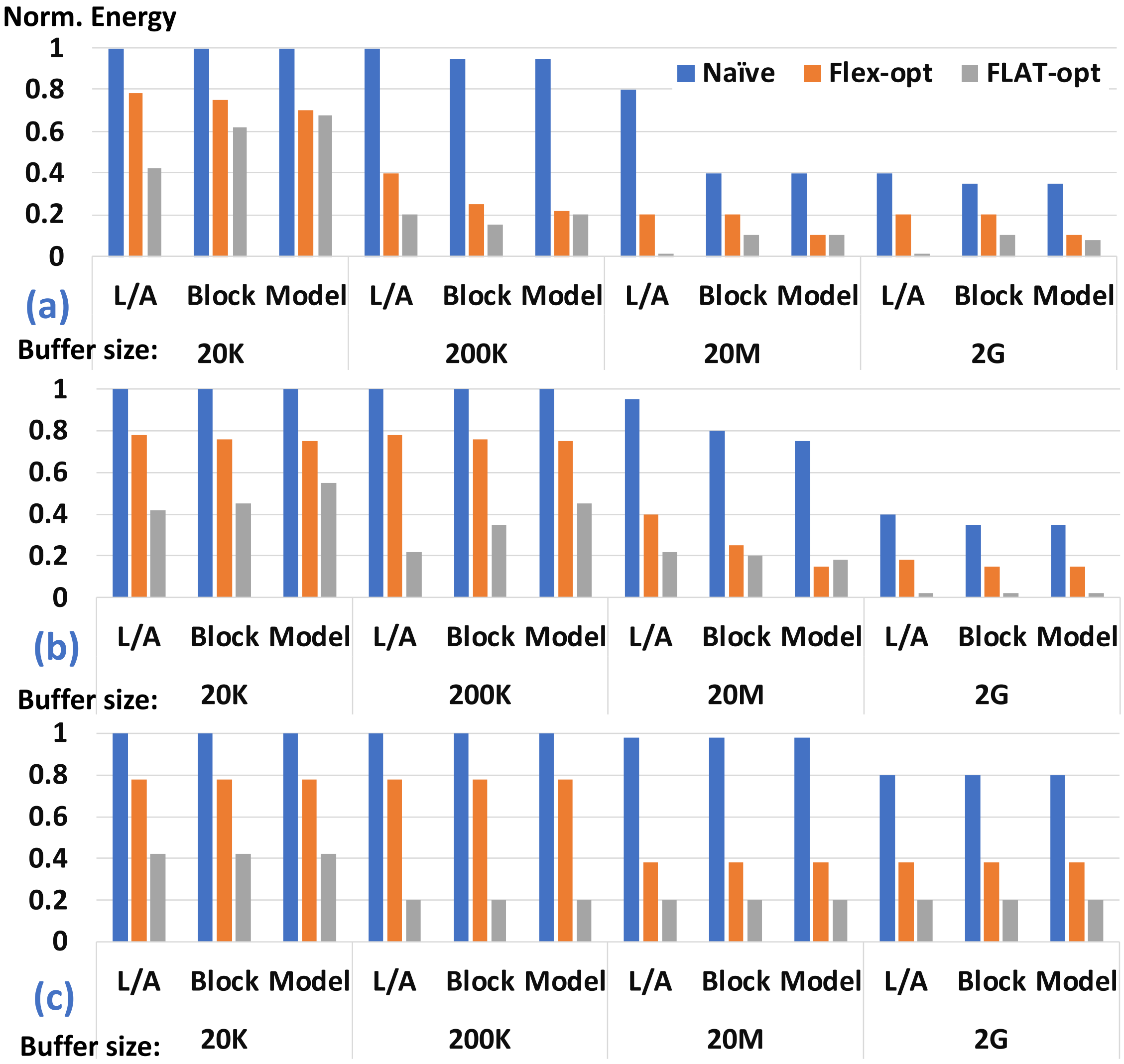


Fig. A.10: The corresponding energy consumption of each of the data-point in Figure A.8. The energy numbers are normalized by the largest energy number in each sub-plot.

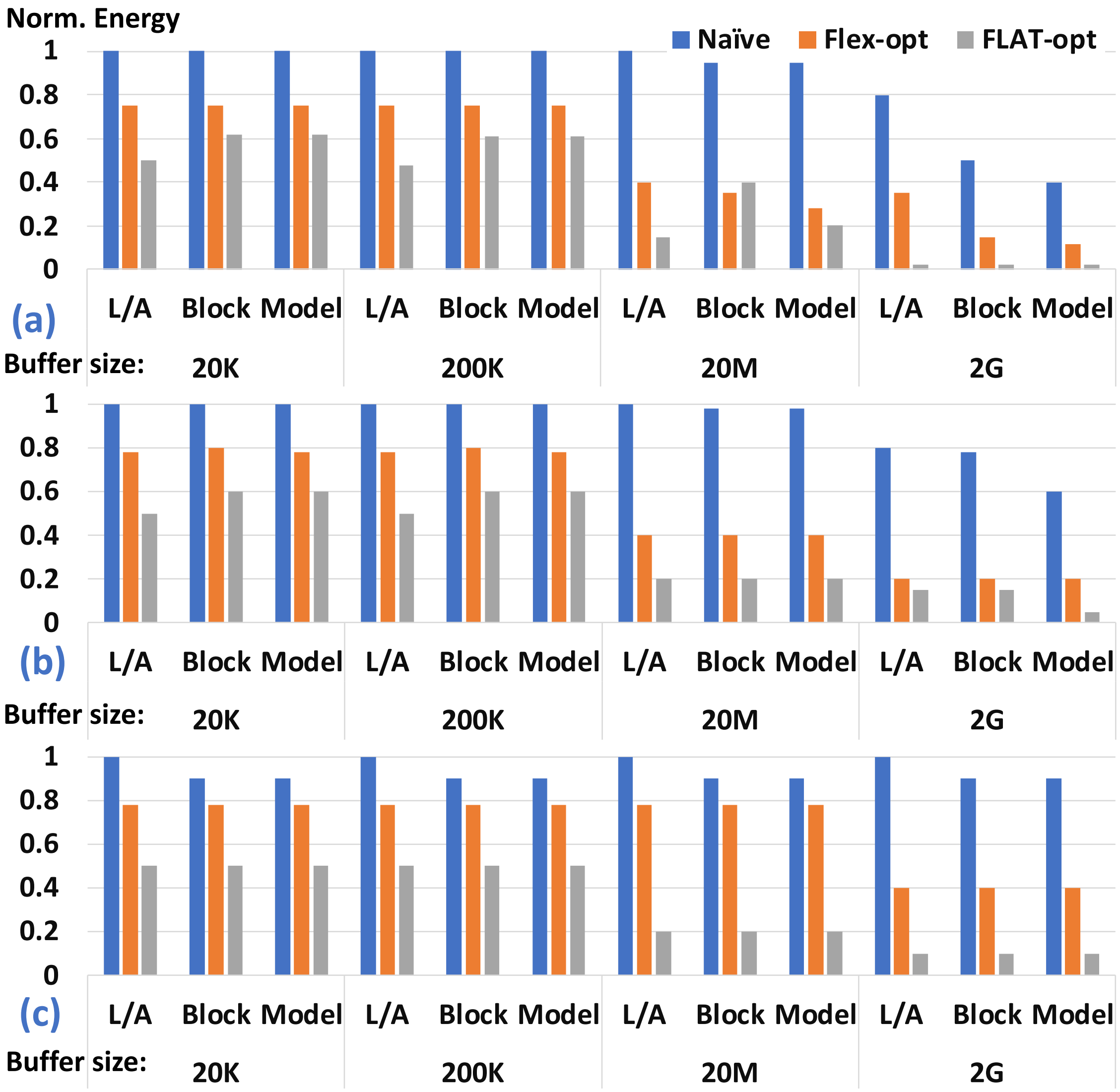


Fig. A.11: The corresponding energy consumption of each of the data-point in Figure A.9. The energy numbers are normalized by the largest energy number in each sub-plot.

A.4.2 Energy Consumption

Figure A.10 and Figure A.11 show the corresponding energy consumption for the data points in Figure A.8 and Figure A.9, respectively. It is worth to mention that high utilization does not directly translate to better energy savings; however, highly correlated. Data points with high compute utilization generally employ better memory access patterns (e.g., less off-chip memory access and better data reuse) and thus impose less cost in terms of memory access energy, the dominant contributor to the overall energy consumption of DNN accelerators. In Figure A.10 and Figure A.11, we observe that FLAT-opt reduces the energy consumption by around 1.5×-2.0× comparing to Flex-opt.

A.4.3 Map Space Exploration

Figure A.12 shows a holistic view of the entire design space of FLAT dataflow. The top-left corner of the diagram indicates high utilization with the least memory footprint. For each dataflow, there are abundance of parameters that can be tuned under different optimization objectives and design constraints. For example, while in this work, we focus on maximizing the compute utilization, one may choose other objectives such as maximizing utilization normalized to memory footprint size, leading to points in the top-left corner, or the least memory footprint size, leading to points in the left-most region. From Figure A.12, we can see that different dataflow configurations in the design space indeed represent notable differences in performance and memory requirement. This highlights the impact and importance of the design choices and dataflow optimizations.

A.5 Evaluation II: Accelerator Comparison

As shown in Table A.3, we pick two common hardware configuration design points [284, 289, 1, 295, 296] and evaluate them for both edge and cloud accelerators. We fix the on-chip buffer capacity to 512KB [296] and 32MB [284] for edge and cloud accelerators, respectively. Analyzing these accelerators across different dataflow spaces, namely Naive,

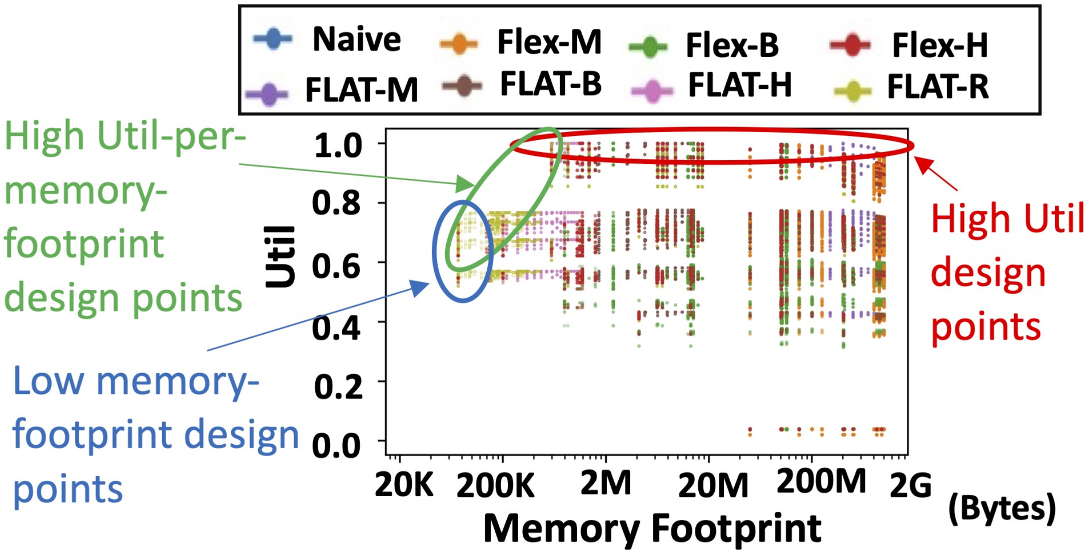
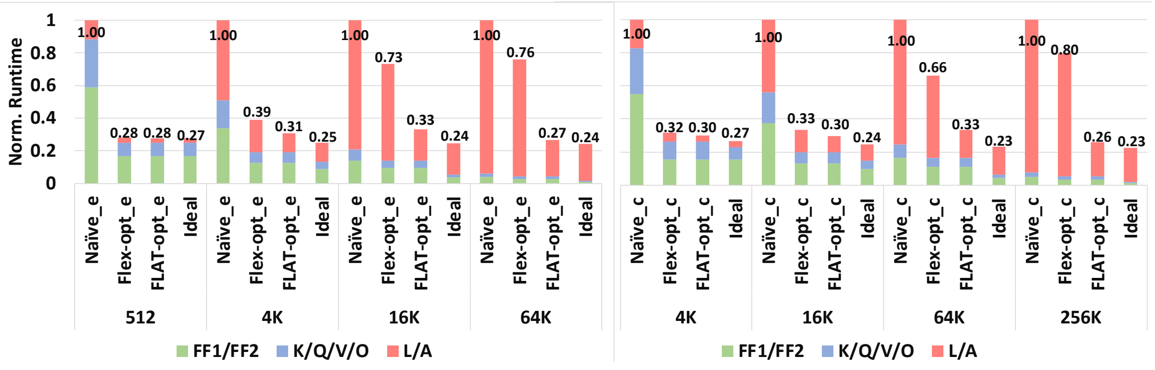


Fig. A.12: The design space of FLAT when running BERT with sequence length of 512 under edge platform resources. Flex dataflow (Flex-X): Flex dataflow with X-granularity. FLAT dataflow (FLAT-X): Flex dataflow with X-granularity, where X could be M (batch-Multi-head), B (Batch), H (head), or R (row). The design-point with the highest utilization, given a buffer constraint represents Flex-Opt and FLAT-Opt (Table A.2).

Flex, and FLAT, forms a concrete and reasonably realistic accelerator design space. Similar to previous sections, we call the optimal accelerator design point in each design space: Naive _edge, Flex-opt_edge, and FLAT-opt_edge for edge accelerator, and Naive _cloud, Flex-opt_cloud, and FLAT-opt_cloud for cloud accelerator, respectively.

Accelerator performance. As show in Figure A.13(a), Flex-opt_edge and FLAT-opt_edge share the same normalized runtime for K/Q/V/O and FF1/FF2. This similarity in performance is because in FLAT-opt_edge, both K/Q/V/O and FF1/FF2 are treated as non-fused operators, and hence the map space for them are the same as the one in Flex-opt_edge. In edge accelerator, when the sequence length is 512, FLAT-opt_edge and Flex-opt_edge both reach a near optimal performance. However, when the sequence length increases to 4K, 16K, and 64K, the performance gap between FLAT-opt_edge and Flex-opt_edge widen. For example, at sequence length of 64K, FLAT-opt_edge runs 2.8× faster than Flex-opt_edge, showing the efficiency of our dataflow optimization. In the cloud accelerator (Figure A.13(b)), the performance difference between FLAT-opt_cloud and Flex-opt_cloud exaggerates even further. For example, at sequence length of 64K, FLAT-opt_cloud runs



(a) (b)

Fig. A.13: End-to-end latency breakdown running: (a) BERT on Edge accelerator, (b) XLM on Cloud accelerator. Naive_e(dge): Edge with Naive dataflow. Flex-opt_e(dge): Edge with optimal Flex dataflow. FLAT-opt_e(dge): Edge with optimal FLAT dataflow. Similarly, for Naive_c(loud) and so on. Flex-opt represents the best possible (SOTA) intra-operator dataflow obtained through exhaustive search of the intra-operator map-space. FLAT performs inter-operator L-A fusion to relieve memory-bandwidth boundedness and achieves lower run time. While FLAT realizes benefits even at modest sequence lengths, the gains are pronounced at larger sequence lengths.

Edge	**Speedup (Ave. 1.75)**					**Energy Consumption Ratio (Ave. 0.56)**				
SeqLen	512	4K	16K	64K	256K	512	4K	16K	64K	256K
Bert	1.02	1.27	2.21	2.84	3.10	0.98	0.78	0.44	0.34	0.31
TrXL	1.02	1.23	2.06	2.75	3.07	0.98	0.81	0.48	0.35	0.31
FlauBert	1.01	1.11	1.62	2.26	2.67	1.00	0.90	0.61	0.43	0.36
T5	1.03	1.34	2.40	2.93	3.13	0.97	0.74	0.41	0.33	0.31
XLM	1.00	1.05	1.35	1.87	2.38	1.00	0.95	0.74	0.52	0.31
Average	**1.02**	**1.20**	**1.89**	**2.50**	**2.85**	**0.99**	**0.83**	**0.52**	**0.39**	**0.32**
Cloud	**Speedup (Ave. 1.65)**					**Energy Consumption Ratio (Ave. 0.45)**				
Bert	1.16	1.38	1.46	2.23	2.72	0.71	0.68	0.11	0.34	0.27
TrXL	1.13	1.34	1.45	2.20	2.71	0.73	0.27	0.13	0.35	0.27
FlauBert	1.07	1.21	1.42	2.21	2.93	0.87	0.80	0.72	0.49	0.37
T5	1.18	1.43	1.48	2.26	2.73	0.69	0.66	0.50	0.33	0.27
XLM	1.02	1.06	1.13	1.98	3.09	0.97	0.89	0.78	0.50	0.31
Average	**1.11**	**1.28**	**1.38**	**2.17**	**2.83**	**0.79**	**0.61**	**0.33**	**0.40**	**0.30**
Speedup and energy consumption ratio of: (left) FLAT-opt_e over Flex-opt_e, and (right) FLAT-opt_c over Flex-opt_c.										

Fig. A.14: The end-to-end speedup and energy-consumption ratio of FLAT-opt_e(dge) over Flex-opt_e(dge) and FLAT-opt_c(loud) over Flex-opt_c(loud) on different models.

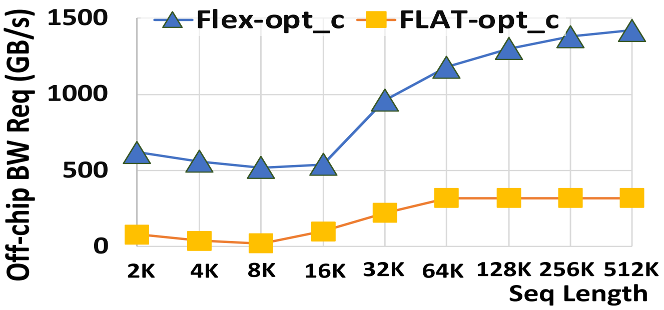


Fig. A.15: The required BW to reach compute utilization rate higher than 0.95 in the most BW-intensive L-A operator when running XLM using Flex-opt_c(loud) or FLAT-opt_c(loud).

3.07× faster than Flex-opt_cloud. That is partly because of the larger model size for the cloud accelerator that enables FLAT-opt_cloud to better utilize the on-chip hardware resources.

Comparisons across different models. Figure A.14 compares the performance of different dataflow optimizations across various transformer models. Compared to Flex-opt_edge, FLAT-opt_edge delivers 1.75× speedup in edge accelerator, while significantly reducing the energy consumption by 44%. In cloud accelerator, FLAT-opt_cloud achieves 1.65× speedup and 55% energy savings over Flex-opt_cloud. These results show the broad application of FLAT in improving the performance of various attention-based models under different design constraints.

Memory bandwidth requirement. Effectively using a limited off-chip bandwidth is an critical factor in the scalability of the hardware accelerator. That is because most DNN operations are often memory-bound and the off-chip memory bandwidth is often shared across different microarchitectural components in the system. In Figure A.15, we show the peak off-chip bandwidth requirement to achieve a compute utilization over 0.95 for L and A attention operators. The left hand side of the U-shape of Figure A.15 comes from the increase in the operational intensity and thus decrease of the bandwidth-boundedness as sequence length increases. The right hand side of the U-shape of Figure A.15 is caused by the quadratic and linear increase of on-chip memory requirement as sequence length increases for Flex and FLAT, respectively. On average, FLAT-opt_cloud reduces the off-chip bandwidth requirement by 82% against Flex-opt_cloud. Similarly, when evaluated under the edge scenario running BERT, FLAT-opt_edge achieves 71% reduction, on average, in the off-chip bandwidth requirement against Flex-opt_edge.

A.6 Background

Dataflow and mapping. Most work on DNN hardware dataflow techniques focus on individual CONV [80, 81, 83, 5, 4, 8, 12, 29, 10, 9, 13, 14, 15, 18, 19, 20] or GEMM [82, 21] operators. Some recent works consider fusion of multiple CONV operators and leveraged pipelined execution [285, 67]. Andrei et al. [297] studied operation fusion in Transformers– however, they only target operation fusion between MatMul operators and element-wise operators. Recently, Nvidia presented fused multi-head attention [298, 299]. Fusing multiple heads of the attention operators primarily involves adding an additional loop over the H *independent* heads, and considering the resultant operator as a single operator to explore intra-operator dataflow. This is captured by the Flex dataflow we evaluate. They, however, do not explore dependent MatMul-softmax-MatMul fusion, which is more complicated. FLAT targets such fusion and enables much higher performance.

Algorithmic optimization. Techniques such as quantization [300, 301, 302, 303], pruning [304, 39, 40, 41, 42], and distillation [212, 305, 306, 307] are used for compressing Attention-based models. There are a large body of algorithmic changes to attention mechanism [308, 309, 310, 311, 294], learned sparsity [312, 286, 313, 314] low-rank and kernel methods [315, 316, 317, 288], and others [70, 294, 287]. These techniques impact model quality and are orthogonal to the ideas developed in this paper. FLAT can be leveraged in association with these techniques when deployed on DNN accelerators to further improve run time and energy.

Attention accelerators. A^3 [318] and ELSA [319] propose dedicated attention accelerators and leverage approximate computation to accelerate attention layers, which trade-off run time performance with model quality. FLAT, by contrast, does not impact model quality, and is a generic yet powerful dataflow technique that can be leveraged on most existing accelerators.

Compiler optimizations. Fusion is a classic compiler technique widely employed in HPC [320, 321, 322, 323, 324] and ML compilers [24, 32, 325, 326]. However, in contrast to our work, ML compilers employ fusion in a limited fashion to fuse matrix operators (FC, CONV) with element-wise operators [327].

A.7 Summary

We identify that running attention-based models with long sequences is challenging because of low reuse in certain attention operators and quadratic growth of intermediate memory footprint, both of which compound memory bandwidth requirements. We propose FLAT, a novel dataflow for attention layers employing inter-operator fusion (the first work to investigate this for attention layers), interleaved execution, and efficient tiling to enhance the operational intensity and provide high compute utilization, reduced off-chip bandwidth requirements and scalability to long sequence lengths.

REFERENCES

[1] Y.-H. Chen, 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*, vol. 9, no. 2, pp. 292–308, 2019.

[2] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[3] E. Qin *et al.*, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training."

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

[5] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 2020, pp. 1–9.

[6] K. Hegde, P.-A. Tsai, S. Huang, V. Chandra, A. Parashar, and C. W. Fletcher, "Mind mappings: Enabling efficient algorithm-accelerator mapping space search extended abstract,"

[7] Q. Huang *et al.*, "Cosa: Scheduling by constrained optimization for spatial accelerators," *arXiv preprint arXiv:2105.01898*, 2021.

[8] Y. S. Shao *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *MICRO*, 2019, pp. 14–27.

[9] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "Dmazerunner: Executing perfectly nested loops on dataflow accelerators," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–27, 2019.

[10] X. Yang *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *ASPLOS*, 2020, pp. 369–383.

[11] P. Tillet, H.-T. Kung, and D. Cox, "Triton: An intermediate language and compiler for tiled neural network computations," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019, pp. 10–19.

[12] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, “Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks,” in *International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[13] M. Gao *et al.*, “Tangram: Optimized coarse-grained dataflow for scalable nn accelerators,” in *ASPLOS*, 2019, pp. 807–820.

[14] ——, “Tetris: Scalable and efficient neural network acceleration with 3d memory,” in *ASPLOS*, 2017, pp. 751–764.

[15] S. Venkataramani *et al.*, “Deeptools: Compiler and execution runtime extensions for rapid ai accelerator,” *IEEE Micro*, vol. 39, no. 5, pp. 102–111, 2019.

[16] N. Suda *et al.*, “Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks,” in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 16–25.

[17] Y. Shen, M. Ferdman, and P. Milder, “Maximizing cnn accelerator efficiency through resource partitioning,” in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2017, pp. 535–547.

[18] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing fpga-based accelerator design for deep convolutional neural networks,” in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 161–170.

[19] S. Venkataramani *et al.*, “Scaledeep: A scalable compute architecture for learning and evaluating deep networks,” in *MICRO*, 2017, pp. 13–26.

[20] L. Song *et al.*, “Hypar: Towards hybrid parallelism for deep learning accelerator array,” in *HPCA*, IEEE, 2019, pp. 56–68.

[21] X. Wei *et al.*, “Automated systolic array architecture synthesis for high throughput cnn inference on fpgas,” in *DAC*, 2017, pp. 1–6.

[22] M. Song *et al.*, “Towards efficient microarchitectural design for accelerating unsupervised gan-based deep learning,” in *HPCA*, IEEE, 2018, pp. 66–77.

[23] A. Stoutchinin, F. Conti, and L. Benini, “Optimally scheduling cnn convolutions for efficient memory access,” *arXiv preprint arXiv:1902.01492*, 2019.

[24] T. Chen *et al.*, “Learning to optimize tensor programs,” in *Advances in Neural Information Processing Systems*, 2018, pp. 3389–3400.

[25] Q. Xiao, S. Zheng, B. Wu, P. Xu, X. Qian, and Y. Liang, “Hasco: Towards agile hardware and software co-design for tensor computation,” in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2021, pp. 1055–1068.

[26] B. Reagen *et al.*, “A case for efficient accelerator design space exploration via bayesian optimization,” in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, IEEE, 2017, pp. 1–6.

[27] B. H. Ahn, P. Pilligundla, and H. Esmaeilzadeh, “Reinforcement learning and adaptive sampling for optimized dnn compilation,” *arXiv preprint arXiv:1905.12799*, 2019.

[28] S. Zheng, Y. Liang, S. Wang, R. Chen, and K. Sheng, “Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system,” in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 859–873.

[29] Z. Jia, M. Zaharia, and A. Aiken, “Beyond data and model parallelism for deep neural networks,” *arXiv preprint arXiv:1807.05358*, 2018.

[30] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, “Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines,” *Acm Sigplan Notices*, vol. 48, no. 6, pp. 519–530, 2013.

[31] N. Vasilache *et al.*, “Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions,” *arXiv preprint arXiv:1802.04730*, 2018.

[32] R. Baghdadi *et al.*, “Tiramisu: A polyhedral compiler for expressing fast and portable code,” in *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, IEEE, 2019, pp. 193–205.

[33] T. Grosser, H. Zheng, R. Aloor, A. Simbürger, A. Größlinger, and L.-N. Pouchet, “Polly-polyhedral optimization in llvm,” in *Proceedings of the First International Workshop on Polyhedral Compilation Techniques (IMPACT)*, vol. 2011, 2011, p. 1.

[34] M. Tan *et al.*, “Mnasnet: Platform-aware neural architecture search for mobile,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[35] H. Cai, L. Zhu, and S. Han, “Proxylessnas: Direct neural architecture search on target task and hardware,” *arXiv preprint arXiv:1812.00332*, 2018.

[36] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[37] C. Liu *et al.*, "Progressive neural architecture search," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 19–34.

[38] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[39] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," *arXiv preprint arXiv:2012.09852*, 2020.

[40] F.-M. Guo, S. Liu, F. S. Mungall, X. Lin, and Y. Wang, "Reweighted proximal pruning for large-scale language representation," *arXiv preprint arXiv:1909.12486*, 2019.

[41] Z. Wang, J. Wohlwend, and T. Lei, "Structured pruning of large language models," *arXiv preprint arXiv:1910.04732*, 2019.

[42] H. Sajjad, F. Dalvi, N. Durrani, and P. Nakov, "Poor man's bert: Smaller and faster transformer models," *arXiv preprint arXiv:2004.03844*, 2020.

[43] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[44] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[45] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[46] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, 2020.

[47] A. Parashar *et al.*, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, IEEE, 2019, pp. 304–315.

[48] Y.-H. Chen, 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*, vol. 9, no. 2, pp. 292–308, 2019.

[49] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *Proceedings of the*

Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), ACM, 2018, pp. 461–475.

[50] Y. Zhang, N. Zhang, T. Zhao, M. Vilim, M. Shahbaz, and K. Olukotun, "Sara: Scaling a reconfigurable dataflow accelerator," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2021, pp. 1041–1054.

[51] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[52] R. Wei, L. Schwartz, and V. Adve, "Dlvm: A modern compiler infrastructure for deep learning systems," *arXiv preprint arXiv:1711.03016*, 2017.

[53] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, "The tensor algebra compiler," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–29, 2017.

[54] A. Klöckner, "Loo. py: Transformation-based code generation for gpus and cpus," in *Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, 2014, pp. 82–87.

[55] M. Steuwer, T. Remmelg, and C. Dubach, "Lift: A functional data-parallel ir for high-performance gpu code generation," in *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, IEEE, 2017, pp. 74–85.

[56] N. D. Lane *et al.*, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, IEEE, 2016, pp. 1–12.

[57] Y. Ma, 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*, 2017, pp. 45–54.

[58] *Maestro tool*, http://maestro.ece.gatech.edu/, 2020.

[59] *Gamma tool*, http://https://github.com/maestro-project/gamma/, 2020.

[60] X. Yang *et al.*, "Dnn dataflow choice is overrated," *arXiv preprint arXiv:1809.04070*, 2018.

[61] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance fpga-based accelerator for large-scale convolutional neural networks," in *2016 26th*

International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2016, pp. 1–9.

[62] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, “Energy-efficient cnn implementation on a deeply pipelined fpga cluster,” in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, 2016, pp. 326–331.

[63] W. Jiang, E. H.-M. Sha, Q. Zhuge, L. Yang, X. Chen, and J. Hu, “Heterogeneous fpga-based cost-optimal design for timing-constrained cnns,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2542–2554, 2018.

[64] W. Jiang *et al.*, “Xfer: A novel design to achieve super-linear performance on multiple fpgas for real-time ai,” in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 305–305.

[65] X. Zhang *et al.*, “Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas,” in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2018, pp. 1–8.

[66] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, “Cloud-dnn: An open framework for mapping dnn models to cloud fpgas,” in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 73–82.

[67] X. Wei, Y. Liang, X. Li, C. H. Yu, P. Zhang, and J. Cong, “Tgpa: Tile-grained pipeline architecture for low latency cnn inference,” in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[68] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, “Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

[69] A. Vaswani *et al.*, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[70] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, “Transformer-xl: Attentive language models beyond a fixed-length context,” *arXiv preprint arXiv:1901.02860*, 2019.

[71] G. Lample and A. Conneau, “Cross-lingual language model pretraining,” *arXiv preprint arXiv:1901.07291*, 2019.

[72] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable detr: Deformable transformers for end-to-end object detection," *arXiv preprint arXiv:2010.04159*, 2020.

[73] P. Bhattacharyya, C. Huang, and K. Czarnecki, "Self-attention based context-aware 3d object detection," *arXiv preprint arXiv:2101.02672*, 2021.

[74] P. Sun *et al.*, "Transtrack: Multiple-object tracking with transformer," *arXiv preprint arXiv:2012.15460*, 2020.

[75] M. Chen *et al.*, "Generative pretraining from pixels," in *International Conference on Machine Learning*, PMLR, 2020, pp. 1691–1703.

[76] N. Parmar *et al.*, "Image transformer," *arXiv preprint arXiv:1802.05751*, 2018.

[77] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," *arXiv preprint arXiv:2012.09841*, 2020.

[78] C.-Z. A. Huang *et al.*, "Music transformer: Generating music with long-term structure," in *International Conference on Learning Representations*, 2018.

[79] W.-Y. Hsiao, J.-Y. Liu, Y.-C. Yeh, and Y.-H. Yang, "Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs," *arXiv preprint arXiv:2101.02402*, 2021.

[80] *Nvdla deep learning accelerator*, http://nvdla.org, 2017.

[81] Z. Du *et al.*, "Shidiannao: Shifting vision processing closer to the sensor," in *International Symposium on Computer Architecture (ISCA)*, 2015.

[82] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *International Symposium on Computer Architecture (ISCA)*, IEEE, 2017, pp. 1–12.

[83] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *JSSC*, vol. 52, no. 1, pp. 127–138, 2016.

[84] M. Pellauer *et al.*, "Buffets: An efficient and composable storage idiom for explicit decoupled data orchestration," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 137–151.

[85] *Definition of affine transform*, https://en.wikipedia.org/wiki/Affine_transformation, 2021.

[86] N. Ahmed, N. Mateev, and K. Pingali, “Tiling imperfectly-nested loop nests,” in *SC’00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, IEEE, 2000, pp. 31–31.

[87] A. Hartono *et al.*, “Parametric multi-level tiling of imperfectly nested loops,” in *Proceedings of the 23rd international conference on Supercomputing*, 2009, pp. 147–157.

[88] T. Krishna, H. Kwon, A. Parashar, M. Pellauer, and A. Samajdar, “Data orchestration in deep learning accelerators,” *Synthesis Lectures on Computer Architecture*, vol. 15, no. 3, pp. 1–164, 2020.

[89] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, “A dynamically configurable coprocessor for convolutional neural networks,” in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 247–257.

[90] C. Farabet *et al.*, “Neuflow: A runtime reconfigurable dataflow processor for vision,” in *Cvpr 2011 Workshops*, IEEE, 2011, pp. 109–116.

[91] S. Park *et al.*, “4.6 a1. 93tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big-data applications,” in *2015 ISSCC Digest of Technical Papers*, IEEE, 2015, pp. 1–3.

[92] L. Cavigelli *et al.*, “Origami: A convolutional network accelerator,” in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, 2015, pp. 199–204.

[93] S. Gupta *et al.*, “Deep learning with limited numerical precision,” in *ICML*, 2015, pp. 1737–1746.

[94] M. Peemen *et al.*, “Memory-centric accelerator design for convolutional neural networks,” in *31st ICCD*, IEEE, 2013, pp. 13–19.

[95] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.

[96] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, “An empirical evaluation of deep architectures on problems with many factors of variation,” in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 473–480.

[97] N. Pinto, D. Doukhan, J. J. DiCarlo, and D. D. Cox, “A high-throughput screening approach to discovering good forms of biologically inspired visual representation,” *PLoS computational biology*, vol. 5, no. 11, 2009.

[98] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[99] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[100] S. Zhong, Y. Shen, and F. Hao, "Tuning compiler optimization options via simulated annealing," in *2009 Second International Conference on Future Information Technology and Management Engineering*, IEEE, 2009, pp. 305–308.

[101] T. Chen *et al.*, "Tvm: An automated end-to-end optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[102] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[103] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic superoptimization," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 305–316, 2013.

[104] M. Pelikan, D. E. Goldberg, E. Cantú-Paz, *et al.*, "Boa: The bayesian optimization algorithm," in *Proceedings of the genetic and evolutionary computation conference GECCO-99*, vol. 1, 1999, pp. 525–532.

[105] D. Stamoulis, E. Cai, D.-C. Juan, and D. Marculescu, "Hyperpower: Power-and memory-constrained hyper-parameter optimization for neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2018, pp. 19–24.

[106] D. Stamoulis *et al.*, "Designing adaptive neural networks for energy-constrained image classification," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[107] M. Parsa, A. Ankit, A. Ziabari, and K. Roy, "Pabo: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design," *arXiv preprint arXiv:1906.08167*, 2019.

[108] K. V. Price, "Differential evolution," in *Handbook of Optimization*, Springer, 2013, pp. 187–214.

[109] K. Price *et al.*, *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[110] S. Das *et al.*, "Recent advances in differential evolution–an updated survey," *Swarm and Evolutionary Computation*, vol. 27, pp. 1–30, 2016.

[111] A. K. Qin *et al.*, "Self-adaptive differential evolution algorithm for numerical optimization," in *2005 IEEE Congress on Evolutionary Computation*, vol. 2, 2005, 1785–1791 Vol. 2.

[112] S. Das *et al.*, "Automatic clustering using an improved differential evolution algorithm," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 38, no. 1, pp. 218–237, 2008.

[113] I. Rechenberg, "Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. frommann-holzbog, stuttgart, 1973," *Step-Size Adaptation Based on Non-Local Use of Selection Information. In PPSN3*, 1994.

[114] N. Hansen, "The cma evolution strategy: A comparing review," in *Towards a new evolutionary computation*, Springer, 2006, pp. 75–102.

[115] N. Hansen *et al.*, "Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009," in *Proceedings of the 12th GECCO*, 2010, pp. 1689–1696.

[116] M. Hellwig and H.-G. Beyer, "Evolution under strong noise: A self-adaptive evolution strategy can reach the lower performance bound-the pccmsa-es," in *International Conference on Parallel Problem Solving from Nature*, Springer, 2016, pp. 26–36.

[117] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, IEEE, vol. 4, 1995, pp. 1942–1948.

[118] R. Kuo *et al.*, "Application of particle swarm optimization algorithm for solving bi-level linear programming problem," *Computers & Mathematics with Applications*, vol. 58, no. 4, pp. 678–685, 2009.

[119] R. J. Kuo *et al.*, "Application of particle swarm optimization to association rule mining," *Applied Soft Computing*, vol. 11, no. 1, pp. 326–336, 2011.

[120] C. C. Cox, "A comparison of active and passive portfolio management," 2017.

[121] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, "Deep reinforcement learning for general video game ai," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE, 2018, pp. 1–8.

[122] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[123] H. Kwon, M. Pellauer, A. Parashar, and T. Krishna, "Flexion: A quantitative metric for flexibility in dnn accelerators," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 1–4, 2020.

[124] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[125] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[126] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

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

[128] *Https://https://github.com/georgia-tech-synergy-lab/maeri*.

[129] S. G. B. H. A. Joon *et al.*, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks,"

[130] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized coarse-grained dataflow for scalable nn accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 807–820.

[131] M. Putic, S. Venkataramani, S. Eldridge, A. Buyuktosunoglu, P. Bose, and M. Stan, "Dyhard-dnn: Even more dnn acceleration with dynamic hardware reconfiguration," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[132] N. P. Jouppi *et al.*, "A domain-specific supercomputer for training deep neural networks," *Communications of the ACM*, vol. 63, no. 7, pp. 67–78, 2020.

[133] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "Dmazerunner: Executing perfectly nested loops on dataflow accelerators," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–27, 2019.

[134] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *ICCAD*, 2020.

[135] P. Chatarasi *et al.*, "Marvel: A data-centric compiler for dnn operators on spatial accelerators," *arXiv preprint arXiv:2002.07752*, 2020.

[136] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–12.

[137] S. Winograd, "Arithmetic complexity of computations," in *Cyclic Convolution and Discrete Fourier Transform*, ch. 7, pp. 71–92.

[138] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. W. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18, Los Angeles, California, 2018, pp. 674–687.

[139] *Freepdk15:contents*, https://www.eda.ncsu.edu/wiki/FreePDK15:Contents, 2019.

[140] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," 2013.

[141] M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[142] K. He, 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*, 2016, pp. 770–778.

[143] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[144] M. Naumov *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[145] "Cuda c++ programming guide," 2021.

[146] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[147] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, 2016, 262–263.

[148] D. P. Kingma *et al.*, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[149] I. Sutskever *et al.*, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[150] R. C. Corrêa, A. Ferreira, and P. Rebreyend, "Scheduling multiprocessor tasks with genetic algorithms," *IEEE Transactions on Parallel and Distributed systems*, vol. 10, no. 8, pp. 825–837, 1999.

[151] E. S. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed systems*, vol. 5, no. 2, pp. 113–120, 1994.

[152] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," in *5th IEEE heterogeneous computing workshop (HCW'96)*, 1996, pp. 86–97.

[153] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," in *5th Heterogeneous Computing Workshop (HCW'96)*, 1996, pp. 98–117.

[154] L. Wang, H. J. Siegel, and V. P. Roychowdhury, "A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments," in *Proc. Heterogeneous Computing Workshop*, 1996, pp. 72–85.

[155] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[156] ——, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[157] J. Rapin and O. Teytaud, *Nevergrad - A gradient-free optimization platform*, https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[158] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2014.

[159] C. Zhang *et al.*, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *FPGA'15*, 2015, pp. 161–170.

[160] S.-C. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *2020 53rd Annual*

IEEE/ACM International Symposium on Microarchitecture (MICRO), IEEE, 2020, pp. 622–636.

[161] S.-C. Kao, M. Pellauer, A. Parashar, and T. Krishna, "Digamma: Domain-aware genetic algorithm for hw-mapping co-optimization for dnn accelerators," *Design, Automation and Test in Europe Conference (DATE)*, 2022.

[162] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[163] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1160–1174, 2021.

[164] S.-C. Kao, S. Subramanian, G. Agrawal, and T. Krishna, "An optimized dataflow for mitigating attention performance bottlenecks," *arXiv preprint arXiv:2107.06419*, 2021.

[165] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, "Sparseloop: An analytical, energy-focused design space exploration methodology for sparse tensor accelerators," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2021, pp. 232–234.

[166] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[167] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[168] S.-C. Kao and T. Krishna, "Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores," *IEEE International Symposium on High-Performance Computer Architecture (HPCA-28)*, 2022.

[169] C.-E. Lee *et al.*, "Stitch-x: An accelerator architecture for exploiting unstructured sparsity in deep neural networks," in *SysML Conference*, vol. 120, 2018.

[170] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "Snap: An efficient sparse neural acceleration processor for unstructured sparse deep neural network inference," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 2, pp. 636–647, 2020.

[171] T.-H. Yang *et al.*, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 236–249.

[172] A. Parashar *et al.*, “Scnn: An accelerator for compressed-sparse convolutional neural networks,” in *International Symposium on Computer Architecture (ISCA)*, 2017, pp. 27–40.

[173] S. Zhang *et al.*, “Cambricon-x: An accelerator for sparse neural networks,” in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–12.

[174] K. Kanellopoulos *et al.*, “Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations,” in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 600–614.

[175] S.-C. Kao and T. Krishna, “E3: A hw/sw co-design neuroevolution platform for autonomous learning in edge device,” in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2021, pp. 288–298.

[176] S. Pal *et al.*, “Outerspace: An outer product based sparse matrix multiplication accelerator,” in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2018, pp. 724–736.

[177] A. Mirhoseini *et al.*, “A graph placement methodology for fast chip design,” *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[178] Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in *International Symposium on Computer Architecture (ISCA)*, 2016.

[179] F. Nogueira, *Bayesian Optimization: Open source constrained global optimization tool for Python*, 2014–.

[180] C. Hao *et al.*, “Fpga/dnn co-design: An efficient design methodology for 1ot intelligence on the edge,” in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2019, pp. 1–6.

[181] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar, “Design space exploration of fpga-based accelerators with multi-level parallelism,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, IEEE, 2017, pp. 1141–1146.

[182] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, “Design space exploration of fpga-based deep convolutional neural networks,” in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2016, pp. 575–580.

[183] W. Jiang *et al.*, “Achieving super-linear speedup across multi-fpga for real-time dnn inference,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–23, 2019.

[184] G. Santoro, M. R. Casu, V. Peluso, A. Calimera, and M. Alioto, “Energy-performance design exploration of a low-power microprogrammed deep-learning accelerator,” in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2018, pp. 1151–1154.

[185] B. Reagen *et al.*, “Minerva: Enabling low-power, highly-accurate deep neural network accelerators,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2016, pp. 267–278.

[186] Y. S. Shao, S. L. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks, “Co-designing accelerators and soc interfaces using gem5-aladdin,” in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–12.

[187] J. Cong and J. Wang, “Polysa: Polyhedral-based systolic array auto-compilation,” in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2018, pp. 1–8.

[188] X. Wei *et al.*, “Automated systolic array architecture synthesis for high throughput cnn inference on fpgas,” in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[189] S.-C. Kao and T. Krishna, “Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning,” in *2020 53th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2020, pp. 1–14.

[190] A. Mirhoseini *et al.*, “Chip placement with deep reinforcement learning,” *arXiv preprint arXiv:2004.10746*, 2020.

[191] H. Wang *et al.*, “Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[192] Cerebras, *Cerebras cs-1*, 2020.

[193] Google, *Cloud tpu*, 2020.

[194] E. Baek, D. Kwon, and J. Kim, “A multi-neural network acceleration architecture,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2020, pp. 940–953.

[195] N. P. Jouppi *et al.*, “A domain-specific supercomputer for training deep neural networks,” *Commun. ACM*, vol. 63, no. 7, pp. 67–78, Jun. 2020.

[196] H. Kwon, L. Lai, T. Krishna, and V. Chandra, “Herald: Optimizing heterogeneous dnn accelerators for edge devices,” *arXiv preprint arXiv:1909.07437*, 2019.

[197] Y. Choi and M. Rhu, “Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units,” in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2020, pp. 220–233.

[198] V. Mnih *et al.*, “Asynchronous methods for deep reinforcement learning,” in *International conference on machine learning*, 2016, pp. 1928–1937.

[199] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[200] *Pytorch.* https://pytorch.org/, 2020.

[201] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[202] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[203] C. Szegedy *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[204] K. He, 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*, 2016, pp. 770–778.

[205] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[206] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[207] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.

[208] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[209] M. Lewis *et al.*, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[210] L. Martin *et al.*, "Camembert: A tasty french language model," *arXiv preprint arXiv:1911.03894*, 2019.

[211] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher, "Ctrl: A conditional transformer language model for controllable generation," *arXiv preprint arXiv:1909.05858*, 2019.

[212] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[213] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[214] H. Le *et al.*, "Flaubert: Unsupervised language model pre-training for french," *arXiv preprint arXiv:1912.05372*, 2019.

[215] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[216] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving language understanding by generative pre-training*, 2018.

[217] M. Junczys-Dowmunt *et al.*, "Marian: Fast neural machine translation in C++," in *Proceedings of ACL 2018, System Demonstrations*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 116–121.

[218] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[219] Y. Jernite, *Explain anything like i'm five: A model for open domain long form question answering*, 2020.

[220] Y. Liu *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[221] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

[222] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[223] G. Lample and A. Conneau, "Cross-lingual language model pretraining," *arXiv preprint arXiv:1901.07291*, 2019.

[224] A. Conneau *et al.*, "Unsupervised cross-lingual representation learning at scale," *arXiv preprint arXiv:1911.02116*, 2019.

[225] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in neural information processing systems*, 2019, pp. 5753–5763.

[226] G. Zhou *et al.*, "Deep interest network for click-through rate prediction," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1059–1068.

[227] G. Zhou *et al.*, "Deep interest evolution network for click-through rate prediction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 5941–5948.

[228] D. Kalamkar, E. Georganas, S. Srinivasan, J. Chen, M. Shiryaev, and A. Heinecke, "Optimizing deep learning recommender systems' training on cpu cluster architectures," *arXiv preprint arXiv:2005.04680*, 2020.

[229] H.-T. Cheng *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st workshop on deep learning for recommender systems*, 2016, pp. 7–10.

[230] U. Gupta *et al.*, "Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference," *arXiv preprint arXiv:2001.02772*, 2020.

[231] X. Zhu, S. Chen, S. Chen, and G. Yang, "Energy and delay co-aware computation offloading with deep learning in fog computing networks," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, IEEE, 2019, pp. 1–6.

[232] F. Fu, Y. Kang, Z. Zhang, F. R. Yu, and T. Wu, "Soft actor-critic drl for live transcoding and streaming in vehicular fog computing-enabled iov," *IEEE Internet of Things Journal*, 2020.

[233] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, "Intelligent resource allocation in dynamic fog computing environments," in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, IEEE, 2019, pp. 1–7.

[234] H. Du, S. Leng, F. Wu, X. Chen, and S. Mao, "A new vehicular fog computing architecture for cooperative sensing of autonomous driving," *IEEE Access*, vol. 8, pp. 10 997–11 006, 2020.

[235] X. Li, Y. Qin, H. Zhou, D. Chen, S. Yang, and Z. Zhang, "An intelligent adaptive algorithm for servers balancing and tasks scheduling over mobile fog computing networks," *Wireless Communications and Mobile Computing*, vol. 2020, 2020.

[236] Transcend, *The tranfer rate of ddr1, ddr2, ddr3, and ddr4*, 2020.

[237] PCI-SIG, *Pce spec*, 2020.

[238] Micron, *Ddr5 sdram*, 2020.

[239] H. Jun *et al.*, "Hbm (high bandwidth memory) dram technology and architecture," in *2017 IEEE International Memory Workshop (IMW)*, IEEE, 2017, pp. 1–4.

[240] E. S. Chung *et al.*, "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[241] S. Bang *et al.*, "14.7 a 288 u w programmable deep-learning processor with 270kb on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, IEEE, 2017, pp. 250–251.

[242] S. Yin *et al.*, "A 141 uw, 2.46 pj/neuron binarized convolutional neural network based self-learning speech recognition processor in 28nm cmos," in *2018 IEEE Symposium on VLSI Circuits*, IEEE, 2018, pp. 139–140.

[243] S. Zheng *et al.*, "An ultra-low power binarized convolutional neural network-based speech recognition processor with on-chip self-learning," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 12, pp. 4648–4661, 2019.

[244] Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, and L. Tang, "Prophet: Precise qos prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 17–32.

[245] M. Cheong, H. Lee, I. Yeom, and H. Woo, “Scarl: Attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster,” *IEEE Access*, vol. 7, pp. 153 432–153 444, 2019.

[246] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Job scheduling for multi-user mapreduce clusters,” Technical Report UCB/EECS-2009-55, EECS Department, University of California . . ., Tech. Rep., 2009.

[247] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, “Libra: A computational economy-based job scheduling system for clusters,” *Software: Practice and Experience*, vol. 34, no. 6, pp. 573–590, 2004.

[248] T. Beisel, T. Wiersema, C. Plessl, and A. Brinkmann, “Cooperative multitasking for heterogeneous accelerators in the linux completely fair scheduler,” in *ASAP 2011-22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*, IEEE, 2011, pp. 223–226.

[249] W. Joo and D. Shin, “Resource-constrained spatial multi-tasking for embedded gpu,” in *2014 IEEE International Conference on Consumer Electronics (ICCE)*, IEEE, 2014, pp. 339–340.

[250] Z. Wu, Z. Ni, L. Gu, and X. Liu, “A revised discrete particle swarm optimization for cloud workflow scheduling,” in *2010 International Conference on Computational Intelligence and Security*, IEEE, 2010, pp. 184–188.

[251] G. Emadi, A. M. Rahmani, and H. Shahhoseini, “Task scheduling algorithm using covariance matrix adaptation evolution strategy (cma-es) in cloud computing,” *Journal of Advances in Computer Engineering and Technology*, vol. 3, no. 3, pp. 135–144, 2017.

[252] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016, pp. 50–56.

[253] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, “Learning scheduling algorithms for data processing clusters,” in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

[254] L. Thamsen, B. Rabier, F. Schmidt, T. Renner, and O. Kao, “Scheduling recurring distributed dataflow jobs based on resource utilization and interference,” in *2017 IEEE International Congress on Big Data (BigData Congress)*, IEEE, 2017, pp. 145–152.

[255] *Stars and bars (combinatorics)*, https://en.wikipedia.org/wiki/Stars_and_bars_(combinatorics).

[256] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[257] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, “Hardware accelerated convolutional neural networks for synthetic vision systems,” in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, IEEE, 2010, pp. 257–260.

[258] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, “Cnp: An fpga-based processor for convolutional networks,” in *2009 International Conference on Field Programmable Logic and Applications*, IEEE, 2009, pp. 32–37.

[259] M. Sankaradas *et al.*, “A massively parallel coprocessor for convolutional neural networks,” in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, IEEE, 2009, pp. 53–60.

[260] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[261] S. Fujimoto, H. Van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” *arXiv preprint arXiv:1802.09477*, 2018.

[262] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *arXiv preprint arXiv:1801.01290*, 2018.

[263] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, “Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation,” in *Advances in neural information processing systems*, 2017, pp. 5279–5288.

[264] *Openai gym*, https://gym.openai.com/, 2016.

[265] I. Popov *et al.*, “Data-efficient deep reinforcement learning for dexterous manipulation,” *arXiv preprint arXiv:1704.03073*, 2017.

[266] Y. Wu and Y. Tian, “Training agent for first-person shooter game with actor-critic curriculum learning,” 2016.

[267] Y. Wu *et al.*, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *arXiv preprint arXiv:1609.08144*, 2016.

[268] A. Vaswani *et al.*, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[269] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[270] R. C. Whaley and J. J. Dongarra, "Automatically tuned linear algebra software," in *SC'98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, IEEE, 1998, pp. 38–38.

[271] D. Novillo, "Samplepgo-the power of profile guided optimizations without the usability burden," in *2014 LLVM Compiler Infrastructure in HPC*, IEEE, 2014, pp. 22–28.

[272] P. P. Chang, S. A. Mahlke, and W.-M. W. Hwu, "Using profile information to assist classic code optimizations," *Software: Practice and Experience*, vol. 21, no. 12, pp. 1301–1321, 1991.

[273] W. Jiang *et al.*, "Hardware/software co-exploration of neural architectures," *arXiv preprint arXiv:1907.04650*, 2019.

[274] M. S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, and N. D. Lane, "Best of both worlds: Automl codesign of a cnn and its hardware accelerator," *arXiv preprint arXiv:2002.05022*, 2020.

[275] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On neural architecture search for resource-constrained hardware platforms," *arXiv preprint arXiv:1911.00105*, 2019.

[276] L. Yang *et al.*, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," *arXiv preprint arXiv:2002.04116*, 2020.

[277] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[278] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[279] W. Yonghui *et al.*, "Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[280] A. Mirhoseini *et al.*, "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 2430–2439.

[281] Y. Gao, L. Chen, and B. Li, "Spotlight: Optimizing device placement for training deep neural networks," in *International Conference on Machine Learning*, 2018, pp. 1676–1684.

[282] A. Paliwal *et al.*, "Reinforced genetic algorithm learning for optimizing computation graphs," 2020.

[283] S.-C. Kao, X. Huang, and T. Krishna, "Dnnfuser: Generative pre-trained transformer as a generalized mapper for layer fusion in dnn accelerators," *arXiv preprint arXiv:2201.11218*, 2022.

[284] N. P. Jouppi *et al.*, "A domain-specific supercomputer for training deep neural networks," *Communications of the ACM*, vol. 63, no. 7, pp. 67–78, 2020.

[285] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–12.

[286] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[287] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap, "Compressive transformers for long-range sequence modelling," *arXiv preprint arXiv:1911.05507*, 2019.

[288] K. Choromanski *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.

[289] N. Tesla, "V100 gpu architecture," *Online verfügbar unter http://images. nvidia. com/content/volta-architecture/pdf/volta-architecture-whitepaper. pdf, zuletzt geprüft am*, vol. 21, 2018.

[290] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2019.

[291] V. Sze, Y. Chen, T. Yang, and J. S. Emer, *Efficient Processing of Deep Neural Networks*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2020.

[292] H. Le *et al.*, "Flaubert: Unsupervised language model pre-training for french," *arXiv preprint arXiv:1912.05372*, 2019.

[293] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

[294] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The long-document transformer,” *arXiv preprint arXiv:2004.05150*, 2020.

[295] A. Yazdanbakhsh, K. Seshadri, B. Akin, J. Laudon, and R. Narayanaswami, “An evaluation of edge tpu accelerators for convolutional neural networks,” *arXiv preprint arXiv:2102.10423*, 2021.

[296] *Coral ai*, https://coral.ai/, 2020.

[297] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, “Data movement is all you need: A case study on optimizing transformers,” *arXiv e-prints*, arXiv–2007, 2020.

[298] *Nivdia fastertransforemr*, https://github.com/NVIDIA/FasterTransformer, 2021.

[299] *Boosting nvidia mlperf training v1.1 performance with full stack optimization*, https://developer.nvidia.com/blog/boosting-mlperf-training-v1-1-performance.-with-full-stack-optimization/, 2021.

[300] S. Shen *et al.*, “Q-bert: Hessian based ultra low precision quantization of bert,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 8815–8821.

[301] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, “I-bert: Integer-only bert quantization,” *arXiv preprint arXiv:2101.01321*, 2021.

[302] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, “Q8bert: Quantized 8bit bert,” *arXiv preprint arXiv:1910.06188*, 2019.

[303] W. Zhang *et al.*, “Ternarybert: Distillation-aware ultra-low bit bert,” *arXiv preprint arXiv:2009.12812*, 2020.

[304] H. Wang, Z. Zhang, and S. Han, “Spatten: Efficient sparse attention architecture with cascade token and head pruning,” in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2021, pp. 97–110.

[305] X. Jiao *et al.*, “Tinybert: Distilling bert for natural language understanding,” *arXiv preprint arXiv:1909.10351*, 2019.

[306] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, “Mobilebert: A compact task-agnostic bert for resource-limited devices,” *arXiv preprint arXiv:2004.02984*, 2020.

[307] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, “Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers,” *arXiv preprint arXiv:2002.10957*, 2020.

[308] T. Shen, T. Zhou, G. Long, J. Jiang, and C. Zhang, "Bi-directional block self-attention for fast and memory-efficient sequence modeling," *arXiv preprint arXiv:1804.00857*, 2018.

[309] N. Parmar *et al.*, "Image transformer," in *International Conference on Machine Learning*, PMLR, 2018, pp. 4055–4064.

[310] J. Qiu, H. Ma, O. Levy, S. W.-t. Yih, S. Wang, and J. Tang, "Blockwise self-attention for long document understanding," *arXiv preprint arXiv:1911.02972*, 2019.

[311] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[312] G. M. Correia, V. Niculae, and A. F. Martins, "Adaptively sparse transformers," *arXiv preprint arXiv:1909.00015*, 2019.

[313] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient content-based sparse attention with routing transformers," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 53–68, 2021.

[314] Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, Z. Zhao, and C. Zheng, "Synthesizer: Rethinking self-attention in transformer models," *arXiv preprint arXiv:2005.00743*, 2020.

[315] S. Wang, B. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[316] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International Conference on Machine Learning*, PMLR, 2020, pp. 5156–5165.

[317] K. Choromanski *et al.*, "Masked language modeling for proteins via linearly scalable long-context transformers," *arXiv preprint arXiv:2006.03555*, 2020.

[318] T. J. Ham *et al.*, "Aˆ 3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2020, pp. 328–341.

[319] T. J. Ham *et al.*, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2021, pp. 692–705.

[320] M. J. Wolfe, "Optimizing supercompilers for supercomputers," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1982.

[321] R. Allen and K. Kennedy, "Vector register allocation," *IEEE Computer Architecture Letters*, vol. 41, no. 10, pp. 1290–1317, 1992.

[322] G. Gao, R. Olsen, V. Sarkar, and R. Thekkath, "Collective loop fusion for array contraction," in *International Workshop on Languages and Compilers for Parallel Computing*, Springer, 1992, pp. 281–295.

[323] C. Ding and K. Kennedy, "Improving effective bandwidth through compiler enhancement of global cache reuse," *Journal of Parallel and Distributed Computing*, vol. 64, no. 1, pp. 108–134, 2004.

[324] K. Kennedy and K. S. McKinley, "Maximizing loop parallelism and improving data locality via loop fusion and distribution," in *International Workshop on Languages and Compilers for Parallel Computing*, Springer, 1993, pp. 301–320.

[325] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, "The tensor algebra compiler," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–29, 2017.

[326] *Tensorflow xla*, https://www.tensorflow.org/xla, 2021.

[327] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren, "Dnnfusion: Accelerating deep neural networks execution with advanced operator fusion," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 883–898.

VITA

Sheng-Chun (Felix) Kao was born on September 1st, 1922, in Taiwan. He received his Bachelor and Master Degrees Electrical and Engineering in 2017 from National Taiwan University. He received his PhD from the School of Electrical and Computer Engineering at Georgia Institute of Technology. He was advised by Professor Tushar Krishna, the principal investigator of Synergy Lab where Felix was a member.

Felix's research interest spans across machine learning and computer architecture. His research focus is on develop different machine learning technique to solve the DNN accelerator design problem including HW resource allocation and map space exploration.